N90-29032

A Hybrid Architecture for the Implementation of the *Athena* Neural Net Model

by

C. Koutsougeras
Computer Science Department
Tulane University
New Orleans, LA 70118

and

C. Papachristou
Computer Science Department
Case Western Reserve University
Cleveland, OH 44106

Abstract

In this paper the implementation of an earlier introduced neural net model for pattern classification is considered. Data Flow principles are employed in the development of a machine that efficiently implements the model and can be useful for real time classification tasks. Further enhancement with optical computing structures is also considered here.

1. Introduction

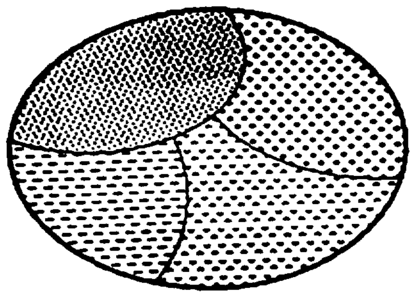
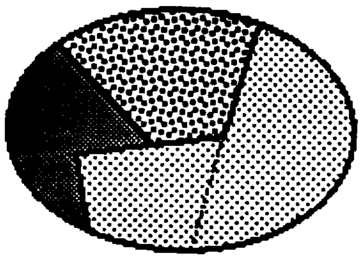
Present day computers depend for their performance on programming. Before any specific task can be carried out by a computer, a programmer has to understand the nature of both the task and the domain of reference. He/she must determine those features of the referenced domain which are pertinent to the task. He/she must also define the basic steps which carry out the task, and the "data structures" which are appropriate for representing the relevant information. Therefore targets of conventional computers were mainly well defined tasks for which complete information can be encoded into the explicit steps of a program. A number of interesting applications however, would require