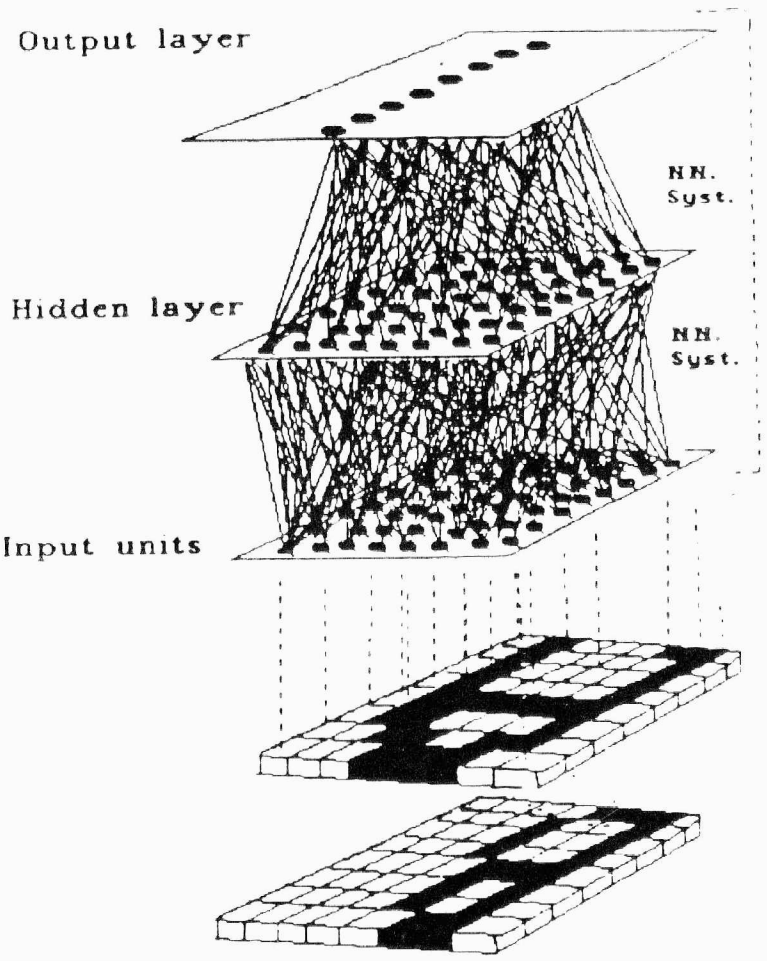
IDG

NEURAL NETWORK WORLD

International Journal on Neural and Mass-Parallel Computing and Information Systems

VOLUME 1 **1991** **NUMBER 6**



NEURAL NETWORK WORLD is published in 6 issues per annum by the IDG Company, Czechoslovakia, 160 00 Prague 6, Lužná 2, Czechoslovakia, the member of the IDG Communications, USA.

Institute of Computer and Information Science, Czechoslovak Academy of Sciences, 18207 Prague, Pod vodárenskou věží 2, Czechoslovakia.
Phone: (00422) 82 16 39, (00422) 815 2080, (00422) 815 3100

Fax: (00422) 85 85 789,

E-Mail: CVS15@CSPGCS11.BITNET

Price Information:
Subscription rate 399 US$ per annum.
One issue price: 66.50 US$.
Subscription adress: IDG Co., Czechoslovakia, 160 00 Prague 6, Lužná 2, Czechoslovakia.

Advertisement: Ms.Ing.H.Vančurová, IDG Co., Czechoslovakia, 160 00 Prague 6, Lužná 2
Phone: (00422) 34 78 95, Fax: (00422) 34 78 95.

Scanning the Issue

Papers:

Tutorial:

ISSN 1210-0552

HIERARCHICAL LEARNING IN NEURAL NETWORKS: A NEW PARADIGM WITH POSSIBLE APPLICATIONS TO AUTOMATED DATA PROCESSING

M.J. Whitcomb, M.F. Augusteijn[1]

Abstract: The application of neural network learning to automated data processing is explored. The requirements for learning methods in this domain are discussed. Two new learning methods, safisfying these requirements, are introduced. Both methods dynamically allocate the necessary number of hidden nodes. The first method is called the Flat Learning Procedure because it builds a single layer of hidden nodes. Performance of this procedure is compared with that of the Generalized Delta Rule. The Flat Learning Procedure requires less training but uses more hidden nodes. The generalization properties of this procedure are unsatisfactory. The second method, the Hierarchical Learning Procedure, builds a hierarchical structure of hidden nodes. This method is capable of learning a hierarchy of concepts. Due to this property it is able to generalize well while maintaining the favorable characteristics of the flat procedure with respect to fast learning. The introduction of these learning procedures is a first step towards the application of neural nets to automated programming.

Key words: *learning method, hidden node, hierarchical structure, automated programming.*

Received: July 23, 1991
Revised and accepted: December 9, 1991

1. Introduction

Currently, the demand for automated systems exceeds industry's ability to produce them. Since the introduction of the microprocessor, the trend towards further automation has accelerated and the burden of producing automated systems has shifted almost entirely to the software discipline. Unfortunately, there has been little gain in software development productivity. Progress has been made, but not enough to compensate for the increasing demand and shorter development times required. Against this background, any viable means to increase system development productivity would be very beneficial.

The recent resurgence of interest in artificial neural networks has given rise the notion of building automated systems through learning, instead of programming. If a machine could readily learn its intended functions, there could be a great potential for increasing the productivity of system development. Some recent progress in the development of learning procedures, such as the Generalized Delta Rule (GDR) [14], offers hope that useful systems might be built through learning instead of programming.

In order to devise a neural network strategy for this type of application, it is first necessary to determine what kind of organization a network would need if were to take on traditional types of data processing tasks. For that purpose, it is useful to consider the techniques that are currently used for modeling data processing systems. One such technique that is frequently used during requirements analysis and early software design is Structured Analysis [1] [5] [17].

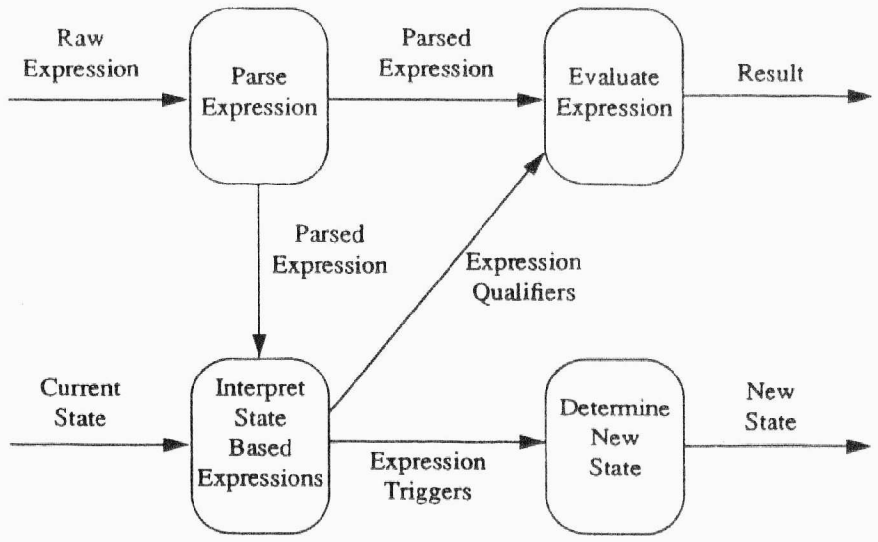


Fig. 1: *An example data flow diagram that depicts a data flow model*

Structured Analysis employs a data flow approach to describing the function of a system. Data flow techniques

[1]Mark J. Whitcomb, Marijke F. Augusteijn
Department of Computer Science, University of Colorado at Colorado Springs, Colorado Springs, Colorado

decompose a process into a series of transforms that are applied to the data as they proceed from input to output. A data flow diagram illustrating this approach is shown in Figure 1, where the individual bubbles represent the data transforms and the arcs between them show the flow of information. When using a data flow model for a system that has been decomposed into data transforms at a low enough level, pattern association neural networks could be considered to implement each transform. A pattern association network is a feedforward network in which neurons are grouped in layers. Each layer receives its inputs, makes decisions based on these inputs and passes the results to the next layer. If pattern association networks can be taught to perform the desired transformations, then potentially the function of an entire system could be taught to a machine rather than programmed.

Alternative network architectures may also be considered, such as feedback networks [7, 8], or competitive networks that use lateral inhibition [15, 6, 9], but the most straightforward neural network implementation of a data flow model is the feedforward pattern association. The pattern association model was used as the basis for deriving the learning paradigms presented here. There are other architectural issues that will also be important. For example, a data flow model represents a system in terms of data stores, as well as data transforms. Strategies for implementing data stores need to be discussed, but these are beyond the scope of this paper. This paper focuses on the requirements for learning strategies and methods meeting these requirements.

2. Requirements on the learning methodology

The following is a list of characteristics which were deemed necessary or desirable for a learning strategy suitable for traditional types of automated systems. Some items are actually simplifications allowed in this application.

1. Only binary inputs and outputs need to be considered. For traditional automation, this is reasonable since almost all current automation tasks are performed with binary data.

2. The procedure must learn exact mappings from inputs to outputs. Furthermore, it must be capable of learning arbitrary mappings from binary inputs to binary outputs. Precision is required for most traditional automation tasks. Thus, the learning will be supervised with a trainer specifying all input/output pairs.

3. The procedure should converge to a solution rather quickly. An important consideration is the order in which training time grows with the size of the problem. A learning procedure in which the cost to learn an arbitrary transform does not remain tractable when scaled up is practically useless.

4. The procedure should be able to support incremental changes to the training set, in order to enable step-wise refinement of the system's function. The alternative would be to completely retrain the network whenever any change (large or small) needs to be made, which seems undesirable.

5. The procedure should converge to a solution regardless of the initial state. To facilitate step-wise refinement, a network may need to unlearn the effects of earlier training and converge to a new solution.

6. The procedure should be able to dynamically add hidden nodes as required to support greater degrees of complexity. The need to predetermine the amount of resources to solve a given problem leads to an unnecessary and difficult constraint on the trainer.

7. The procedure should be able to generate solutions in which the number of required hidden nodes remains reasonable. The rate at which the number of hidden nodes increases with respect to the size of the problem is a driving consideration in whether the learning procedure is practically useful or not.

8. The procedure must be capable of generalizing in such a way as to learn the underlying inten behind the training set with a minimum number of training pairs. The size of the required training set must grow at a tractable rate with respect of the size of the problem.

Two new methods were developed, specifically designed for the automated data processing application. In each of these techniques, the input and output layer are prespecified but the internal structure of hidden nodes is built during learning. The first method is called the flat learning procedure because it builds a single (flat) layer of hidden nodes. The main features distinguishing the new learning methods from existing strategies were developed during the design of this procedure. However, the generalization characteristics of the flat learning procedure were found to be rather weak. The second method was then developed to specifically improve on generalization. It is called the hierarchical learning procedure because it organizes its hidden nodes in a hierarchical manner.

3. The flat learning procedure

A basic idea behind the flat learning procedure is to create and train a set of hidden nodes so that each of them learns a linear separation function in the tradition of the perceptron [13, 11]. Since an arbitrary number of hidden nodes can be allocated during training, the entire set will be capable of learning any separation function which can be approximated by linear segments. A major advantage is that a fast converging variation of the delta rule can be used to update the weights. This variation will be referred to as the "Greedy Delta Rule". In order to achieve this, the input set needs to be partitioned into

linearly separable subsets. Each hidden node must learn a single subset. A separation mechanism has been designed that accomplishes this partitioning and, in addition, aids in the generalization properties of the net.

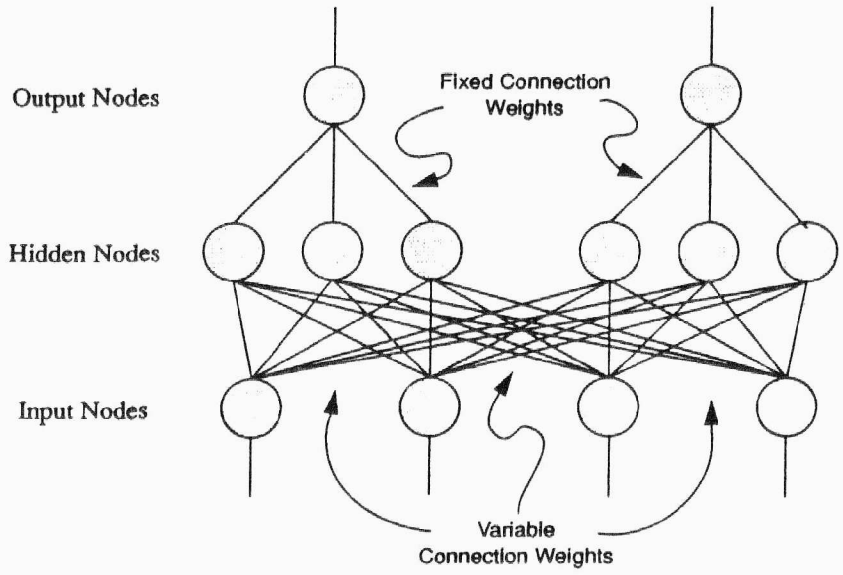


Fig. 2: *Architecture for the Flat Learning Procedure*

The specific architecture used by this learning procedure is shown in Figure 2. Each output node has its own independent pool of hidden nodes and each hidden node is completely connected to the inputs. The output node essentially applies an OR operation to its hidden nodes. When the desired output of a node equals 1, one of its hidden nodes must be taught to produce a 1. When the desired output equals 0, each of its hidden nodes must produce a 0. In this, a mapping from binary inputs to binary outputs is learned by constructing a piece-wise linear approximation of the required function, with each hidden node learning one linear segment. The connection weights between hidden and output nodes are all fixed and are generally set to 1.

The weights on the connections between input and hidden nodes are determined through learning. During the learning phase, the training pairs are presented in an incremental fashion to support step-wise refinement of the system's functionality. The main features that enable the required learning are discussed in the following subsections. Several small improvements to the essential ideas were discovered during the extensive testing of this procedure. Not all of them are discussed in detail but the complete algorithm, as used in the experiments, is given in subsection 3.6.

3.1 Incremental Training Presentations

The learning procedure should be capable of incremental modification of the system data transforms. It should not be necessary to present all required training pairs together since it will often be the case that the system functionality will need to be expanded a little bit at a time. With an incremental training order, the presentation of each new training pair is followed by the presentation of all previously learned pairs until no more errors occur. This presentation order can be stated more precisely as follows:

1. Partition the training set into two parts *old* and *new*. The *old* set consists of all training pairs that have been presented before and is initially empty. The *new* set consists of all training pairs that the network has never seen, and initially consists of the entire training set.

2. Select a training pair from the *new* set and present it to the network. If the *new* set is empty, learning is complete.

3. If the network correctly produces the specified outputs from the given inputs, remove the new training pair from the *new* set, place it in the *old* set and repeat step (2).

4. If the network does not produce the correct outputs, initiate learning for this pair. After this pair has been learned, cycle through all training pairs in the *old* set (in the same order as originally placed in the set) and initiate learning for each pair that produces an error. If any pairs from the *old* set resulted in an error, present the new pattern again and repeat this step until no more errors occur. When this has been accomplished, remove the new training pair from the *new* set, place it in the *old* set and repeat step (2).

A disadvantage of this approach is that the network is only slowly exposed to a large portion of the training set. It cannot, therefore, anticipate the impact of patterns later in the set. But, for this application, all the training pairs must be learned exactly, and the training set is, therefore, assumed to be noiseless. Given those conditions, the use of an incremental presentation order should not be detrimental.

3.2 The Greedy Delta Rule

Generally, when the Generalized Delta Rule is used to implement learning, the network will not be allowed to move quickly in any particular direction. The weight manifold (or equivalently, the error function) may be quite complex and a fast learning algorithm may not converge to the correct solution. However, when each node only learns linearly separable patterns, a much more greedy approach can be taken. In this case, the weight manifold will only contain a single global minimum. The learning procedure can then be made to move rapidly towards that minimum.

The Delta Rule was modified to adjust the weights for each training pair so that the output will be correct for that pattern after a single iteration. The new procedure does not use a learning rate parameter. Given the desired output of a hidden node, the amount that each weight will be adjusted is computed from:

$$Dw_i = ((desired - output - actual - output)\, x_i)/ \\ number - of - input - ones$$

where Dw_i is the change in weight from input i to the hidden node, the *actual-output* equals the current activa-

tion of the hidden node, x_i is the value of the input from node i, and *number-of-input-ones* is the sum of all the x_i values in the current pattern. Thus, the weights are adjusted enough to bring the activation to the exact value desired for that one pattern.

Another characteristic is that whenever a node produces the correct output, regardless of its activation value, the weights are not modified. The goal here is not to find a subtle separation function that maximizes the number patterns that are correctly classified by a single hidden node, but to rapidly and deterministically achieve a correct output for the entire network.

3.3 Separation Mechanism

The separation of patterns across different hidden nodes serves two purposes. One is to distinguish the patterns that are linearly inseparable, the other is to control generalization. An important criterion in the partitioning is similarity between patterns. Input patterns that are similar to each other should be learned by the same hidden node. The mechanism to accomplish this is lateral inhibition. Lateral inhibition serves to select the hidden node that has the greatest activation.

Without an adequate separation mechanism, some poor types of generalization may occur. As an example, consider the input pattern (0 1 1 0 0 0) and (0 0 0 1 1 0) which both correspond to an output (1). If these patterns would be allocated to the same hidden node, it would dutifully learn to produce a (1) output for the specified inputs as well as for all cross combinations of these patterns. As an example, the pattern (0 1 0 1 0 0) would also produce a (1) output.

This form of generalization does not seem appropriate since the original two patterns are completely disjoint. This is an example of over-generalization that should be avoided. However, if the two training pairs had been more similar the same type of generalization would have been reasonable.

The mechanism for separating patterns is based on a measure of their similarity. The activation value of a hidden node will be used as the similarity measure. It should be noted that not all linearly inseparable training sets involve disjoint patterns. However, the activation of a hidden node depends on the weights as well as on the inputs, and the weights can be modified so that patterns that need to be separated will appear dissimilar. Consider the following training set:

input: (0 0 1 0 0)	output:	(0),
input: (0 1 1 0 0)	output:	(1),
input: (0 1 1 1 0)	output:	(0),
input: (0 0 1 1 0)	output:	(1).

The second and fourth training pairs are not disjoint and they produce the same output pattern, so it seems reasonable, from a generalization standpoint, to assign them to the same hidden node. If they combined together, however, the entire training set will not be separable. But before the fourth training pair is presented to the network, the third one will introduce some inhibition on the fourth bit. That inhibition will lower the activation value for the fourth training pattern, so that it will no longer appear to be very similar to previously learned patterns.

Based on this type of reasoning, a threshold mechanism has been defined to separate input patterns, both when they are non-linearly separable and to control generalization. Each hidden node has a unification threshold which determines whether a new pattern will be unified with those already represented by a hidden node. When a new pattern is presented to the net, all hidden nodes with activations exceeding their unification threshold enter a competition. This competition has been implemented through lateral inhibition. The winning hidden node is the one selected to learn the new pattern.

If the patterns previously learned by all hidden nodes are not similar enough to the new input (their activation does not exceed their unification thresholds), none of the hidden nodes will enter the competition. When this occurs, a new node must be allocated. Allocation of a new node can occur naturally with this mechanism. If a pool of unallocated nodes exists initially, with each one having its inification threshold at 0 and its weights to the inputs as small random values, all hidden nodes will initially enter a competition. Their initial activation will be small, so they will be unlikely to win any competition unless no other nodes have activations that exceed their unification thresholds. In this manner, a previously unallocated node will win a competition and become part of the solution when a new input pattern is not sufficiently similar to any previously learned input patterns.

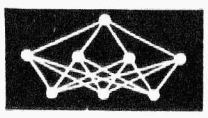
As a consequence of this separation mechanism, the learning procedure will not always derive the smallest network possible. The approach consciously separates patterns that are fairly dissimilar in order to control generalization. In essence, the learning procedure is more concerned with generalization than with minimization. The manner in which the unification threshold is set and adjusted is as follows:

1. Initially, the unification threshold for all hidden nodes is 0.

2. When a node learns to produce a 1 output for any input pattern, its unification threshold is increased by some percentage of the total error; that is output threshold minus activation value.

3. The unification threshold is never lowered, except when a node is deleted, as described later. On deletion, the unification threshold is set back to 0.

In this manner, unification will require a greater degree of similarity as more patterns are learned by the same hidden node. Thus, the domain of a single node becomes increasingly more focused around certain input patterns that are very similar.



3.4 Short-Cut Inhibition

Inhibitory connections, required when a hidden node produces a 1 while a 0 is desired, can be learned in a short-cut fashion. An example will illustrate this feature. Consider the training pairs:

input: (0 0 1 1 0 0) output: (1),
input: (0 0 1 1 1 0) output: (0).

Because of the similarity of the inputs, the second pattern will initially produce the wrong output. In ordeer to learn the correct result for both patterns, the third and fourth input connections must acquire excitatory (positive) weights and the fifth connection must obtain an inhibitory (negative) weight. These connections can be learned in a single pass through the training set if, on presentation of the second training pair, the weight for the fifth bit will be the only one manipulated (lowered to a negative value).

In general, to learn a 0 output, if any of the weights are zero (or very small) and an input is present, those weights can adjusted down immediately, without affecting much of the previous learning. As a matter of fact, this is the only reliable way that was found to turn a node off when the separation mechanism described previously is used. The basic problem is that the meaning of similarity becomes unclear when other weights are lowered. If the unification threshold were to be lowered when the weights are lowered, it could happen that patterns would be unified that should not. This could occur because a lower threshold allows a much lower degree of similarity for unification. On the other hand, if the original value of the unification threshold were to be maintained, previously learned patterns could sometimes not be learned anymore by this node. When only the weights with zero values are lowered, most previously learned patterns will not be affected and the unification threshold need not be changed.

3.5 Node Deletion

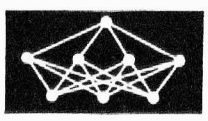
There will be cases in which there are no unassigned weights that can be lowered when a node needs to be shut off. Then, it will not be possible to learn the training pairs in the order presented. The solution is to delete the node that could not be inhibited and reorder the training set so that the training pair requiring the inhibition is now presented first. Incremental training presentations can then be restarted from the beginning.

There is an additional benefit to the ability to reorder the training set. If there are contradictions in this set, they will also result in deletion and reordering. Then, the contradictory patterns will become the first two training pairs in the reordered set. Once both contradictory training pairs have been moved to the beginning, it is easy to recognize them.

3.6 The flat learning algorithm

The flat learning procedure can be summarized as follows:

1. Initialize the net with no hidden nodes. The input and output nodes are not yet connected.
2. Present the training pairs in the incremental fashion described. (In the case of reordering, remove all patterns from the *old* set and put them back into the *new* set.)
3. For each training pair, and for every output bit:
   (a) If the desired output is 1:
      i. If the actual output is 1, do nothing.
      ii. If the actual output is 0, try to identify the hidden node, aassociated with the output, that has the greatest activation and whose activation is greater than its unification threshold.
         A. If identification is successful, increase all weights on the connections receiving a 1 input of the identified node by equal amounts until its activation is greater than or equal to the output threshold (Greedy Delta Rule). Increase its unification threshold as described.
         B. If identification is unsuccessful (no hidden nodes have an activation greater than their unification thresholds), create a new hidden node. Connect this node with the corresponding output using a fixed weight. Also, connect this node to all inputs and proceed as if identification had been successful with the newly created node as the identified node.
   (b) If the desired output is 0:
      i. For all hidden nodes associated with the output, if the activation is less than the output threshold:
         A. If the activation is less than the unification threshold, do nothing.
         B. If the activation is greater than or equal to the unification threshold, but less than the output threshold, increase the unification threshold until it is greater than the activation. This is to prevent possible future input patterns that are similar to the current input pattern and that require a 1 output from being learned by the hidden node and invalidating the 0 output for the current pattern.
      ii. If the activation is greater than or the same as the output threshold, and one or more unassigned (zero or very small) weights exist



Problem	Generalized Delta Rule			Flat Learning Procedure		
	learning_ rate Parameter	Number of Hidden Nodes	Presentations/ Iterations	*similarity* Parameter	Number of Hidden Nodes	Presentations/ Iterations
"xor"	0.5	1	2232/ 558	50%	2	10/ 2.5
4 bit odd parity	0.5	4	45200/ 2825	50%	8	239/14.9
6 bit symmetry	0.1	2	77312/ 1208	50%	8	1072/16.8
3 bit negation	0.25	3	<80000/<5000	50%	6	62/ 3.9
2 bit add	- couldn't reliably solve			50%	8	103/ 6.4

Tab.1: *Comparison of the performance of the Generalized Delta Rule versus the Flat Learning Procedure*

where the inputs are 1, lower those weights until the activation is below the unification threshold.

iii. If the activation is greater than or the same as the output threshold, but no unassigned weights exist where the inputs are 1, delete the node. Reorder the training set so that the new training pair is at the beginning. Start the incremental training presentations over again.

3.7 Results and Analysis of the flat learning procedure

The flat learning procedure was tested on a set of familiar bit pattern manipulation problems. The results of these tests are shown in Tables 1 and 2. Table 1 compares the results of flat learning with those of the GDR procedure as published by [14]. For the GDR, the learning-rate parameter (which controls the size of each increment in the gradient descent process) is a measure of the speed of convergence of this procedure. The flat learning procedure does not use a learning-rate parameter and the similarity parameter is used instead. The similarity parameter specifies the rate of growth of the unification threshold. A 50% value means that when two input patterns unify, they must share at least half of each other's 1 bits. A lower value would allow more unifications to take place and fewer hidden nodes might be allocated. The drawback is that generalization can be poorer and separation takes more time to accomplish. The 50% value was experimentally found to be adequate for the test cases in Tables 1 and 2.

Table 1 compares the number of hidden nodes for both methods. This number is prespecified for the GDR and derived during learning for the flat learning procedure. It is seen that the number of hidden nodes for the GDR is less in all cases. This is to be expected since the flat procedure does not explicitly try to minimize the number of hidden nodes. It should be noted, though, that the solutions obtained by the flat procedure for the symmetry and parity problems show combinatorial growth in the number

Problem	Number of Hidden Nodes	Number of Deletions	Presentations/ Iterations	Prediction Accuracy
or	2	0	7/ 1.8	50%
and	1	0	7/ 1.8	75%
xor	2	0	10/ 2.5	25%
4 bit symmetry	4	0	106/ 6.6	25%
6 bit symmetry	8	0	1072/ 1[illegible]	50%
8 bit symmetry	16	0	10456/ 40.8	69%
4 bit odd parity	8	1	239/ 14.9	6%
8 bit odd parity	128	94	111909/ 437.1	2%
3 bit negation	6	0	62/ 3.9	50%
2 bit add	8	0	103/ 6.4	50%
3 bit add	17	0	1027/ 16.0	59%
4 bit add	42	8	19251/ 75.2	64%
2 bit subtract	8	1	158/ 9.9	19%
3 bit subtract	17	2	1500/ 23.4	38%
4 bit subtract	41	9	21031/ 82.2	56%

Tab.2: *Complete test results for the Flat Learning Procedure*

of hidden nodes. This is a result of the piece-wise linear solution that the network constructs.

The performance of the two procedures is also compared with respect to the total number of presentations of training pairs required for adequate learning. The GDR cycles through the entire set until the total error for the entire training set has become acceptable. The flat learning procedure follows the incremental presentation order, so the term *iteration* is not that meaningful. For comparison purpose, an *iteration* is defined as the total number of presentations divided by the total number of training pairs.

All of the training sets listed in Table 1 contained all possible input patterns for the specific problem. In all cases, these training sets were ordered according to the binary counting sequence for the input patterns. It is evident that there is a significant speed improvement using the flat learning procedure over the GDR. It should also be noted that the results obtained by the flat procedure were not prone to local minima. The GDR, on the other hand, did not easily avoid local minima when attempting to solve the 2 bit add problem.

Table 2 shows the complete set of results for the flat learning procedure. In all cases, the similarity parameter was set to 50% . The number of deletions listed is the number of times the learning procedure was forced to delete a hidden node and restructure the training set. The prediction accuracy column indicates the percentage of the total number of training pairs for which the network guessed the correct output, without modifying any weights, at the first presentation of that input pattern. Although not listed, a training set including a contradiction was also presented and the contradiction was recognized as expected.

It is obvious from these results that the generalization properties of the flat learning procedure are not strong. For the smaller problems, the prediction accuracy is not a reliable measure, but for the larger problems, this accuracy is probably the best measure of performance. It is seen that the prediction accuracy is increasing with the problem size (except for the parity problem), but it is not increasing nearly as fast as the number of possible input combinations. As an example, the prediction accuracy improves only 5% from the 3 bit add problem to the 4 bit add problem, but the number of possible input patterns increases by a factor of 4.

4. The hierarchical learning procedure

Structured learning is commonly perceived in humans. Lower level concepts are learned first and then used for the acquisition of higher level concepts. If this type of learning could be simulated by a neural network, a learning method requiring fewer training pairs and showing stronger generalization properties can result. As an example, consider the addition problem. If a learning procedure could uncover the hidden, hierarchical carry function, it could learn the solution to the addition problem from fewer training pairs. In more general terms, it would be of great benefit if a neural net could learn a hierarchy of concepts.

The hierarchical learning procedure that will be presented is an attempt to model structured learning. It should be noted that, in the case of an artificial neural network, all learning information is represented by the training set without any other contextual clues that could aid the learning process. The two aspects of a training set that can be exploited to support hierarchical learning are the content of the set and the order in which the training pairs are presented.

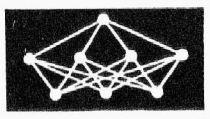
In general, the training set can be structured so that related concepts are taught together and low level concepts are taught first. The restriction that all related concepts be taught together is a mechanism to provide context. That limited type of context can be a great help in deciding how to organize a solution by providing some clues as to which concepts belong together. The restriction that low level concepts be taught before high level ones is obvious. For higher level concepts to be properly learned, they should be based on all appropriate lower level concepts. If such is not the case, generalization (understanding) will likely be poor, even for humans.

The hierarchical learning procedure is in many aspects similar to the flat learning method, but it has better generalization properties. Like the flat learning procedure, it uses the notion of the incremental training presentation order, the Greedy Delta Rule to learn the excitatory weights to the hidden nodes, a unification threshold to measure the degree of similarity between patterns, and the idea of only changing free (zero) weights when a hidden node needs to be turned off. The incremental presentation order is particularly important to the notion of hierarchical learning. With this order, it can be ensured that low level concepts are completely learned before any higher level concepts are introduced.

The architecture of the hierarchical learning procedure is, however, quite different from that of the flat learning procedure. The architecture is characterized by a multilevel, self-organized structure. The learning procedure creates a network with a hierarchically arranged set of hidden nodes whose organization is completely determined during learning. Figure 3 shows the essence of this architecture.

Each hidden node learns a concept, based on the network inputs and the outputs from lower level hidden nodes (which represent lower level concepts). In this manner, hidden nodes are visible to higher level hidden nodes and the output nodes, but not to any lower level hidden nodes. It is important to note that the hidden nodes are not arranged in layers but form a hierarchical sequence. Each hidden node may have connections to all inputs and to the outputs of lower lever hidden nodes, but a hidden node does not have any input connections from higher level hidden nodes. This network architecture is strictly feedforward.

Contrary to the flat architecture, each output node does not have its own pool of hidden nodes. A hidden node may connect to all outputs. The connection weights between the hidden and output nodes are set when a hidden node



learns its first pattern (and are not all equal to 1 as in the flat procedure). The values of these weights will be determined by the Greedy Delta Rule. Once a hidden node has learned its first pattern, the weights from it to the output nodes will not change.

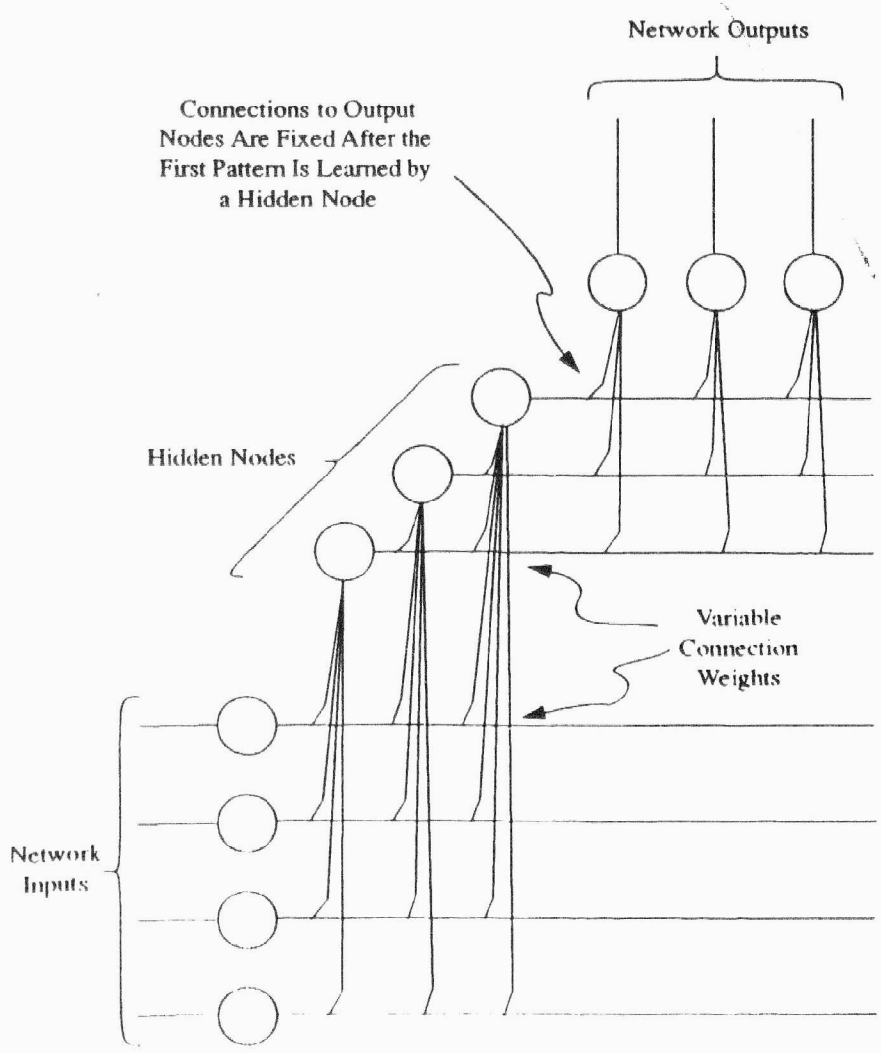


Fig.3: *Architecture for the Hierarchical Learning procedure*

4.1 Building Internal Concepts

When an input pattern is presented, adjustments will be made unless the corresponding outputs are already produced by the net. If adjustments are necessary, the input pattern will either be unified onto an existing hidden node, or a new hidden node will be created. A pattern will only unify onto a hidden node if that node is not already firing, the node's activation is greater than its unification threshold, and the output of the entire network would be correct if the hidden node were to fire. Alternatively, if a hidden node is firing and shutting it off would make the output exactly correct and there is a free connection (with zero weight) that could be used to inhibit the hidden node, the inhibitive connection will be learned and the hidden node shut off.

If neither of these options exist for any hidden node, a new node will be allocated directly above all other hidden nodes. This new hidden node will be connected with the net's inputs, the outputs of the existing hidden nodes, and the output nodes of the net. Allocation of a new node is called discrimination, because the new pattern represents what is deemed to be a new concept (since it couldn't unify onto any existing hidden node), and the new node discriminates the new concept from all previously learned concepts.

The weights between the new hidden node and the output nodes will be adjusted so that all outputs will be correct for the presented input. The output nodes also have biases which may be adjusted if needed. In particular, if an output node is asked to produce a 1 when the inputs are all Os,the bias for that output node will be adjusted to accomplish this goal.

Whether unification or discrimination has taken place, only the input weights to the hidden node that learns the new pattern will be adjusted. There are two types of input connections, those to the inputs of the net and those to the outputs of lower level hidden nodes. In general, not all of them will be adjusted and the decision about which weights will be changed is crucial to hierarchical concept building. An example will demonstrate this. Consider the training pairs:

input: (0 0 1 1 1 0) output: (0 1 1 0),
input: (1 1 0 0 0 0) output: (0 1 0 1),
input: (1 1 1 1 1 0) output: (1 0 0 0).

Figure 4 shows the net that will built during presentation of this set. After presentation of the first two pairs, the network will have learned two distinctly different concepts by two different hidden nodes. The third pair should also create a new hidden node since both existing hidden nodes will fire, but the output will be incorrect. The question is which of the connection weight should be adjusted. Since the two hidden nodes will both be firing, the options are to adjust the weights to the network inputs only, adjust the weights to the outputs from the two lower level hidden nodes only, or to adjust the weights to both the network inputs and the outputs from the two existing hidden nodes.

If the training set has been properly specified, the third training pair should be interpreted as a new concept built from the lower level concepts represented by the first two training pairs. This will be the case because the third pattern is a combination of the two previous patterns. Therefore, the outputs from the first two hidden nodes should be used to help discriminate the third concept. Furthermore, the weights from the first two hidden nodes should be the only ones adjusted. If the third hidden node would be allowed to establish excitatory weights directly to the network inputs, then there would be no hierarchical building of concepts.

Experimentation with this learning procedure has shown that generalization is improved by adjusting the weights as described. Once one or more concepts have been unified together (represented by one hidden node), only one training pair with one of those original patterns needs to be presented to represent all of the patterns in the unified concept. In this way, higher level concepts can be made to apply to an entire set of lower level patterns by presenting only one representative pattern.

The desired hierarchical structure will be built if new high level hidden nodes adjust their weights to only the

highest level hidden nodes (or inputs) that are firing immediately below them. New hidden nodes should not learn associations to any concepts or to the network inputs, unless they represent the highest level concepts. To achieve this result, some mechanism is needed to restrict which weight adjustments a hidden node will be allowed to make.

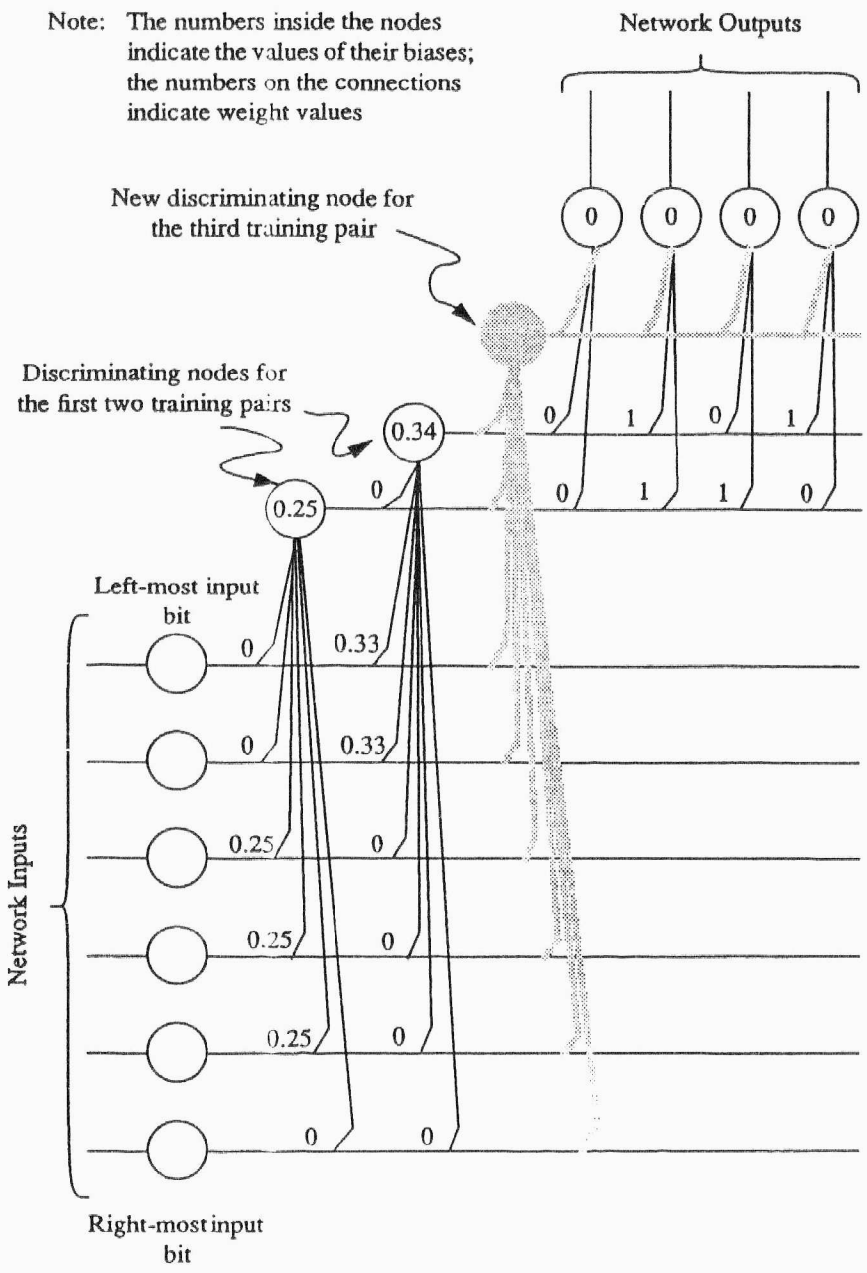


Fig.4: *Network resulting presentation of the first two training pairs in the example training set*

4.2 The Occlusion Mechanism

The mechanism devised to restrict which weight adjustments can be made has been termed an occlusion mechanism, because it occludes, or hides, certain network inputs or hidden node outputs from other hidden nodes. The occlusion mechanism is derived from the simple notion that if a hidden node is firing, no new associations should be made to the inputs that led the hidden node to fire. A new association is made when a zero connection weight is initially adjusted to some non-zero value. The occlusion mechanism, therefore, controls when zero connection weights can be adjusted. Connection weights that are already non-zero can be adjusted as needed. In this manner, the occlusion mechanism forces higher level hidden nodes to learn new associations only with the highest level concepts below them.

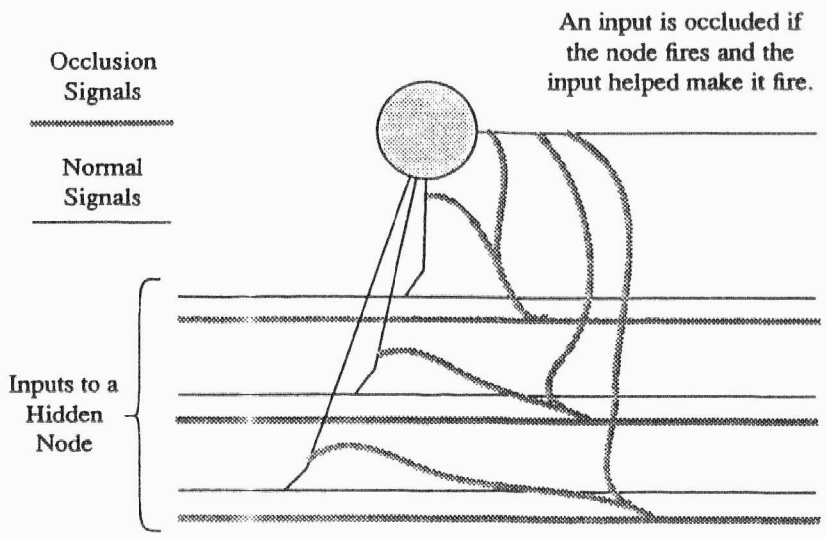


Fig. 5: *Mechanism for occlusion of input signals*

Figure 5 depicts a means to implement the occlusion mechanism. An extra signal line follows each input and each hidden node's output. These lines flag which inputs and hidden node outputs cannot be used to establish any new associations. When a hidden node fires, its output is also to activate the occlusion signal line for any of its inputs wherever an excitatory connection exists between that input and the hidden node. In this way, the inputs to a hidden node that caused it to fire can be kept from establishing any new associations.

It should be realized that the occlusion mechanism does not stop the inputs or the hidden node outputs from firing higher level hidden nodes. The output of the network will be the same whether the occlusion mechanism exists or not. The mechanism is only used to determine which input weights should be adjusted and which ones should remain unaffected in learning a new pattern.

There are three additional aspects to the occlusion mechanism. The first one arises from the possibility that a hidden node may fire, while all of its inputs are occluded. In this case, the output of the hidden node should also be occluded. This possibility may happen when independent patterns are learned by a network. It is possible for multiple hidden nodes that were established independently to end up occluding all the inputs to another hidden node. In this case, the hidden node with all of its inputs occluded is not providing any additional information for discrimination purposes. The other concepts should be the highest level concepts for the purposes of establishing new associations. Thus, a hidden node should occlude its own output in the case that all of its inputs are occluded.

The second aspect occurs when several input patterns have been unified onto a hidden node and a newly created node finds all its inputs occluded except for the output of that particular hidden node. In this case, it can only establish a connection to this hidden node. With this single connection present, the new hidden node will not be able to discriminate input patterns from those causing the previously existing hidden node to fire. More than one hidden node must be unoccluded in order to discriminate patterns. As an example, consider the training set:

input: (1 1 1 0 0) output: (0 1 1),
input: (0 1 1 1 0) output: (0 1 1),
input: (1 1 1 1 0) output: (0 0 1).

The first two inputs unify onto a single hidden node and the third pair creates a second hidden node that can only connect to the first one. The third pattern cannot be discriminated from the previous two. In order to solve this problem, the occlusion mechanism was modified. This mechanism was made selective, such that only the inputs to a hidden node that are minimally necessary to cause that node to fire will be occluded. In the example, one of the input connections to the second hidden node will then not be occluded and the third pattern can be discriminated from the first two.

The third and final aspect of the occlusion mechanism follows directly from the selective mechanism just discussed. Experimentally, conditions were found in which the selective occlusion mechanism could force the learning procedure into an infinite loop. If inputs to a hidden node are selectively not occluded and they are subsequently used to inhibit another hidden node, it is possible for the inhibition to keep a hidden node from ever firing. That would in turn force the creation of a new discriminator hidden node, which is subsequently inhibited, and so on. An example of this situation is given by the following training set:

input: (0 0 1 1 0) output: (0 1),
input: (0 0 0 1 1) output: (0 1),
input: (0 1 0 1 1) output: (1 0),
input: (0 1 1 1 1) output: (0 1).

The resulting network (taking the selective occlusion mechanism into account) is illustrated in Figure 6. In this example, the first two patterns unify together in the first hidden node. The third pattern results in a second hidden node that learns connections to the second input bit and to the first hidden node. When the fourth training pair is presented, the second hidden node will inhibit itself as a means to produce the desired output. Unfortunately, the selective occlusion mechanism does not know which of the input bits in the first two patterns should not be occluded. Based on the Greedy Delta Rule, it sees larger weights for the first pattern and, therefore, selectively does not occlude the fifth bit (from the second pattern). The second hidden node will then inhibit itself, based on the presence of that fifth bit. But, of course, the fifth bit will inhibit the second node all of the time.

The solution is to modify the selective occlusion mechanism to only partially occlude inputs that are not minimally necessary for a node to fire. Inputs that are *partially occluded* are allowed to have excitatory connections established, but not inhibitory connections.

4.3 Node Deletion

The need to delete nodes occurs when the output of the network is not correct and only a single hidden node output has not been occluded. In this case, the new pattern cannot be discriminated and the network has overgeneralized. In response, the network deletes the lowest level hidden node that is currently firing and all hidden nodes above it. The new training pattern is moved to the front of the training set and training is resumed.

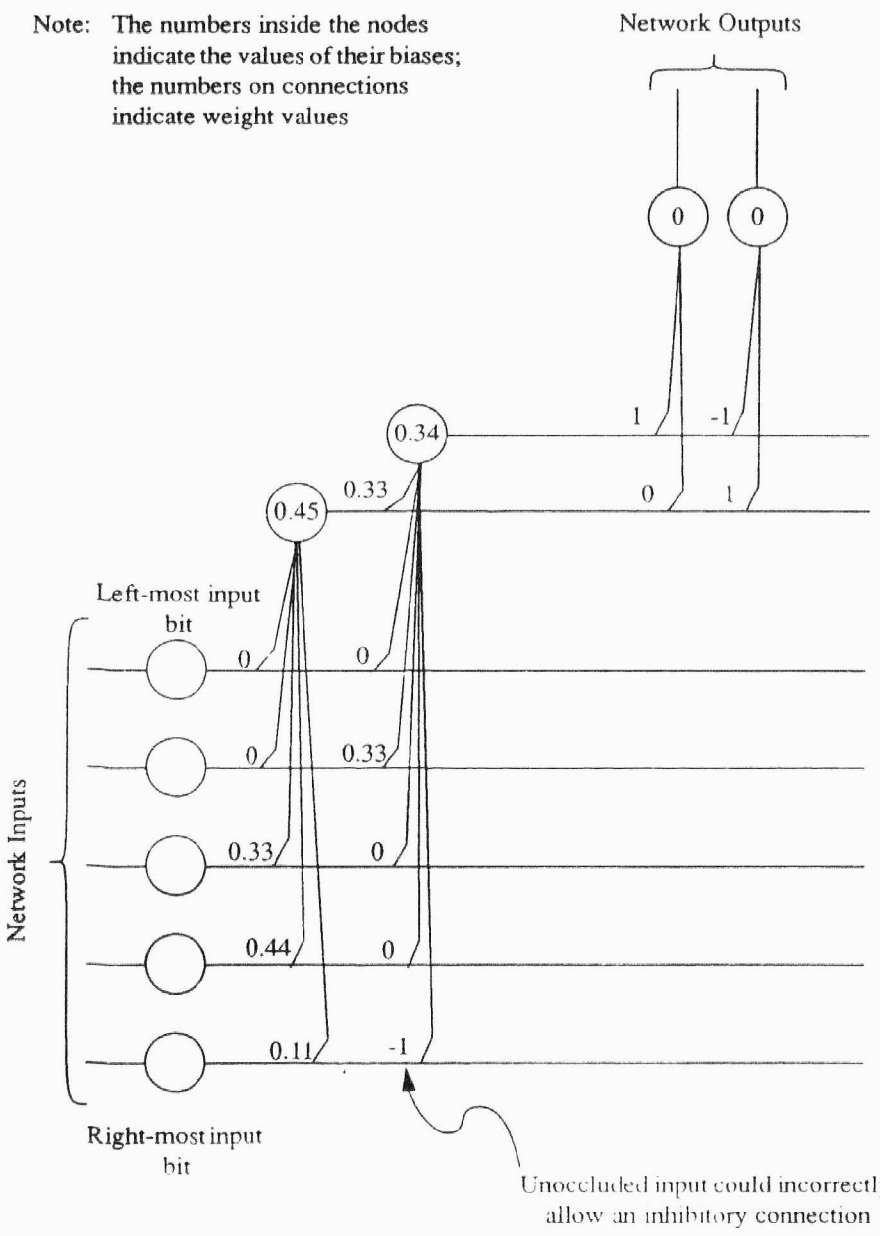


Fig.6: *Erroneous use of selective occlusion mechanism to establish inhibitions*

It should be noted that, like the flat learning procedure, the node deletion and reordering process leads to the easy detection of contradictions. Contradictory training pairs will again end up as the first two pairs in the reordered training set, where they are easy to spot.

4.4 The Overall Learning Procedure

The overall learning procedure can be summarized as follows:

1. Initially, set up the network with no hidden nodes. The output nodes have no input connections, and their bias is set to 0.

2. Present the training pairs in an incremental fashion with the possibility for reordering as in the flat learning procedure.

3. For each training pair:

   (a) Compute the entire network output and all occlusion signals.

   (b) If the network produces the correct outputs, do nothing.

   (c) If the network does not produce the correct outputs:

      i. If the highest level hidden node that has been allocated is active, and if the output would be correct if it were shut off, and if one or more inputs to the node are present and not occluded or partially occluded that have zero weights:

         A. Equally decrease the weights to all those free inputs until the activation of the node reaches 0.

      ii. Else if the highest level node that has been allocated is not firing, but its activation is greater than its unification threshold, and if the output of the network would be correct if the node were on:

         A. Equally increase all existing excitatory weights to inputs that are present and all previously 0 weights to all unoccluded inputs that are present, until the activation of the node is greater than or equal to its output threshold.

         B. Increase the unification threshold for that node by a preselected percentage of the total error.

      iii. Else if any input is unoccluded or if 2 or more hidden node outputs are unoccluded:

         A. Allocate a new hidden node immediately above all other hidden nodes in the hierarchy. Set its unification threshold to 0, set all of its input weights to 0. Set all of the weights between it and the output nodes to 0.

         B. Equally adjust the weights for all unoccluded inputs to that new hidden node until the activation of the node is greater than or equal to its output threshold.

         C. Increase the unification threshold for the hidden node by a preselected percentage of the total error.

         D. Equally adjust the connection weights from the new hidden node to all output nodes until the activation of all output nodes is exactly as desired.

      iv. Else:

         A. Delete the lowest level hidden node that is firing and all hidden nodes above it in the hierarchy.

         B. Reorder the training set so that the new training pair is at the beginning. Start the incremental training presentations over again.

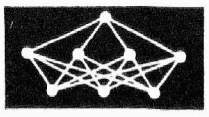
4.5 Results and Analysis

For this learning procedure, two different types of test sets were run. The first type contained all of the possible input combinations in binary counting order. This corresponds to exactly the same test data used for the flat learning procedure. The second type of training set contained just those training pairs that were minimally necessary (and in the order necessary) to teach the network the underlying cencepts.

Table 3 lists the results from the unstructured training sets, and cross lists them with the results for the flat learning procedure. Table 3 also lists some additional tests. The similarity parameter that governs the modification of the unification threshold was again set to 50% . The number of deletions shows the number of times that the network was forced to delete one or more nodes and reorder the training set. The number of training iterations is defined as the total number of training presentations divided by the total number of training pairs in the training set. Finally, the prediction accuracy column indicates the percentage of the total number of training pairs for which the network guessed the correct output at the first presentation of the corresponding input pattern.

From this table, some definite improvement over the flat learning procedure can be seen in the prediction accuracy in almost all cases. Only the 2 bit add problem is slightly worse. Also, the training times are generally shorter. This general improvement is also reflected in the very small number of deletions.

The hierarchical learning procedure often generated more hidden nodes than the flat learning procedure. In particular, the subtraction problem generated many more hidden nodes. In this case, the training procedure traded resources for better prediction accuracy and greater speed. The training sets were not well enough structured, however, to uncover the minimum solution. In the case of the symmetry problem, the number of hidden nodes decreased to a linear function of the number of bits in the problem. This is obvious from the type of solution generated, since it is now able to create building blocks that can be combined together to eliminate the combinatorial expansion. The solution to the 8 bit parity is somewhat better than the flat procedure. There are roughly 25% fewer hidden nodes and the prediction accuracy has improved over that of the flat procedure. As a result, the training time dropped by an order of magnitude. The small and elegant solution found by the GDR should still not have been expected, however, because the procedure is a greedy approach based on similarity of the patterns. The parity problem is difficult to solve using this type of approach because input patterns that are only one bit different require a different output. Contradictory data could easily be detected as before.



Problem	Flat Learning Procedure				Hierarchical Learning Procedure			
	Number of Hidden Nodes	Number of Deletions	Presentations/ Iterations	Prediction Accuracy	Number of Hidden Nodes	Number of Deletions	Presentations/ Iterations	Prediction Accuracy
or	2	0	7/ 1.8	50%	2	0	7/ 1.8	50%
and	1	0	7/ 1.8	75%	1	0	7/ 1.8	75%
xor	2	0	10/ 2.5	25%	3	0	10/ 2.5	25%
4 bit symmetry	4	0	106/ 6.6	25%	6	0	46/ 2.9	56%
6 bit symmetry	8	0	1072/ 16.8	50%	9	0	190/ 3.0	84%
8 bit symmetry	16	0	10456/ 40.8	69%	12	0	766/ 3.0	94%
4 bit odd parity	8	1	239/ 14.9	6%	8	0	122/ 7.6	12%
8 bit odd parity	128	94	111909/ 437.1	2%	98	0	13498/ 52.7	53%
3 bit negation	6	0	62/ 3.9	50%	7	0	62/ 3.9	56%
2 bit add	8	0	103/ 6.4	50%	7	0	81/ 5.1	43%
3 bit add	17	0	1027/ 16.0	59%	18	0	697/ 10.8	65%
4 bit add	42	8	19251/ 75.2	64%	44	0	5838/ 22.8	80%
2 bit subtract	8	1	158/ 9.9	19%	9	0	104/ 6.5	25%
3 bit subtract	17	2	1500/ 23.4	38%	23	0	935/ 14.6	50%
4 bit subtract	41	9	21031/ 82.2	56%	62	0	8498/ 33.2	68%
4 bit increment	--	--	-- --	--	9	0	153/ 4.8	71%
4 bit decrement	--	--	-- --	--	8	0	267/ 8.3	56%
4 bit compare	--	--	-- --	--	50	0	6090/ 23.8	76%
2 bit multiply	--	--	-- --	--	5	0	61/ 3.8	68%
3 bit multiply	--	--	-- --	--	21	0	838/ 13.1	67%
4 bit multiply	--	--	-- --	--	91	2	28577/ 111.6	63%
6/3 bit divide	--	--	-- --	--	90	4	27271/ 121.7	55%

Tab.3: *Performance of the Flat Procedure versus the Hierachical procedure; unstructured training set*

Problem	Unstructured Training Sets				Structured Training Sets			
	Number of Training Pairs	Number of Hidden Nodes	Presentations/ Iterations	Prediction Accuracy	Number of Training Pairs	Number of Hidden Nodes	Presentations/ Iterations	Prediction Accuracy
4 bit symmetry	16	6	46/ 2.9	56%	7	6	28/ 4.0	56%
6 bit symmetry	64	9	190/ 3.0	84%	10	9	55/ 5.5	84%
8 bit symmetry	256	12	766/ 3.0	94%	13	12	91/ 7.0	95%
3 bit negation	16	7	62/ 3.9	56%	7	7	28/ 4.0	56%
2 bit add	16	7	81/ 5.1	43%	8	6	36/ 4.5	50%
3 bit add	64	18	697/ 10.8	65%	13	9	91/ 7.0	80%
4 bit add	256	44	5838/ 22.8	80%	18	12	171/ 9.5	93%
2 bit subtract	16	9	104/ 6.5	25%	8	6	46/ 5.8	50%
3 bit subtract	64	23	935/ 14.6	50%	17	10	177/ 10.4	73%
4 bit subtract	256	62	8498/ 33.2	68%	32	16	578/ 18.1	88%
4 bit increment	32	9	153/ 4.8	71%	9	9	45/ 5.0	72%
4 bit decrement	32	8	267/ 8.3	56%	14	8	105/ 7.5	56%
4 bit compare	256	50	6090/ 23.8	76%	24	11	338/ 14.1	91%

Tab.4: *Performance of the Hierachical procedure; unstructed training sets versus structured training sets*

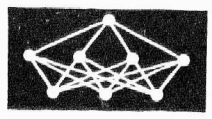


Table 4 lists the results for structured training sets and cross lists them with the results from the unstructured training sets. All of the structured training sets had many fewer training pairs than the total number possible. For these training sets, the learning procedure had a 0% prediction accuracy, as was desired. Each new training pair in a structured training set represents some new aspect of the problem that the network will not already know. The prediction accuracy given in Table 4 for the structured training sets was, therefore, calculated on the assumption that all other possible input patterns would be presented immediately after the structured training set itself.

From Table 4, in the add, substract, and compare problems, the prediction accuracy has been further improved by large margins. The number of hidden nodes has also dropped dramatically for those problems. These problems benefited greatly from the additional structuring of the training set. In the remaining problems, it is apparent that the normal binary counting order was already a good way to order the training pairs. In all cases, by specifying only the minimum required training pairs, the training speed also dramatically improved.

5. Comparison with Other Research

There is currently a great deal of research in the area of neural net learning. It is not the objective here to give an exhaustive account of all recent efforts. Only a few learning strategies that have similar objectives or results will be mentioned here for comparison with the flat and hierarachical learning procedures.

A non-deterministic learning strategy was developed by [10]. Their approach is similar to the Boltzmann machine. This network has also been applied to the add problems that were used to test the flat and hierarchical procedures. They report the ability of their method to solve the 2-bit add problem in 12 iterations with 18 hidden nodes, the 3-bit add problem in 46 iterations with 26 hidden nodes, and the 4-bit add problem in 76 iterations with 47 hidden nodes. Comparing this procedure with flat learning, it is seen that its performance is worse for small problems, but both learning procedures approach the same type of performance for the more complex problems. However, the hierarchical learning procedure is clearly superior for this type of application with respect to the number of iterations required. The method by [10] lacks the deterministic features, as desired for the automated data processing application, and requires long execution time on conventional computers. The generalization properties of this network were not published.

The Multiple RCE (Restricted Coulomb Energy) network ([12] et al.) appears to be, in essence, very similar to the flat learning method. The network uses an analogous mechanism to the unification threshold described here to separate linearly inseparable patterns, and the network uses fixed connection weights for each output node that applies the *or* operator to the outputs of all hidden nodes. The separation mechanism does not appear to have the same type of generalization control, but over large training sets, the RCE network is likely to perform similarly to the flat learning procedure. However, the RCE net does not have the generalization characteristics or learning speed of the hierarchical learning procedure.

Several architectures employing a hierarchical organization of the hidden nodes have recently been developed. [2] have developed a hierarchical approach which is based on the observation that patterns that are not linearly separable in one binary coding scheme can, in fact, be linearly separable in a different coding scheme. Their method uses a transform technique (the Fast Real Discrete Fourier Transform) to map input bits to another coding scheme, if a pattern produces too large an error in the original scheme. Thus, multiple classifying nodes operate in parallel, based on data that have been recoded in a cascaded, or hierarchical fashion. Experimentation is probably required to better determine the performance and generalization characteristics of this approach for comparison purposes.

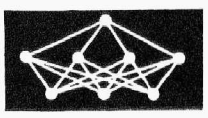
[16] present a hierarchical, self-organizing network that appears to be able to construct arbitrarily deep networks that seem to have good generalization characteristics. The specific approach they describe, however, is based on the use of continuous data and higher-order functions, and relies on the use of simulated annealing to converge. Their network is, therefore, not deterministic, and is not appropriate for incremental learning.

[4] have developed a hierarchical architecture called Cascade-Correlation. It was designed to improve the slow learning characteristics of backpropagation. [3] (1990) has designed the Upstart Algorithm which shows definite similarities to the hierarchical learning strategy. This network also constructs an exact association between binary patterns. However, it does not build a hierarchy of concepts as described in this paper.

6. Conclusions and Future Work

Based on the results achieved with the hierarchical learning procedure, it appears possible to make neural networks learn fairly quickly and to achieve the required type of generalization. If the hierarchical training procedure proves to be robust for a large variety of problems, the concept of learning instead of programming may become within reach.

The hierarchical procedure, as currently developed, still has some limitations. The procedure does not perform well when the training set is not properly structured. Automated structuring of the training set has not yet been accomplished. An inherent limitation in achieving this goal is found in the incremental presentation order. While this order has definite advantages, it prevents a network from comparative analysis as a means to discriminate between low and high level concepts. Another disadvantage is that, when the size of a problem is scaled up, the training performance will decrease due to the fact that the learning



procedure always reconsiders all previous training pairs to ensure that previous learning isn't impacted. The tests performed showed that in many cases the rechecking passes through the training set weren't actually required. It may be possible to employ heuristics to decide when such reconsiderations are not necessary.

Finally, a more rigorous, formal development of the learning procedure is needed in order to determine whether the procedure will always converge to a solution. Further, a formal means to determine the upper bound on the number of hidden nodes and the number of training pairs necessary for different types of problems would be of great benefit.

References

[1] De Marco, T.: Structures Analysis and System Specification, New York: Yourdon Press, 1979.

[2] Ersoy, O.K., and Hong, D.: Parallel, Self-Organizing Hierarchical Neural Networks, IEEE Trans. Neural Networks, Vol.**1**, 1990, 167-178.

[3] Frean, M.: The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks, Neural Computation Vol. **2**, 1990, 198-209.

[4] Fahlman, E.S., and Lebiere, C.: The cascade-Correlation Learning Architecture, Techn. Report CMU-CS-90-100, Carnegy Mellon University, Pittsburgh, PA., 1990.

[5] Gane C., and Sarson, T.: Structured Systems Analysis: Tools and Techniques, Saint Louis, Mo.: McDonnell Douglas Professional Services Co, 1985.

[6] Grossberg, S.: Competitive Learning: From Interactive Activation to Adaptive Resonance, Cognitive Science, Vol.**11**, 1987, 23-63.

[7] Hopfield, J.J.: Neural Networks and Physical Systems with Emergent Collective Computational Abilities, Proc. Natl. Acad. Sci. USA, Vol. **79**, 1982, 2554-2558.

[8] Hopfield, J.J.: Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons, Proc. Natl. Acad. Sci. USA, Vol.**18**, 1984, 3088-3092.

[9] Kohonen, T.: Self-Organization and Associative Memory, Berlin: Springer-Verlag, 1984.

[10] Lansner, A., and Ekeberg, O.: An Associative Network Solving the '4-Bit ADDER' Problem, Proc. IEEE First Int. Conf.Neural Networks, Vol.**II**, 1987, 549-556.

[11] Minsky, M.L., and Papert, S.A.: Perceptrons (Expanded Edition), Cambridge, Mass.: MIT Press, 1988.

[12] Reilly, D.L., Scofield, C., Elbaum, C., and Cooper, L.N.: Learning Systems Composed of Multiple Learning Modules, Proc. IEEE First Int. Conf. Neural Networks, Vol.**II**, 1987, 495-503.

[13] Rosenblat, F.: Principles of Neurodynamics: New York, Spartan Books, 1962.

[14] Rumelhart, D.E., Hinton, G.E., and Williams, R.J.: Learning Internal Representations by Error Propagation, In D.E. Rumelhart, and J.L. McClelland (Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol.**1**: Foundations, 318-364, Cambridge, Mass.: MIT Press, 1986.

[15] Rumelhart, D.E., and Zipser, D.: Feature Discovery by Competitive Learning. In D.E. Rumelhart, and J.L. McClelland (Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Congnition Vol.**1**: Foundations, 318-364, Cambridge, Mass: MIT Press, 1986.

[16] Tenorio, M.F., and Lee, W.: Self-Organizing Network for Optimum Supervised Learning, IEEE Trans. Neural Networks, Vol.**1**, 1990, 100-110.

[17] Yourdon, E.: Modern Structured Analysis. Englewood Cliffs, New Jersey: Prentice Hall, 1989.

NEURAL ALGORITHM FOR CONSTRUCTING MINIMUM SPANNING TREE OF A FINITE PLANAR SET

Andrew I. Adamatzky[1]

Abstract: We present a local parallel algorithm for constructing minimum spanning tree of a finite planar set. Our algorithm is based on mechanisms of a dendritic tree growth during neuronal ontogenesis. We assume that some neuron wishes to make synaptic terminals on the all points of a given set. Neuron solves this problem by growing and sprouting its dendritic tree at plane. We implement our algorithm in a cellular automata processor, which architecture is similar to neural one. Cellular automata processor is the perspective specimen among massively-parallel computers. Offered algorithm runs in $0(h)$ time and it requires $0(n)$ processors, when h is a number of the given points in the longest branch of minimum spanning tree, n is a number of the given planar points.

Key words: *Neuron, Dendritic Tree, Cellular Automata Processor, Parallel Algorithm, Minimum Spanning Tree, Complexity.*

Received: July 7, 1991
Revised and accepted: December 9, 1991

1. Introduction

We'll consider a modification of well-known Steiner's problem, i.e. the problem of connecting a given set S of n points in the Euclidean plane by straight lines, that there is a path of lines between every pair in the set, so as to minimize the total length of the lines used. Line segments connecting points of a given set must not be intersected anywhere in the plane except points of a given set S. By that way, we construct minimum spanning tree (MST), i.e. a connected planar graph with n nodes, $n-1$ edges, without cycles, so as the sum of weights (lengths) of all edges is minimal.

In the base work [1] it was shown that the relative neighborhood graph and Delanuay triangulation are a supersets of the MST. It leads to use of constructing MST in computing relative neighborhood graph and Dealanuay triangulation. Furthermore, constructing MST is a one of the basic algorithms for the following problems: routing [2], network synchronization, breadth-first-search, deadlock resolution, leader selection, traveling salesman [3,4,5]. MST problem has an important application to designing of the distributed networks in which the nodes represent cities and the edges represent electrical power, water and natural gas lines, roads etc.

Modern algorithms for MST problem use the ideas of the pioneer work of Boruvka [6,7]. The majority of approaches to constructing MST are based on Kruskal's algorithm [8], Prim-Dijkstra algorithm [9,10] and Sollin's algorithm [11]. The distributed algorithms for MST problem are discussed in [12]. In the majority of distributed algorithms it is assumed that a processor exists at each node of the graph: it means that data are mapped onto processor network by one-to-one correspondence. The known algorithms for parallel constructing MST have time complexity $0(n\ logn)$ and space complexity $0(n)$ [13, 14]. Huang's algorithm [15] uses a one-dimensional systolic array of $(n-1)$ cells and it runs in $0(m+n)$ time, for computing MST of a given graph with n nodes and m edges. Asynchronous implementation of parallel algorithm leads to $\theta(n)$ time [5].

In section 2 we present an algorithm for constructing MST by single neuron using growth of dendritic tree. Implementation of this algorithm in a cellular automata processor is discussed in Section 3.

2. Neuronal Constructing MST

Along the whole section we concern the subject of a dendritic tree growth in single neuronal cell. The notion of growth cones is widely used by us. Roughly speaking, each growth cone (GC) is a bulb at the tip of neuron's sprout (branch) filled with all cellular organels except nucleus and which is able to motion similarly to amoeba. GC communicates with neuronal soma by using macro molecular transport. These messages need in a great time, the-

[1] A.I. Adamatzky,
Biophysics Department, Sankt-Petersburg University, Universitetskaya emb. 7/9, Sankt-Petersburg 199164, Russia,
and GALAFOX, Sankt-Petersburg, Russia

refore GCs are practically independent on their host neuron, i. e. host neuron fulfills some energetical support only.

At the beginning we'll consider the MST problem from the ontogenesis point of view. First of all, we point out three main stages of neuron transformating that can be distinguished in the differentiation of neural cells: medulloblast → neuroblast → neuron. Medulloblast is characterized by ability to cell division whereas each neuroblast has neurofibrils, large nucleus and small volume of cytoplasm (moreover, it can move); at last, neuron has axon and dendritic tree shaped and electrical activity. We mean that the transformation "neuroblast → neuron bases on the process of axon and dendritic tree grows. The following propositions, which are well known and sometimes folklor among neuromorphologists, are basical to understanding algorithm offred by us:

1. the GCs of dendritic tree moves towards afferent terminals,

2. in the beginning, synapses are formed on apical dendrites and after that on proximal dendrites and soma,

3. each neuroblast (when it is transforming to neuron) maked much more branches than it is needed; "unnecessary" branches pull in soma (or in other branches) when neuroblast "matures",

4. neurons obtain chemical labels at early stage of ontogenesis; these labels help GCs to find target-cells or target-branches.

Let a neural tissue is cutted by plane P which is perpendicular to n afferent fibers passing through this tissue. Assume, a some neuroblast $\mathcal{N}$ must transform itself to neuron which obtains information from each of n afferent fibers, i.e. $\mathcal{N}$ must form synaptic contacts with n axons by dendritic tree growing. The principal limitation on grown dendritic tree is a minimal summary length of all branches. We can sure that neuroblast $\mathcal{N}$ finds solution of this task without any problems.

Initially,$\mathcal{N}$ crowls up to a site where density of afferent fibers is greatest. And after that $\mathcal{N}$ begins the growth of dendritic tree. Each GC of $\mathcal{N}$ has the wide powers for moving and branching out, i.e. GC chooses a direction of movement and time when branching out itself only.

Now, show the process of spanning planar set S with n points by dendritic tree in details. Let the following variables and constants are ascribed to each GC $\mathcal{C}$: (1) γ is an upper degree of branching, i.e. output degree of node from the graph-theoretic point of view, (2) ρ is a maximal distance at which $\mathcal{C}$ can recognize point of a given set S, (3) E is an "energy" of $\mathcal{C}$, (4) θ is a threshold of energy E decreasing, (5) δ is a small real, (6) x is a current position of $\mathcal{C}$ at plane, $x \in R^2.\gamma, \rho, \delta$ and θ are constant, but E and x are variable. Note, that energy E of $\mathcal{C}$ is an anlog of distance between $\mathcal{C}$ and host neuron $\mathcal{N}$ (or neuroblast for initial stages).

Each GC $\mathcal{C}$ at unit time step can fulfill one of the operations: crowl up to next position x, generate not more than γ new GCs (successors), detect point of S and crowl up to its position, settle a conflict (when it occurs) between $\mathcal{C}$ and other GCs for point of S. The energy E is decreased at each time step of $\mathcal{C}$ motion as: $E^{(0)} = E_0$, $E^{(t)} = E^{(t-1)} - 1$, E_0 is enough great. Variable E is necessary to count the length of path from current position of $\mathcal{C}$ to host neuron $\mathcal{N}$ along branch, i.e. E describes "age" of GC because at each time step GC must move towards one of the all possible directions (note, "age" is analogue of length of branch from host neuron or branching point to GC position). This quantity is used for resoluting conflicts.

Let, three GCs $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ have crowled up to a given point p, and at moment of meeting, their energies are equal to E_1, E_2 and E_3, such that $E_1 > E_2 > E_3$ (see Fig.1,A). Each of $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ recognize the leader on energy, i.e. cone $\mathcal{C}_i$ which energy is a maximum $E_i = \max_{1 \le j \le 3} E_j$ and such GCs that are not leaders pull back in predecessor or continue motion, but leader occupies position of point p (see Fig.1,B).

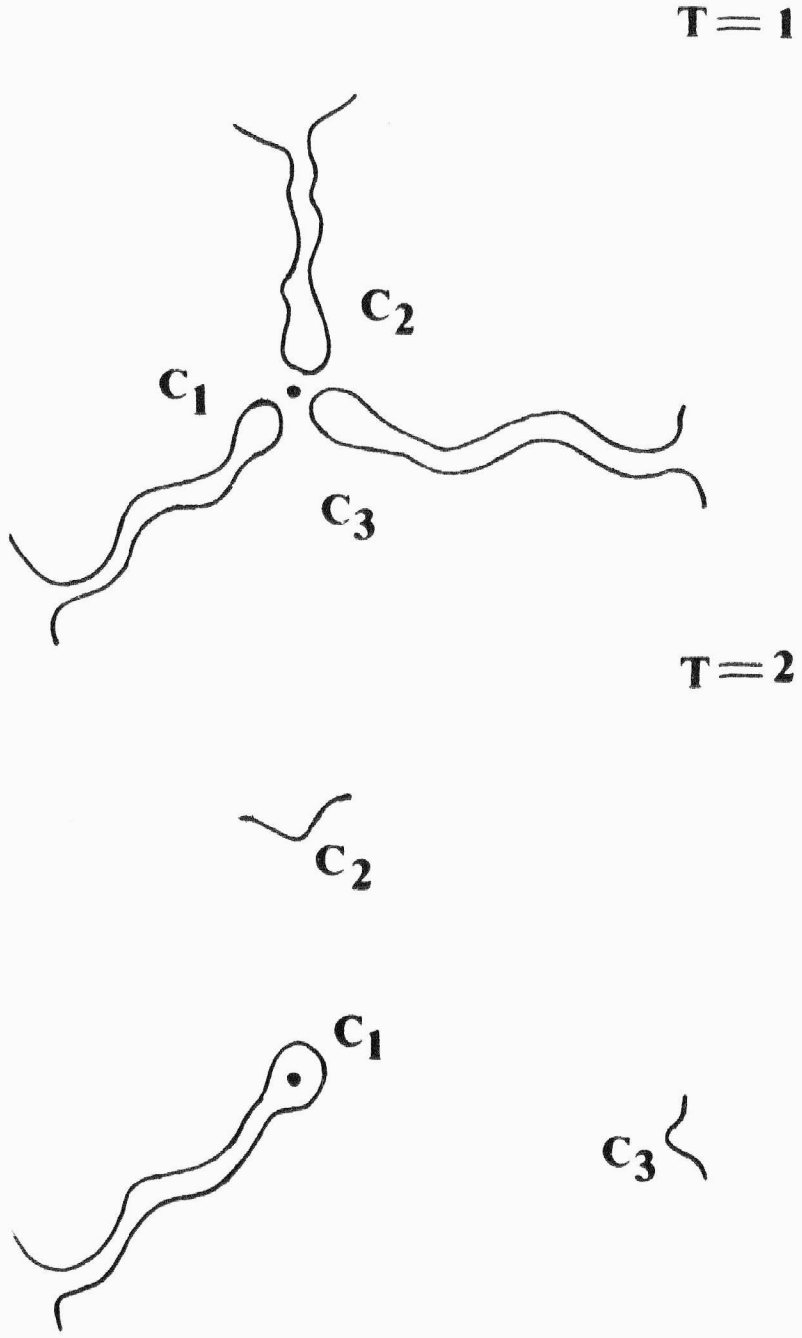


Fig.1: *Resolution of conflict between $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$.*

Show the typical situations in a tree growth (see Fig.2). GC $\mathcal{C}$ has came to position of point $p \in S$ (see Fig.2,A). Note, $r = \min\{dist(p,q) : q \in S\}$. After that $\mathcal{C}$ generates seven GC-successors which find the next points of S in a

some neighborhood of point p (see Fig.2,B). Let, some of these GCs have found points of S and fixed themselves at their positions, but other GCs continue searching until energy of each GC $\mathcal{C}_j$ holds the condition $|E_j - \theta| \leq \delta$ (see Fig.2,C). After that they pull back in $\mathcal{C}$ (see Fig.2,D).

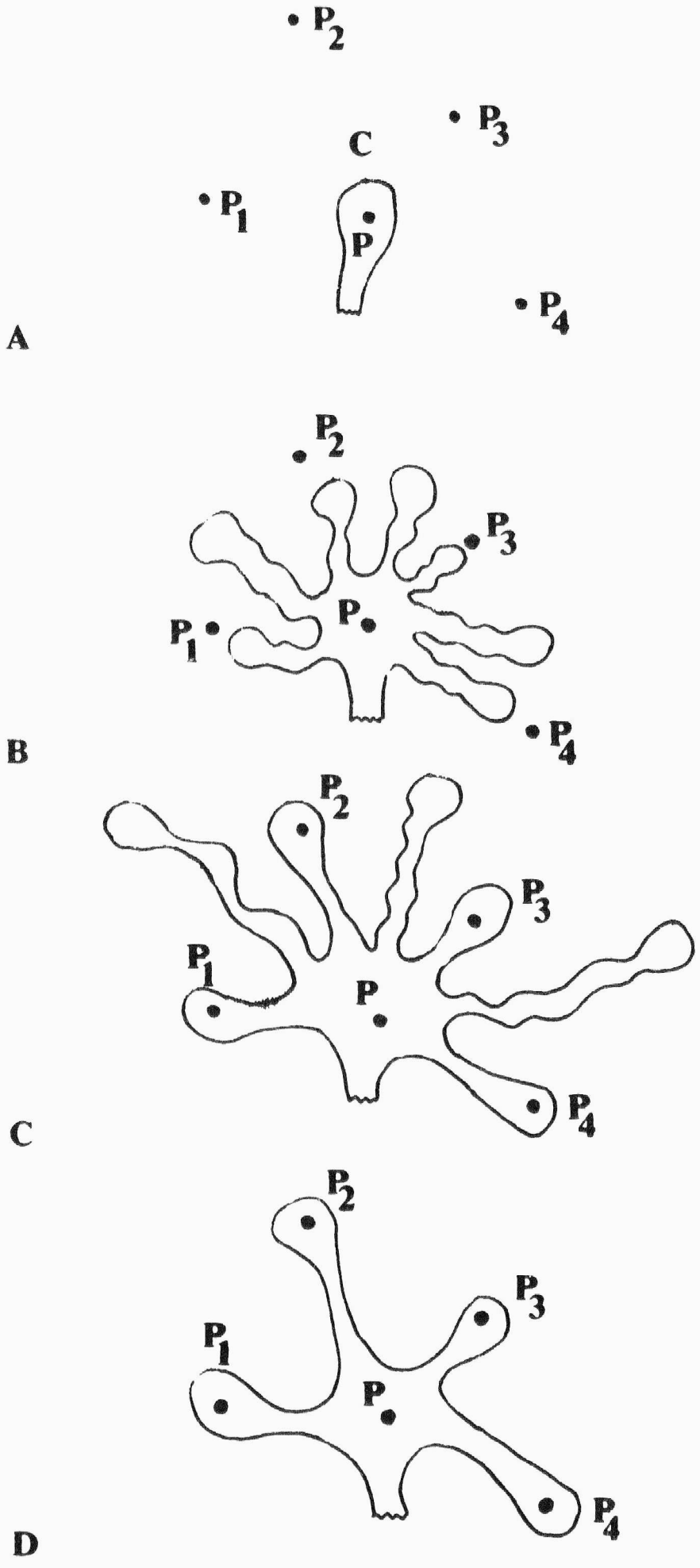


Fig.2: *Stages of spanning points p_1, p_2, p_3 and $p_4 \in S$ by routes of GC $\mathcal{C}$.*

Each GC $\mathcal{C}$ stops any action in the case when all of its GCs-successors pulled back in it. Neuron $\mathcal{N}$ stops the growth of dendritic tree when all of its GCs have stoped any actions.

As a results, we can design the following pascal-like program for a single GC. Note, $D(x, \rho$ is a planar disc with center x and radius ρ.

```
Program Growth_ Cone; (*assume it is the program for single GC C*)
  INPUT: S;
  OUTPUT: MST (S);
  const    δ = δ0; ρ = ρ0; θ = θ0;  (*δ0, ρ0, θ0 are natural *
  var x:array [1..2] of integer;
       E:integer;
       B:boolean;
       C: set of integer;
  Begin
  E := E0; B:FALSE; x := x0; C := 0; (*E0 is enougth great natural*)
  Repeat
    Begin
     E := E − 1;
     Repeat x := next(x) Until x is not occupied by any branch;
     Generate C; (*C is a set of energy values of such GCs that vish to
         occupy position choosen by GC C*)
     If D(x, ρ) ⋂ S ≠ 0 then
      Begin
       If C = 0 or max{E' ∈ C} < E then B:=TRUE
      End
     End;
     Until B or |E − θ| ≤ δ;
     If B then
      Begin
       Occupy position x and stop growing;
       Branch out (Generate at least γ successors);
      End
          else Pull in predecessor
  End.
```

The result of computation is a dendritic tree which is the MST of a given set S.

We do not compute the bounds of complexity for dendritic tree growth because of a number of informal terms (as "motion", "branching out", conflict", "pull back" etc.) used in our algorithm description. Complexity of neuronal algorithm will be discussed in the next section using the most certain notion of the local parallel computations.

3. Cellular Automata Implementation

This section presents the realization of dendritic tree growth, spanning points of a given set, in the one class of the massively-parallel computers, - cellular automata processor. Cellular automata processor (CAP) has medial position between massively-parallel computers with very local (CLIP) and global (Connection Machine) interprocessors connections [16, 17].

The cellular automata (i.e. local massively parallel) algorithms had been used by us for solving such problems of computational geometry as Voronoi diagram [16], relative neighborhood graph, influence graph and Gabriel graph [17] constructing.

CAP is an integer grid which nodes are elementary processors having its own local memory of 0(1) size and cellular automata structure of connections. All processors of CAP are uniform and each processor performs the same local algorithm and it has its own instruction counter, local set of registers. Each processor P_{ij} of CAP can communicate with k neighbor processors $P_{i-Rj-R}, \ldots, P_{i+Rj+R}$ such that $R \geq 1, k = (2R+1)^2$. Set of these processors is called a neighborhood of P_{ij} and it is wrote as $U_R(P_{ij})$. Note, R is radius of neighborhood and k is a neighborhood size.

We mean that CGs crowl up on an integer grid. This discrete motion is simulated by CAP such that, when GC is in cell (i,j) of grid, processor P_{ij} is activated. hence, in this case, CGs do not rove on plane for searching points of S, but each of them jumps from one point of S to another. The searching of points for branching out or jumping for GC at cell (i,j) is performed by processor P_{ij}. Ascribe to each processor P_{ij} of CAP the following variables and constants:

1. ρ, θ, γ, δ are as in GC definitions,
2. D is a length of path from an initial point (root) of tree to a current point of S stored by a given processor P_{ij}, D is an analog of energy E,
3. L is a list of coordinates (addresses) of processor P_{ij} neighbors containing the such points of S which are the targets of ramifying (branching out),
4. LP is a list of coordinates of points contained in processors of L,
5. L_{mst} is a list of points neighboring with point belonging to P_{ij} at MST(S),
6. P_{mst} is a list of coordinates of processors containing elements of L_{mst},
7. A is a boolean variable.

Example. Let us assume P_{ij} is a processor of CAP which has neighborhood radius $R = 1$. Assume neighborhood of P_{ij} is equal to $U_1(P_{ij}) = (P_{i-1\,j-1}, P_{i\,j-1}, P_{i+1\,j-1}, P_{i-1\,j}, P_{i-1\,j+1}, P_{i\,j+1}, P_{i+1\,j+1})$. Let (P_{ij}) contains coordinate of point $p_1 \in S$ and processors $P_{i\,j-1}$, $P_{i+1\,j-1}$, $P_{i\,j+1}$ contain coordinates of points $p_2, p_3, p_4 \in S$ correspondingly, whereas processors $P_{i-1\,j-1}, P_{i-1\,j}, Pi+1\,j, P_{i-1\,j+1}, P_{i+1\,j+1}$ contain nothing. Then $L = \{(i,j-1),(i+1,j-1),(i,j+1)\}$ and $LP = \{p_2, p_3, p_4\}$. Assume processor P_{ij} had connected points in pairs (p_1,p_2) and (p_1,p_4) by edges of MST. In that case we have $L_{mst} = \{p_2, p_4\}$ and $P_{mst} = \{(i,j-1),(i,j+1)\}$.2

So processor P_{ij} with coordinates of $p \in S$ acts as follows. If boolean variable A is true, it means that P_{ij} is occupied by a virtual GC, then P_{ij} scans clockwise neighborhood by increasing neighborhood radius $r = 1, 2, 3, \ldots$ and performs the operations. Adds to set L addresses of processors in neighborhood, containing points of S, which are at distance not more than $\min\{dist(p,q)+\varepsilon : q \in LP\}$. Notes, that points contained in element of L are added to set LP. After that P_{ij} writes its own coordinates (address) and coordinates of p in the sets L_{mst} and P_{mst} of processors of L. Except this, P_{ij} changes the value variable A from FALSE to TRUE for each processor of L, i.e. processor P_{ij} activates processors of L.

The process of computation is finished globally when there are no procceors of CAp with true variable A.

Let us assume that points of a given set S are distributed in the processors of CAp in a natural order such that the arbitrary points p and q, which are neighbors with one another at plane, are contained in the neighboring processors.

So processor $P \equiv P_{ij}$ containing coordinates of point $p \in S$ performs the following program.

```
Algorithm Cell:
const γ := γ0; ρ := ρ; δ := δ0; ε := ε0; θ := θ0; R := 1;
var   A: Boolean;
      a, r, D:Integer; (* r is a neighborhood current radius*)
          (* a is an auxiliary variable *)
      L, LP, Lmst, Pmst : Set of integer;
Begin
 L := 0; LP = 0,
 If A then;
  Begin r:=0;
   Repeat
    Begin
     r := r + 1; Scan neighborhood Ur(Pij);
     If ∃P'i'j' ∈ Ur(P) and P'i'j' ∉ Pmst and P'i,j, contains
     coordinates of p' ∈ S then Begin L := L + {(i', j')};
     LP := LP + {p'} End;
    End
   Until r = R or |LP| = γ; (* R is superior limit to increasing r *)
   Find such a ∈ LP that dist(p, a) = min{dist(p,q) : q ∈ LP};
   For each q ∈ LP do If dist(p, q) > a + ε then LP := LP - {q};
   For each (i, j) ∈ L do If ∃q ∈ LP that Pij contains coordinates of
       q then else L := L - {(i, j)};
   For  each (i*, j*) ∈ L do If D > D* then
   LP := LP - LP∩LP*; L := L - L∩L*; Pmst := L, Lmst := LP,
   Active processors which addresses are in L; (* analog of branching
   out *)
   A:=FALSE (* transition of Pij to passive state *)
  Begin
End.
```

It is clear that if processor P_{ij} is not active ($A = FALSE$) then it implements empty sequence of operations.

Before discussing complexity of constructing MST, show extreme structure of data. Let n points of S belong to the only straight line and MST begins to growth from a tip point (see Fig. 3, left). This is the worst case and CAP needs in $n-1$ steps (see Fig. 3, right).

Fig.3: *Extreme case of data structure.*

This case of data structure shows that upper bound of time complexity of constructing MST in CAP is equal to

$0(n)$. But it is not the most precision bound. We understand that computation in CAP is finished when each processor of CAP has $A = FALSE$. It leads to time complexity $0(h)$, where h is the maximal number of given points forming single sprout of MST, i.e. time complexity of constructing MST is bounded by time needed for growth of longest branch of MST.

Theorem. Minimum spanning tree of a planar set with n points is computed in cellular automata processor of size $0(n)$ in $0(h)$ time by the algorithm of a dendritic tree growth, when h is a maximal number of vertices in single sproute of tree. **2**

We can give concrete example concerning application of our algorithm to practice. It is the constructing MST of a multi-pin net in wire routing design [20]. The algorithm shown was included into the system for serial-parallel routing SPROUTE based on parallel co-processor L51V (Scientific Research Corp. GALAFOX, Sankt-Petersburg) (see also [18, 19, 20]).

Acknowledgment. The author would like to thank the referees for their careful reading of my paper and valuable suggestions.

References

[1] Toussaint, G.T. The relative neighborhood graph of a finite planar set. Pattern Recogn. Vol.**12**, 261-268, 1980.

[2] Gilbert, E.N. A solvable routing problem. Networks Vol.**19**, 587-594, 1989.

[3] Held, M. and Karp, R. The traveling salesman problem and minimum spanning trees: Part II. Math. Program. Vol.**1**, 6-26, 1970.

[4] Awerbuch, B. Complexity of network synchronization. J. ACM Vol.**32**, 804-823, 1985.

[5] Awerbuch, B. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems, in Proc. 19th Annual ACM Symposium on Theory of Computing, ACM, NY, 230-240, 1987.

[6] Borůvka, O. Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí. Eletotechnický obzor, **15**, 153-154, 1926.

[7] Graham, R.L. and Hell, P. On the history of the minimum spanning tree problem. Ann. Hist. Comput. Vol.**7**, 43-57, 1985.

[8] Kruskal, J.B. On the shortest subtree of a graph and the traveling salesman problem. Proc. Amer. Math., Sec **7**, 48-50, 1956.

[9] Prim, R.C. Shortest connection networks and some generalizations. Bell Syst. Tech.J. Vol.**36**, 1389-1401, 1957.

[10] Dijkstra, E.A. A note on two problems in connexion with graphs. Namer. Math. Vol.**1**, 269-271, 1959.

[11] Goodman, S.E. and Hedetniemi, S.T. (Eds.) Introduction to the Design and Analysis of Algorithms. McGrow-Hill, NY, 1977.

[12] Quinn, M.J. and Deo, N. Parallel graph algorithms. Comput. Surveys. Vol.**16**, 3190340, 1984.

[13] Chin, F. and Ting, H.F. Improving the time complexity of message-optimal distributed algorithms for minimum-weight spanning trees. SIAM J. Comput., Vol.**39**, 612-626, 1990.

[14] Gallager, R.G., Humblet, P.A., and Spira, P.M. A distributed algorithm for minimum weight spanning trees. ACM Trans. Programming Languages and Syst. Vol.**5**, 66-77, 1983.

[15] Huang, S.T. A fully pipelined minimum-spanning tree constructor. J. Parall. Distrib. Computing Vol.**9**, 55-62, 1990.

[16] Adamatzky, A.I. Local parallel algorithm for constructing Voronoi diagram. Elektronnoe Modelirovanie (Kiev), Vol.**1**, 1992. (In Russian).

[17] Adamatzky, A.I. Local parallel constructing graphs of a finite planar set. Elektronnoe Modelirovanie (Kiev), 1992 (to appear. In Russian).

[18] Ivanov, S.F. Cellular Automata Processor, GALAFOX Report, 7, 1990.

[19] Bronnikov, V.A., Ivanov, S.F., Adamtzky, A.I., Solovyov, M.V. Assively-parallel computer LOCON. Project, GALAFOX, Leningrad, 1989 (in Russian).

[20] Adamatzky, A.I. and Ivanov, S.F. Algorithms for parallel routing. XIII All-Union Workshop on Homogenous Computing Media and Systolic Structures, L'vov. October 1991.

Interesting and Coming Events

In this section of our Journal the information on some interesting and coming conferences, symposiums and seminaries are given:

DECEMBER 1991

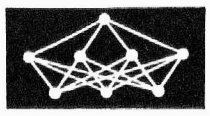
1991 IEEE Workshop on Speech Recognition, Dec.15-18, 1991, Harriman, NY. Contact: Jay G.Wilpon (201)582-3559.

World Congress on Expert Systems, Dec.16-19, 1991, Orlando, Fla. Cosponsors: Int'l Assoc. of Knowledge Engineers et al. Contact World Congress on Expert Systems, c/o Congress Secretariat, Congress (USA), Inc., 7315 Wisconsin Ave., Suite 404E, Bethesda, MD 20814, phone (301)469-3355, fax (301)469-3360.

8th Israeli Conference on Artificial Intelligence and Computer Vision, Dec.30-31, 1991, Tel-Aviv, Israel. Contact: H.Wolfson, 8th IAICV, School of Mathematical Sciences, Tel-Aviv University, Tel-Aviv 69978. Israel.

JANUARY 1992

25th Annual Hawaii International Conference on System Sciences (HICSS - 25), Jan. 7-10, 1992.



Kauai, Hawaii. Contact: Dr. Bhushan Saxena, Department of Computing, Hong Kong Polytechnic, Hung Hom, Kowloon, Hong Kong, e-mail: cssaxena @ hkpcc.hkp.hk

Second Int'l Symposium on AI and Mathematics, Jan.5-8,1992. Fort Lauderdale, Fla. Contact: L-L.Lassez, IBM T.J.Watson Research Center, PO Box 704, Yorktown Heights, NY 10598, fax(914)784-6307, e-mail: jll@watson.ibm.com.

FEBRUARY 1992

Singapore International Conference on Intelligent Control and Instrumentation, Feb.18-21, 1992. Singapore. Contact: C.C.Hang, Technical Programme Chair, SICICI'92, IEEE Singapore Section, 200 Jalan Sultan, # 11-03 Textile Centre, Singapore 0719, e-mail: fenghcc@nus3090.bitnet.

MARCH 1992

IEEE Int. Conf. on Fuzzy Systems, March 8-12, 1992. San Diego, CA. Contact FUZZ-IEE'92, Meeting Management, 5665 Oberlin Drive, Suite 110, San Diego, CA 92121, phone (619)453 6222, fax (619)535 3880.

1992 IEEE International Workshop on Intelligent Signal Processing and Communication Systems, March 19-20, 1992. International Convention Center, Taipei, Taiwan, ROC. Information: Dr. Naohisa Ohta, NTT Transmission Systems Labs, 1-2356, Take Yokosukashi 238-03 Japan. phone +81 – 468 – 59 – 2072, fax +81 – 468 – 59 – 3014, e-mail: naohisa@nttsd.ntt.jp.

1992 Intl. Conference on Acoustics, Speech, and Signal Processing, March 23-26, 1992. San Francisco Marriott. Paper Proposals due August 2, 1991. Information: Sally Wood, EECS Dept, Santa Clara Univ., Santa Clara CA 95053.408/554-4058, swood@scu.bitnet.

APRIL 1992

International Conference on Information - Decision - Action Systems in Complex Organisations-IDASCO'92, April 6-8, 1992. Zoology Department University of Oxford, UK. Contact Miss Jane Chopping Conference Organiser IEE Conference Services, Savoy Place, London WC2R0BL, phone (071)240 1871, fax (071)497 3633.

ICCL 92, Int'l Conf. on Computer Languages, April 20-23, 1992, San Francisco. Sponsor: IEEE Computer Soc. Technical Commiteee on Computer Languages. Contact: Mario Barbacci, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA 15213, phone (412)268-7704, fax (412)268-5758.

MAY 1992

IEEE Infocom 92, 11th Conf. on Computer Comm. May 4-8, 1992,Florence, Italy. Contact L.Fratta, Politecnico di Milano, c/o Cefriel, Via Emanueli, 15, 20126 Milano, Italy, phone 39(2)2399-3578, fax 39(2)2399-3587, e-mail: fratta@imicefr.bitnet.

IEEE International Symposium on Circuits and Systems - ISCAS'92, May 10-13, 1992, Sheraton Harbor Island Hotel, San Diego, California; Contact: Dr. Stanley A.White,433 Avenida Cordoba, San Clemente, CA 92 672, phone (001 714) 498 5519

AI 92: Canadian Artificial Intelligence Conference, May 11-15, 1992, Vancouver, Canada. Contact: Fred Popowich, School of Computing Science, Simor Fraser University, Burnaby, B.C. V5A 1S6, Canada.

ECCV-92: Second European Conference on Computer Vision, May 18-23, 1992, Santa Margherita Ligure, Italy, Information: P.Ponte, ECCV-92 Secretariat, Consorzio Genova Ricerche, Via dellÁcciaio 139, 16152 genova, Italy, phone +39 10 651-4000, fax +39 10 603-801.

SEPTEMBER 1992

IEEE International Workshop on Robot and Human Communications, September 1-3, 1992, Hosei University, Tokyo, Japan; Contact: Prof. Hisato Kobayashi, Dept. of Electrical Engineering, Hosei University, Kajinocho, Koganei, Japan, phone (0081) 423 87 6187, fax (0081) 423 87 6122.

OCTOBER 1992

1992 RNNS/IEEE Symposium on Neuroinformatics and Neurocomputing, October 7-10, 1992, Rostov on Don, USSR; Contact: Prof.W.Dunin-Barkowski, Symposium Chairman, Research Institute of Neurocybernetics, 344104 Rostov on Don, pr-t Stachki 194/1, phone (095) 8632280588; Phone/Fax (Moscow): (095) 3660394. Authors outside USSR should submit papers to Prof. W.E. Snyder, Department of Radiology, Bowman Gray School of Medicine, Winston-Salem, NC27157-1022 USA.

JANUARY 1993

IEEE International Symposium on Information Theory, January 10-15, 1993, San Antonio, Texas, Contact: Mr. Costas N. Georghiades, Texas A& M University, Department of Electrical Engineering, College Station, TX 77843-3128, phone (001 409) 845 7408.

GRAVIDAL
(A GRAPHICAL VISUALIZATION TOOL FOR TRANSPUTER NETWORKS)

O. Vornberger, K. Zeppenfeld[1]

Abstract: Large distributed algorithms are hard to design and to implement. One difficulty is the potential complexity of the interactions among the large number of parallel processes. One way to help these problems as small as possible is visualization of the dynamic behavior of such distributed algorithms. This article describes GRAVIDAL, a graphical visualization environment for occam programs running on arbitrary transputer networks. It provides animated user defined views of algorithms during their runtime. The user has only to place some macro calls in his source code and GRAVIDAL then generates a visualized version of his algorithm. Therefore, a modular and specially adapted visualization for any distributed algorithm can be easily achieved. Time measurements show that the overhead for the animated version is in an acceptable range. Besides visualization and debugging, GRAVIDAL can be used to improve teaching and learning of the science of distributed algorithms.

Key words: *parallel processes, distibuted algorithms, visualization environment, occam programs, transputer networks*

Received: July 15, 1991
Revised and accepted: December 9, 1991

1. Introduction

Graphical animation or visualization is an invaluable help in understanding the dynamic behaviour of algorithms. Especially for distributed algorithms graphical visualization is necessary, because during the execution of several processes in parallel the user loses the overall view of the activities in his network. To obtain this survey in large transputer networks extremely difficult routing or multiplexing techniques for every network and application have to be implemented because only one processor, the host processor, is connected to the keyboard while screen while the network processors have no direct connections to these resources.

[1] Oliver Vornberger, Klaus Zeppenfeld
Department of Mathematics and Computer Science, University of Osnabrück, Albrechtstr.28, D-4500 Osnabrück, Germany
E-mail: Klaus @ dosuni.uvcp

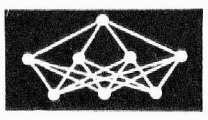
Another important advantage of the graphical visualization, especially for new users, is the reduction of the difficulties involved in understanding the complex behaviour of distributed systems and parallel algorithms. With a graphical visualization environment they get a simple introduction to these topics.

For this reason our objectives was to develop a software environment for distributed algorithms running on arbitrary transputer networks (see [3], [4]). It should provide animated user-defined views into distributed algorithms during their runtime. And so, the herefore unfulfilled wish of transputer and occam users to look into their networks becomes reality, and by simple means. Combined with this ability to look, we achieve a monitor facility which can be very helpful during the design phase of distributed algorithms. The development of a modular and easy to use graphical visualization environment for distributed algorithms, named GRAVIDAL (**Gra**phical **V**isualization of **D**istributed **Al**gorithms) is described in the following sections.

Section 2 gives a general view of the basic concepts of GRAVIDAL. The transformation from a normal occam program into an animated one is shown in Section 3. Section 4 presents some of the modular features supported by GRAVIDAL. Graphical outputs, examples and overhead measurement are the topics of Section 5. Finally, our conclusions are presented in Section 6.

2. Basic concepts

There are a lot of existing systems which show the behaviour of sequential algorithms with graphical output. They can be divided into two main categories. On one side there are the so called *on-line* systems, which show information about the program during runtime. The others, let's call them *off-line* systems, only collect and store the data of the program during the runtime. After the program has finished they show the information on the screen. The BALSA system from. M.H. Brown [2], for example, is a typical member of the first category. The MOVIE-STILLS system from Bentley and Kernighan [1] and GRAIL (Graphical representation of activity, interconnection and loading) from Stepney [7] represent the second group of systems.



Another criterion for graphical visualization systems is the style of algorithm behaviour graphical output and the selection of data which must be drawn. One important example of these different types in the literature is the graphical visualization of sequential sorting algorithms. They are animated only with vertical bars, which show the user the point of the array where the numbers which have to be sorted are found. The user has no chance during the whole animation to change the representation. Also, the user only sees the sorted data and no other details of the algorithm. If it does not work propertly, for example a variable violates array boundaries, it is very useful to have an additional look at the values of certain variables. During the design phase of the algorithms these views could have a great influence on error detection. However, we have to mention that these views can not replace a debugger that not only looks at variables but also allows their manipulation.

The discussion shows that the graphical visualization of sequential algorithms already has a number of possibilities to show the behaviour of the algorithms. These possibilities are more complicated for distributed algorithms which run in parallel. From the above-mentioned ideas and from the experience with distributed algorithms written in occam on transputer networks, which we have accumulated over the years, we decided to develop a graphical visualization environment that meets our demands, and those of our students. The GRAVIDAL environment combines some of the above-mentioned facilities.

GRAVIDAL is an *on-line* system that represents the behaviour of distributed algorithms during their execution phase. The specification of the data to be represented, is done in a similar manner as in the MOVIE-STILLS system. In this system the user can specify certain views of his program by placing some statements into his source code. During the execution phase of his program these statements produce special outcode, which must be compiled after the program has halted, to produce the so called script file. With the help of the script file the user-defined views of the program are animated.

We also used the idea of user-defined views because the user knows exactly which parts of his program are important and should be animated on the screen. But we have some important differences in contrast to the sequential idea of Bentley and Kernighan. GRAVIDAL visualizes the user programs during their execution. The graphical output task on the screen and the event collection task in the network are strictly separated. The advantage of this separation is the large modularity of GRAVIDAL, because these tasks can be enlarged or reduced as the user wants. Additionally, they can be modified or changed to run on other parallel computers or graphic screens.

The user has a choice between several macro calls, hidden in libraries, which can be placed in his program source. During the runtime of the distributed algorithm, GRAVIDAL routes the data to the screen. To get information from inside the net the user normally has to implement routing and monitoring techniques by himself. This work done from now on by GRAVIDAL and this is another advantage of the system.

At the moment, GRAVIDAL runs on the hardware configuration shown in Figure 1. The transputer network contains up to 32 T800 processors. The host transputer is placed on a VMTM-board (VME-Multi Transputer Module) from Parsytec. The connections are made via the VME-bus. Data can be sent in two directions between the host transputer and the Sun, which runs under UNIX. The incoming data on the Sun are displayed with the well known SunView graphics environment, running under Sun-tools. Unfortunately SunView is not device-independent. Therefore, the graphical output can be shown only on the screen of the Sun which is connected to the host. But we are working on a graphical output environment, which uses the device-independent facilities of the X-Window system.

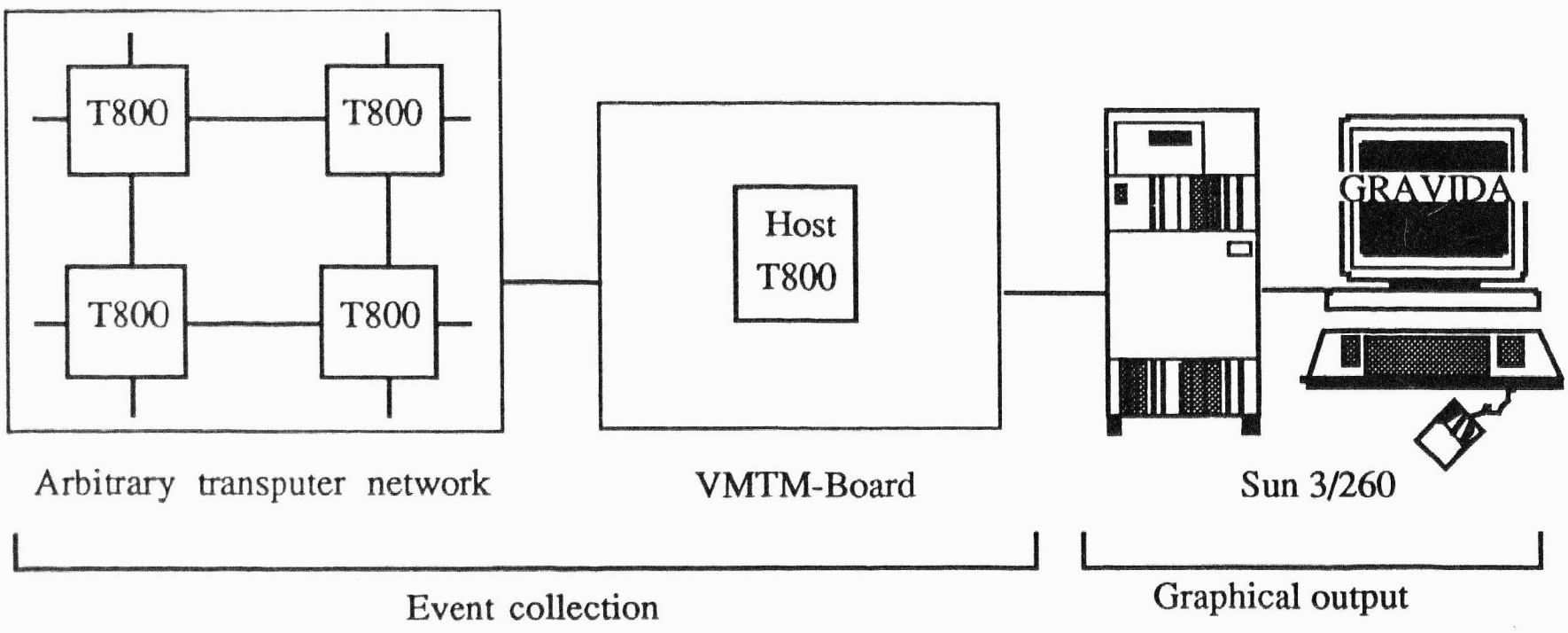


Fig.1: *Hardware configuration for GRAVIDAL*

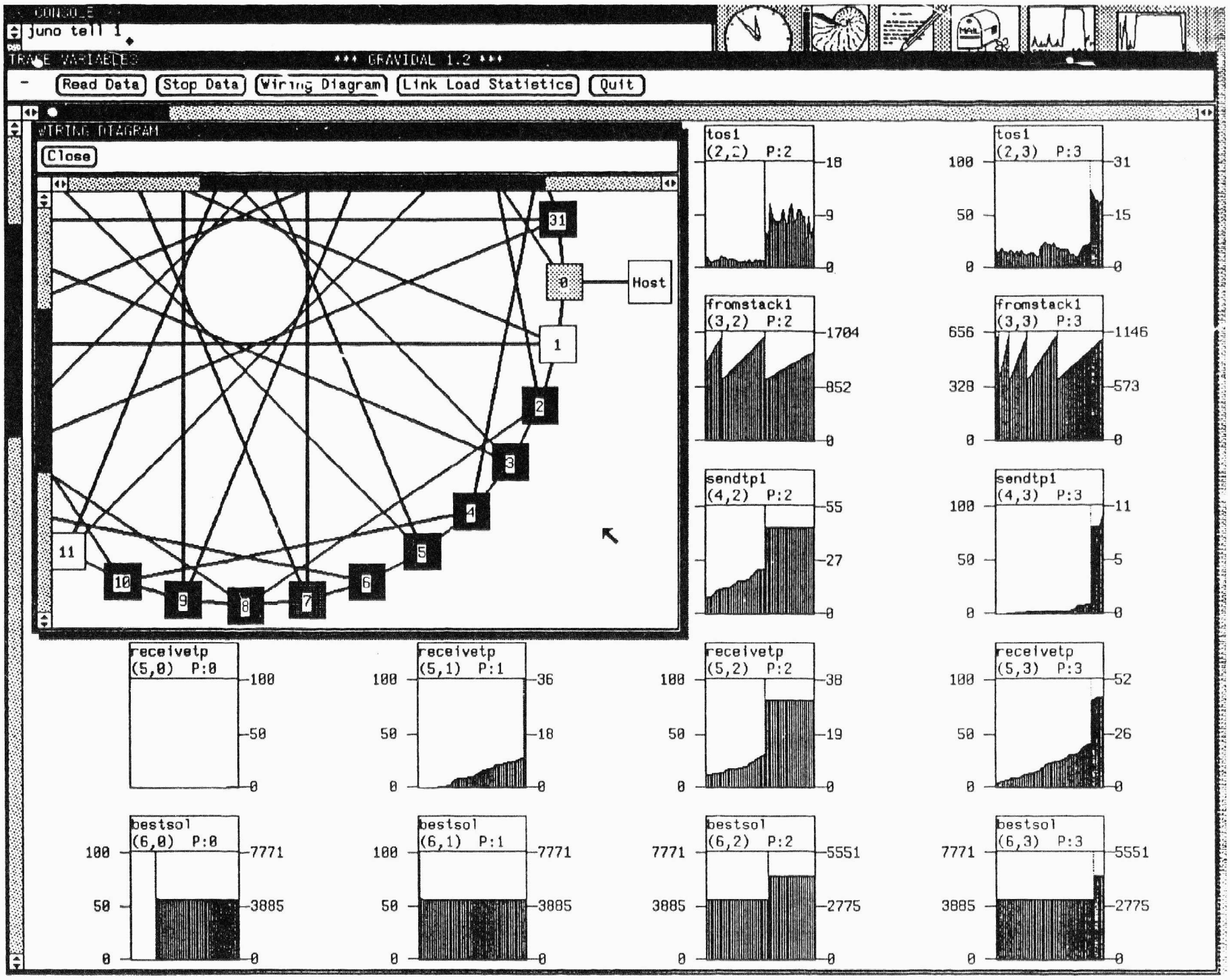


Fig.2: *Typical GRAVIDAL output*

A typical output of GRAVIDAL can be seen in Figure 2. It shows the network topology as a graph, the CPU-load of some processors and some user-defined traces of variables.

3. How to get a Visualized Algorithm

In this section we describe the steps that convert a distributed algorithm, written in occam, into an animated one.

- First, the user writes his distributed occam program as usual.
- Second, he adds some macro calls (for example CPU_Load, LINK_Load, VAR_Out, etc.) into his source code.
- Compilations of the EXE and PROGRAM folds are done as usual.
- The CONFIG INFO fold, in which the WIRING DIAGRAM and the BOOT PATH information are hidden is created.
- At the end, GRAVIDAL converts this program into a new source code in which the user-defined outputs are automatically routed to the host transputer and then sent to the Sun workstation, on which the outputs are drawn.

The last step is very complex and is explained in detail in the next subsections.

3.1 Scanning

The original program, with the embedded macro calls, is first scanned from GRAVIDAL to put some technical help macros and statements into the source code. In addition, a so-called supervisor process is built around the old user process to observe the in going and outgoing messages. These modifications are necessary to relieve the user of

some technical details of the macros and of the supervisor process. He only sees the user-friendly calls of the macros and has, for example, only to add the call "VAR_OUT(x)" into his source code to get the value of variable x on a network processor. Also the whole supervisor technique is invisible to the user and is added automatically by the scanner.

3.2. Boot path and Wiring Diagram Information

After changing the user program the information from the WIRING DIAGRAM fold and the BOOT PATH fold are scanned from GRAVIDAL.

From the WIRING DIAGRAM fold GRAVIDAL gets information about the topology of the user network. With this information, the network topology is automatically drawn on the screen. If the user does not accept this representation he can draw a new one, save the coordinates of this layout and load this file instead of the GRAVIDAL representation. The BOOT PATH fold contains information about the boot path from the host processor to each of the network processors in the from of a minimum spanning tree. So, for every network processor GRAVIDAL knows the shortest way to the host processor.

With this information, GRAVIDAL can run on any arbitrary transputer network.

3.3. The Supervisor Process

The original user process, which runs on one processor, and has up to four links, is embedded into a so-called supervisor process that simulates the behaviour of the old user node and, in addition, multiplexes user messages with event messages that were generated by the macro calls. These event messages have to be routed through the network to the host transputer. To make a clear decision between the event messages and the normal user messages the supervisor process creates new message headers so that the other processor in the network know exactly where a message must be sent to. An overview of the supervisor node is given in Figure 3.

The user application is the old program, written by user. GRAVIDAL reads information about the ingoing and outgoing links of the user application from its wiring diagram and the PROC definition. Then, it creates at least one, and up to four, *switch_in* and *mux_out* processes. If necessary, the *switch_in* processes are connected to two *buffer* processes in which messages are stored. If no message is sent to the *swith_in* processes they are descheduled. If a message does arrive it is sent via the *buffer* process to the *util* or *user application* process, according to its message type. All user messages are sent to the *user application* process and all event messages to the *util* process.

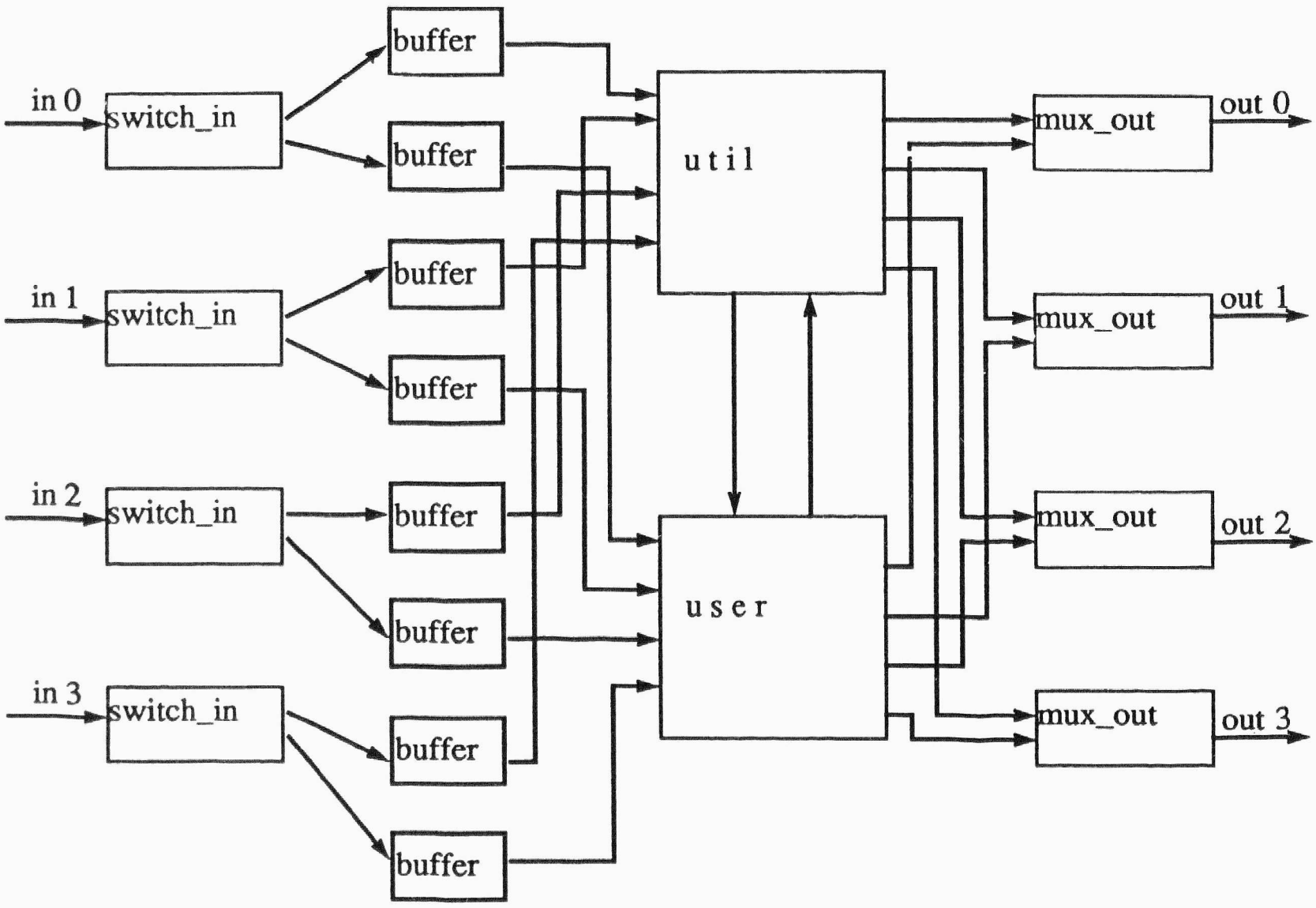


Fig.3: *Supervisor process*

This process then routes the event messages via the *mux_out* processes to the host. The *user application* process is responsible for its own user messages. Event messages which are created by the *user application* process are sent to the *util* process. To have a relation between the macro calls and the output windows on the screen, the macro calls inside the *user application* process generate special message headers, so that the graphic program on the Sun knows exactly which value is to be drawn in which window. All added processes (*switch_in, buffer, util, mux_out*) are active only if they send or receive a message. Otherwise, they are descheduled and do not waste processor time.

After the *user application* process has terminated, an end message is created and sent from every transputer to the host transputer. The host then sends an end message to all network processors, and on receipt of this message the network processes terminate. If some user processes are deadlocked, the corresponding user application processes from GRAVIDAL are deadlocked too, but graphical output from non-deadlocked transputer are routed through the network and displayed on the screen. So, GRAVIDAL has no influence on the termination of the original algorithm, but with the help of the graphical output we achieve a monitor facility which shows deadlocked processors on the screen.

3.4. Ordering of Events

To get the chronological order of the user defined events in the distributed system we have implemented a time stamp algorithm according [6], which simulates a logical global clock.

In every processor, there is a logical clock which assigns numbers to event, the number being the time at which the event occured. The logical clocks are initialized at the beginning of the animation. The definition that an event on processor X happens before an event on processor Y can be done if the processors communicate with each other. Otherwise these events are considered to be concurrent. If communication takes place, the time stamps which are included in each message can be compared and possibly adjusted. From these time stamps it can be whether an event A on processor X occurs before an event B on processor Y.

Every event message gets an additional time stamp. On the Sun workstation, GRAVIDAL then reorders, if necessary, these time stamps to obtain a consistent order of the events in the system.

The functions mentioned in subsections 3.1. - 3.4. are performed automatically by GRAVIDAL. So, the user gets new EXE and PROGRAM folds which run after compilation with the user-defined graphical views of his algorithm.

4. Macros

Some selected macros which the user can place into his source code are now explained in detail.

4.1. VAR_OUT(x)

One of the important and most-used macros is the VAR_OUT(x) macro. As its name suggests, this macro outputs the value of the variable x. It is possible that the name of the variable x exists in several processes, but GRAVIDAL automatically scans the scope of the variable which should be traced. The user is responsible for the names of the variables in one processor.

VAR_OUT(x) is macro which should be called out of a SEQ statement. Macro calls which output values from PAR and ALT statements are also available.

4.2. CPU_LOAD

To achieve high performance in the network many distributed algorithms use load balancing techniques. In other words, with the help of heuristics, they try to dynamically distribute subtasks to idle processors. GRAVIDAL supports a software implementation of a cpu-load monitor which measures the cpu-load of a processor by measuring the idle times. The code of the measurement consists of a *cpu_supervisor* process which must be run, at low priority, in parallel with the code of the program being measured.

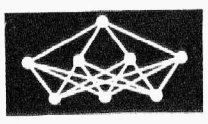
The *cpu_supervisor* process counts how often it finds the processor idle, and returns a value representing the percentage utilization of the processor. It works command driven, which means that the information of which point the measurement should be taken comes over a command channel, connecting the *cpu_supervisor* process with the program being measured.

The *cpu_supervisor* process works by counting the number of times it finds no processes in the low priority process queue of the transputer. When it is executed, it makes a note of the timer and deschedules itself, placing itself at the back of the ready queue. The next time the process comes to the head of the queue and is executed, it compares the present time with the last noted time. If the difference is sufficiently small, we can assume that only the *cpu_supervisor* process is in the ready queue and the processor is therefore idle. So the *cpu_supervisor* process is only active if the original program has nothing to do, e.g. because the processes are waiting for communication.

Our measurements show that a program which runs in parallel to the *cpu_supervisor* process has an average runtime overhead of only 1% . However, there are some more tricks in the implementation of the CPU-LOAD macro which could not be shown here. A detailed description of its implementation can be found in [5].

4.3. LINK_LOAD

To monitor the traffic along the communication channels GRAVIDALsupports a LINK_LOAD macro. This macro counts the bits and bytes which are transferred between two processors via the hardware links and compares them with the maximum link transfer rate which goes from 5 Mbits/sec up to 20 Mbits/sec for the T800 trans-



puter. These transferrates can be achieved under optimal hardware circumstances, but measurements in performance-maximized occam programs show that these hardware transferrates can not be achieved with normal user programs. So, the LINK_LOAD macro, which returns a percentage utilization of the link-load, is not as accurate as the CPU_LOAD macro.

The link transfer, say of an integer variable, works as follows. After division into four bytes, these bytes are sent, in addition to a start-, a data- and an end-bit, from processor X to processor Y. These eleven bits, resulting from one byte, are always acknowledged by two bits, which are sent from processor Y to processor X. The LINK_LOAD macro takes this data-transfer technique into account, and returns a percentage utilization of the link-load.

5.0. Graphical outputs, Examples and Time Measurement

One of the important application fields of GRAVIDAL is analysis of the performance of distributed algorithms. Many of these algorithms use load balancing techniques, i.e. with the help of heuristics they try to dynamically distribute subtasks to idle processors. For example, distributed algorithms from the field of operations research use distributed heap management techniques. With GRAVIDAL, the heap weights can be shown graphically during the whole network computation and therefore, anomalies in the distributed heap management can be detected very easily.

Figures 2, 4 and 5 show some GRAVIDAL layouts. They are snapshots of the rutime phase of a distributed Branch-

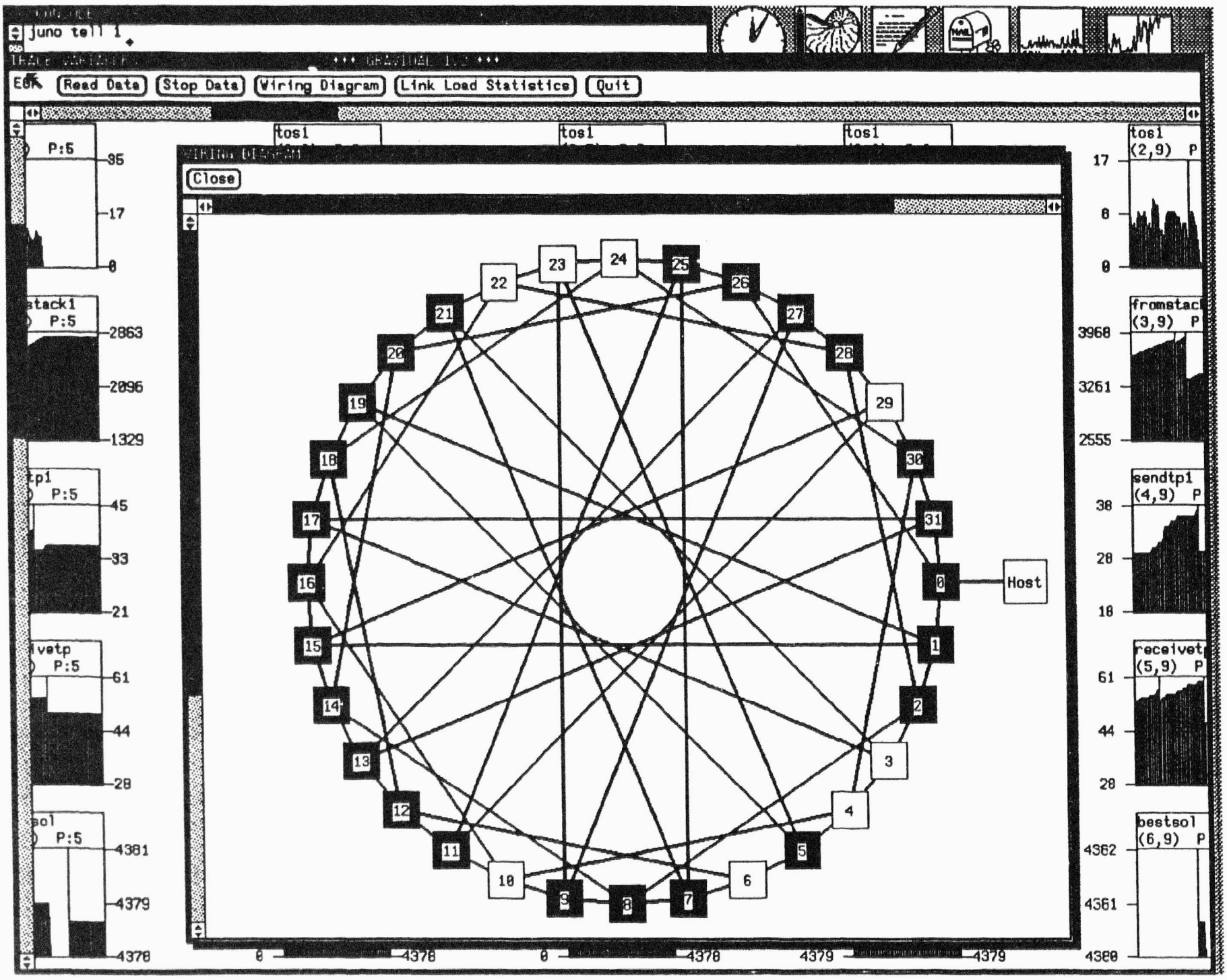


Fig.4: *CPU-load output*

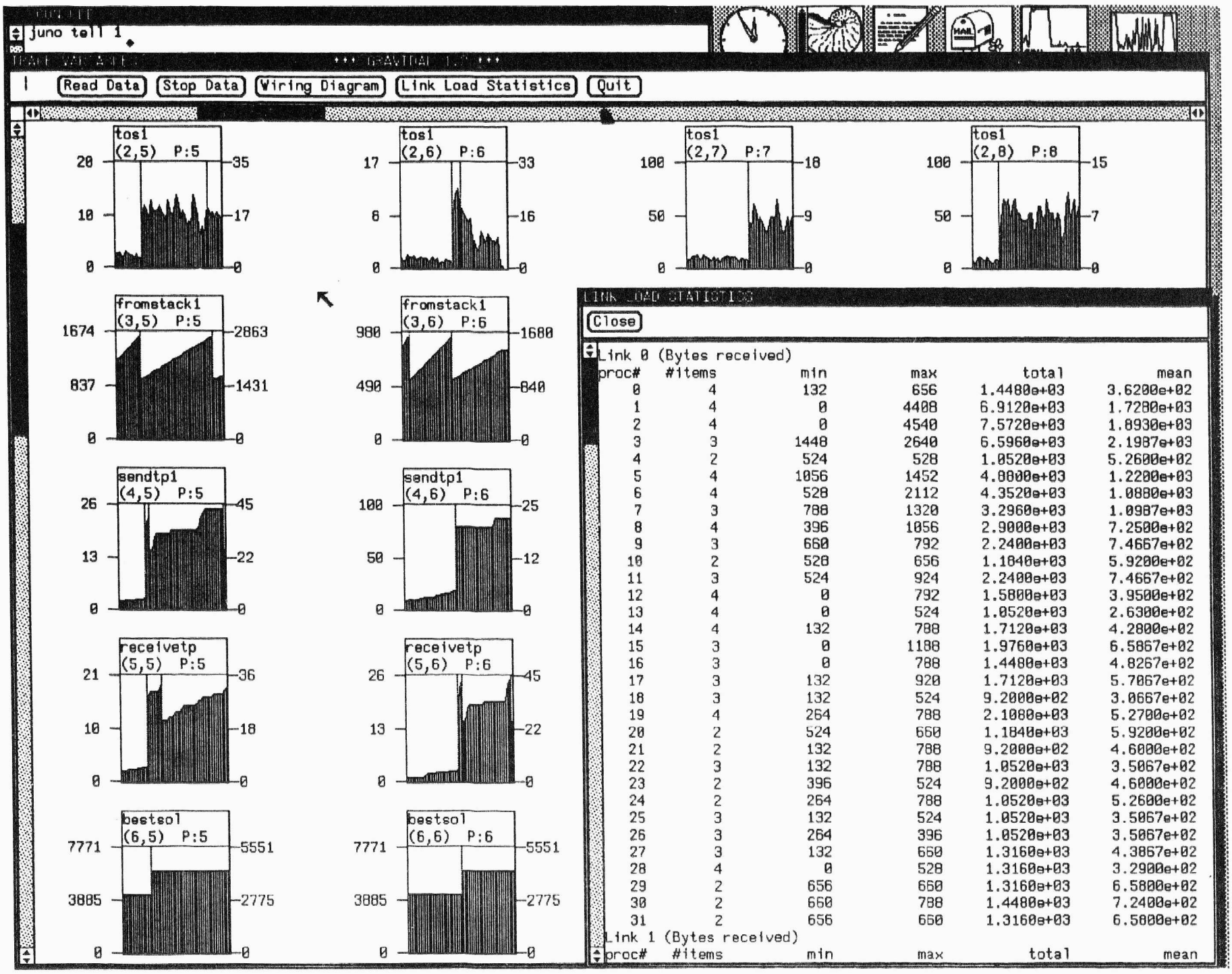


Fig.5: *Traced variables*

&-Bound algorithm, which calculates the NP-complete "m processor scheduling" problem.

In Figure 2, a typical GRAVIDAL layout is shown. Behind a little part of the network topology one can see a lot of parameters in which the user defined views of the values of certain variables are visualized. In every parameter the names of the variable and the processor number are shown. The boundaries of a single parameter are self-adjustable so that the closest boundaries are always shown. Additionally, the user can modify the names, the placements, and the boundaries of parameters during the runtime of his algorithm.

Figure 4 shows the complete 32 transputer network which was used for the testruns. The layout of this network is automatically drawn by GRAVIDAL after reading the BOOT-PATH and WIRING-DIAGRAM-fold. The different gray tones of the processor icons of the network topology show the CPU-load of each processor. A white icon indicates 0% CPU-utilization and a black icon shows 100% CPU-utilization. With this graphical visualization, the user can identify the cold spots or hot spots in his network and he gets an impression how good or bad his load balancing strategy is.

To see the LINK-LOAD statistics, the user has a choice between a representation as large number columns or as a perfmeter (see Figure 5). But here it can be seen that visualization as a perfmeter is much easier to recognize grasp than large number columns.

GRAVIDAL offers many another graphical features which canot be shown in detail here.

To see what price must be paied for a visualized distributed algorithm both GRAVIDAL, we evaluated the run time difference between some user programs and their animated counterparts and call this the *overhead* of the graphical visualization.

GRAVIDAL is a software environment and therefore it is clear that if the number graphic and user messages increases, the overhead increases too. Therefore, the graphi-

cal visualization overhead is user-dependent. If the user wants to see, for example, the values of all variables in his network, then he has to take into consideration that the runtime of his distributed algorithm increases dramatically.

Our measurements show that for the above-mentioned example and for a lot of other testruns that the overhead lies between 10% - 15% if the macro calls produce less than 2KBytes/sec of graphical information. If more than 2KBytes/sec information is produced, the overhead increases accordingly. But the measurements show that Figures 2, 4 and 5 are produced with less than 2KBytes/sec. Additionally it can be seen that the user loses control over the things which are visualized if the graphical messages produce more than 2KBytes/sec information. If the user wants only to see the CPU-Load and some values of certain variables, the overhead is extremely low.

Normally, the user has to implement his own routing and monitoring algorithms to know the behaviour of his network. Therefore, he spends his own time and a lot of processor time too. So the everhead measurement has to be seen in relation to these times.

6.0. Conclusions

GRAVIDAL represents a new concept in graphical visualization of distributed algorithms.

The system architecture of GRAVIDAL is modular so that the user can build his own visualization for every application from a wide range of possibilities.

The time measurement of the overhead is in an acceptable range, if we take into consideration that for a "home made" output the user would have to implement all these features by himself.

Therefore, with GRAVIDAL the user has an easy-to-use software tool for the analysis and the design of his distributed algorithms.

Acknowledgments. Our student Thorsten Telljohann did an excellent job in programming GRAVIDAL.

References

[1] Bentley, J.L., Kernighan B.W.: A system for Algorithm Animation, Tutorial and User Manual, AT & T Bell Laboratories Computing Science Technical Report No.132, 1987.

[2] Brown M.H.: Algorithm Animation, ACM distinguished dissertation 1988, The MIT Press.

[3] Nurns A: Programming in Occam 2, Addison-Wesley, 1988.

[4] INMOS Limited, Transputer Development System, Prentice Hall, 1988.

[5] Jones G. with contributions of Rabagliati A., Zeppenfeld K. and Goldsmith M,: Measuring the Business of a Transputer, Occam User Group newsletter No.12, Jan.1990, 57-64.

[6] Lamport, L.: Time, clocks, and the ordering of events in a distributed system, CACM Vol.21, No.7, 558-565, July 1978.

[7] Stepney, S.: GRAIL: Graphical Representation of Activity, Interconnection and Loading, Processings of the 7th OUG Meeting, Sep.1987.

PATTERN CLASSIFIER, AN ALTERNATIVE METHOD OF UNSUPERVISED LEARNING

A.E. Günhan [1]

Abstract: This model and its learning algorithm is based on the neurophysiological activity of real neurons. Natural Neural Networks have an extremely powerful self-organizing property. In the present model, this self-organization property emerges not only as a consequence of mutual inhibition of neurons, but also as a result of a very simple and plausible learning principle governing the individual neurons.

Key words: *learning algorithm, real neurons, self-organization.*

Received: September 4, 1991
Revised and accepted: December 9, 1991

1. Architecture of the Network

The network is composed of two feed forward layers. 64 input units, a hidden layer with 64 hidden units and an output layer with 8 units (see Fig.1). The layers are fully interconnected to the next layer. The output and hidden layers have internal lateral inhibitions. We may consider the network as a combination of two different single layer networks.

1.1. Features of the network

The first part that is between input units and hidden layer accepts input patterns one at a time. In other words, when we present the first pattern to this part, only this part of the network will be trained until we obtain internal representations in the hidden unit for this pattern. Before we can present the second pattern this part of the network will be initialized. After we have obtained an internal representation for the first pattern in the hidden layer, this representation is presented to the second part of the network as an input for final classification of the pattern.

The second part of the network can be considered as another single layer network between the hidden layer and output layer. The function of this second network differs from the first part of the network in terms of the way of training. This part accepts the internal representation as input and the layer is trained untill the pattern is classified. The next internal representation, which is the result of the second pattern that is presented in the first part, will also be accepted as second input pattern to the second part of the network without any initialization. In this way, the second part of the network is capable of training and memorizing a sequence of patterns as in traditional training.

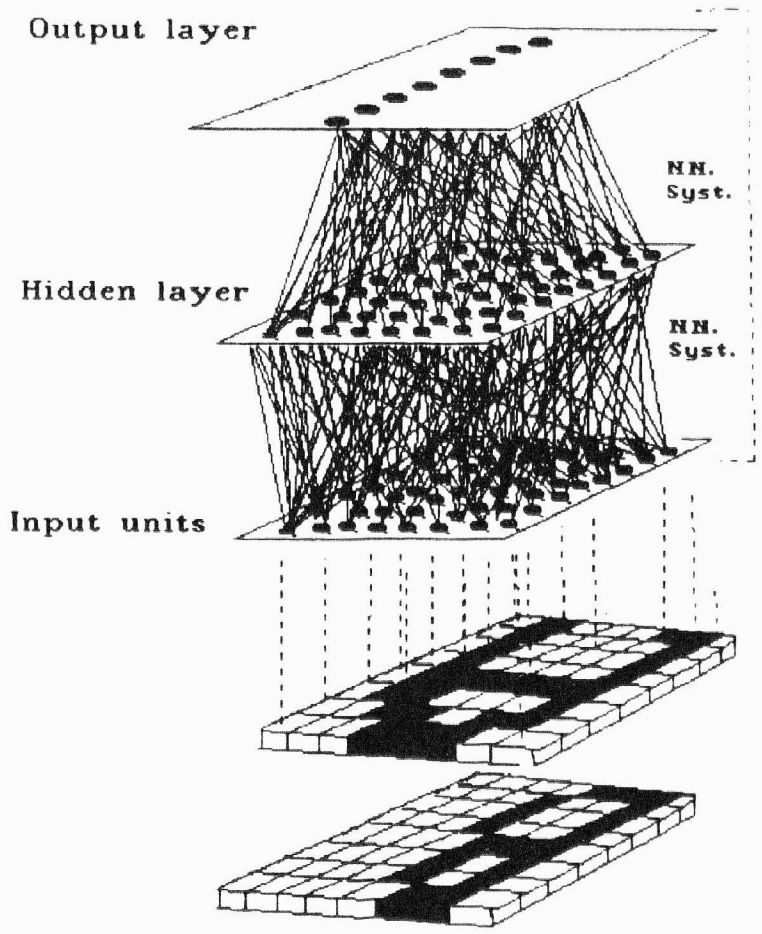


Fig.1: *Architecture of multi layer network*

A *"neighbour inhibition"* method with an Eight-directional Inhibition Strategy (see Fig.2) is used in the layer of the first part of the network that is in fact the hidden layer for the whole network. In this strategy most active unit inhibits the activity of its closest neighbour units. Units in the layer are organized in a two dimensional array. Positional overlapping is not allowed by the application program. Therefore the number of neighbour units

[1] Atilla E. Günhan
Department of Information Science, University of Bergen, Hightechnology center, N-5020 Bergen, Norway

can vary from 3 to 8. In the output layer of the network, the *"Winner-take-all"* method is used.

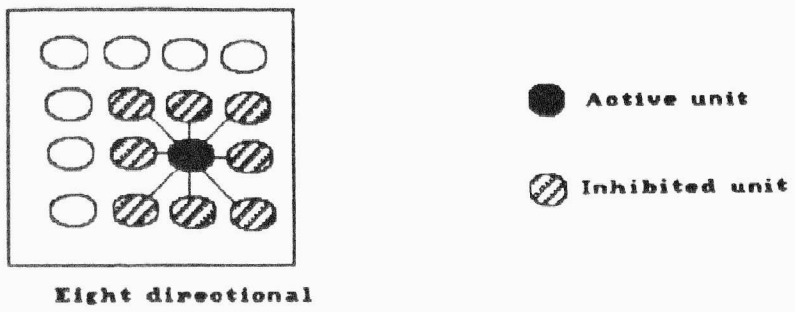


Fig.2: *Eight-directional Inhibition Strategy. Most active unit inhibits the activity of its closest neighbour units.*

1.2. Lateral inhibition

Lateral inhibition is a special form of neural interaction. In the 1959's some theories have been developed to describe the lateral inhibition mechanism and to show its possible importance in perception. This provides an ability to enhance contrast, peaks and edges of incoming patterns. Lateral inhibition can also be used in preprocessing noisy data. In the present work the lateral inhibition property is based on Purkinje cell activity in the neurophysiology. This property is studied in ref. 4, 5.

According to the principle of lateral inhibition [3] the activity of a unit not only depends upon the stimulus received by itself, but also upon the activity of units in its neighborhood. Such an interaction has been observed in the retina, auditory pathways and Purkinje cells of vertebrates.

1.2.1. Neighbour neuron inhibition

Neighbour neuron inhibition is actually accepted as real lateral inhibition mechanism [3] which is similar to neurophysiological aspects. In this work; at a given time the neurons which are not fired will keep their different inhibited values and will join the competition continuously until they are fired (see Fig.3). The fired neurons will return to their minimum value. This minimum value is called *resting-membrane-potential* in real neurons. In this way one can build more complex systems which allow different neurons to fire simultaneously in a given time.

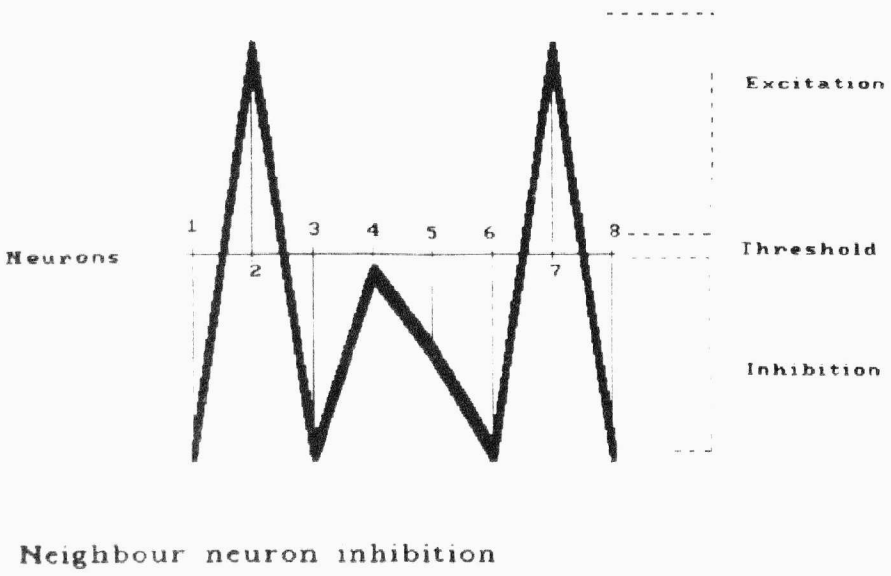


Fig.3: *Neighbour neuron inhibition. In figure neuron numbers 2 and 7 are fired. Neuron number 4 is approaching the threshold*

1.2.2. Winner-take-all

This method is used in most known unsupervised learning algorithms [7,8,10]. In the *winner-take-all* method each unit in the output layer inhibits every other unit in the same layer. There is only one unit in a given time which wins the competition and this unit remains activated while the others are pushed to their minimum value (see Fig.4).

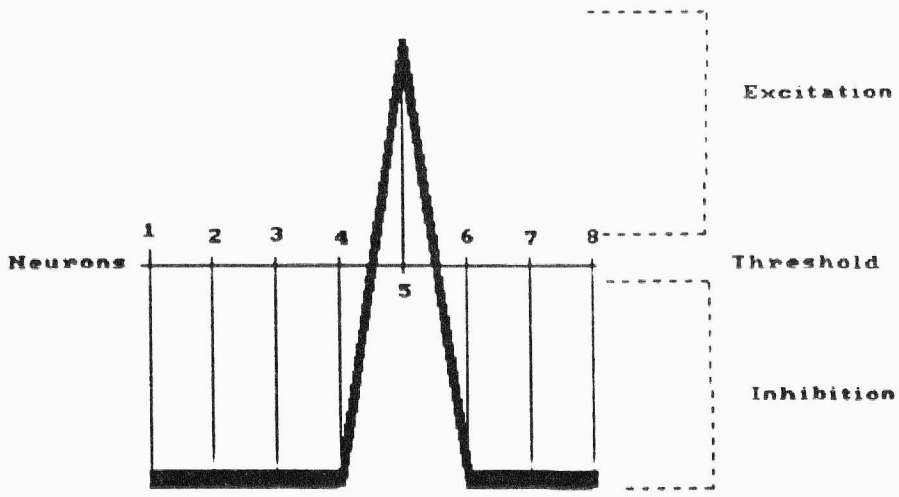


Fig.4: *Winner-take-all function. In figure neuron number 5 is the winner and it pushes other neurons into a constant minimum value*

1.3. The training process

In the beginning of the process the weights in each part of the network are initialized with small random numbers between 0.1 and 0.9. Afterwards weight vectors in each part of the network are normalized according to the normalization condition which is explained in the next sections. Input patterns in each part of the network are also normalized during the whole process.

The first part of the network is trained for the first incoming normalized pattern until internal representation is achieved. Then this internal representation for the first pattern is presented to the second part of the network as normalized input pattern. The second part of the network is then trained for this input pattern until final classification is achieved.

Before presenting the second pattern to the network the weights in the first part of the network are initialized and normalized while the weights in the second part of the network remain with the latest modified values of the weights. The process continues until the network is trained with all patterns.

The main property of this network is that: The first part of the network behaves as a *temporary storage* and the second part behaves as a *permanent storage*. This is somewhat similar to *short term* and *long term* memory concepts in biological systems [2].

2. Learning Algorithm of the Network

There is a large number of fibers that provide synaptic connection to a given Purkinje cell (see Fig.5) in a Natural Neural Network [1,9]. We denote the number of all parallel fibers by N. The input carried by the k^{th} parallel fiber at the time t is denoted by $s_k(t)$; it can have the value of 1 or 0 according to whether the fiber carries an impulse or not. The effect of the input from the k^{th} fiber on the i^{th} Purkinje cell is determined by the synaptic coupling strength $W_k^{(i)}$. The output of the network is carried by the axons of the M Purkinje cells and acts to inhibit muscle activity. The strength of the effective inhibition of a cell denoted by g. This activity of the i^{th} Purkinje cell is characterized by the quantity $a^{(i)}(t)$, which is 1 or 0 depending on whether the cell it originates at is active (*i.e.* has fired) or not. The activity state of the Purkinje cell in turn depends on its axon hill potential, $v^{(i)}(t)$.

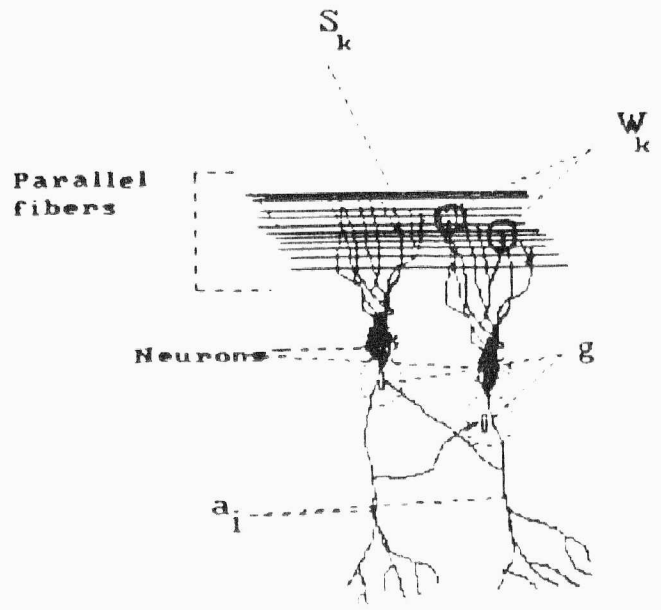


Fig.5: *Symbolic notations for two Purkinje cells.*

2.1. Neuronic equation (Netinput)

To describe the behaviour of a neuron, I will employ a formulation based on ref. 4. The dynamic behaviour of a neuron is governed by the *neuronic equation:*

$$v^{(i)}(t+\tau) = \Big[(1-\lambda)v^{(i)}(t) + \sum_{k=1}^{N} W_k^{(i)}(t)s_k(t) - \sum_{j=1}^{M} g_j^{(i)}a^{(j)}(t)\Big](1-a^{(i)}(t)), \quad (1)$$

where

- $v^{(i)}(t+\tau)$: Hill potential at time $(t+\tau)$ of cell i. (In natural systems this time is $\tau \approx 1$ msec.)
- $(1-\lambda)v^{(i)}(t)$: The remaining potential of neuron i from time t at time $(t+\tau)$. If no other input arrives from fibers then the potential of the neuron decreases exponentially with λ decay constant ($\lambda = 0.1$ in my appl.)
- $W_k^{(i)}(t)s_k(t)$: The excitation arriving from the k^{th} parallel fiber to neuron i at time t.
- $g_j^{(i)}a^{(j)}(t)$: The inhibition arriving from other neurons to neuron i at time t.
- $(1-a^{(i)}(t))$: A multiplicative factor. This term resets the potential to its resting value after the firing and keeps it there the refractory period (the minimum time between two firings), during which the neuron is not excitable.

A neuron will become active, if its potential $v^{(i)}$ at the axon hill exceeds a threshold value, $\theta^{(i)}$, ($\theta = 1$ in my appl.). In that case the neuron will emit a non-decreasing pulse of fixed duration τ, along its axon making inhibitory action at all of its synaptic connections.

This activity is described as,

$$a^{(i)}(t) = \begin{cases} 1, & \text{if } v^{(i)}(t) \geq \theta^{(i)} \\ 0, & \text{otherwise} \end{cases}$$

Given the time-dependent input $s_k(t)$, the response of the network can be calculated with eq. (1). For a shorter period, the synaptic strengths $W_k^{(i)}$ and $g_j^{(i)}$ can be considered as constants. If the incoming pattern remains steady for several time steps; those Purkinje cells whose synaptic strength vectors have the best overlap with the input pattern vector will be most likely to fire; in turn, they will then inhibit their neighbouring neurons as a consequence of the lateral inhibition $g_i^{(j)}$. If the membrane potential, v, of a neuron exceeds a given threshold value, θ, then neuron becomes active (fired) and inhibits its neighbours with a certain inhibition value. Active neuron also resets itself into resting membrane potential value.

2.2. Memory equation (Learning)

On a longer time scale the coupling strengths may change, thus providing a learning ability for the net. When we assume that the inhibitory synapses are fixed, the excitatory synapses leading from the parallel fibers to the Purkinje cells may change according to Hebb's rule [6], which we formulate in the following *memory equation:*

$$W_k^{(i)}(t+\tau) = q^{(i)}\big[(1-\varepsilon)W_k^{(i)}(t) + \delta a^{(i)}(t)s_k(t-\tau)\big], \quad (2)$$

where

- $W_k^{(i)}(t+\tau)$: Synaptic strength of coupling between fiber k and neuron i at the time $(t+\tau)$.
- $q^{(i)}$: Normalization constant for neuron i.
- $(1-\varepsilon)W_k^{(i)}(t)$: Exponential decay of synaptic strength ($\varepsilon = 0.001$ in my appl.)

- $\delta a^{(i)}(t)s_k(t-\tau)$: If the neuron i is active at time t (i.e.,$a = 1$), the connections between neuron i and fiber k will be strengthened by δ learning rate.

The normalization condition

$$q^{(i)} = \eta^{(i)}/(\sum_{k=1}^{N}[(1-\varepsilon)W_k^{(i)}(t) + \delta a^{(i)}(t)s_k(t-\tau)]) \quad (3)$$

where $\eta^{(i)}$ is a constant (i.e., $\eta^{(i)} = 1$). This condition ensures that the total synaptic strength remains constant,

$$\sum_{k=1}^{N} W_k^{(i)}(t) = \eta^{(i)}.$$

This learning mechanism simply describes the effect that if an excitatory impulse is arriving to a synaptic coupling which will lead in the following time step to a firing of the post-synaptic cell, then that connection will be strengthened by δ. Due to the normalization $\eta^{(i)}$, and the slow exponential decay of the couplings, all "non-successful" synapses will be somewhat weakened at the same time.

Exponential decay $(1-\lambda)$ in *neuronic equation* and $(1-\varepsilon)$ in *memory* equation are ignored in the second layer of the network to obtain plausible classification of patterns.

Learning algorithm for the second layer of network is as follows

neuronic equation

$$v^{(i)}(t+\tau) = [v^{(i)}(t) + \sum_{k=1}^{N} W_k^{(i)}(t)s_k(t) - \sum_{j=1}^{M} q_j^{(i)}a^{(j)}(t)](1-a^{(i)}(t)), \quad (4)$$

and *memory equation,*

$$W_k^{(i)}(t+\tau) = q^{(i)}[W_k^{(i)}(t) + \delta a^{(i)}(t)s_k(t-\tau)]. \quad (5)$$

3. Discussion

This model has many advantages compared with other models. The learning algorithm and architecture of the network explained above are different from other network models. In competitive learning the output unit with maximum value is chosen as the winner and all activation for the other units are set to zero value. Competitive learning does not produce internal representations of input patterns. Choices for learning parameter values and the number of training iterations are adjusted experimentally to obtain the best results.

In Kohonens' network model, distances between all input and output units are calculated. The unit with lowest distance is the winner. The weights of the units within a square with the winning unit in the center are adjusted.

The advantages of the two layer model presented above are:

Ability of prepossessing the incoming pattern.

The network has ability to preprocess the incoming patterns and reduce the linear dependency among these patterns (see Fig. 6). The network has ability to learn to respond in different parts to differences in input signals by using the neighbour inhibition mechanism. These responses are represented in the hidden layer as internal representation of a given pattern.

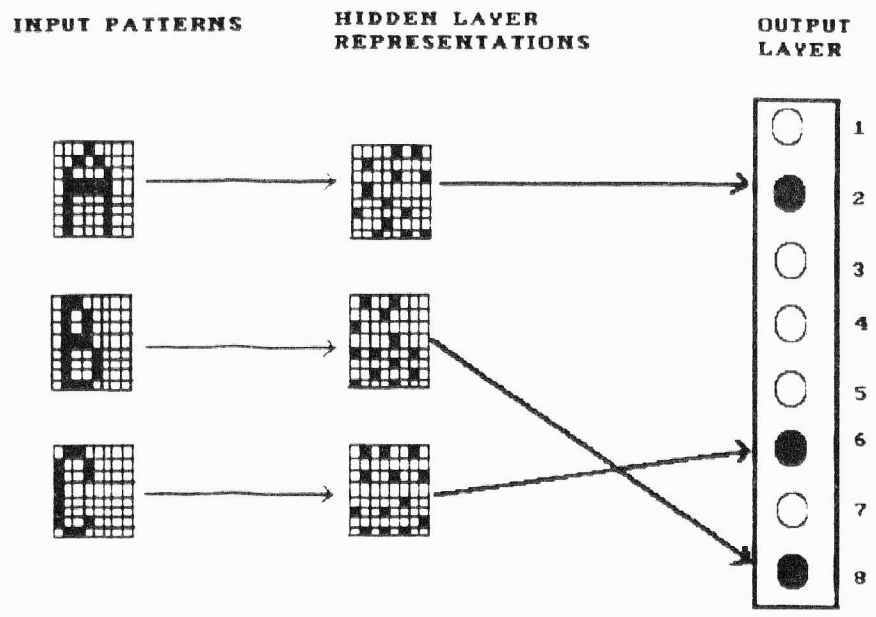


Fig.6: *Input patterns A,B,C, their internal representations and response of the network in the output layer.*

Experiments indicate that the network has ability to classify all input vectors as long as those vectors have less than 50% common elements with each other. When the patterns **A,B,C,1** and **2** presented to the network (see Fig.7), patterns **C** and **2** have been classified into same class.

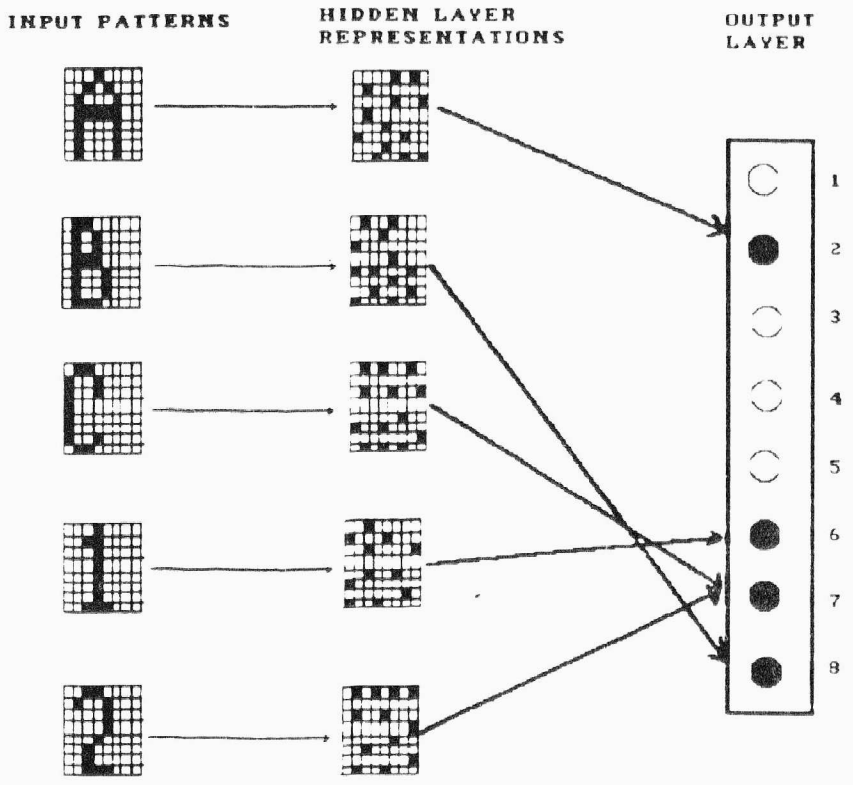
The reason is that common elements of these two patterns are more than 50% of the number of representation elements. Patterns **A** and **C** had 4 common elements but they have been classified into their own classes since their uncommon elements were in majority. The second layer of the network is limited with properties of "*winner-takes-all*" method.

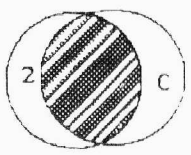
Network architecture and learning algorithm are similar to biological mechanisms.

The first part of the network is a temporary storage for each pattern. The task for this part of the network is to preprocess incoming patterns like a filter, in other words, to reduce the linear dependency among patterns by determining their relevant representations. These representations are the input patterns for the next part of the network. The second part of the network classifies incoming pattern information that comes from the hidden layer. As long as the hidden layer has preprocessed pattern information, the final classification and memorizing process will be easy.

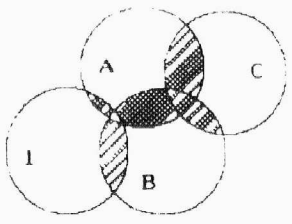
The learning algorithm is based on neurophysiological activity of real neurons. If the membrane potential, v, of a neuron at a given time exceeds a certain threshold va-

lue, θ, then the neuron becomes active (fired) and inhibits its neighbours with a certain inhibition value. Only the weights of the active neuron are adjusted. Active neuron also resets itself into resting membrane potential value.





Pattern 2 is dominated by pattern C



Patterns A, B, C, 1 classified into four classes.

Fig.7: *Input patterns A,B,C,1,2, their Internal representations and response of the network in the output layer.*

Ability to recognize noisy patterns.

Experiments showed that this network also has ability to recognize ambiguous or noisy incomplete input patterns as long as, noise were not dominating these input patterns.

Network accepts repeated patterns.

Repeated patterns during training do not cause any side effect. They activate always the same output units.

Values of parameters are fixed.

Choice of parameter values for learning and the number of training iterations are fixed. The network may be trained with a value of learning rate between 0.1-0.6, 3 epochs and 50 cycle for each pattern. These values are valid for any type of pattern. The number of epochs can be reduced to 2, learning rate to 0.1 and the pattern cycles to 5. These reduced values give the same results but the number of firings is minimized. Inhibition values between the units are set to 0.9 when the method is neighbour inhibition.

Disadvantages of the network were common with all other networks in this field. The capacity problem occurred because the internal representations of an input pattern required many hidden units. This problem may be reduced by increasing the area of inhibited neighbour neurons. The experiments with neighbour inhibition showed that when we increase the direction of inhibition from 2 to 8, common activated units among different input patterns are reduced. The "*winner-takes-all*" method was actually the best method to reduce common elements among the internal representations of input patterns. This method is used for final classifications in the network. If we use this method in the hidden layer with *one at time* training, the pattern classification may be maximized in this multi layer network but this type of mechanism has the solution for only well-defined input patterns as in competitive learning by Rumelhart.

For research on patterns which are not well-defined, we need to observe more representations in the hidden layer by using neighbour inhibition mechanism for future investigations. Eliminating common internal representations of not-well-defined patterns could be the solution for classification of these type of patterns. This method has possibly three disadvantages. First, we need an extra filter mechanism in the network before we may present the internal representations of patterns to the last part of the network. Second, if we increase the number of patterns which will be passed through this filter mechanism, the number of existing active units from the previous filter (internal representations which are results of the training first part of the network) will probably decrease strongly and will cause non-active units at the end of the eliminating process. Third, if the network trains with reduced information, it will not be able to response correctly in a recalling process. Thus, we must study mechanisms like neighbour inhibition and experiment with more amount of hidden units or layers.

Another problem is that the network classified the same pattern differently when we changed the location or size of the input pattern. This is a usual problem for most of the unsupervised learning algorithms.

Conclusion

In the present work, an alternative unsupervised neural network model that may approximate certain neurophysiological features of Natural Neural Systems has been studied. This work is a result of further investigations on ref. 4 and 5. However the present model shows us the ability of machinewritten character recognition. It can also be investigated within the subject of signal processing. This model can work as a feature detector or signal classifier in many application areas like speech recognition, active sonar classification, signal analyses on nuclear power plants etc.

Acknowledgement. I want to thank Professor Svein Nordbotten and Professor László P. Csernai for their usefull discussions and constructive comments to my ideas.

References

[1] Bullock, T.H.: Introductions to Nervous Systems. W.H.Freeman and Company, San Francisco 1977.

[2] Caianiello, E.R.: Outline of Theory of Thought - Processes and Thinking Machines. J.Theoret. Biol., 1961, **2**, 204-235.

[3] Caianello, E.R.: Cybernetic of Neural Process. Ricerca Scientifica, Roma, 1965.

[4] Günhan, E. A., Csernai, L.P., Randrup, J.: Unsupervised Competitive Learning in Neural Networks. International Journal of Neural Systems, World Scientific, London, Vol.**1**, No.**2**, 177-186

[5] Günhan,E. A.: Pattern recognition and Self-Organization of Neural Networks. Dept. of Information Science, University of Bergen, Norway, Master thesis, March 1991.

[6] Hebb, D.O.: The Organization of Behaviour. Wiley, New York, 1949.

[7] Hinton, G.E.: Connectionist Learning Procedures. Technical Report, CMU-CS-87-115, University of Toronto, Canada 1987.

[8] Kohonen, T.: Self-Organization and Associative Memory. Springer Verlag, New York 1984.

[9] Pellionisz, A., Llinás, R.: A computer model of Cerebellar Purkinje Cells. Neuroscience, 1977, Vol.**2**. 37-48.

[10] Rumelhart, D.E., McCelland, J.L. and The PDP research group: Parallel Distributed Processing. The MIT press. Cambridge 1986, Vol.**1-2**.

Literature Survey

The literature on neuroscience increases last few years extremely fast. At present some estimations of more than 20000 existing papers, conference and symposium talks, books and research reports are made. Evidently it is not possible to inform the readers about all the interesting publications, which currently appear. However, we would like to use the existence of the computer oriented Scientific Information System of the Institute of Computer and Information Science in Prague for to present here almost regularly the short survey of the last year records of this base.

Of course, the readers are asked for to be so kind and inform the Editors or the Institute about any publication, which they recommend to insert in this literature survey.

Mundie D.B., Massengill L.W.: Weight Decay and Resolution Effects in Feedforward Artificial Neural Networks

IEEE Trans. on Neural Networks Vol.2, 1991 No.1 pp.168-170

Abstract: This letter presents results from a preliminary study on the effects of weight decay and resolution on the performance of typical three-layer, feedforward neural networks.

Salam F.M.A., Wang Y., Choi Myung-Ryul: On the Analysis of Dynamic Feedback Neural Nets

IEEE Transactions on Circuits and Systems Vol.38, 1991 No.2 pp.196-201

Abstract: We present some theoretical results pertaining to the dynamic properties of dynamic feedback (artificial) neural networks. We also present some conditions to SYSTEMATICALLY specify the equilibria for these networks. Our emphasis throughout will be on the design of feedback artificial neural nets of the Hopfield type and their potential use as classifiers.

Shoemaker P.A.: A Note on Least-Squares Learning Procedures and Classification by Neural Network Models

IEEE Trans.on Neural Networks Vol.2, 1991 No.1 pp.158-160

Key words: Neural Network Models - Classification.

Abstract: In recent years there has been considerable interest in the capabilities of neural network models applied to tasks such as classification, which typically require some a posteriori judgment of likehood based upon probabilistic data. This has led to analyses of network learning and function in a statistical context.

Shynk J.J., Bershad N.J.: Steady-State Analysis of a Single-Layer Perceptron Based on a System Identification Model with Bias Terms

IEEE Transactions on Circuits and Systems Vol.38, 1991 No.9 pp.1030-1042

Key words: neural networks; single-layer; perceptron based; bias terms.

Abstract: A stochastic analysis is presented of the steady-state convergence properties of a single-layer perceptron for Gaussian input signals. A system identification formulation is presented whereby the desired response signal $(+-1)$ is modeled by an unknown linear FIR system F plus an unknown bias, followed by a signum function nonlinearity.

Sietsma J., Dow R.J.F.: Creating Artificial Neural Networks That Generalize

Neural Networks Vol.4, 1991 No.1 pp.67-79

Key words: back-propagation; pattern recognition; generalization; hidden units; pruning.

Abstract: We develop a technique to test the hypothesis that multilayered, feed-forward networks with few units on the first hidden layer generalize better than networks with many units in the first layer. Large networks are trained to perform a classification task and the redundant units are removed ("pruning") to produce the smallest network capable of performing the task.

STATISTICAL MODELS AND TESTS OF INTRAFASCICULAR NERVE FIBER ARRANGEMENTS

Ove Frank[1]

Abstract: In order to gain insight into the structure and organization of human peripheral sensory nerve fascicles, it is possible to use a microneurographic method described in a recent article by Hallin, Ekedahl and Frank. This method uses a specially devised concentric needle electrode for obtaining intrafascicular recordings of nerve activity. The statistical analysis of such recordings is discussed here, and stochastic models are developed for testing various hypothesis on nerve fiber arrangements. The general approach is illustrated by analyzing experimental data collected for studying particular segregation and clustering phenomena of fibers in human sensory nerve fascicles.

Key words: *microneurographic models, peripheral nerve organization, nerve fiber segregation.*

Received: September 4, 1991
Revised and accepted: December 9, 1991

1. Introduction

Peripheral nerve fascicles consist of bundles of nerve fibers of different modalities. The intrafascicular organization of bundles and fibers is of neurological and physiological interest [1,3,5,6,7].

A neuron can be described as a cell having short branches or dendrites which receive information or impulses from other neurons and a long branch or axon which transmits information to other neurons. The axon is the core of a nerve fiber. It is surrounded by a myelinated isolation except at certain points where impulses may pass along the axon. These transmission points are called the nodes of Ranvier.

In order to gain insight into the structure and organization of human peripheral sensory nerve fascicles, a microneurographic method was developed by Hallin and Wiesenfeld [4]. This method uses a specially devised concentric needle electrode for obtaining intrafascicular recordings of nerve activity. In two recent articles by Frank [2] and Hallin, Ekedahl and Frank [3] the statistical analysis of such recordings is described. The findings give significant evidence against a random distribution of fibers of different modalities and support the hypothesis of segregation by modality of the fibers. These experimental results should preferably be analyzed further to find our whether or not there are random or systematic locations of the nodes of Ranvier in neighboring fibers and bundles of fibers. For that purpose the previous statistical models and methods have to be extended.

A possible extension is described in the following. The next section specifies a stochastic model of intrafascicular nerve fiber arrangement which makes it possible to alternative fiber arrangement hypotheses. By comparing such simulated results with real experimental results, the hypotheses can be statistically tested as elaborated on in Section 3. Section 4 illustrates the method by using data from the experiments described in Hallin, Ekedahl and Frank [3]. Further results using this method are presently being analyzed and will be published elsewhere.

2. A Model of Intrafascicular Fiber Arrangements

We consider k different modalities of nerve fibers in a fascicle, and these modalities are labeled by integers $1, \ldots, k$. Let $P_1, \ldots, P_k$ be the relative frequencies by which the modalities occur. Here $0 < P_j < 1$ for $j = 1, \ldots, k$ and $P_1 + \ldots + P_k = 1$.

A perpendicular path of the recording electrode through a fascicle can be considered as a screening procedure by which the modalities of the fibers encountered along the path can be specified. Let $U_1, \ldots, U_n$ be such a sequence of modality labels along a path crossing n fibers. All these labels cannot be identified, through. The special recording device can identify the modality label of a fiber only if this fiber has a node of Ranvier sufficiently close to the path. Therefore we introduce a sequence $V_1, \ldots, V_n$ of indicator variables, such that V_i is 1 or 0 according to whether or not the modality of the ith fiber along the path can be identified. Thus, for any fiber i, the modality U_i can be identified if and only if the indicator $V_i = 1$. Consequently, we can define a sequence of signals $X_1, \ldots, X_n$ such that X_i is equal to the modality label of the ith fiber when it

[1] Ove Frank
Department of Statiscs, Stockholm University,
S-10691 Stockholm, Sweden

is identifiable and equal to 0 otherwise; the signal X_i is simply an ordinary product of U_i and V_i :

$$X_i = U_i V_i \quad \text{for } i = 1, \ldots, n.$$

If the path hits a fiber at a random location, the probability p that the modality of the fiber can be identified should be approximately equal to the proportion of the internodal distance along a fiber that is covered by the path.

Now the special recording device cannot observe the signals $X_1, \ldots, X_m$ one by one. Observations are obtained at certain sites along the path. Each site records m consecutive signals, but the sites might overlap; that is, the same signal might belong to several consecutive sites. If we assume that the sites are regularly distributed along the path with a displacement of h fibers and an overlap of $m - h$ fibers between neighboring sites, then a path of $n = m + rh$ fibers provides $r + 1$ observations. There is the initial observation consisting of the signals $X_1, \ldots, X_m$ at site 0, the next observation consisting of the signals $X_{1+h}, \ldots, X_{m+h}$ at site 1, the next observation consisting of the signals $X_{1+2h}, \ldots, X_{m+2h}$ at site 2, et cetera. The final observation consists of the signals $X_{1+rh}, \ldots, X_{m+rh}$ at site r. At each site, there are m signals that can be observed but the order between the signals cannot be observed. Thus, the observation at the ith site can be given as the frequency distribution $Y_{i0}, \ldots, Y_{ik}$ of the signals $X_{1+ih}, \ldots, X_{m+ih}$. Here Y_{ij} is the number of signals equal to j at site i. The total is $Y_{i0} + \ldots + Y_{ik} = m$ and the subtotal $Y_{i1} + \ldots + Y_{ik} = m - Y_{i0}$ is equal to number of identified modalities at site i.

A general stochastic model for the frequencies $Y_{ij} (i = 1, \ldots, r$ and $j = 1, \ldots, k)$ is obtained by considering $U_1, \ldots, U_n$ and $V_1, \ldots, V_n$ as stochastic processes with specified properties and specified dependence. The properties of the U_i-process should capture the intrafascicular modality distribution, and the properties of the V_i-process should capture the distribution of the nodes of Ranvier. For instance, a pure random distribution of the fibers of different modalities can be specified by letting the U_i be independent identically distributed random variables with a common probability distribution given by the occurrence proportions $P_1, \ldots, P_k$. A pure random distribution of the nodes of Ranvier can be specified by letting the V_i be independent identically distributed Bernoulli (p) variables where p is that proportion of length of a fiber for which it is identifiable by the recording device. A pure random distribution of both modalities and nodes, with independence between the two, can be considered as a basic hypothesis of randomness. Interesting deviations from this randomness include a modality segregation hypothesis and a node clustering hypothesis. Modality segregation implies that fibers of the same modality tend to be neighbors more often than under pure randomness. Node clustering implies that the nodes of neighboring fibers of the same modality tend to be close together more often than under pure randomness.

To specify these hypotheses, we have to make some technical assumptions. Assumptions of varying degree of sophistication can certainly be made to achieve modality segregation and node clustering. The ones given here are very simple but sufficient for illustrate purposes. We illustrate later how the segregation and clustering tendencies can affect the data. We also suggest statistics that might be useful for discriminating between the hypotheses.

Modality segregation seems possible to describe by letting the U_i-process be a Markov chain. Here we choose with a probability $1 - \theta$ to repeat the previous modality and with a probability $1 - \theta$ to select the new modality independently of the previous one. The probability of a transition from modality i to j is given by

$$P_{ij} = \theta \delta_{ij} + (1 - \theta) P_j$$

where δ_{ij} is 1 or 0 according to whether or not $i = j$.

Node clustering seems possible to describe by specifying the V_i-process as a Markov chain conditional on the U_i-process. Here we choose the simple rule that $V_{i+1} = V_i$ if $U_{i+1} = U_i$; that is, all neighboring fibers of the same modality have a common identification indicator. This means that the nodes of Ranvier are at the same location along neighboring fibers of the same modality. The values of V_i corresponding to distinct runs of values in the U_i-process are assumed to be independent identically distributed Bernoulli (p)-variables.

With the specifications of randomness, modality segregation and node clustering given above we need to know the implied effects on the observed frequencies Y_{ij} of fibers of modality j at site i for $j = 1, \ldots, k$ and $i = 0, \ldots, r$. We also need to find statistics that are useful for discriminating between randomness and various hypotheses involving modality segregation and node clustering.

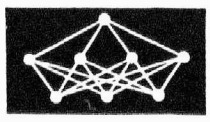
3. Test Statistics

Under randomness, the waiting time for a site with at least one identified modality is a random variable W with probability distribution

$$\begin{aligned} P(W = 0) &= 1 - q^m \\ P(W = i) &= q^{m+(i-1)h}(1 - q^h) \quad \text{for } i = 1, \ldots, r \\ P(W > r) &= q^{m+rh} \end{aligned}$$

Here the waiting time is given as the number of screened recording sites before the first identified modality is encountered. If no more than r replacements from the initial site are allowed, then the waiting can be in vain with a probability q^{m+rh}, and the waiting time distribution when the waiting is not in vain is given by the conditional distribution

$$\begin{aligned} P(W = 0 | W \leq r) &= (1 - q^m)/(1 - q^{m+rh}) \\ P(W = 1 | W \leq r) &= q^{m+(i-1)h}(1 - q^h)/(1 - q^{m+rh}) \\ &\quad \text{for } i = 1, \ldots, r. \end{aligned}$$



Let α be the probability that at least one modality can be identified. Under randomness it follows that

$$\alpha = 1 - q^{m+rh}.$$

For $j = 1, \ldots, k$, let $Z_j = Y_{Wj}$ be the number of identified fibers of modality j at the first site having any identified modality. Let further

$$Z \; Z_1 + \ldots + Z_k = m - Y_{W0}$$

be the total number of identified modalities at the first site having any identified modalities. The probability distributions of Z and Z_j conditional on that at least one modality is identified are denoted by

$$\begin{aligned} P(Z = z|W \leq r) &= \beta(z) \\ P(Z_j = z|W \leq r) &= \beta_j(z) \end{aligned}$$

for $z = 0, 1, \ldots, m$. Probabilistic arguments show that under randomness

$$\beta(z) = \frac{b(z : m, p)}{1 - q^{m+rh}} + \frac{q^m(1 - q^{rh})b(z; h, p)}{(1 - q^h)(1 - q^{m+rh})}$$

for $z = 1, \ldots, m$

where

$$b(k, n, p) = \binom{n}{k} p^k (1 - p)^{n-k}$$

is a binomial distribution. For $\beta_j(z)$, $z = 1, \ldots, m$ there is a similar expression with p replaced by pP_j in the two binomials. Furthermore

$$\beta_j(0) = 1 - \beta_j(1) - \ldots - \beta_j(m) > 0.$$

Modality segregation and node clustering can be expected to cause the distributions of Z and Z_j to be shifted upwards.
For $j = 1, \ldots, k$, let

$$T_j = Y_{0j} + \ldots + Y_{rj}$$

be the total number of identified fibers of modality j at all visited sites possibly counting the same fiber more than once), and let

$$S_j = Y_{0j}Y_{1j} + Y_{1j}Y_{2j} + \ldots + Y_{r-1,j}Y_{rj}$$

be the sum of pairwise products of the modality counts at neighboring sites. Put

$$T = T_1 + \ldots + T_k$$

and

$$S = S_1 + \ldots + S_k.$$

These statistics can be expected to give some quidance for discriminating between randomness, modality segregation and node clustering. The probability distribution of T under randomness can be given as the distribution of a linear combination of independent binomials:

$$T = \sum_{k=1}^{s+1} k \; bin(n_k; p)$$

where $\sum n_k = n = m + rh$ and $m = sh + t, 0 \leq t < h$. This leads to

$$ET = m(r + 1)p$$

and

$$VarT = \sum_{k=1}^{s+1} k^2 n_k pq \leq m(r + 1)(s + 1)pq$$

so that a value of T larger than

$$m(r + 1)p + 2\sqrt{m(r + 1)(s + 1)pq}$$

would be evidence against randomness. Similar arguments can be applied to T_j.

We illustrate the distributions of Z_j, Z, T_j, T, S_j and S numerically by quoting some results from computer experiments according to the following four hypotheses:

H_{00}	Randomness
H_{01}	Modality segregation
H_{10}	Node clustering
H_{11}	Node clustering and modality segregation.

Lead by the application described in [3] we fix the parameters at the following values:

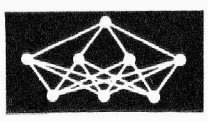
$k = 4$	$m = 8$	$r = 7$	$h = 3$
$p = 0.06$	$q = 0.94$		
$P_1 = 0.40$	$P_2 = 0.25$	$P_3 = 0.20$	$P_4 = 0.15$

Computer experiments provided the results given in Tables 1-3.

According to the computer experiments it seems as if the upper tails of the conditional probability distributions of the statistics are useful for discriminating between the hypotheses H_{00}, H_{01}, H_{10} and H_{11}. The probability of no identified modality is quite different according to which one of these hypotheses is true. It seems useful to consider the probability distributions of S_j, T_j, S and T conditional on $T > 0$. Tests could be based on the number of experiments having at least one identified modality and on the number of experiments having Z_j, Z, T_j, T, S_j or S above a chosen threshold. An illustration is given in the next section.

4. An Illustration

Tables 4-6 present various results from the experiments reported in [3]. Table 4 refers to the modality counts at the first site with at least one identified fiber, Table 5 to the sums of modality counts at all the sites, and Table 6



Tab.1.: *Probability distributions (%) of the number of identified modalities at the first site of activity conditional on at least one identified fiber.*

Hypothesis H_{00} (randomness)

Number of indentified fibes	Modality 1	2	3	4	All modalities
0	57	72	78	83	0
1	41	27	22	16	87
2	2	1	0	0	12
≥ 3	0	0	0	0	1

Hypothesis H_{10} (node clutering)

Number of indentified fibes	Modality 1	2	3	4	All modalities
0	65	71	76	80	0
1	24	23	20	17	72
2	8	5	3	2	21
≥ 3	3	1	0	0	7

Hypothesis H_{01} (modality segragation)

Number of indentified fibes	Modality 1	2	3	4	All modalities
0	58	73	80	84	0
1	40	25	20	16	89
2	2	1	1	1	10
≥ 3	0	0	0	0	1

Hypothesis H_{11} (node clutering and modality segregation)

Number of indentified fibes	Modality 1	2	3	4	All modalities
0	65	73	78	81	0
1	16	14	12	11	49
2	10	8	6	5	28
≥ 3	9	5	4	3	23

Tab.2.: *Probability distributions (%) of the total number of identified modalities conditional on at least one identified fiber.*

Hypothesis H_{00} (randomness)

Number of indentified fibes	Modality 1	2	3	4	All modalities
0	40	57	64	72	0
1	8	7	6	6	7
2	16	14	12	9	15
≥ 3	35	22	18	13	77

Hypothesis H_{10} (node clutering)

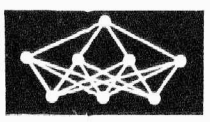
Number of indentified fibes	Modality 1	2	3	4	All modalities
0	52	60	66	72	0
1	6	6	5	5	9
2	10	11	10	8	15
≥ 3	32	23	19	15	76

Hypothesis H_{01} (modality segragation)

Number of indentified fibes	Modality 1	2	3	4	All modalities
0	42	60	67	74	0
1	8	6	5	4	7
2	17	13	11	9	17
≥ 3	34	21	17	12	76

Hypothesis H_{11} (node clutering and modality segregation)

Number of indentified fibes	Modality 1	2	3	4	All modalities
0	59	68	72	77	0
1	3	3	3	2	7
2	5	6	5	5	13
≥ 3	32	23	20	16	79



Tab.3.: *Probability distributions (%) of the total number of identified pairs of egual modalities conditional on at least one identified fiber.*

Hypothesis H_{00} (randomness)

Number of indentified fibes	Modality 1	2	3	4	All modalities
0	49	64	70	77	9
1	17	14	12	10	18
2	21	16	14	11	26
≥3	12	6	4	2	48

Hypothesis H_{10} (node clutering)

Number of indentified fibes	Modality 1	2	3	4	All modalities
0	60	67	72	77	12
1	9	10	9	8	15
2	11	12	11	10	21
≥3	19	11	7	5	52

Hypothesis H_{01} (modality segragation)

Number of indentified fibes	Modality 1	2	3	4	All modalities
0	50	66	73	78	8
1	18	13	11	9	19
2	20	15	13	10	25
≥3	13	6	4	3	48

Hypothesis H_{11} (node clutering and modality segregation)

Number of indentified fibes	Modality 1	2	3	4	All modalities
0	65	73	77	81	13
1	4	4	4	3	10
2	6	5	5	5	13
≥3	25	17	15	11	63

Tab.4.: *Distribution (%) of 30 experiments with at least one identified fiber according to number of identified fibers of each and every modality at the first site with at least one identified fiber.*

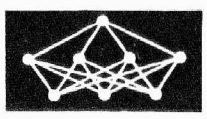
Number of indentified fibes	Modality 1	2	3	4	All modalities
0	57	50	90	77	0
1	27	33	3	20	57
2	7	17	7	3	23
≥3	10	0	0	0	20

Tab.5.: *Distributions (%) of 30 experimens with at least one identified fiber according to toal number of identified fibers of each and every modality at all the visited sites.*

Number of indentified fibes	Modality 1	2	3	4	All modalities
0	30	33	83	70	0
1	30	37	3	23	17
2	17	23	13	7	23
≥3	23	7	0	0	60

Tab.6.: *Distributions (%) of 30 experiments with at least one idetified fiber according to total number of identified pairs of fibers of each and every modality.*

Number of indentified fibes	Modality 1	2	3	4	All modalities
0	77	80	97	97	43
1	7	13	3	3	10
2	3	3	0	0	20
≥3	13	3	0	0	27



to the sums of pairwise products of modality counts at all neighboring sites. These experimental distributions should be compared to the corresponding probability distributions obtained according to the hypotheses H_{00}, H_{01}, H_{10} and H_{11} in order to discriminate between the hypotheses.

By comparing Table 1 and TAble 4 it is fairly evident that hypothesis H_{11} yields a better model than does H_{00}, H_{01} and H_{10}. A formal significance test based on the number of experiments giving large values of Z, say $Z \geq 2$, would under H_{00} assing a probability of the order 4.10^{-6} to an experimental outcome at least as extreme as the observed one, while the same probability under H_{01}, H_{10} and H_{11} is of order 4.10^{-6}, 3.10^{-2} and 8.10^{-1} respectively. This means that data are strongly in favor of H_{11}.

The same conclusion can be drawn by considering formal tests based on S and T. The power seems to be higher for tests based on Z.

Acknowledgements The numerical results for the various models presented here were obtained by using computer programs developed by Mr. Martin Karlberg. The experimental data were provided by Dr. Rolf Hallin.

References

[1] Dykes, R.W, Terzis, J.K.: Spinal nerve distribution in the upper limb: The organization of the dermatome and afferent myotome. Phil Trans R Soc London, Vol.**B293**, 1981, 509-554.

[2] Frank, O.: Statistical models of intraneural topography. In Proceedings of the International Symposium on Neural Networks and Neural Computing, Neuronet'90, Czechoslovak Academy of Sciences, Prague, 1990, 87-89.

[3] Hallin, R., Ekedahl, R., Frank, O.: Segregation by modality of myelinated and unmyelinated fibers in human sensory nerve fascicles. Muscle and Nerve, Vol. **14**, 1991, 157-165.

[4] Hallin, R., Wiesenfeld, Z.: A standardized electrode for percutaneous recording of A and C fiber units in conscious man. Acta Physiol Scand, Vol. **113**, 1981, 561-563.

[5] Rethelui, M., Szentagothai, J.: Distribution and connections of afferent fibers in the spinal cord. In A. Iggo (ed.): Handbook of Sensory Physiology, Vol.**II**, Springer - Verlag, Berlin, 1973, 207-252.

[6] Roberts, W.J., Elardo, S.M.: Clustering of primary afferent fibers in peripheral nerve fascicles by sensory modality. Brain Research , Vol.**370**, 1986, 149-152.

[7] Schady, W.J.L., Torebjörk, H.E., Ochoa, J.L.: Peripheral projections of nerve fibers in the human median nerve. Brain Research, Vol.**277**, 1983, 249-261.

Book Alert

The following books can be interesting for the readers of our Journal

Advances in Neural Information Processing Systems, 3. Edited by R.A.Lippmann, J.E.Moody and D.S.Touretzky. San Mateo, Calif. 1991. ISBN 1-55860-184-8. US $ 49.95.

AI and Expert Systems: C Language. R.I.Levine, D.E.Drang and B.Edelson. New York, McGraw-Hill 1990. 289 pp. ISBN 0-07-037500-3. US $ 24.95.

AI and Expert Systems: Turbo Pascal. R.I.Levine, D.E.Drang and B.Edelson. New York, McGraw-Hill 1990. 292 pp. ISBN 0-07-037500-3. US $ 29.95.

Analysis and Control of Industrial Processes. Edited by D.Popovic. Wiesbaden, Vieweg 1991. 281 pp. ISBN 3-528-06340-8. DM 98.00.

Artificial Intelligence at MIT. Vols.1 and 2. P.H. Winston and S.A.Shellard. Cambridge, Mass. MIT Press 1990. Vol.1, 656pp., Vol.2, 634 pp. ISBN 0-262-23150-6. US $ 70 for both volumes.

Neural Networks and Speech Processing. Morgan,D.P., Scofield, Ch.L. and Cooper,L.N. Norwell, MA, Kluwer Academic Publishers 1991. 416 pp. ISBN 0-7923-9144-6. US $ 69.95.

Parallel Processing in Neural Systems and Computers. Edited by R.Eckmiller, G.Hartmann and G.Hauske. Amsterdam, Elsevier Science Publishers 1991. 626 pp. ISBN 0-444-88390-8. US $ 68.50.

The 119 contributions in this book cover a range of topics, including parallel computing, parallel processing in biological neural systems, simulators for artificial neural networks, neural networks for visual and auditory pattern recognition as well as for motor control, AI, and examples of optical and molecular computing.

The Perception of Multiple Objects. A Connectionist Approach. Michael C.Mozer. Neural Network Modeling and Connectionism Series. Cambridge, MIT Press 1991. 176 pp. US $ 22.50.

Progress in Neural Networks. Edited by Omid M. Omidvar. Nerwood, NJ Ablex Publishing Corporation 1990. 240 pp.ISBN 0-89391-610-2. US $ 45.00(Inst.) $ 32.50(Pers.)

Symbols versus Neurons? Edited by Joachim Stender and Tom Addis. Amsterdam, IOS Press 1990. 252 pp. ISBN 90-5199-039-1. US $ 65.00.

Visual Perception: The Neurophysiological Foundations. Edited by Lothar Spillmann and John S.Werner. London, Academic Press 1990. 531 pp.

ANALYSIS OF DECISION-MAKING PROCESSES IN DISCRETE SYSTEMS BY FUZZY PETRI NETS

V. Olej, J.Chmúrny[1]

Abstract: This paper presents possibility of access to uncertainty in analysis of decision-making processes in discrete systems by fuzzy Petri nets (FPN).

Key words: *Petri nets, parallel algorithm, OCCAM, transputer*

Received: November 11, 1991
Revised and accepted: November 11, 1991

1. Introduction

The analysis of decision-making processes in discrete systems with uncertainty can be realized by neural nets which are represented by FPN. Fuzzy Petri nets described [1,2,3] represent special kind of neural nets in which transitions are neurons and places are conditions of the realization [4] of neurons. To every condition and neuron FPN va lue of the membership function [5] is assigned. Parallel algorithm of the realization FPN (written in parallel programming language OCCAM 2) represents a possibility of technical realization of an analyzer on the basis of transputers.

2. Fuzzy Petri Nets

For the analysis of decision-making processes in discrete systems with uncertainty FPN can be applied. The properties of FPN are described in [1,2,3,6,7,8].

Fuzzy Petri net is a bichromatic oriented graph defined as 7-tuple

$FPN =< C, N, I, Q, M\,o, T, D >$

where:

- C is a finite set of vertices called conditions represented by circles, $C = \{c1, c2, \ldots, ck\}$;
- N is a finite set of vertices called fuzzy vector of the neurons (FVN) represented by lines, $N = \{n1, n2, \ldots, nl\}$, $C \bigcap N = 0$;
- $I : CxN \rightarrow A, (A = \{0, 1, 2, \ldots\})$ is the forward incidence function. It states for each neuron $n_i \in N$ a set of input conditions $I(n)$, $I(n) = \{c \in C / I(c, n) \neq 0\}$;
- $Q : NxC \rightarrow A$, is a backward incidence function. It states for each neuron $n_i \in N$ a set of output conditions $Q(n)$, $Q(n) = \{c \in C / Q(n, c) \neq 0\}$;
- Mo is a initial marking asssigned number of tokens to conditions;
- $T = \{t_1, t_2, \ldots t_k\}$ is the finite set of values of membership function called fuzzy truth vector (FTV) assigned to each condition $c_i \in C$;
- $D = \{d_1, d_2, \ldots, d_l\}$ is the finite set of values of membership function called fuzzy decision-making vector (FDMV) assigned to each FVN $n_i \in N$.

Fuzzy set $B \subset X$ is any representation of the set X into interval $< 0, 1 >$ in the form

$$u_B : X \rightarrow < 0, 1 > .$$

Function u_B that is identified with the set B is called a membership function [5] and the number $u_B(x)$ where $x \in X$ is called an element of fuzzy set B. A membership function can be obtained by expert estimation or modelling.

The discrete system with uncertainty is described by a set of k conditions $C = \{c_1, c_2, \ldots, c_k\}$ on the basis of which its rules can be constructed. Conditions may be conjuncted and disjuncted in a natural way to allow the firing of the neurons. The conjunctions are represent by operation MIN and disjunction are represent by operation MAX [5,9].

The truth of the conditions (tokens in conditions $c_i \in C$ in FPN) $C = \{c_1, c_2, \ldots, c_k\}$ determines FTV $T = \{t_1, t_2, \ldots, t_k\}$ by which is initialized FVN $N = \{n_1, n_2, \ldots, n_l\}$ assigned to neurons in FPN. Fuzzy decision-making vector $D = \{d_1, d_2, \ldots, d_l\}$ can be assigned to FVN. Fuzzy decision-making vector represents threshold of feasibility of FVN.

3. Sequential Realization of Decision-Making Processes in Discrete Systems with Uncertainty

Sequential algorithm of analysis of the decision-making processes in discrete systems with uncertainty by FPN can

[1]Doc.Ing.Vladimír Olej,CSc., Prof.Ing.Ján Chmúrny,DrSc.
Departmen of Computer Science, Technical University,
031 19 Liptovský Mikuláš, Czechoslovakia

be described by the following way where input quantities of algorithm are:

- FVN N initialized by value one, FTV T, FDMV D;
- forward incidence function I which is represented by forward incidence matrix $I(k,l)$;
- backward incidence function Q which is represented by backward incidence matrix $Q(l,k)$.

Forward and backward incidence function describes the structure of discrete systems. Sequential algorithm can be realized as follows:

```
begin
repeat
   cycle for processing of rows and columns
   of forward incidence matrix I(k, l)
   for j = 1 to l do
     begin
       NN(j) = N(j)
        for i = 1 to k do
realization MIN operation between FVN and FTV
        if I(i, j) = 1 then begin
                  if NN(j) ≥ T(i)
                  then NN(j) = T(i)
                end
     end
      cycle for comparing of j − th components of new FVN
      NN and j − th components of FDMV D
      for j = 1 to l do
        if NN(j) < D(j) then NN(j) = 0
      cycle for assigning of j − th components of new FVN
      NN to j − th componets of FVN N
      for j = 1 to l do
        N(j) = NN(j)
      cycle for processing of rows and columns
      of backward incidence matrix Q(l, k)
       for j = 1 to k do
         begin
           NT(j) = T(j)
           for i = 1 to l do
realization MAX operation between FTV and FVN
           if Q(i, j) = 1 then begin
                  if NT(j) ≤ N(i)
                   then NT(j) = N(i)
                end
         end
         cycle for assigning of j − th components of new FTV
         NT to j − th components of FTV T
         for j = 1 to k do
           T(j) = NT(j)
         cycle for up-to-dating of i-th components of FVN N
         for i = 1 to l do
           if N(i) = 0 then N(i) = 1
until (T(k) ≠ 0 AND N(l) ≠= 0)
end
```

4. Example of Application Fuzzy Petri Net in Analysis of Decision-Making Processes with Uncertainty

The structure of a simple discrete system can be described by:

- set of conditions C_i, $i = 1, 2, \ldots, 6$;
- rules which result from conditions c_1 AND $c_2 \Rightarrow c_4; c_4 \Rightarrow c_6; c_5 \Rightarrow c_3; c_5 \Rightarrow c_1$;

On the basis of conditions and rules can be fuzzy Petri net described (see Fig.1).

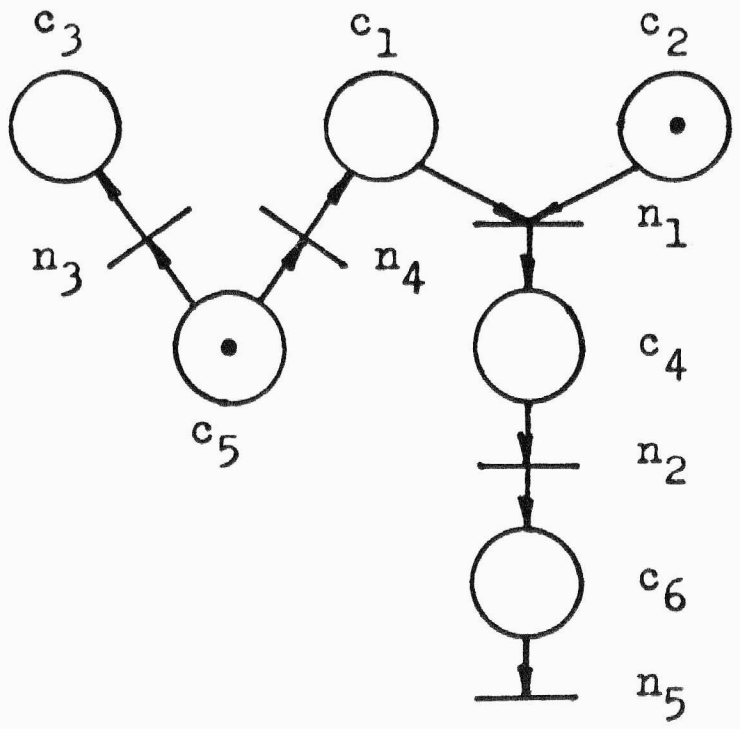


Fig.1: *Fuzzy Petri net of simple decision-making of systems*

- forward incidence matrix

$$I(k,l) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix};$$

- backward incidence matrix

$$Q(l,k) = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix};$$

- FTV $T_i, i = 1, 2, \ldots, 6$;
- FDMV $D_i, i = 1, 2, \ldots, 6$;

Then analysis of decision-making processes of discrete systems by sequential algorithm is given in the table 1.

5. Parallel Realization od Decision-Making Processes in Discrete Systems with Uncertainty

Development of the present computing systems is characterized by increasing of their speed on the basis of parallel realization of the computing process. The necessity of parallelization follows from the need to realize the response of a computing system in real time. This requirement is often so demanding that cannot be met even by computing means designed on the basis of the most

advanced technology providing the computing process is realized sequentially.

Step 1

NN T NN MIN T = NN

$$\begin{bmatrix}1\\1\\1\\1\\1\end{bmatrix}\begin{bmatrix}.0\\.8\\.0\\.0\\.5\\.0\end{bmatrix}\quad\begin{bmatrix}.0\\.0\\.5\\.5\\.0\end{bmatrix}$$

NT N NT MAX N = NT

$$\begin{bmatrix}.0\\.8\\.0\\.0\\.5\\.0\end{bmatrix}\begin{bmatrix}.0\\.0\\.5\\.5\\.0\end{bmatrix}\quad\begin{bmatrix}.5\\.8\\.5\\.0\\.5\\.0\end{bmatrix}$$

Step 2

NN T NN MIN T = NN

$$\begin{bmatrix}1\\1\\.5\\.5\\1\end{bmatrix}\begin{bmatrix}.5\\.8\\.5\\.0\\.5\\.0\end{bmatrix}\quad\begin{bmatrix}.5\\.0\\.5\\.5\\.0\end{bmatrix}$$

NT N NT MAX N = NT

$$\begin{bmatrix}.5\\.8\\.5\\.0\\.5\\.0\end{bmatrix}\begin{bmatrix}.5\\.0\\.5\\.5\\.0\end{bmatrix}\quad\begin{bmatrix}.5\\.8\\.5\\.5\\.5\\.0\end{bmatrix}$$

Step 3

NN T NN MIN T = NN

$$\begin{bmatrix}.5\\1\\.5\\.5\\1\end{bmatrix}\begin{bmatrix}.5\\.8\\.5\\.5\\.5\\.0\end{bmatrix}\quad\begin{bmatrix}.5\\.5\\.5\\.5\\.0\end{bmatrix}$$

NT N NT MAX N = NT

$$\begin{bmatrix}.5\\.8\\.5\\.5\\.5\\.0\end{bmatrix}\begin{bmatrix}.5\\.5\\.5\\.5\\.0\end{bmatrix}\quad\begin{bmatrix}.5\\.8\\.5\\.5\\.5\\.5\end{bmatrix}$$

Step 4

NN T NN MIN T = NN

$$\begin{bmatrix}.5\\.5\\.5\\.5\\1\end{bmatrix}\begin{bmatrix}.5\\.8\\.5\\.5\\.5\\.5\end{bmatrix}\quad\begin{bmatrix}.5\\.5\\.5\\.5\\.5\end{bmatrix}$$

Tab.1: *Analysis of a discrete system based on fuzzy Petri net*

The present development of the production technology of elements with high scale integration (mainly transputers) enabes to design parallel computing systems at relatively available prices in various architectures so that they may suit particular applications to their best.

Parallel algorithm of analysis of decision-making processes in discrete systems on the basis of FPN can be realized in parallel programming language OCCAM 2. The structure of this programming language corresponds to the hardware architecture of transputers. It enables programming of parallel computing trasputer systems. Parallel algorithm of analysis of decision-making processes in discrete systems by means of FPN can be described as follows:

```
SEQ
  WHILE (T(k) ≠ 0 AND N(l) ≠ 0)
    SEQ
      PAR j = 1 FOR l
        NN(j) = N(j)
        PAR i = 1 FOR k
          IF
            I(i,j) = 1
              IF
                NN(j) ≥ T(i)
                  NN(j) = T(i)
                TRUE
                  SKIP
          TRUE
            SKIP
      PAR j = 1 FOR l
        IF
          NN(j) < D(j)
            NN(j) = 0
          TRUE
            SKIP
      PAR j = 1 FOR l
        N(j) = NN(j)
      PAR j = 1 FOR k
        NT(j) = T(j)
        PAR i = 1 FOR l
          IF
            Q(i,j) = 1
              IF
                NT(j) ≥ N(i)
                  NT(j) = N(i)
                TRUE
                  SKIP
            TRUE
              SKIP
        PAR j = 1 FOR k
          T(j) = NT(j)
        PAR i = 1 FOR l
          IF
            N(i) = 0
              N(i) = 1
            TRUE
              SKIP
```

6. Conclusion

Fuzzy Petri nets which are special kind of neural nets enable to analyze of decision-making processes in discrete systems with uncertainty. Possibilities of this formal mean are based on the knowledge of fuzzy matrices [9]. By means of FPN can be modelling concurrency and conflict in discrete systems. Fuzzy Petri nets can be applied in fuzzy rule-based reasoning and deducing of control systems for real time decision-making with application to controls and pattern recognition.

Technical realization of the design of FPN analyzer architecture on the basis of the transputers can be considered as highly perspective in the given field due to their high performance, simplicity of multitransputer systems, simple possibility of expansion and possibility of unified approach to the design of technical and programming means.

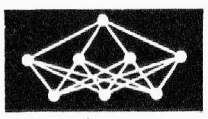
References

[1] Looney, C.G.: Fuzzy Petri Nets for Rule-Based Decision- Making. IEEE Transactions on Systems, Man, and Cybernetics, Vol.SMC-**18**, No1, 1988, 173-183.

[2] Chmúrny J., Olej V.: Application of Fuzzy Petri Nets in Analysis of Decision-Making Processes of Discrete Sys tems. Electrical Engineering Journal, Vol.**41**, No.5, 1990, 388-392.

[3] Olej V., Chmúrny J.: The Analysis of Decision-Making Processes in Discrete Systems. Proceedings of Conf. Elektrotechnika 90, EF SVT, Bratislava, 1990, 67-70.

[4] Peterson J.L.: Petri Net Theory and Modelling of Systems. Prentice Hall, Inc. 1981.



[5] Novák V.:Fuzzy Sets and their Applications. SNTL, Praha, 1990.

[6] Chmúrny J., Olej V., Mokriš I.: Analysis of Discrete Systems by Fuzzy Petri Nets. Electrical Engineering Journal, Vol.**39**, No.3, 1988, 226-230.

[7] Olej V., Chmúrny J., Mokriš I.: Analysis of Parallel Discrete Systems by Discrete Stochastic and Fuzzy Petri Nets. Computers and Artificial Intelligence, Vol.**10**, No.3, 1991, 221-237.

[8] Olej V., Strelec J., Chmúrny J.: Analysis of Deterministic, Stochastic and Fuzzy Discrete Systems by Generalized Petri Net. International Fuzzy Engineering Symposium, Proceedings IFES " 91, Yokohama, Japan, 13.-15. Nov., 1991 (in Press).

[9] Chmúrny J., Olej V.: Fuzzy Matrices and Possibility of their Application. Electronic Horizont, Signals-Systems-Informatics, Vol.**51**, No.9, 1990, 355-382.

Literature Survey

Smith H.L.: Convergent and Oscillatory Activation Dynamics for Cascades of Neural Nets With Nearest Neighbor Competitive or Cooperative Interactions

Neural Networks Vol.4, 1991 No.1 pp. 41-46

Key words: convergent dynamics; oscillatory dynamics; cascade.

Abstract: It is shown that a cascade of the neural net N0 into the net N1 has oscillatory (convergent) dynamics if the nets N0 and N1 have oscillatory (convergent) dynamics and the net N1 has a special structure.

Soucek B.: Neural and Intelligent Systems Integration. Chichester

John Wiley & Sons LTD, 1991, 560 p.ISBN 0471 53676 8

Key words: intelligent systems.

Abstract: This book discusses well-defined intelligent modules that can be integrated into an intelligent system. Among these modules are; intelligent algorithms and programs, neural networks and computer elements, fuzzy data comparators and expert systems. They are the result of the 5th generation research effort mounted in Japan as well as ongoing research.

Stark J.: A Neural Network to Compute the Hutchinson Metric in Fractal Image Processing

IEEE Trans. on Neural Networks Vol.2, 1991 No.1 pp.156-158

Key words: Fractal Image Processing.

Abstract: The Hutchinson metric is a natural measure of the discrepancy between two images for use in fractal image processing. This paper describes a neural network which can quickly calculate this metric.

Sudharsanan S.I., Sundareshan M.K.: Equilibrium Characterization of Dynamical Neural Networks and a Systematic Synthesis Procedure for Associative Memories

IEEE Transactions on Neural Networks Vol.2, 1991 No.5 pp.509-521

Key words: neural networks; dynamical; associative memories.

Abstract: Several new results cencerning the characterization the equilibrium conditions of a continuous-time dynamical neural network model and a systematic procedure for synthesizing associative memory networks with nonsymmetrical interconnection matrices are presented.

Takefuji Y., Lee K.Ch.: Artificial Neural Networks for Four-Coloring Map Problems and K-Colorability Problems

IEEE Trans. on Circ. and Systems Vol.3, 1991 No.3 pp.326-333

Key words: neural networks - colorability problems.

Abstract: The computational energy is presented for solving a fourcoloring map problem. The map-coloring problem is defined that one wants to color the regions of a map in such a way that no two adjacent regions are of the same color.This paper presents a parallel algorithm based on the McCuloch-Pitts binary neuron model and the Hopfield neural network.

Troudet T.P., Walters S.M.: Neural Network Architecture for Crossbar Switch Control

IEEE Transactions on Circuits and Systems Vol.38, 1991 No.1 pp.42-56

Abstract: A Hopfield neural network architecture is proposed for the real-time control of a crossbar switch for switching packets at maximum troughput. The network performance and processing time are derived from a numerical simulation of the transitions of the neural network.

Tsao T.R., Shyu H.J., Libert J.M., Chen V.C.: A Lie Group Approach to a Neural System for Three-Dimensional Interpretation of Visual Motion

IEEE Trans. on Neural Networks Vol.2, 1991 No.1 pp.149-155

Key words: Neural Network - Visual Motion.

Abstract: This paper presents a novel approach to neural network computation of three-dimensional rigid motion from noisy two-dimensional image flow. It is shown that the process of 3-D interpretation of image flow can be viewed as a linear signal transform.

AN OPTIMAL STOPPING CRITERION FOR BACKPROPAGATION LEARNING

Martin A. Kraaijveld, Robert P.W. Duin[1]

Abstract: A common problem of iterative learning procedures like the backpropagation algorithm is the lack of insight in the learning phase. Therefore, it is difficult to decide at what moment the learning phase should be terminated. For many ad hoc criteria that are used in practice, it can be shown that they suffer from serious defects, especially for recognition problems in which the distributions of the classes have some overlap. By the application of a technique from the statistical pattern recognition literature, the editing algorithm [3], a learning set can be transformed to a data set in which the overlap of the classes is effectively removed. This results in an optimal stopping criterion for iterative learning procedures, and a number of experiments indicate a moderate improvement in learning speed for the backpropagation algorithm. Moreover, because it can be proven that an edited data set yields a performance which is close to Bayes-optimal for the nearest-neighbor classifier, it is very likely that a classifier which is based on an iterative learning procedure and which classifies *all* samples in the edited learning set correctly, is also close to Bayes-optimal.

Keywords: *iterative learning procedure, statistical pattern recognition, neural networks, backpropagation learning, editing algorithm.*

Received: May, 5, 1991
Revised and accepted: December 9, 1991

1. Introduction

In the last few years an enormous interest in learning procedures for neural networks has emerged. An important development in this area was the invention of the backpropagation algorithm by Rumelhart [9]. Practical applications (e.g.[2], [4], [11], etc.) as well as benchmarking studies (e.g.[8] have shown that classifiers based on the backpropagation algorithm have a good performance. Multi-layer networks therefore seem to offer a reasonable alternative for various parametric and non-parametric techniques of the statistical pattern recognition literature. Especially the feature that complex shaped decision functions can be represented by a limited number of units and weights, resulting in a very compact representation of a classifier, facilitates fast operation for real time applications [7].

However, from a practical and theoretical viewpoint, there are still some drawbacks:

1. Training a multi-layer network with the backpropagation algorithm is slow. In comparison with other methods (e.g. the nearest-neighbor classifier), multi-layer networks require large amounts of CPU time to obtain a reasonable classifier. This is due to a number of specific properties of the algorithm:

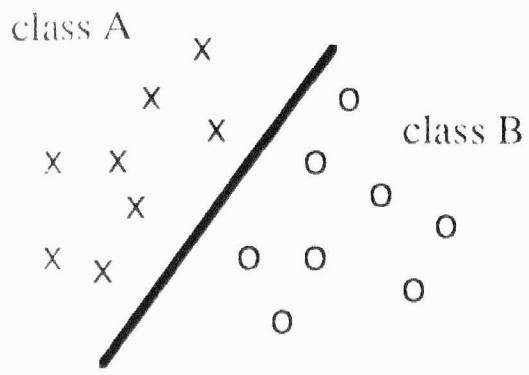
   - The algorithm is iterative in nature.
   - The learning rate η should be small in order to improve convergence. However, a small η results in a long learning phase.

2. The algorithm is essentially based on a gradient descent in an error space which contains local minima. Therefore, the learning phase can get stuck into a local minimum, resulting in a suboptimal performance of the classifier. It is important here to distinguish the cases of recognition problems with overlapping and non-overlapping classes (see Fig.1):

   When there is no overlap between the classes in the learning set, it is desirable that the network, for the learning set as well as for any test set, finally correctly classifies 100% of the samples. Getting stuck in a local minimum implies that the performance is less than 100% , and is therefore suboptimal.

   When the classes in the learning set do overlap, however the situation is completely different. When the learning set has a Bayes error (i.e. an intrinsic overlap) of ε % , no classifier will ever achieve a performance higher than (100 - ε)% . Therefore, when a large multi-layer network has achieved a performance of 100% on the learning set, it must have made many small decision boundaries around samples in the overlapping part of the learning set and will certainly have a suboptimal performance on a test set. As the backpropagation algorithm tries to minimize the

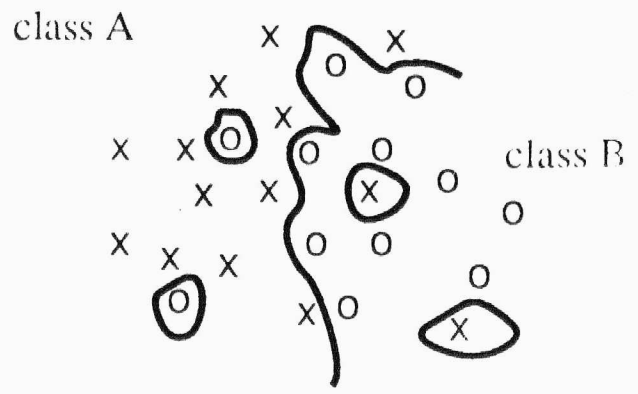
[1] Martin A.Kraaijveld, Robert P.W.Duin
Pattern Recognition Group, Faculty of Applied Physics, Delft University of Technology, P.O.Box 5046, 2600 GA Delft, The Netherlands



non-overlapping classes:

Mean squared mapping error minimal

Classifier optimal



overlapping classes:

Mean squared mapping error minimal

Classifier **SUB**optimal

Fig.1: ***For a learning set with non-overlappings classes, a global minimum in the mean squared mapping error corresponds to an optimal classifier. For a learning set with overlapping classes, the global minimum in the mean squared mapping error corresponds to a suboptimal classifier***

mean squared mapping error, it does in fact tend to a situation with decision boundaries around outliers, because this corresponds to a lower point in error space.

From this we conclude that the optimal classifier for a learning set with overlapping classes is surprisingly found in a *local* minimum of the error space. For overlapping classes it is therefore highly desirable that the learning phase gets stuck in a local minimum! (N.B. notice that for a learning set which is not sufficiently representative for the underlying distributions of the classes, the optimal classifier might not even correspond to a local minimum in error space).

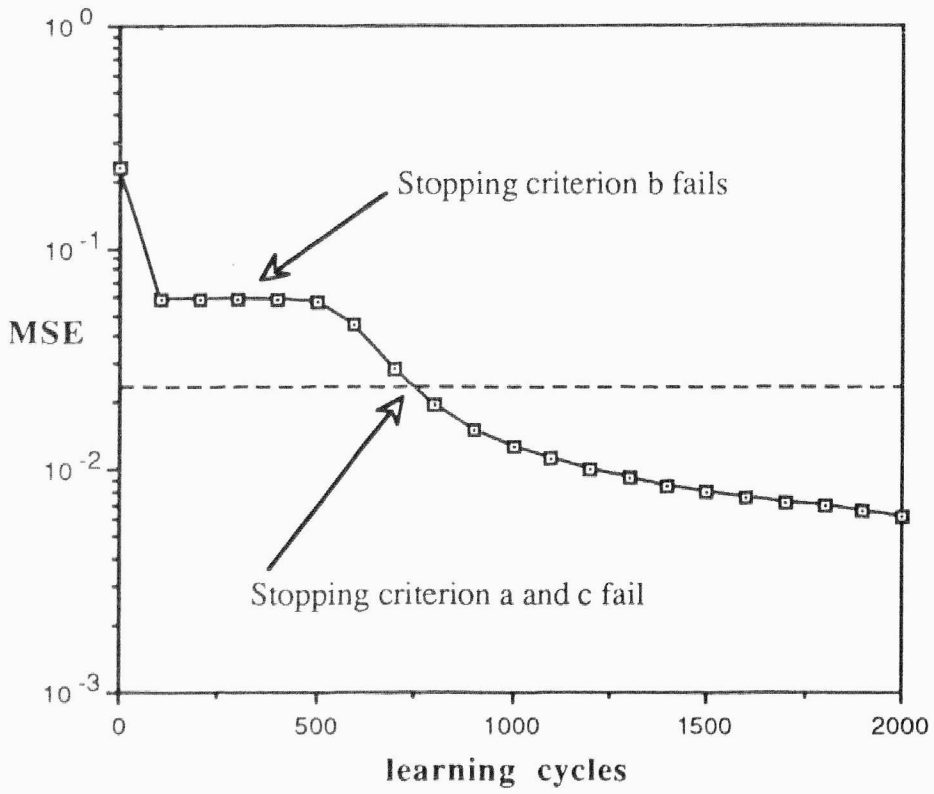


Fig.2: *Ad hoc stopping criteria fail to stop the learning phase at the correct moment.*

3. A third problem is that there is no insight in the learning process. Unless there is some explicit knowledge of the underlying distributions of the classes in the decision problem, it is very difficult to decide in which state the learning phase remains. Is the current state the global error minimum? Is the current state close to a ravine in error space? Is the network currently learning an outlier of the classes in the learning set?, etc.

Furthermore, it is not clear if learning should continue until all samples are classified correctly (in the case of separable classes), or until another stopping criterion is reached (in the case of intrinsic overlap of the classes). Due to the lack of a stopping criterion that takes into account the state of the network in error space, ad hoc criteria are applied. Examples of these are (see Fig.2):

(a) continue the learning phase until the the mean squared error is "sufficiently" low.

(b) continue the learning phase until the time-derivative of the mean squared error is sufficiently low.

(c) continue the learning phase until a sufficient percentage of the learning set is classified correctly.

Especially the first problem that is sketched above, bears a close parallel to problems that are related to the nearest neighbor classifier. For the nearest neighbor classifier, outliers of a different class will generate regions in which samples will be assigned to the wrong class. This deteriorates the performance of the resulting classifier. For a multi-layer network, the learning procedure will try to

separate these small regions around the outliers, because this decreases the mapping error.

A solution for the problem of the outliers with the nearest-neighbor rule is described by [3], and is called "editing". The editing algorithm effectively removes the overlap in case of overlapping classes, i.e. it effectively separates the classes. This results in a learning set from which all outliers are removed. The main contribution of this paper is to investigate the behavior of the backpropagation algorithm on edited data sets.

2. Learning of Edited Data Sets

The editing technique [3] is an algorithm that is used to improve the performance of the nearest neighbor classifier as well as to reduce the number of samples in the learning set. The algorithm removes the intrinsic overlap (if any) between the classes in the learning set in such a way that it hardly affects the position of the optimal decision boundary. However, erroneously classified samples are removed (see Fig.3). In the context of nearest neighbor pattern classification this largely improves the performance of the resulting nearest neighbor classifier. In fact it can be proven that, provided that the learning set is sufficiently large, the performance of the 1-nearest neighbor rule on an edited data set is very close to Bayes-optimal.

For the experiments described in this paper, a special version of the editing algorithm was used, which is called the *multi-edit algorithm*. It consists of 5 steps:

1. **Diffusion:** Make a random partition of the learning set S into N subsets $S_1, \ldots, S_N$, $N > 2$.
2. **Classification:** Classify the samples in S_i using the 1-nearest neighbor rule with $S_{(i+1)Mod\ N}$ as a training set, $i = 1, \ldots, N$.
3. **Editing:** Discard all samples that were misclassified at step 2.
4. **Diffusion:** Pool all the remaining data to constitute a new set S.
5. **Termination:** If the last I iterations produced no editing then exit with the final set S, else go to step 1.

Using an edited data set instead of non-edited data set for the training of a multi-layer network is expected to have the following consequences:

- By editing the data set, a very deep minimum is in error space is introduced. Because the learning set is now completely separable (although not necessarily linearly separable), this error minimum is also a global minimum and has an error value of zero. The stopping criterion for the learning phase has now become trivial: *as long as not all samples are classified correctly the learning phase should be continued.*
- Because all outliers in the region of the other class are removed from the learning set, there is no danger that the performance of the classifier will deteriorate by learning outliers, even when the network has a (very) large number of hidden units.
- Although it is somewhat difficult to measure, it appears that learning an edited data set is faster than learning a non-edited data set. The difficulties with the measurements are caused by the fact that the learning time of an non-edited data set is based on a certain (ad hoc) stopping criterion. Any figure for the learning time can be produced, however, when the parameters of this stopping criterion are changed. Learning is therefore certainly faster in the sense that we can make an earlier decision to stop the learning phase. Notice that this improvement in speed is derived by pre-processing of the data instead of an adaptation of the learning procedure.

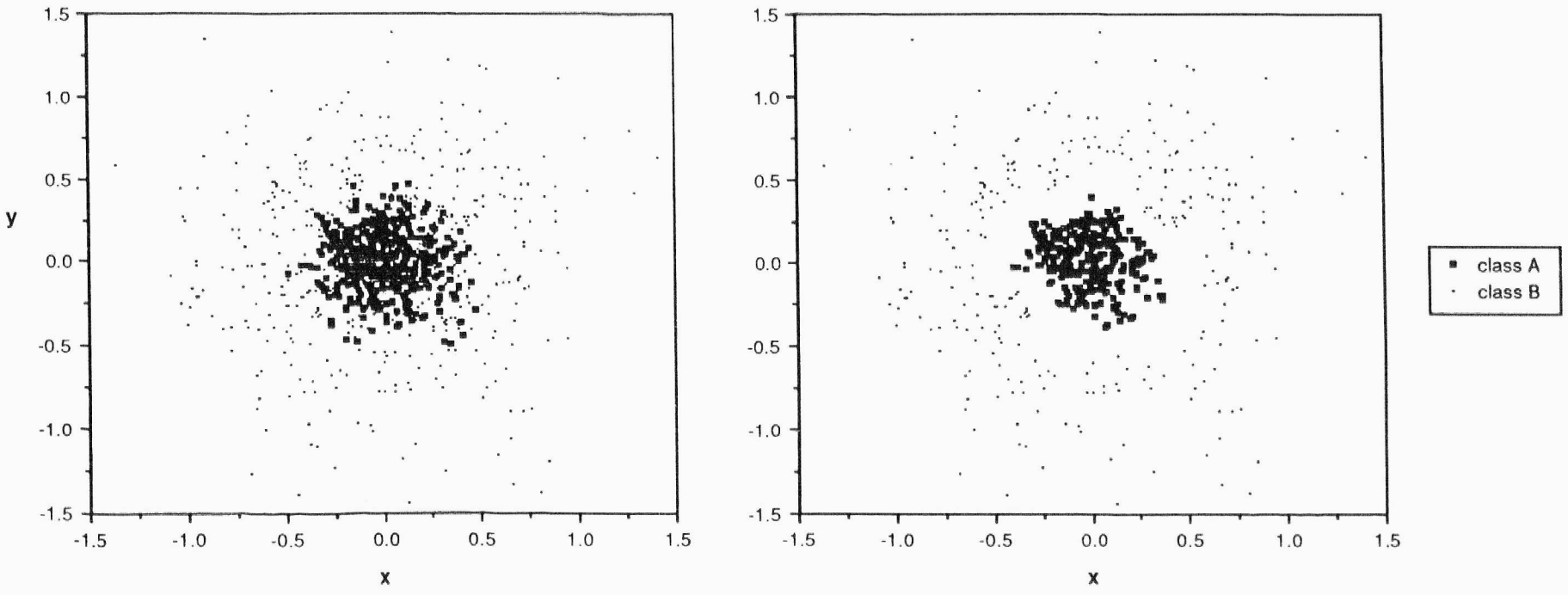


Fig.3: *Data set 1 (left), and the data set after editing*

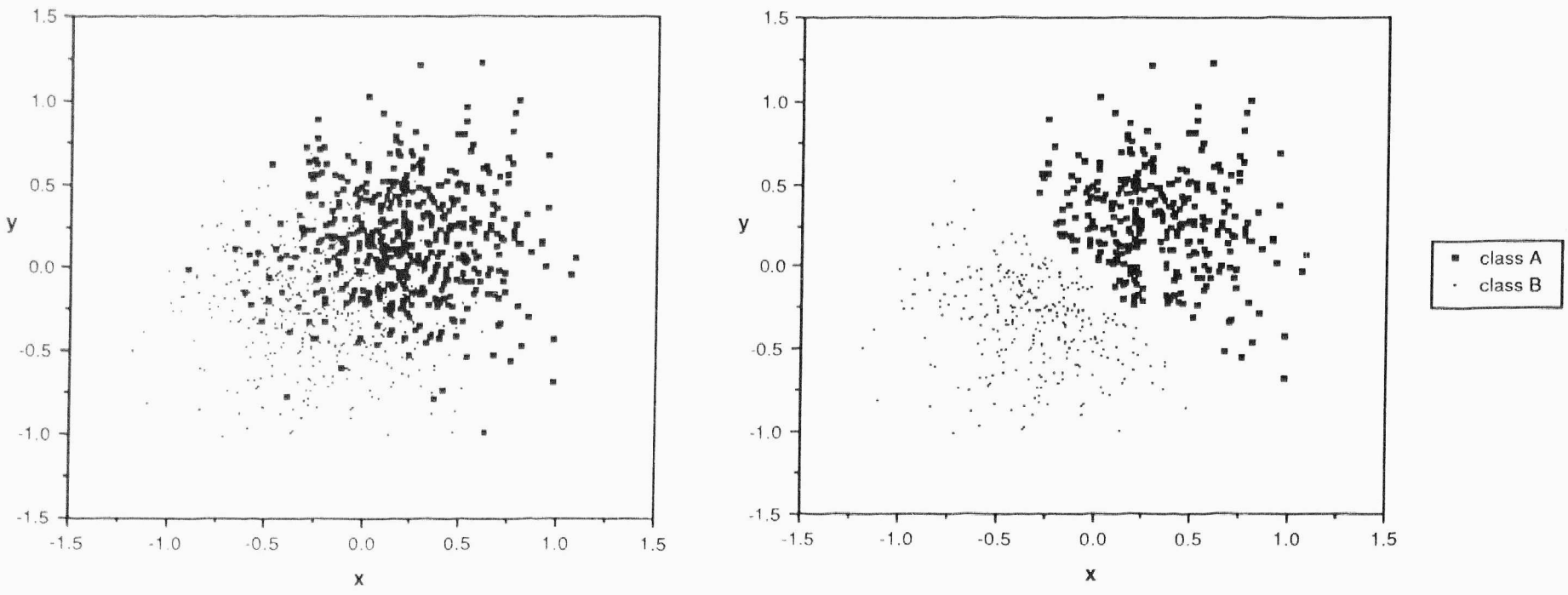


Fig.4: *Data set 2 (left), and the data set after editing*

- When there is no overlap in the classes, the editing algorithm will not affect the learning set. Therefore, the editing algorithm can be applied in any case, overlapping or non-overlapping, to assure that the resulting set is separable.

- The editing algorithm can be considered to perform a transformation of the underlying removing the overlapping part of the densities. It is important to realize that it is this transformation that enables the use of the new stopping criterion. By transforming the original densities to the edited densities we can solve the problem of the stopping criterion in the original domain.

- The editing algorithm, applied to network training as well as nearest-neighbor classification, can be considered as a pre-processing step. However, there are differences in the goals to be reached in both cases. For the nearest neighbor classifier the goal is the decrease of the size of the learning set and the increase of the performance of the nearest neighbor classifier. For network training, however, editing a data set aims at influencing the error space of the learning process such that a good criterion to finish the learning phase can be found.

3. Experiments

To investigate the behavior of the backpropagation algorithm on edited data sets, some experiments were performed on two heavily overlapping circular symmetric Gaussian distributions. Data set 1 (see Fig.3) has both means on the origin (0,0) and standard deviations 0,4702 and 0,1736. Data set 2 (see Fig.4) consists of two classes with means (0,2, 0.2) and (-0.2, -0.2) and equal standard deviations 0.333. The Bayes error for both learning sets is 20% . Each set consists of 1000 samples.

For both learning sets, the original set and its edited version were used as a training set for a number of multilayer feedforward networks. The networks consist of 2 input units, 10,20,50 or 100 hidden units and 1 output unit. The parameters of the backpropagation algorithm were: learning rate η and momentum term α. A sample was considered to be correct when the difference between the actual output and the desired output was smaller than 0,5. The learning phase for the edited data sets was terminated when all samples were classified correctly.

Table 1 shown the perfomance of a trained network on a large test set. Figure 5 shows a plot of the average performance on the learning set as a function of the learning time.

4. Discussion

From the experiments that were performed, it appears that the backpropagation algorithm with an edited data set is a factor three faster when compared to a non-edited data set, *provided that we had a good criterion to stop the learning phase for a non-edited data set.* In practical situations therefore, the optimal stopping criterion results in a considerable improvement in speed. What we also gained with editing is the relative certainty that a reasonable classifier is found, due to the introduction of a large near-Bayes global minimum in the error space. Though, we are still not able to guarantee that the learning process finishes in the desired minimum; in fact for large networks one can still imagine *global* minima which are very undesirable, see [1].

Tab.1: *The average performance of 50 trained networks on a test set of 25,000 samples.*

average performance	data set 1 (figure 3)		date set 2 (figure 4)	
	non-edited (percent)	edited (percent)	non-edited (percent)	edited (percent)
10 hidden units	80.7	79.1	80.1	80.0
20 hidden units	80.3	79.0	80.1	80.1
50 hidden units	80.2	79.0	80.1	80.0
100 hidden units	80.1	78.7	80.0	80.0
nearest neighbor classifer	74.7	78.9	74.0	79.8

Furthermore, it is remarkable that the performance of a multi-layer network is not seriously deteriorated by the overlap in the learning set, as is the case with the nearest neighbor classifier. It is clear that the algorithm does not form small decision boundaries around outliers, as can be expected from a procedure that is based on the minimization of the mapping error. The search for a global minimum of the backpropagation algorithm is apparently constrained by a mechanism that prevents that too small groups of outliers form a separate decision boundary. If this is also the case for the Boltzmann machine [5] is doubtful, since the search for an error minimum of the Boltzmann machine is based on an optimization method which was designed to escape from local minima in the error space [6]. It is therefore to be expected that the Boltzmann machine more seriously suffers from defects as sketched in figure 1. However, it will therefore benefit even more from the editing procedure. Essentially, training a network with an edited data set provides a stopping criterion for *any* iterative learning procedure which optimizes the mean squared mapping error, whether this is backpropagation, some faster variant of backpropagation, the Boltzmann machine or any other iterative procedure. Furthermore, it is interesting to realize that our method

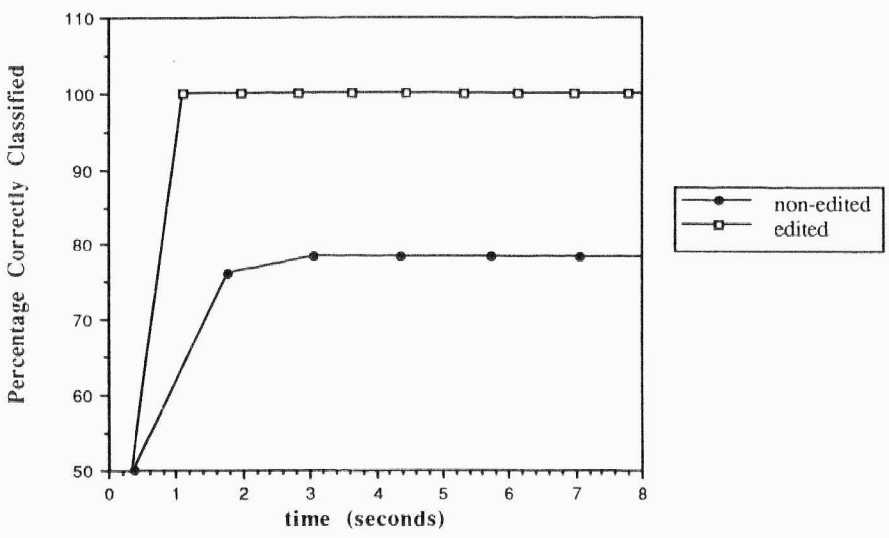


Fig.5: *The average performance of a network with 10 hidden units on the leraning set during the learning phase. Notice that leraning the edited data set stabilises at t=1, whereas leraning the non-edited data set stabilizes at t=9. Every point in this plot was a averaged over 50 instances of a network. The simulations where performed on a SUN 4/280 with a simulator written in C.*

changes the behavior of the learning rule, not by changing the learning rule itself, but by changing the data set. It might therefore be interesting to search for other operations on the data set which also influence the learning rule.

Some additional remarks on the editing algorithm: In the first place, the editing step requires a certain amount of computational effort. This is for the data sets of figure 3 and figure 4 typically 20 seconds of CPU time on a SUN 4/280. Although this implies that the total requirements with respect to CPU time for the presented method are higher, the advantage is the certainty of the new stopping criterion. Without editing, one could train the network for minutes without ever being sure that the performance could not be improved.

In the second place, the editing algorithm requires that the data is sufficiently representative for the underlying distributions of the classes; i.e. when the data set is divided over N subsets, all these subsets must on their own be representative for the distributions.

Returning to the nearest-neighbor classifier, the step which usually follows the editing of a data set is called condensing [3]. The condensing algorithm removes all samples that are not close to the discriminating function, thereby heavily reducing the size of the learning set. For the nearest-neighbor classifier this results in a dramatic improvement in classification speed; instead of computing the distance to all samples in the learning set, only the distance to a few (discriminating) samples has to be determined.

It is interesting to notice a parallel of the condensing algorithm and the multi-layer perceptron: both represent the discriminating surface only (instead of the complete learning set). This leads to a large reduction of data resulting in an improvement of classification speed.

One can wonder if it is possible to speed up the learning phase of the backpropagation algorithm by training the network with a condensed learning set. The small subset containing the discriminating samples would concentrate the learning effort on the discriminating surface only. A little thought learns, however, that its is not always the case; the Euclidean metric which is used by the nearest-neighbor classifier to determine the distance to a sample, is not used by a multi-layer network. This is caused by the

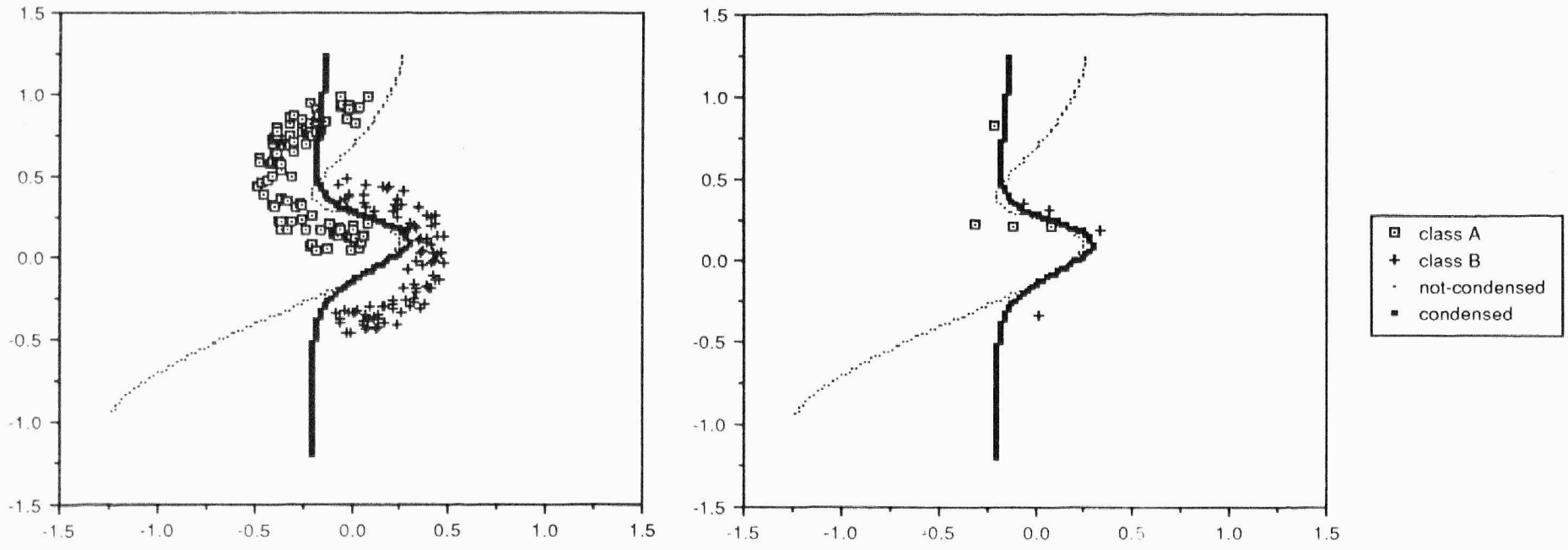


Fig.6: *A data set (left), and the data set after condensing. In both data sets the decision function of a network that was trained with the original data set, and the decision function of a network that was trained with the condensed set are drawn. Notice that the decison function of the condensed set does not correctly classify the original set*

inherent non-linearities of a multi-layer network. Whereas the backpropagation algorithm simply aims at separating classes, the nearest-neighbor classifier aims at separating the classes by drawing a line exactly in the middle between two samples (resulting in a Voronoi-tesselation of the features space). For the backpropagation algorithm it is not essential that the decision surface is exactly between two samples. As figure 6 shows, the differences between these metrics are so large that learning with a condensed learning set can give rise to a suboptimal classifier.

5. Conclusions

The editing algorithm effectively introduces a near-Bayes global minimum in the error space of the backpropagation algorithm. This has the advantages that it is very likely that the learning phase will end in this minimum and that the learning speed is moderately improved. Furthermore this editing technique results in an optimal stopping criterion. Since this stopping criterion is based on an operation on the data, instead of a change of the learning rule, it can be applied for any iterative learning procedure which optimizes the mean squared mapping error.

Experiments indicate that the search for the global minimum in the backpropagation algorithm is essentially constrained by a mechanism that prevents the formation of decision boundaries around small groups of outliers. This results in a very good performance for overlapping classes.

Acknowledgements This work was sponsored by the Dutch Government as a part of the SPIN/FLAIR-DIAC project, and by the Foundation of Computer Science in the Netherlands (SION) with financial support from the Dutch Organization for Scientific Research (NWO).

References

[1] Baum, E.B.: On the Capabilities of Multilayer Perceptrons. Journal of Complexity, Vol.**4**, 193-215, 1988.

[2] Bounds, D.G., and Lloyd, P.J.: A Multi-Layer Percpetron Network for the Diagnosis of Low-Back Pain. Proceedings of the SGAICO conference 1988, University of Zürich, OCt., 1988.

[3] Devijver, P.A., and Kittler, J.: Pattern Recognition, A Statistical Approach. Prentice Hall, 1982.

[4] Gorman, R.P., and Sejnowski, T.J.: Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets. Neural Networks, Vol.**1**, No. 1, 1988.

[5] Hinton, G.E., and Sejnowski, T.J.: Learning and Relearning Boltzmann machines. Chapter 7 in: Rumelhart, D.E., and McClelland, J.L.: Parallel Distributed Proceesing: Explorations in the miciro structure of cognition. Vol.**1**, MIT Press, 1986.

[6] Kirkpatrick, S., Gelatt, C.D.Jr., and Vecchi, M.P.: Optimization by Simulated Annealing. Science, Vol.**220**, 671-680, 1983.

[7] Kraaijveld, M.S.: On the Application of Connectionist Models for Pattern Recognition, Robotics and Computer Vision: a Technical Report. Delft University Press, Delft, The Netherlands, 1989.

[8] Kohonen, T., Barna, G., and Chrisley, R.: Statistical Pattern Recognition with Neural Networks: Benchmarking Studies. Proceesdins of the Neuro '88 conference, Paris, June 1988.

[9] Rumelhart, D.E., Hinton, G.E., and Williams, R.J.: Learning Internal Representations by Error Propagation. Chapter 8 in: Rumelhart, D.E., and McClelland, J.L.: Parallel Distributed Processing: Explorations in the micro structure of cognition. Vol.**1**, MIT Press, 1986.

[10] Sejnowski, T.J., and Rosenberg, C.R.: Parallel Networks that learn to Pronounce English Text. Complex Systems, Vol.**1**, 145-168, 1987.

[11] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K. and Lang, K.: Phoneme Recognition Using Time-Delay neural Networks. IEEE Trans. on ASSP, Vol.**37**, March 1989.

MOLECULAR-LEVEL NEUROELECTRONICS

A.V. Samsonovich[1]

Abstract: New ideas of modern neural network models molecular-level (ML) implementation based on Coulomb correlated electron tunneling in molecular media are proposed; estimations of reachable parameters have been obtained. The neuronet approach proves to be more relevant for molecular-level computers than the von Neumann one.

Key words: *network models; Coulomb correlated electron tunneling; molecular media; molecular-level computers;*

Received: June 15, 1991
Revised and accepted: December 9, 1991

1. New Promise in ML Information Processing

While the earliest ideas of molecular electronics have aroused hopes of fantastic possibilities for ML computing [1-5] (spatial density of logical or memory elements up to $10^{18}cm^{-3}$ in conjunction with an operation frequency of the order of $10^{12}s^{-1}$),it has been recognized that there is no promising way to make use of these ideas in the framework of the von Neumann approach or its counterpart - multi-processor architectures - in the future [6].

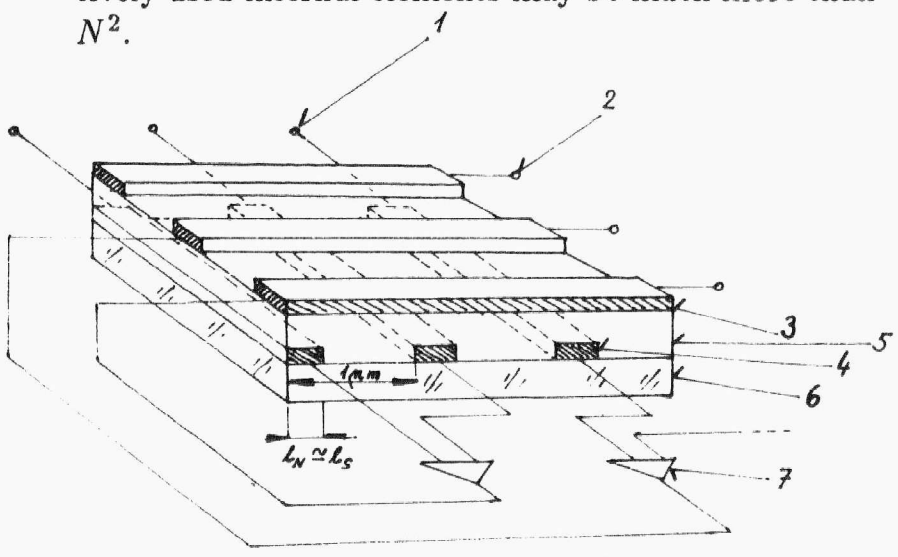
On the other hand, the most promising trends in informatics, knowledge engineering and artificia! intelligence concern quite a different possibility, namely, the neuronet (NN) approach [7-9]. Its hardware requirements are adequate to ML element base features: very large degree of integration and high parallelism are claimed; randomly generated architecture of interconnections and low reliability of elements are to be allowed.

No doubt, quite new hardware is necessary for the successful development of the NN approach; but is ML necessary? Usually the integration, or the number of internal elements in the device, is restricted by the number of interface channels, which should meet traditional microelectronics requirements. For example, in Hopfield type NN models [8,9] we need only $N_s = 0(N^2)$ internal elements - synapses where N is the number of interface channels coinciding with the number of neurons, each of which has its own interface channel in the case of parallel input/output. In the planar scheme of the crossed-strips type the synapse dimension l_s may be equal to the cross-section of interface channel (a strip) l_N, which is usually on a micron scale. Thus, ML gives no gain in this case. When is ML NN implementation necessary? There are a few possible scenarios:

1. to use a 3-dimensional scheme: in the case of surface interface, the requirement $N_s \simeq N^2$ leads to $l_s \sim l_N^{3/4} L^{-1/3} \ll l_N$ (disregarding the volume occupied by axons and dendrites), where L is the device dimension, l_s^3 the volume per synapse, l_N^2 the area of the surface per neuron; but due to the spatial packing problem the volume per synapse l_s^3 appears to be much larger than the volume occupied by the synapse itself, see Section 2.3;

2. to realize a sequential interface at ML, taking the proper model - see Section 2.4;

3. to use new NN models in which the number of effectively used internal elements may be much more than N^2.



The cross-strips type scheme of possible submicrom-level molecular-electronic implementation of the Hopfield model [4]: 1...input terminal, 2...output terminal, 3...conducting metallic strip, 4...conducting strip, 5...learning molecular layer, 6...substrate, 7...neuron (CMOS based).

The latter possibility is more attractive because current promising NN models are exactly of this type. There are two interesting directions in this field:

1. high-order nets [10], in which $N_s \gg N^2$; and Boolean nets [14], requiring a large memory matrix per neuron. In these models the number of neurons (nodes of the net) remains equal to the number of interface channels.

[1] A.V.Samsonovich
Institute for Microelectronics Technology, USSR Academy of Sciences, Chernogolovka, Moscow District, 142432 USSR

2. multi-layer (perceptron-type) cascade or cyclic nets [11] and multi-nuclear models [12]. Here internal layers or nuclei are local subnets consisting of hidden neurons having no direct interface. Hierarchical models [13] arouse a great interest in the framework of the second direction.

Realization of the latter models may possibly be achieved on a basis of a neurolike medium, i.e. a macroscopically uniform structure consisting of neuron-like elements of different sorts, each having a large number of random interconnections of restricted length. Such a model appears to be more useful for ML realization than Hopfield's nets with global interconnections. It can involve interface via its surface shell only. More generally, it may be any suitable non-linear multi-stable active medium, in which information can be transmitted long distances and memorized. In such a medium, the hierarchical or multi-nuclear structure can be obtained by a self-organization process. Possible ML implementations of random cellular automata, which are similar to this (except for that they are usually considered to be non-learning), have been discussed by Conrad [15]. In our case, the interface problem could be solved if the path of excitation in the medium (produced by the input pattern) would depend on its informational content. A concept of the management of spatio-temporal trajectories of activity peaks in the neural network based on a triangle lattice of analogous neurons has been introduced by Eckmiller [16]. Unfortunately, he considers a model involving global interconnections.

Another interesting way to effectively use a large-volume neurolike medium can be based on a parallel interface with its domains: each local subnet obtains the same input, but due to subnet specialization, only one of them will answer. The response (provided it appears) should be different for different subnets. Subnet specialization can be achieved by means of competitive conditions which should be established during the learning procedure. For this case, the ART models introduced by Grossberg [17] and probably their modifications appear to be very useful.

As for the neurolike medium architecture, besides the contiguous subnets, it may be assembled from a large number of separate small-size neuronet devices with standard microelectronic interfaces between them - thus we can obtain "a neurotransputer". Both possibilities can be interesting from the practical point of view.

Thus we have two tasks for NN hardware designing which can be solved at ML: 1) a small-size, inexpensive perceptron- or Hopfield-type net comprising about $10^3 \div 10^5$ neurons, which can be used as a nucleus in a large network or as an expert part of an intelligent system; 2) a large-volume neurolike medium, possessing unbounded possibilities of development.

It should be noted that if we know how to obtain access and effectively use a large-volume neurolike medium with internal bonds of restricted length via its surface shell only, and if molecular element synthesis is mastered, then a qualitative leap can be performed in the field of informatics tools. As for other models, ML implementations can lead to a quantitative gain only; the integration degree will be restricted by the possibilities of interface or global interconnections. Thus, following the principle used by Hawking [23] in the development of his theory of topological space-time structures, "to seek the key under a lamp", we should believe in the concept of a neurolike medium.

2. Mathematical Principles of the NN Approach and Possible ML NN Architectures

2.1. General Principles

In general, the task of ML neuroelectronics is to develop an associative memory device based on an attractor neurolike system (ANS) consisting of molecular-size elements. The concept of the ANS [12] implies that relaxation of the system to an attractor can be treated as pattern recognition, while learning consists of attractors and its basins adjustment. Hence, the system should have at least two groups of dynamic variables, or two subsystems, representing respectively the short time memory (STM) and the long time memory (LTM). An input pattern should be represented in the system as a state of the STM. Its evolution, depending on the LTM state, should lead to an output signal production. During learning, the STM should adjust the LTM state to agree the STM input with its desired output given from outside. In fact, the latter point involves the adjustment of attractors itself as well as its basins. This adjustment should be governed by local learning rules, i.e. changes in each part of the LTM should depend on the state of parts of the STM intermediately connected with it.

2.2. Hopfield's Networks and Beyond: A Brief Review and Statement of the ML NN Elements Designing Task

First, let us briefly review the main principles of current NN models [8-11], which will be used below.

The typical dynamic equations of a Hopfield type model in discrete time t are

$$\sigma_i^{t+1} = sgn(h_i^t - \vartheta_i), \tag{2.1}$$

$$h_i^t = \sum_j J_{ij}\gamma_j^t, \tag{2.2}$$

where J_{ij} is synaptic efficacies, h_i the membrane (or postsynaptic) potential; ϑ_i the neuron activation threshold and σ_i^t the Boolean variable taking values ± 1 (which may also be gradual; in this case we should use a sigmoid function f instead of sgn), attributed to the i-th formal neuron; $\gamma_j^t \equiv \sigma_j^t$. The μ-th input pattern is represented as a set of $\{\sigma_i^o\} = \{\xi_i^\mu\}$.

The local learning rules for associative memory have a typical form (for example, the Widrow-Hoff form)

$$\Delta J_{ij}^{\mu} \alpha (\sigma_i^{\nu} - h_i^{\mu}) \gamma_j^{\mu}, \qquad (2.3)$$

where μ is the pattern number (which may coincide with time during the learning process), ΔJ_{ij}^{μ} the increase in J_{ij} after $\mu - th$ learning cycle, $\nu \equiv \mu$, and $\{\sigma_i^{\mu}\}$ patterns to be stored.

In the case of a high-order net aimed at temporal sequences storing and retrieving [10] one introduces randomly chosen nonlinear and time-delayed synapses, i.e., substitutes into (2.2),(2.3) the vector γ assembled from randomly chosen few-argument Boolean functions:

$$\{\gamma_j^t\} = \{\sigma_i^t,\ \sigma_i^{t-1},\ \sigma_i^t \sigma_k^t,\ \sigma_i^t \sigma_k^{t-1}, \ldots\}, \quad j = 1 \ldots M,\ M \gg N, \qquad (2.4)$$

and substitutes $\nu = \mu + 1$ into (2.3).

In the case of the Boolean net [9] the state of each node σ_i^{t+1} is a Boolean function of c randomly chosen (once and for all) other nodes $\sigma_{j_{i,1}} \ldots \sigma_{j_{i,c}}$. In contrast to the high-order Hopfield-type model, these Boolean functions are learning themselves and we need 2^c bits of memory per node.

In layered feed-forward (or perceptron-like) models [6] one usually chooses gradual σ_i and writes instead of (2.1.),(2.2)

$$\sigma_{l,i}^{t+l} = f(h_{l,1}^t), \quad h_{l+1,i}^t = \sum_j^{N_i} J_{i,j}^l \sigma_{l,j}^t, \qquad (2.5)$$

where $l = 1 \ldots L$ is the layer number. Hence, now only neighbouring layers interact with each other. The learning rules are analogous to (2.3).

Hierarchical structure may be set by dividing layers into clusters: now each hidden (internal) neuron receives signals from neurons of one cluster of the lower layer and sends output to only one neuron of the upper layer [13]; intralayer interactions are inhibitory.

The multinuclear net [12] consists of local subnets described by dynamic equations similar to (2.1.),(2.2.) internuclear connections are usually feed-forward and may be introduced by different ways: 1) as copying of output state of one nucleus to another using it as input state; 2) as adding output of each neuron of one nucleus to the membrane potential of the corresponding neuron of another nucleus at each moment of discrete time; 3) as adding to membrane potentials $h_{l,i}$ (2.2) the term given by (2.5), where $J_{i,j}^l$ corresponds to internuclear communications. The rules, by which learning and retrieval processes as well as internuclear communications are switched on and off, can be controlled by some special neurons inside the nuclei.

The equations (2.1)-(2.5) can be simply reformulated in terms of σ_i taking values $\{0, 1\}$.

Now we can say what kinds of elements are necessary for ML NN implementations. In the case of a completely parallel processing architecture they are:

- **cell body**, or the neuron itself, which is either a threshold/bistable or a non-linear amplifying element; it receives the input signal from the dendrite and supplies the output signal to the axon, in some models this element can also have an LTM (there may be different sorts of these elements in the same network);
- **synapse**, which is a learning bond, transmitting the signal with a definite weight (or probability) between two neurons, either intermediately or with a given time delay; in the case of a high order model it should perform some non-linear transformation on signals received from source neurons and send the weighted result to the destination neuron; its learning consists in adjustment of its output weight, controlled by the neurons connected;
- **dendrite**, or the dendritic tree, which should collect signals coming to a neuron from synapses and sum them; the function of non-linear signals mixture can be attributed as well to the dendritic tree incorporating synapses;
- **axon**, the counterpart of the dendrite, which distributes the signal coming out of the neuron to synapses attributed to other neurons.

But this point of view may be not suitable for ML NN implementations. Alternative possibilities can be based on the dynamic and/or distributed rather than structural realization of these elements. In a definite sense models which we consider below are of that type.

2.3. Scaling Laws and the Hausdorf Dimension of ML Neurolike Structures: Global vs. Local Interconnections

Let us assume that we want to realize a Hopfield type network, i.e. a network with global interconnections, as a D-dimensional scheme (D=2 or 3). In the models under consideration, the number of interconnections N_s should be of the order of N^2, or more generally, $N_s \sim N^{\chi}$ (N is the number of neurons, $\chi > 2$ for high-order networks). The LTM is provided by synapses, and $N_s \gg N$. Without loss of generality we may assume that synapses are implemented as separate localized objects or space domains. It seems to be natural to suppose that integration degree will be restricted by synapse dimensions r. The value of r may be of the order of $10nm$ in the case of ML synapses implementation and is taken as unity in our further consideration. Another significant value is axon/dendrite cross section dimension d. It will be assumed to be of the same order or less. The question is: how large may be the effectively used part of the device volume $V = L^D$, or how large may be the number of effectively used ML elements (synapses) contained in the device volume at given device dimension $L \gg 1$ (in units of r)?

To proceed in the analysis of this question we introduce some general notations and relations. We denote by $N_0 \simeq L^D$ the maximum number of synapses which may

be packed in the volume $V = L^D$. A half set of synapses connected with the same cell body via axons and dendrites together with these axons, dendrites and cell body we call "a neuron". Let us imagine that the volume L^D is divided into unitary cells by introducing any regular or random space lattice with cell dimension $a \simeq r = 1$. The Hausdorf dimension [20] H of a neuron can be defined by the relation $V_n \simeq l^H$ at $l \to \infty$, where V_n is the number of imaginary cells occupied or crossed by the neuron and l is a typical neuron maximal size which in the case of global interconnections should be of the order of L. We assume that neurons are closely packed in the device volume, but not superimposed (i.e. the whole device volume is divided into neurons). It leads to the relation $N \simeq L^{D-H}$. On the other hand, $N^\chi \simeq N_s \leq N_0 \simeq L^D$, hence $N \leq N_{max} \simeq L^{D/\chi}$. Regardless of the spatial packing problem we can obtain the optimal value of H from the condition that N reaches its maximum: $H_{opt} = (1 - 1/\chi)D$. Thus, the result is: If $H = H_{opt}$ and the spatial packing problem is solved, then the volume is effectively used, i.e., a significant part of the volume is occupied by actually used synapses.

But the spatial packing problem remains open. It involves accomplishment of the next conditions: 1) the condition of connectivity of a neuron; 2) the condition at which each neuron should touch a large part of the other neurons (or different couples of other neurons in the case of high-order models) by its synapses. These conditions lead to some restrictions on the possible value of H.

Let $\chi = 2$: this case corresponds to "linear" synapses. In the planar case ($D = 2$, $H_{opt} = 1$), geometrical solution of the packing problem is straightforward and is given by the scheme of crossed strips (see Fig.0). As for $D = 3$ case, at least one (rather involved) solution of the packing problem providing $H = H_{opt} = 3/2$ does exist. But in general some disorder in ML elements packing is unavoidable. When we put neurons into the volume randomly to a certain extent, we obtain $H > H_{opt}$ (for example, in the case of random closely packed linear chains $H = 2$ [21]). It leads to $N \simeq L^{D-H} \ll N_{max} \simeq L^{DH_{opt}}$.

How close can be H to H_{opt}? In general, the solution of this question demands computer simulations [16]. The answer depends not only on a geometry itself but also on technology of ML device assembly. Another restriction arises from the condition that a signal path along an axon (dendrite) should not be too long. For example in the case of linear chains their length in our units is $L_s \simeq L^H$. If $l \sim 1cm$, $r \sim 10nm$ and $H = 2$, we obtain $L_s \sim 10km$. It would take a long time to transmit a signal along such a chain. Thus, dendritic trees are necessary rather than linear chains. This circumstance also enlarges the minimum possible value of H.

If $H > H_{opt}$, then the effectively used volume N_s becomes negligibly small in comparison with L^D at $L \to \infty$, because $N_s \simeq L^{\chi(D-H)}$, $N_s/N_0 \simeq L^{\chi(D-H)-D} = L^{\chi(H_{opt}-H)}$. In other words, the Hausdorf dimension of the effectively used device volume $H_{ef} = \chi(D - H)$ becomes less than D. In this case the total volume per synapse is $V_s = L^{\chi(H-H_{opt})} \gg 1$. In contradiction with our previous assumption the volume L^D appears to be occupied by axons and dendrites rather than by synapses, and volume per synapse is not of ML. In this case, the integration degree is restricted by dendrite/axon cross-section size d (which in fact has been taken as unity in our considerations) rather than the synapse dimension r - the latter is not relevant until it becomes considerably large, $r \geq L^{(H-H_{opt})\cdot\chi/D}$ in the units of d.

The situation is quite different if we use a model with interconnections of restricted length. In this case, obviously, $H_{ef} = D$, and ML can be retained in the limit $L \to \infty$. A small fraction of global interconnections can be admissible if we suppose that the characteristic neuron dimension is $l \simeq L^\alpha$, $\alpha < 1$. It involves some constraint on the interconnections length distribution. In this case we obtain in a similar way that $H_{opt} = (1 - 1/\chi).D/\alpha$. Hence, we may admit the H_{opt} to be sufficiently large.

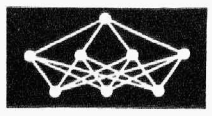
2.4. An Equivalent NN Model with Local Interconnections

The main disadvantage of the models described in Section 2.2 is the necessity of global interconnections which, due to the spatial packing problem and the interface problem, makes it unsuitable for ML implementations (see Sections 1 and 2.3). In spite of the existence of NN models based on the cellular automaton principles [13], the aim of this Section is to describe a possible physical-like model, or net, with local interconnections between nodes and time distributed signal processing, i.e., a learning cellular automaton model, which may be equivalent to models with global interconnections, but is more suitable for ML implementation.

Let us consider a multilayer system consisting of $V \simeq n.L$ internal nodes described by Boolean variables $\gamma_{l,j} \in \{0, 1\}$, $l = 1 \ldots L$, $j = 1, \ldots N_l$, $N_1 = m$, $N_2 \ldots N_L = n$, and some other variables $w_{l,j}$ providing $\{\gamma\}$ is the STM, while $\{w\}$ is the LTM. We assume that these nodes are distributed in a given volume V with mean nearest-neighbor distance a, which we take as unity. For example, in the case of molecular single-electronic implementation (see Section 3.2) we can consider these nodes as molecules having electron localization centers, and γ's the electron occupation numbers of these centers. The values w can be attributed to the chemical structure of the molecules or to its conformation. We assume that the μ-th input pattern $\{\sigma_i\} = \{\xi_i^\mu\}$ is introduced into the system bit at bit each moment of discrete time t as a state of the interface layer $\{\gamma_{1,j}^t\}$:

$$\gamma_{1,j}^t = \sigma_t, \quad j = 1, \ldots m, \quad t = 1, \ldots N. \qquad (2.6)$$

Initially, all γ's are set to zero, and the system obeys such evolution equations that the value of γ each node (except for nodes of the last layer L) is copied to another neighbouring node at each moment of discrete time (which may be given by external field impulses):



$$\gamma^{t+1}_{l',j'} = \gamma^t_{l,j}, \tag{2.7}$$

$$l' = l'(',j,t) \in \{l, l \pm 1\}, \;\; l' \neq 1; \;\; j' = j'(l,j,t) \tag{2.8}$$

with nodes (l,j), (l',j'') being neighbors. We suppose that the mapping (2.8) is randomly generated for each moment of time (it can be determined by molecular structure and external field E_t). Thus, the total set $\{\gamma\}$ is mapped onto itself with randomly chosen local displacements each moment of time. For the sake of simplicity, we assume that this mapping is non-degenerated, i.e., there exists an unambiguous inverse reproduction $l = l(l',j',t)$, $j = j(l',j',t)$ for each t. We may say that the input signals are subjected to a diffusion process in the volume, hence, the mean distance $< r^t_j >$ between the input point $(1, j_0)$ and the current signal (or electron) location point (l_t, j_t) is of the order of $t^{1/2}$ in units of a. The mean distance between two subsequent signals (electrons) originating from the same node is of the same order. We assume that input nodes are spread uniformly in the input layer with mean distance of the order of L between them, $m \simeq n/L^2$.

Hence, at the time $t = T \simeq L^2$, we obtain an approximately uniform, linearly decreasing to zero with $l \to L$, distribution of N input bits $\{\sigma\}$, represented by $\{\gamma\}$ ($\sim m\gamma$'s per σ) in the volume $V \simeq mL^3$ with the mean concentration $c \simeq N/L^3$ per node. If these signals move further according to (2.7), (2.8) during the time $T' = N$, the each node may by passed by a signal approximately $cT' \simeq N^2/L^3$ times. The last value can be close to unity, if we take $n \simeq L^{3/2}$. Then we can say that each node is related to approximately one signal, and each signal being duplicated in m copies, passes approximately $N.m$ nodes during the time $T' = N$.

Each node (l,j) produces an output signal proportional to $w_{l,j}$ each time, provided that its occupation number $\gamma_{l,j} = 1$. This signal may be: an electric current measured in the external circuit, which occurs due to an electron displacement from one layer to another; an optical characteristic of the molecular center, depending on its charge, etc. We assume the interface to be spatially non-sensitive and sequential, i.e., that the total output signal h^t is measured in each moment of discrete time t during output,

$$h^t = \sum_{l,j} w_{l,j}\gamma^t_{l,j}, \;\; t = T+1 \ldots T+T'. \tag{2.9}$$

These values $\{h^t\}$ can be stored in some additional memory and used for determination of $\{\sigma^t\}$ according to (2.1) during the next retrieval cycle. We also might use it without storing, immediately for the next moment of time, using a slightly different variant of input. Thus, we obtain a model which can be equivalent to the Hopfield-type model (2.1)-(2.3), provided that the learning rules have the form

$$< \Delta w^t_{l,j} >= \alpha(\xi_t - h^t)\gamma^t_{l,j}, \tag{2.10}$$

where ξ^t is the desired output and $< \Delta w^t_{l,j} >$ denotes mean value. Fortunately, these learning rules are local in the physical sense, because each node should know only its own occupation number and some global agent strength which is proportional to $\xi_t - h^t$ and can be supplied by some external tools.

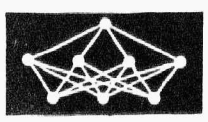
Now we can summarize the general characteristics of this model. The density of actually and independently used memory elements is of the order of molecular centers (nodes) density - the very ML density in 3 dimensions. The time of one input/output or retrieval cycle may be of the order of $t \simeq N^{4/3}$ in units of our discrete time. The number of electron tunnel jumps (as well as ML output signals) occurring in the volume at each moment of discrete time is of the order of mN. The redundancy of information representation is m.

One disadvantage of this model is that input/output and retrieval time is too long (of the order of $N^{4/3}$). This difficulty can be overcome in a similar model involving a small fraction of long-distance interconnections. By that, we may achieve the input time $T \simeq N$.

There are some reasons why this model is non-physical. First, in real system, the states of $\gamma^{t+1}_{l',j'}$, would depend not only on the given $\gamma^t_{l,j}$, but also on other γ's. Thus we obtain a high order model rather than the Hopfield one, because γ's represent some Boolean functions (2.4) depending on several input bits. Second, information should not be lost during the evolution process (2.6). But in a real system the randomly generated transition rules (2.7),(2.8) cannot be truly reversible. If the transition rules are completely randomly chosen, then each signal should encounter approximately $cT \simeq L^{1/2} \gg 1$ other signals during the input time T. If we want the information to be conserved under these collisions, and if we want the model to be of a not too high order, we should provide proper rules for signals passing through each other at a node. This leads to nodal elements being more complicated than simple electron localization centers. Finally, all processes at ML are probabilistic. Hence, we need sufficient redundancy $m \gg 1$ and should write equations in terms of occupation probability distributions rather than occupation numbers. Despite that, in principle it is possible to implement such a model on the basis of a partially ordered system assembled from ML elements having sufficiently good reliability for this purpose. But there exists another possibility, which will be discussed in Section 4.

2.5. A Neurolike Medium Model

Here we consider a model, which in some respects is close to the model of the brain cortex proposed by Eckmiller [11]. Our model consists of two subnets (or layers) of neurons, both involving interconnections of restricted length. One of them, the transport layer, is a randomly connected synchronous cellular automaton, consisting of Boolean "neurons", while the other, the master layer, is exactly the hidden layer of the perceptron-type model (except for some special dilution providing the restricted



bond length). Its input is the state of the transport layer, while its output is a master field having its own nonlinear spatio-temporal dynamics. The master field controls the activity level of the transport layer: it may arouse an activity wave (AW) propagation, or even deflect, split or eliminate it. The wave carries not only activity, but also information significant for the master layer. Hence, its evolution may strongly depend on this information.

The results of the numerical simulations are shown in Fig.1. One sees the stationary states of the transport layer (after approximately 10 iterations) at two different input patterns, which were introduced into the interface shell (top rectangles). These simulations have been made without any training of the master layer: the synaptic efficacies have been chosen randomly once and for all. But spatial configurations of the AW in the medium prove to be dependent on its informational content (compare A and B patterns). Sometimes splitting of the AW self-focusing trajectory occurs (B).

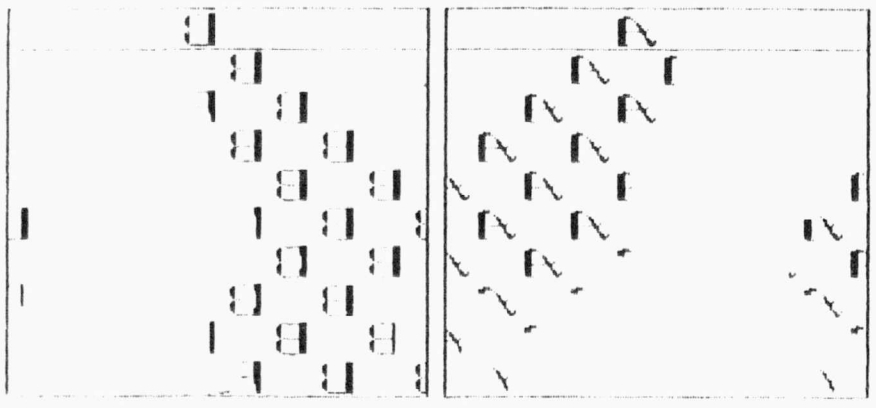


Fig.1: *Computer simulated activity wave propagation in the two-layer neurolike medium. The transport layer is regular in the top patterns and randomly interconnected in the bottom layers.*

It should be noted that even in the case of the randomly interconnected transport layer, the information of the input is not lost and reaches the bottom side of the square, where it may be restored or processed by some additional learning network.

The rate of information decay during AW propagation can be investigated using a simplified model of the transport layer. Let us consider a chain of 7-bit arrays $\{\sigma_{i,j},\ i = 1,\ldots,n,\ j = 1,\ldots,7\}$, $\sigma_{i,j} \in \{0,1\}$. Let the input pattern be represented by $\{\sigma_{1,j}\}$, and let the "dynamic equations" be the next:

$$\sigma_{i+1,j} = XOR(\sigma_{i,j}, \sigma_{i,j+1}), \qquad (2.11)$$

where XOR is exclusive "or" and we mention the periodical

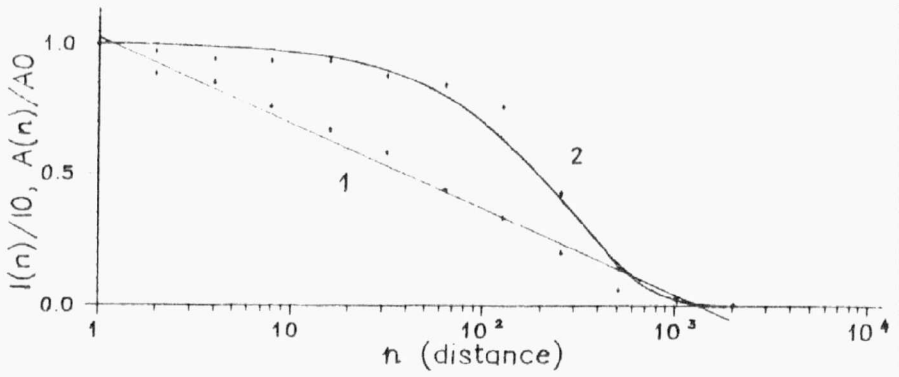


Fig.2: *Informational correlation decay on distance in the simplified model of the transport layer.* $1-I(n)$*, normalized on 7 bit; 2 - the values of mean activity* $<\sigma_{n,j}>_j$ *.*$C = 0.05$.

boundary conditions imposed on σ in j. Then we introduce a static noise: at some nodes (i^*, j^*) the rules will different:

$$\sigma_{i+1,j} = \sigma_{i,j} * \sigma_{i,j+1},\ (i,j) = (i^*, j^*). \qquad (2.12)$$

We choose these nodes randomly with a density C. Then we measure the information correlation between input $\{\sigma_{1,j}\}$ and output $\{\sigma_{n,j}\}$ defined as follows:

$$I(n) = \sum_{\{\sigma\}} P(\{\sigma_1\},\{\sigma_n\}) \log_2 \frac{P(\{\sigma_1\},\{\sigma_n\})}{P(\{\sigma_1\})P(\{\sigma_n\})}, \qquad (2.13)$$

where P is the probability of realizing $\{\sigma\}$. After all, we average this value by realizations of the static noise $\{(i^*, j^*)\}$.

The result is shown on Fig.2. We see the logarithmic decay of the information correlation, while the activity decay rate obeys the exponential law.

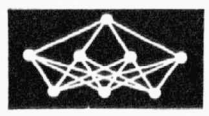
3. Possible Physical Bases of ML NN Implementations

3.1. General Statement of the Task

There are a few conceptions of ML NN implementation, each involving a definite choice of the class of NN models and mathematical tasks, basic physical phenomena and technological methods. All questions should be considered jointly. After having made the choice, we come to the task of designing the elements and architecture which has been outlined in Section 2.

Despite a large variety of physical systems exhibiting some neurolike features, there are too few physical phenomena which can be taken as basis for well-known NN models ML implementations, providing relatively good characteristics. General questions are:

- what class of physical systems can be considered as self-learning ANSs?



- what groups of degrees of freedom can be considered as the STM and the LTM?
- in what manner can a pattern be represented by the STM at ML?
- how can we provide the interface?
- how can the local learning rules be fulfilled, providing the adjustment of attractors and its basins?
- how can the learning process be switched off, providing the LTM state be conserved for a long time?
- etc.

In this Section we present some physical possibilities of answering these questions.

3.2. Molecular Single-Electronics

Here we consider the physical model of a single-electron tunnel transition in organic molecular structures. The term "single electronics", introduced by Likharev, implies Coulomb-correlated single-electron tunneling between submicron metallic particles at low temperatures [24]. There exists a counterpart of this phenomenon at ML which has slightly different physical origin: electron localization at a molecular center in organic medium is due to electron-vibron interactions rather than electron-electron or electron-phonon relaxation. We may speak about the small-radius molecular polaron as a localized unitary charge, which moves from one center to another by instantaneous tunnel jumps (the localization time τ_{loc} is usually less than $10^{-14}s$).

Now we describe a physical model (which is a little rough, but nonetheless clear), which can be taken as a basis for the development of the electron localization centers i and j of one (or two neighbouring) organic molecule(s), incorporated in organic environment. We assume the following:

$$\varepsilon_r \gg T \geq \omega \gg I_{ij}, \tag{3.1}$$

where ε_r is the molecular center reorganization energy, determined by the electron-vibron interactions; T the temperature (we make use of units $\hbar = e = k_B = 1$); ω the vibron mode frequency and I_{ij} the electron tunnel Hamiltonian matrix element. In semiclassical approximation a large distance between the centers r_{ij} we have [26]

$$I_{ij} \simeq I_0 \exp[-\alpha_{ij}(r_{ij} - a)], \tag{3.2}$$

$$\begin{aligned} \alpha_{ij} &\simeq \sqrt{2m\frac{(\varepsilon_c - \varepsilon_t^{ij})(\varepsilon_t - \varepsilon_v)}{\varepsilon_c - \varepsilon_v}} \\ &\simeq \sqrt{2m(\varepsilon_c - \varepsilon_t^{ij}}, \ \varepsilon_c - \varepsilon_t \ll \varepsilon_t - \varepsilon_v, \end{aligned} \tag{3.3}$$

here, a is the well dimension, ε_t^{ij} is the tunneling electron energy [26], m is its effective mass (the effective tunnel mass we assume to be on the order of the free electron mass); ε_c, ε_v are respectively the conduction and the valence band energies of the environment or the first free and the last occupied energy levels of the molecule. The value of I_0 in the case of donor-acceptor coupling and/or in the case of polar environment (like water) can be evaluated as [25]

$$I_0 \simeq 1/\mathcal{H}r^{ij}, \tag{3.4}$$

while in the case of homogeneous centers in non-polar environment for hydrogen-like potentials we have

$$I_0 \simeq (\alpha_{ij} r_{ij})^3/6\mathcal{H}r_{ij}, \tag{3.5}$$

where $\mathcal{H}$ is the dielectric constant.

The electron tunnel jump probability from the center i to the center j under conditions (3.1) can be written as

$$W_{ij} \simeq \nu \exp\left[-\frac{\varepsilon_{ij}^a}{T} - 2\alpha_{ij} r_{ij}\right] \tag{3.6}$$

where (omitting the numerical multiplier)

$$\nu \simeq \frac{I_0^2 \exp(2\alpha a)}{(\varepsilon_r T)^{1/2}}, \tag{3.7}$$

ε_{ij}^a is the activation energy, which can be taken in the following form [28]:

$$\varepsilon_{ij}^a \simeq \begin{cases} \frac{(\Delta\varepsilon_{ij} + \varepsilon_r)^2}{4\varepsilon_r}, & |\Delta\varepsilon_{ij}| \leq 2\varepsilon_r, \\ \varepsilon_r + (\Delta\varepsilon_{ij})\Theta(\Delta\varepsilon_{ij}), & |\Delta\varepsilon_{ij}| \gg 2\varepsilon_r, \end{cases} \tag{3.8}$$

$\Delta\varepsilon_{ij} = \varepsilon_j - \varepsilon_i$, and ε_i is the electron energy level at the center i regarding the electric field (the external field E and local fields):

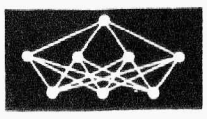
$$\varepsilon_i = \varepsilon_i^0 + Er_i + \sum_j \frac{n_j}{\mathcal{H}r_{ij}}, r_{Ij} = |r_i - r_j|, \tag{3.9}$$

where ε_i^0 is the intrinsic energy level and n_j, r_j are respectively the occupation number and the coordinates of j-th center, $n_j \in \{0, 1\}$, $r_j \equiv (x_j, y_j, z_j)$.

These relations represent the formulation of the physical model, which will be used in our further consideration. Usual values of parameters for typical organic molecules and dielectric organic environments are [27-31]: $\varepsilon_r \simeq 0.1 \div 0.5\ eV$, $\mathcal{H} \simeq 2 \div 4, \omega \simeq 0.01 \div 0.08\ eV$, $\alpha \geq 0.8A^{-1}, a \sim \alpha^{-1}$; and the last inequality in (3.1) is approximately equivalent to the condition [27]

$$\alpha r \geq 7. \tag{3.10}$$

Rough estimations of relative transition probabilities can be obtained without regard to non-exponential dependence of ν on $\varepsilon_i - \varepsilon_j$, r_{ij} and α_{ij}, as well as to the dependence of α_{ij} on the electric field (global or local field). We



also neglect the electron cloud polarization effects produced by electrons jumping to electron energy levels due to exchange interactions [32] because as the distance r grows, the exchange interactions vanishes in comparison with the Coulomb one.

Now let us consider the 3-center chain $i-j-k$ with the valent electron initially located at the i-center (if the k-center is occupied by an electron, we may consider the situation in terms of holes rather than electrons). There are three main possibilities of the transition $i \to k \to j$:

1. Two successing transitions $i \to k, k \to j$: according to (3.2)-(3.8) we obtain the rough estimation

$$W_{ijk} \simeq \min\{W_{ik}, W_{kj}\} \simeq \nu \exp(-\varepsilon^a_{max}/T - 2\alpha r) \tag{3.11}$$

where $\varepsilon^a_{max}, \alpha, r$ are taken for the pair of centers, (i,k) or (k,j), possessing the smallest W.

2. Virtual transition through the l-level (now its occupation number is irrelevant). A simple quantum-mechanical consideration leads to the rough estimation

$$W'_{ikj} \simeq \frac{I_0^2}{\Delta\varepsilon_{ik}^2 + \sigma^2} \nu \exp(-\varepsilon^a_{ij}/T - 2\alpha_{ik} r_{ik} - 2\alpha_{kj} r_{kj}), \tag{3.12}$$

where σ is due to the k-level energy fluctuations and may be taken as a value of the order of ε_r.

3. Transition via thermal activation to the k-level, but without thermalization on it. According to Dogonadze & Kuznetsov [28], and omitting numerical and logarithmic multipliers, we can write

$$W''_{ikj} \simeq (\nu^2/\omega) \exp(-\varepsilon^a_{max}/T - 2\alpha_{ik} r_{ik} - 2\alpha_{kj} r_{kj}) \tag{3.13}$$

where $\varepsilon^a_{max} = \max\{\varepsilon^a_{ki} + \Delta\varepsilon_{ij}\}$.

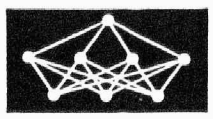
3.3. ML Quantum Electronics

Electrons are not the only pretenders to information carriers at ML. An alternative possibility concerns the utilization of some cooperative excitations in extended molecular structures. They may be chemically or electromagnetically pumped coherend oscillations and boson fields in one-dimensional extended molecular chains, which can form a net. Similar things can be found in the realm of molecular biology.

There exist a few types of one-dimensional biological intracellular structures - components of the cytoskeleton, called filaments. They are: microtubules (MT), actin microfilaments intermediate filaments, microtabecular lattices, etc [34]. It is difficult to say whether coherent excitations exist in these structures and carry any biological function in the cell. Meanwhile, the soft collective vibrational modes should exist in any extended ordered molecular structure; and as soon as the environment is strongly non-equilibrium (i.e., the pumping into the hard modes take place), the soft modes can be activated above the thermal level, and various non-linear phenomena concerning these kind of excitations should exist at proper conditions. They can be used for ML signal transmission and processing. In other words, we can create a laser-like situation (the inverse occupation of modes) in a single molecular chain and thus realize various phenomena of non-linear optics at ML: soliton, propagation along the chain (whether altering the state of the chain or not), bistability of the chain as a whole, amplification of a ML signal, etc. These phenomena can be considered as a possible physical basis for ML MN model implementations.

From general arguments we can claim the following. 1) An energy pumping along the whole length of the filament is necessary to obtain non-dissipative long-range signal propagation along the filament. It may be biochemical pumping produced by the GTP or the ATP hydrolysis, etc. 2) The system should be close to the excitation threshold (or, in general, to a some kind of phase transition) to be sensitive to a ML signal. 3) The interconnections between chains forming a net should be provided by some other ML subunits, for example, the microtubule associated proteins (MAPs) in the case of MT-based network. 4) The interface should be assisted by some external agent (light, microwave irradiation, hypersound, etc), interacting with the soft cooperative modes. 5) The learning process can be imagined as changing the network structure due to polymerization/depolymerization controlled by the soft mode excitation.

There are a lot of possibilities for constructing models of the Frohlich type. The reader should believe that it is possible to construct an abstract theoretical model starting from a Hamiltonian similar to (3.15), but involving some additional high-order interactions, providing its solutions possess desired properties under the proper choise of parameters. A more difficult question is its validity for a particular physical system. The following situations can be necessary for NN implementations, based on the network of randomly connected ML chains: 1) The chain should conduct non-decaying soliton-like waves, which are able to pass through each other and to interact with some ML elements incorporated into the chain. These ML inserts may serve as memory, interface and/or valve elements in the architecture, like the one considered in Section 2.4. 2) On the contrary, the whole chain can be considered as the cell body, if it is bistable by nature and can be switched from one state to another depending on the total signal power coming to it from adjacent chains via local ML learning bonds. In this case, we may obtain the neurolike medium architecture.



4. Physical Models of ML Neurolike Systems

4.1. A Multilayer LB-Film Based ANS

First, let us consider the Langmuir-Blodgett (LB) four-layer fragment structure shown in Fig.3, bearing in mind that we are dealing with a periodical multistructure composed of such chains. This hypothetical multistructure is of the Y-type. 2-4 layers are composed of molecules having conducting tails (i.e., the potential energy U of an electron in this tail is less than that of the environment, see Fig.3b). For example, this tails may be chains with conjugate bonds. There are three sorts of molecular heads: deep donor, shallow acceptor and intermediate acceptor. Molecules with acceptor heads are the main components of the structure. They are diluted by the molecules with donor heads with concentration c. Some donors are occupied by electrons and hence are neutral, others are free and positively charged. The compensating negative charges can be located anywhere within layers and are supposed to be fixed (deep acceptors, not shown). The layer number 1 in Fig.3a differs from the others only by molecular tiles: here molecules with donor heads have only conducting tails, while the molecules with acceptor heads have tails with saturated hydrogen bonds (not shown). This is the static picture.

Dynamics begins when we apply the electric field $E = (0, 0, E_z)$, $E \equiv E_z$ to the structure. First, each electron can jump from one head (or localization center) to another in the vertical direction z provided that it gains energy when jumping. Otherwise the jump probability is exponentially small. We suppose that deep donors have different energy levels, due to the local environment or to variations in its chemical structure. Hence, the condition of jumping depends on the center. Let us suppose that the electric field is applied during a short time t_1, which is sufficiently large in comparison with the time of the allowed vertical transition τ_v, but small in comparison with the intralayer relaxation time τ_h. This condition implies the good inequality $\tau_v \ll \tau_h$, which can be met by an appropriate choice of potential barrier heights between molecules. See Section 3.1.

After the electric fields is removed, electrons which have jumped begin move from center to center in the layer. The probability of coming back to the previous layer is small due to the energy loss during the localization process. This motion, consisting of tunnel jumps, can be roughly considered as a deterministic one if local fields inside the layer produced by charged donors are sufficiently large. Hence, we may speak about intralayer electron drift in the local field. This drift terminates when the electron achieves a donor. It can be the same donor each time when this process occurs for a given initial electron location. For the

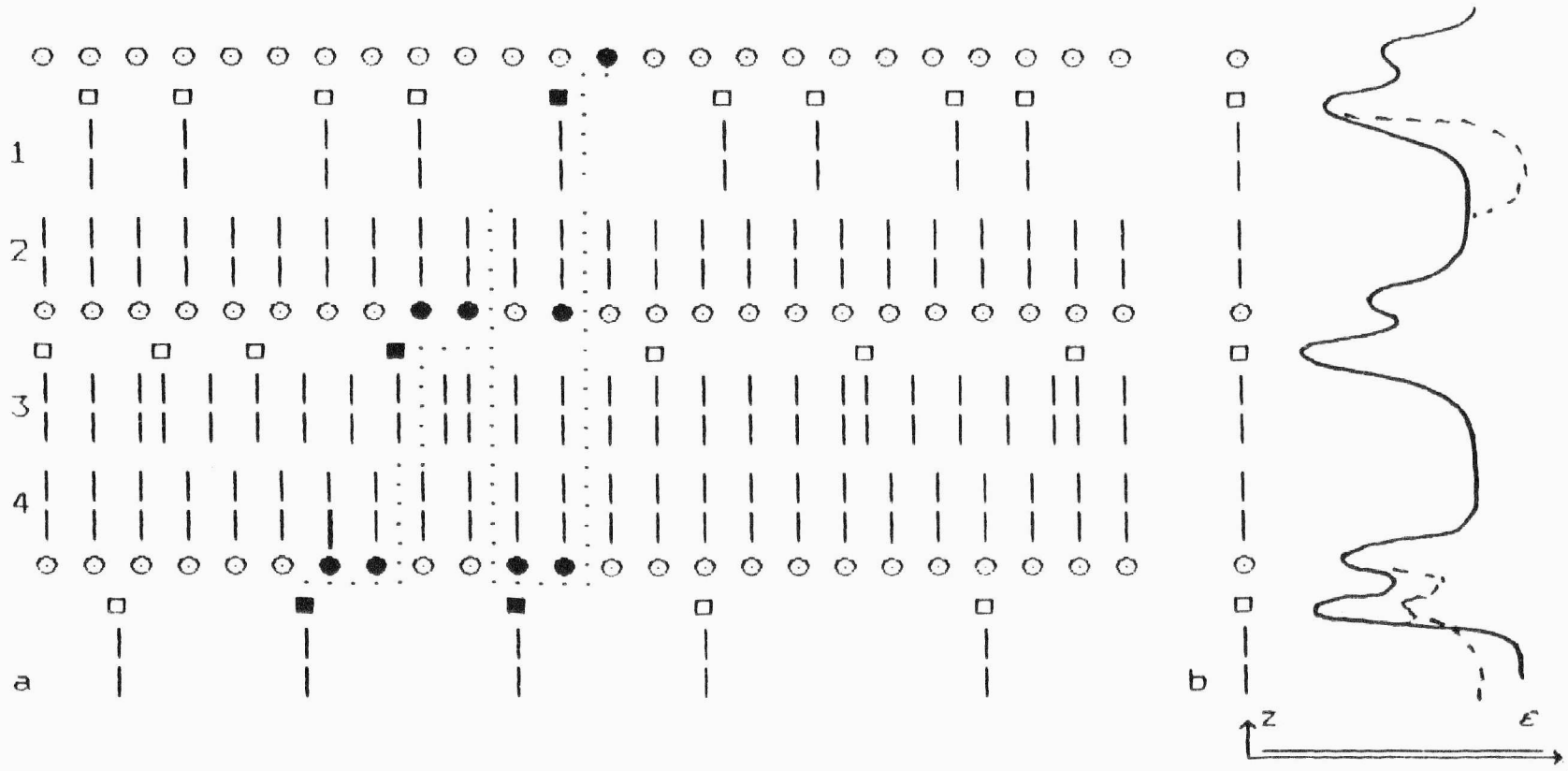


Fig.3: *A fragment of the LB multistructure (a) and its energy diagram (b). Vertical lines are conducting tails, deep donors ⊙ intermediate aceptors, dots - a possible electron trajectory in the alternating electric field* $\{E_t\} = \{E_o, -E_o, E_o\}$, □, ● *- points of localization of the electron.*

probability of error to be sufficiently small, the concentration of donors c should lie within definite limits.

Now let us suppose that we have the multistructure composed from the fragments considered above, which is put between two electrodes. Electrons are injected into the first layer from the top electrode every time the electric field E exceeds some threshold value E_o. The field of different signs and values in a quasi-random sequence is applied periodically by pulses with the period $t_o = 1$. Their further motion is similar to that of the model described in Section 2.4, but is like a drift with diffusion rather than a pure diffusion. Due to this feature we need a large number of extra layers (of the order of $N^{4/3}$) which cannot be used for learning (the main principles of usage of this structure have been described in the Section 2.4). On the other hand, the number of layers is restricted by admissible degree of the stochasticity of the electron trajectory. The problem can be solved by using the cross-strips scheme, in which the interface becomes sequentially-parallel. In this case the necessary number of layers can be less than N.

4.2. About submicron level implementations of NN models

There exist many possibilities for molecular based NN model implementations at the micron and submicron levels, which are more realistic than the ML ones. Optical implementation seems to be the most promising among them. But in this Section we describe some other ideas. Namely, we consider a few variants of the synaptic chip based on crossed strips geometry (see Fig.5). The task is to provide a variable element (resistor or capacitor) in each cross. Sandwich structure of the cross is preferable, but it is not always admissible in physics. For example, let us consider a superionic materials based structure (see Fig.4). We have a variable layers at the interfaces 1-2 can be the crossed strips of Fig.5, and electrode 4 should be connected with another strip via a threshold element (MOSFET, diode, etc). The learning rules realization is straightforward. Learning is provided by passing a definite electric charge in the circut 3-4. The problem is to cut off the electrode 4 after learning for a long time.

Another possibility concerns electron beam access associative memory. We can realize the operation (2.2) of multiplying a vector by a matrix if we store the matrix as a property of the surface which is subjected to the electron beam, and if we cause the beam to scan this surface in direction x during vector input and in direction y during the readout of the result. The application of the cross-strips scheme together with the matrix of catodes may enlarge the degree of parallelism.

5. Discussion: The NN vs. the Von Neumann Approach in ML Electronics

In conclusion, we can say that the NN approach is more appropriate to ML electronics than the von Neumann for many reasons.

- It appears that the ML computer cannot be competitive with traditional semiconductor based computers in usual calculational tasks in the near future. On the other hand, artificial intelligence tasks, which appear as ones of exponential complexity for von Neumann computers, became more and more topical today. They demand application of the NN approach, based on a non-traditional hardware.

- The von Neumann architecture and methods of information processing encounter many problems at ML: 1) it is impossible to build a macroscopic device of a given architecture with atomic accuracy in the near future, even using biotechnology and self-assembling methods; 2) lesion at ML is unavoidable during exploitation (at least due to cosmic radiation effects); 3) problems of organizing parallel logical operations become too difficult as the number of computing elements increases. On the other hand, the NN approach is consistent with high parallelism and partially disordered architectures and allows a relatively large fraction of defects to arise during exploitation.

- Possible physical principles of information representation, transmission, processing and storing at ML are

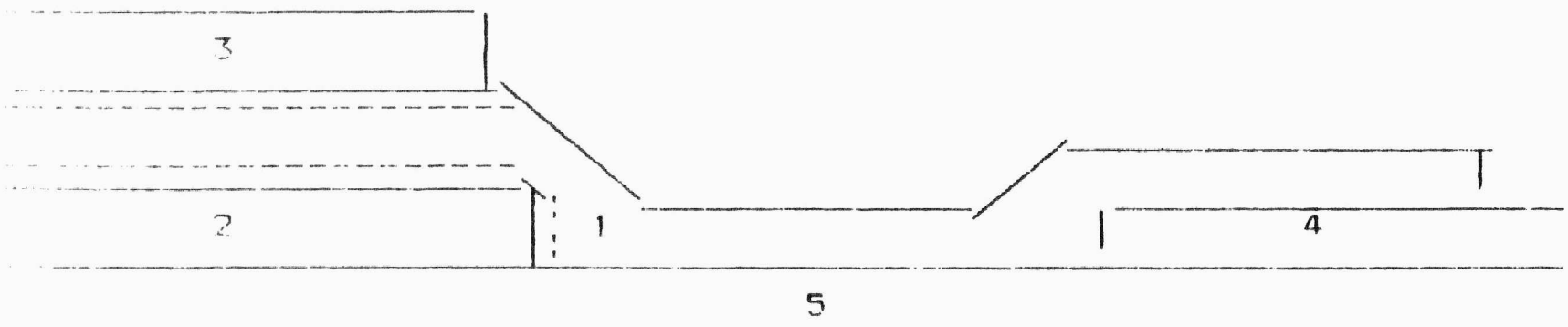


Fig.4: *A superionic conductor (1) based variable capacitor which can be used as a synaptic element. 2,3 - passive electrodes, 4 - reversible electrode, 5 - dielectric substrate.*

more consistent with the NN approach rather than the von Neumann due to its probabilistic and analog nature.

- The traditional von Neumann type computer demands a high degree of reliability of its elements: the probability of error per logical operation should be less than 10^{-25} for a large present-day computer. This is physically impossible for ML elements. The situation becomes more dramatic when the integration degree N increases. Of course, we can introduce a large redundancy and make use of different self-control methods, but it means returning to non-ML. As for ML NN computer, its reliability increases with N, and a high reliability of each element is not necessary.

- The addressing problem is very difficult to solve at ML, when it is necessary to send a given signal to a molecule located at a given space point. But in the case of an NN device, only the reproducibility of randomly generated addressing is necessary. We need not know where the destination molecule is located, and the destination molecule need not know from which source the received signal originates.

- This ML interface problem is rather complicated for the von Neumann computer. The number of interface terminals of a traditional chip grows with N faster than $N^{1/2}$, when N is the integration degree. Thus, N is restricted by interface possibilities even for submicron level device. As for ML neurocomputers, we have seen that the interface problem in principle can be solved.

- Another aspect of the ML interface problem is amplification of a ML signal to the macroscopic level. It is rather difficult when the signal is produced by one molecule. But in the case of NN ML computer a signal received by a neuron is the sum of a large number of ML signals, hence, it is not of ML.

- Present-day basic ML technological methods, such as the LB technique, are inconsistent with the demands of the von Neumann approach at ML, but can be used for ML NN computer assembling, as we have seen in Section 4.

References

[1] Carter, Forrest L., ed. (1982) Molecular Electronic Devices, Marcel Dekker, New York.

[2] Carter, Forrest L., ed. (1987) Molecular Electronic Devices-II, Marcel Dekker, New York.

[3] Carter, Forrest L., et al., eds. (1988) Molecular Electronic Devices, North Holland, Amsterdem.

[4] Emerging technologies no.9. (1983) Molecular electronic: beyond the silicon chip, Technical Insights, Fort Lee, New Jersey, Libary of Congress Catalog No.83-050975.

[5] Aizava, Masuo, ed. (1985) Proc. of the International Symposium on Future Electron Devices (FED BED/MED Symposium). Chiyoda-ku, Tokyo.

[6] Haddon, R.C., and Lamola, A.A. (1985) The MED and the biochip computers: present status. Proc. Natl. Acad. Sci. 222, 1874.

[7] Haken, Hermann, ed. (1988) Neural and Synergetic Computers, Springer-Verlag, Berlin.

[8] Hopfield, John J. (1982) Neural networks and physical systems with emergent collective computational abilities, Proc. Natl. Acad. Sci. 79, 2554-2558.

[9] Hopfield, John J. (1984) Neurons with graded response have collective computational properties like those of 2-state neurons, Proc. Natl. Acad. Sci. 81, 3088-3092.

[10] Personnas, L., Guyon, I., and Dreyfus, G. (1987) High-order neural networks: information storage without errors, Europhys. Lett. 4, 863-867.

[11] Amari, Shun-ichi, (1988) Associative memory and its statistical neurodynamical analusis, in ref. 2, 85-99.

[12] Amit, Daniel J. (1990) Attractor neural networks and biological reality: associative memory and learning, Proc. of Intelligent Autonomous Systems, Univ. of Amsterdam (in press).

[13] Dotsenko, Victor S. (1986) A layered hierarchical model of memory, Sov. Phys. JETP Lett. 44, 151-153.

[14] Wong, K. Y. M., and Sherrington, D. (1989) Theory of associative memory in randomly connected Boolean neural networks, J. Phys. A 22, 2233-2263.

[15] Conrad, Michael, and Hong, Felix T. (1985) Molecular computer design and biological information processing: an electrochemical and membrane reconstitution approach to the synthesis of a cellular automaton, in ref. [1e], 89-94.

[16] Eckmiller, R. (1988) Neural Nets for the Manajement of Sensory and Motor Trajectories, in ref. [2], 229-239.

[17] Carpenter, G.A., and Grossberg, S. (1988) Self-organizing neuronet architectures for real-time adaptive pattern recognition, in ref. [2], 42-75.

[18] Proc. of the IEEE International Joint Conference on Neural Networks. San Diego, July 1989.

[19] Rujan, P. (1988) Cellular automata and moddels of memory, in Jeam Delacour and J.C.S. Levy (eds.), Systems with learning and memory abilities, North Holland, Amsterdam, 571-596.

[20] Mandelbrot, Benoit B. (1976) The Fractal Geometry of Nature, Freeman, New York.

[21] de Gennes, Pierre-Gilles (1979), Scaling Concepts in Polymer Physics, Cornel Univ. Press, Ithaka.

[22] Pietronero, Luciano, and Tosatti, Erio, eds. (1986) Proceedings of the Sixth Trieste International Symposium on Fractals in Physics, North Holland, Amsterddam.

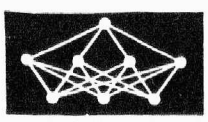
[23] Hawking, Stewen W., Euclidean theory of gravitation, in Recent Developements in Gravitation, Plenum Press, Carges, 1978.

[24] Likharev, Konstantin K., Mikroelekronika 16, 195-209, 1987.

[25] Kuznetsov, Alexander M., priv. comm.

[26] Ivanov, G.K., and Kozhushner, M.A., Theory of interimpurity electron tunneling in solids, Sov. Solid Science, 20, 9-16.

[27] Kapinus, Eugene I. Photonics of Molecular Complexes, Naukova Dumka, Kiev (in russian), 1988.



[28] Dogonadze, K.R., Kuznetsov, A.M., and Marsagishvili, T.A. The present state of the theory of charge transfer processes in condensed phase. Electrochim. Acta 25, 1-28, 1980.

[29] Movaghar, B., J. Mol., El., 3, 183, 1987.

[30] Volkenstein, M.V. Biophysics, Nauka. Moscow, 1988.

[31] Simon, J., and Abdre, J.J. Molecular Semiconductors. Photoelectrical Propesrties and Solar Cells, Springer-Verlag, Berlin, 1985.

[32] Wilson, S. Electron Correlation in Molecules, Clarendon Press, Oxford, 1984.

[33] Larkin, A.I., and Matveev, K.A. Current-voltage characteristics of mesoscopis semiconductor junctions, Sov.Phys. JETP 93, 1030-1038, 1987.

[34] Bershadsky, A.D., and Vasiliev, J.M. Cytoskeleton, Plenum Press, New York and London, 1988. Alberts, B, et al. Molecular Biology of the Cell. Garland Publishing, Inc., New York & London, Vol. 3, Chap.10, 1983.

Literature Survey

Utsugi A., Ishikawa M.: Construction of Inner Space Representation of Latticed Networks Circuits by Learning
Neural Networks Vol.4, 1991 No.1 pp.81-87

Key words: parallel sensor; analog resistive circuit.

Abstract: A position-detective sensor with an intrinsic inner coordinate system using a parallel calculation schema is proposed. The schema keeps the calculation time independent of the resolution or the size.

Wang J.F., Wu Ch-H., Chang S-H., Lee J-Y.: A Hierarchical Neural Network Model Based on a C/V Segmentation Algorithm for Isolated Mandarin Speech Recognition
IEEE Transactions on Signal Processing Vol.39, 1991 No.9 pp.2141-2146

Key words: neural networtks.

Abstract: A novel algorithm simultaneously performing consonant/vowel (C/V) segmentation and pitch detection is proposed. Based on this algorithm, a consonant enhancement method and a hierarchical neural network scheme are explored for Mandarin speech recognition.

Wang L.X., Mendel J.M.: Cumulant-Based Parameter Estimations Using Structured Networks
IEEE Transactions On Neural Networks Vol.2, 1991 No.1 pp.73-83

Key words: structured networks.

Abstract: This paper develops a two-level three-layer structured network to estimate the moving-average (MA) model parameters based on second-order and third-order cumulant matching.

Yan H.: Stability and Relaxation Time of Tank and Hopfield's Neural Network for Solving LSE Problems
IEEE Transactions on Circuits and Systems Vol.38, 1991 No.9 pp.1108-1110

Key words: neural networks; LSE problems; stability; transformations; eigenvalues.

Abstract: The network can also be used for solving linear least squares error problems. It is shown here the stability of the network is guaranteed even under weaker conditions. More accurate formulas are derived for determining the relaxation time of the network.

Yao Y., Freeman W.J., Burke B., Yang Q.: Pattern Recognition by a Distributed Neural Network: An Industrial Application
Neural Networks Vol.4, 1991 No.1 pp.103-121

Key words: autoassociator; back propagation; chaos; feature enhancement; olfactory system; pattern recognition.

Abstract: In this report, a distributed neural network of coupled oscillators is applied to an industrial pattern recognition problem. The network stems from the study of the neurophysiology of the oilfactory system.

Zak M.: An Unpredictable-Dynamics Approach to Neural Intelligence
IEEE, 1991 pp.4-10

Key words: dynamics; neural intelligence; neural networks.

Abstract: This dynamic neural-network architecture takes advantage of the notion of terminal chaos to process information in a way that is phenomenologically similar to brain activity.

Zhou D.N., Cherkassky V., Olson D.E.: A Neural Network Approach to Job-Shop Scheduling
IEEE Trans. on Neural Networks Vol.2, 1991 No.1 pp.175-179

Key words: analog computational network.

Abstract: This paper presents a novel analog computational network for solving NP-complete constraint satisfaction problems, i.e., job-shop scheduling.

A VIEW ON NEURAL NETWORK PARADIGMS DEVELOPMENT

(Part 6)

J.Hořejš [1]

Here we continue in the tutorial paper concerning the neural network paradigm, which first part was published in the Neural Network Word. No. 1, 1991.

Fig.30 shows a possible coverage of two input space classes by the fields of influence. Those hyperspheres,which cover one of them are connected to the same output neuron. Note also the existence of black areas, in which fields intersect (while A and B do not).

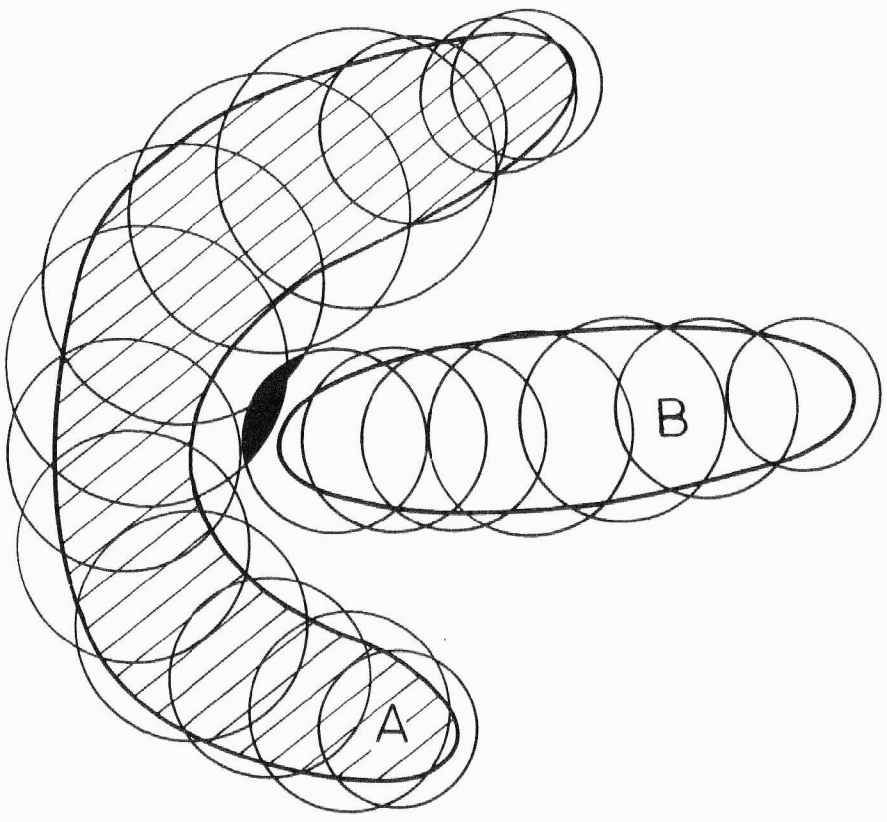


Fig.30: *Coverage of classes A and B*

While the main idea is simple and hopefully has been fully understood, there are still some intricacies to be aware of. When a field of influence (hypersphere) has to shrink because of a newcomer $\mathbf{y}^*$, it may happen that some of former input vectors ($\mathbf{x}'$ in Fig. 31) except the "founder" - *prototype* $\mathbf{x}_1$ can fall out off the field. In that case the "expelled" inputs simply establish new fields of their own with radius equal or slightly less to 1 or to the biggest distance which will not cover alien vectors ($\mathbf{y}$ in Fig. 31); by this *expansion* they can of course cover other inputs of their own category ($\mathbf{x}''$) so that not everybody who was just excommunicated from the paradise is necessarily compelled to build its own field.

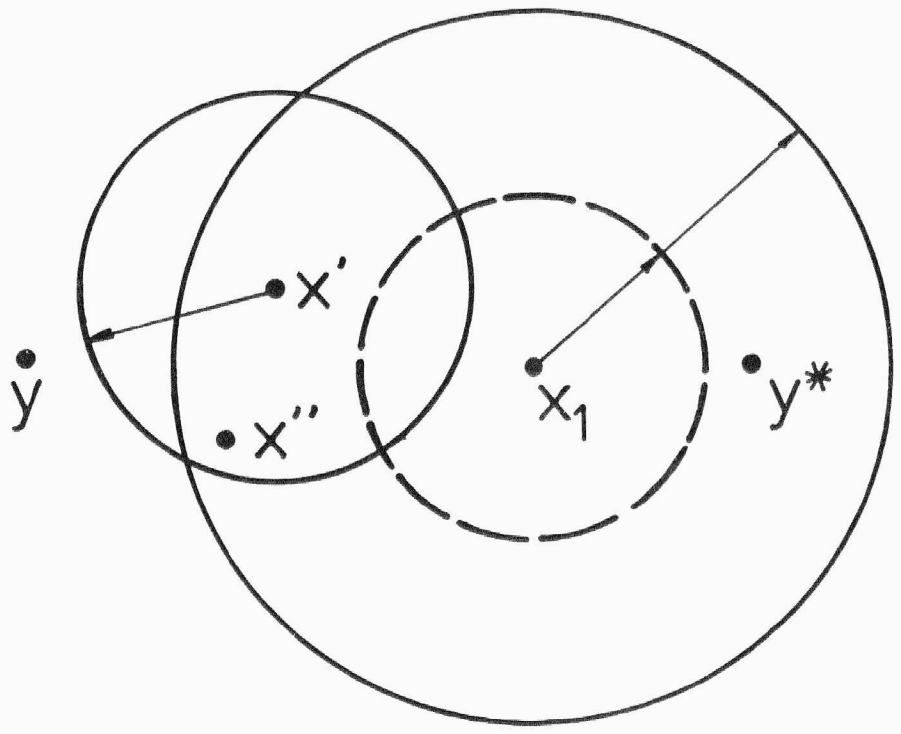


Fig.31: *Corrections of influence fields*

Anyway you have to check sooner or later, but before the working mode is entered, whether shrinking and expanding process has placed all learned inputs in the proper category. The algorithm itself is however so simple and efficient that this can easily be accomplished even in real time applications, when the two modes are interleaved. Compared to iterative methods like back-propagation, it is very quick and if you extend it by canceling fields that are no more necessary (when the remaining ones cover the whole classification class), it is even memory effective. Note also that it does not require complete cross connection of neurons between two adjacent layers, namely hidden and output one.

The only drawback of RCE paradigm is that it is primarily classification oriented, so that it gives boolean outputs only. This lack of generality is on the other hand (partially) compensated by ability of easy and "natural" generalization: if a previously unknown input pattern $\mathbf{x}$ is covered by some hypersphere h, then: if the radius of h is small, then it is close (in sense of Euclidean metrics) to the prototype of h; if it is big, there were in the training set T no counter examples which would justify why not to answer in the same way as to the prototype of h. Of course in a particularly messy environment full of exception, even the best net might be confused.

The RCE net is simple to program and this provokes to build the whole hierarchy of them. Given together, they form the so called NESTOR system, which will be now briefly described. Let us emphasize that, as you have probably noted, even with RCE networks, there are many variants possible; also here we will describe general strategies rather than the particular system (version etc) which may have been chosen when releasing a specific product [described in a company manual].

[1] Prof. Dr. Jiří Hořejš,CSc.,Department of Computer Science, Charles University, 118 00 Prague 1, Malostranské nám.25, Czechoslovakia

Note. If you are not afraid of more complicated calculations, you can replace hyperspheres by other covering hyper objects, like ellipsoids, intersections of hyperspheres etc. Thus in -b) above you could instead of shrinking $h1$ use the difference of $h1$ and $h2$.

III.The Nestor Development System, NDS.

Assume now that you have several RCE nets - *modules*. Every module is assigned a *priority level*. A Class Selection Device, CSD, combines the results of single modules to elaborate the final answer.

From the CSD function we can easily see, that the modules should be different.

One possibility how to make use of it is the following. In order to categorize the input space, several different views can be taken; then we also get different characteristics of the objects to be classified and arrive thus to different input vectors and the features they use. Or, possibly, to divide one big feature vector into several subvectors.

For example we can distinguish different features of people: one sort of input vectors will care about their physical properties, the other will prefer psychological ones; or, if you like to recognize a person just entering the room, one point of view may concentrate on his/her voice, the other on visual appearance, the third one on his/her odor. When recognizing an illness, you may distinguish results of simple X-ray, computer tomography, magnetic resonance and sonar examinations. In the technical control or quality tests you can have data measured by sensors based on different principles or different wave bands. In recognizing true signatures/ underwritings (on checks, say) from forgeries [one of the first commercially success of NESTOR], you can take the whole sequence of direct digitized handwritten characters, or various measures of signing process like total time spent, maximal acceleration of the pen, number of pen lifts, proportion of highest and lowest letter etc.

All of them can be evaluated by different modules; and from the examples it should be also clear that some data are naturally more relevant then others, giving thus natural priority assignments. On the other hand a quite unambiguous statement of several generally less perfect devices can be decisive in case of unconvincing judgment of a more sophisticated one.

And here comes another property of modules into the game. In some cases, the RCE algorithm is not able to come to the unique answer: introducing in working mode an input not from T, more than one or none output neurons can be stimulated; this is due to the fact that you can almost never cover a given input class by finite number of hyperspheres exactly. When several such classes almost touch each other (euclidean distance between them is very small) and their shape is highly irregular, it may be practically problematic even in the adaptive mode requiring to create too small fields of influence! [Typically, prototypes with thresholds below some minimum size are after some time deleted from the memory.] Thus particular modules can - besides showing excited output neurons - also give a summary answer "confused-c" or "unidentified-u" [while we expected "identified-i" tag].

There can be several selection strategies of CSD. If one or more highest priority modules produce the i-tag, their answer is probably correct. If several high priority modules (even confused) share the same class C as an answer, while others at the same priority level share anther one C', simple voting can be accepted. Otherwise you go to the next lower priority level and repeat the procedure. You can however design another strategy, depending among others how important the final verdict is.

There is the possibility to introduce other modules (of low priority, say), if you feel that the answer is not convincing etc.

Again, the described system can be extended to the case, when two or more fields deliberately intersect, see Fig.32. This corresponds to the case that we are not dealing with a single unique classification (a partition of input space into disjoint classes), but admit two or more classification criteria. If for example cars are sorted according to price, motor power, color,..., there may be [expensive, big, blue ...] cars along with [cheap, small, orange ...], [expensive, small, orange, ...] ones etc. Two or more hidden neurons covering in their influence fields all of them can then be attached to more than one categories, i.e. to more than one output neurons. In Fig.32 two classes A nad B are shown, the members from the intersection of which belong to two categories.

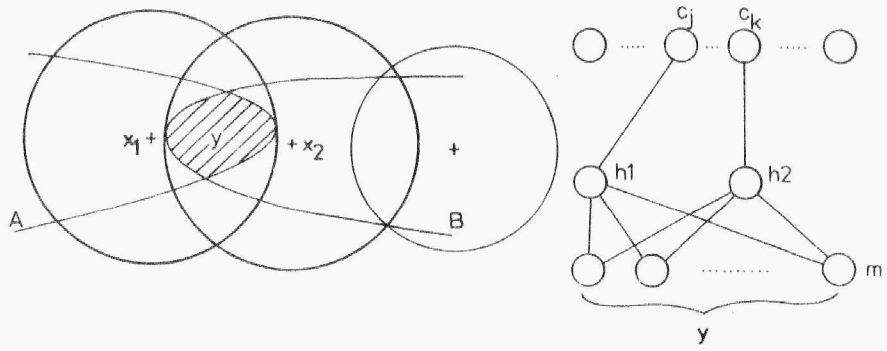


Fig.32: *Objects belonging to two categories*

It is also possible to introduce some fuzziness; in that case the hidden neurons will produce not a boolean output, but any number between 0 and 1, say, where this number is given by a measure of closeness of incoming new input vector $\mathbf{x}'$ to the prototype vector $\mathbf{x}$ of the hidden neuron h with threshold ρ (or its original size $\rho = 1$). The output ξ of h is 0 for $d(\mathbf{x}', \mathbf{x}) > \rho$ and 1 for $d(\mathbf{x}', \mathbf{x}) = 0$ while for $0 < d(\mathbf{x}', \mathbf{x}) < \rho$ will assume a value from the interval (0,1); there are many eligible transfer function of this sort and you can generally choose. Also the role of CSD will be more complicated and you can be inspired by expert systems how to add *credibility* contributions coming from particular hidden neurons to establish the credibility of the summing category output neuron. [Let us recall that this description does not distinguish between a concrete commercially available product of NESTOR, Inc. and general ideas which do or might solve in a different way the tasks under consideration.]

(Continuation)

Instructions to authors

1. Manuscript

Two copies of the manuscript should be submitted to the Editor-in-Chief.

2. Copyright

Original papers (not published or not simultaneously submitted to another journal) will be reviewed. Copyright for published papers will be vested in the publisher.

3. Language

Manuscripts must be submitted in English.

4. Text

Text (articles, notes, questions or replies) double space on one side of the sheet only, with a margin of at least 5 cm, (2") on the left. Any sheet must contain part or all of one article only. Good office duplication copies are acceptable. Titles of chapters and paragraphs should appear clearly distinguished from the text.

Author produced (camera ready) copy is acceptable if typed on special sheets which are available from the Editor, and adherence to the Typing instructions (also available from the Editor) has been taken care of, is emphasized that camera ready text should be typed single space (i.e. with no space between the lines). Complete text records on 5 1/4" floppy discs are also acceptable, if typed according to the instructions available from the Editor.

5. Equations

Mathematical equations inserted in the text must be clearly formulated in such a manner that there can be no possible doubt about meaning of the symbols employed.

6. Figures

The figures, if any, must be clearly numbered and their position in the text marked. They will be drawn in Indian ink on white paper or tracing paper, bearing in mind that they will be reduced to a width of either 7,5 or 15 (3 or 6") for printing. After scaling down, the normal lines ought to have a minimum thickness of 0,1 mm and maximum of 0,3 mm while lines for which emphasis is wanted can reach a maximum thickness of 0,5 mm. Labelling of the figures must be easy legible after reduction. It will be as far as possible placed across the width of the diagram from left to right. The height of the characters after scaling down must not be less than 1mm. Photographs for insertion in the text will be well defined and printed on glossy white paper, and will be scaled down for printing to a width of 7,5 to 15 cm (3 to 6"). All markings on photographs are covered by the same recommendations as for figures. It is recommended that authors of communications accompany each figure or photograph with a descriptive title giving sufficient information on the content of the picture.

7. Tables

Tables of characteristics or values inserted in the text or appended to the article must be prepared in a clear manner, preferably as Camera Ready text. Should a table need several pages these must be kept together by sticking or other appropriate means in such a way as to emphasize the unity of the table.

8. Summaries

A summary of 10 to 20 typed lines written by the author in the English will precede and introduce each article.

9. Required information

Provide title, authors, affiliation, data of dispatch and a 100 to 250 word abstract on a separate sheet. Provide a separate sheet with exact mailing address for correspondence.

10. Reference

References must be listed alphabetically by the surname of the first author. List author(s) (with surname first), title, journal name, volume, year, pages for journal references, and author(s), title, city, publisher, and year for the book references. Examples for article and book respectively:

[1] Dawes, Robyn M. and Corrigan, Bernard: Linear models in decision making, Psychological Bulletin, **81** (1974), 95-106.

[2] Brown, Robert G.: Statistical Forecasting for Inventory Control, New York: McGraw-Hill, 1959.

All references should be indicated in the manuscript by the author's surname followed by the year of publication (e.g., Brown, 1959).

11. Reprints

Each author will receive 25 free reprints of his article.

This is Seagate Technology.

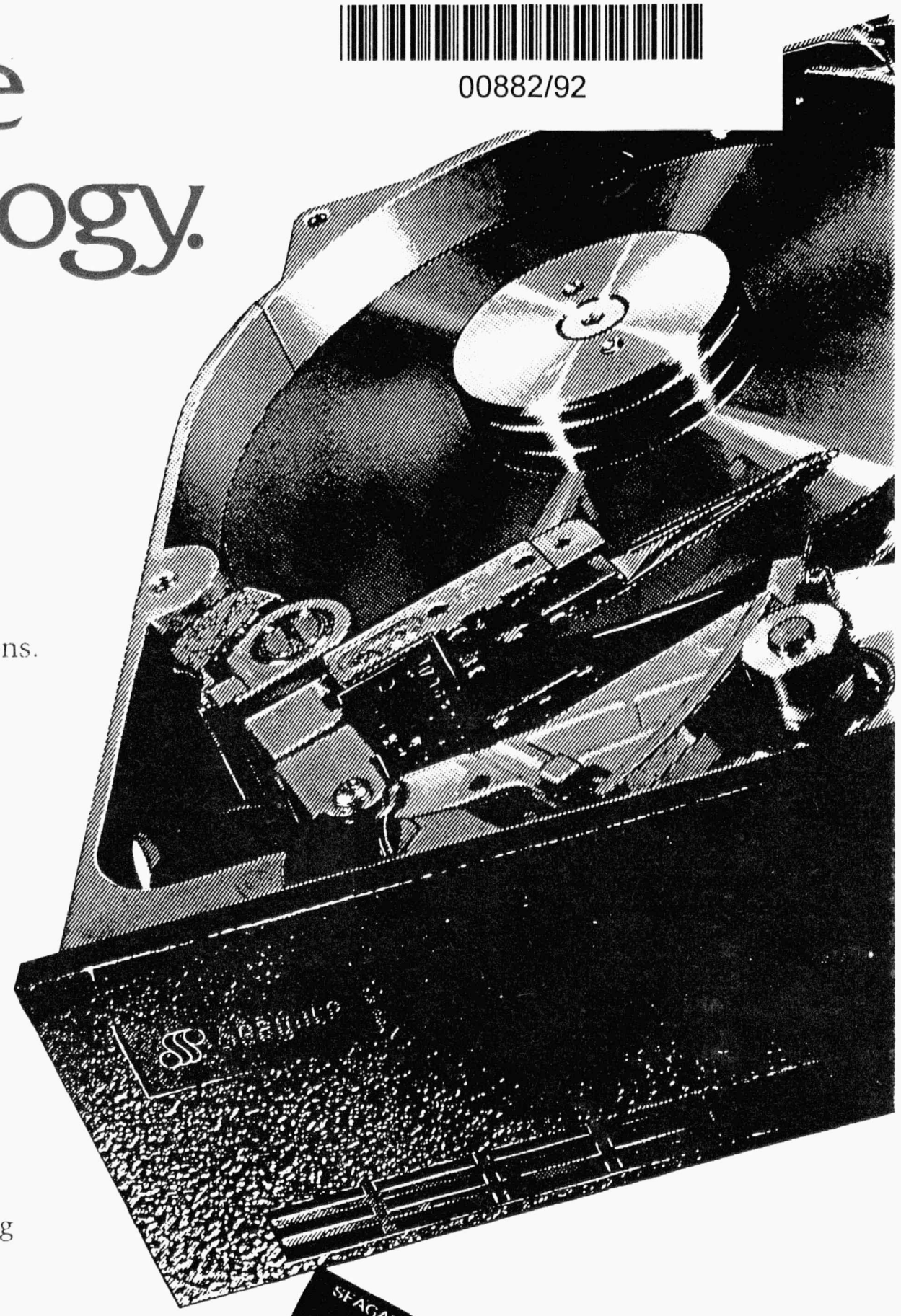
Seagate's line of hard disc drives is packed with high technology. And every one is built to the highest quality and reliability standards in the industry.

And now, Seagate drives are available locally for all your Personal Computer applications.

Only Seagate can offer you full technical support, and a one-year warranty . through our authorised representatives in your country.

Complete technical and interface details are included in the Seagate product brochures, which are free of charge to professional PC buyers and users. Simply use the coupon below to request your copies.

You'll soon see why Seagate has become the world's leading independent manufacturer of disc drives.



Seagate

Seagate Technology Europe
Seagate House, Fieldhouse Lane, Globe Park, Marlow SL7 1LW Great Britain
Tel: 0628 890366 Fax: 0628 890660 Telex: 846218 SEAGAT G

To: Seagate Technology Europe,
Seagate House, Fieldhouse Lane,
Globe Park, Marlow SL7 1LW Great Britain.
Please send me technical details of Seagate disc drives
Name _____
Job Title _____
Organisation _____
Address _____
_____
Country _____
Type of business _____
Number of employees _____ Number of PCs _____
☐ I use a PC ☐ I authorise the purchase of PCs
☐ I am a technical support manager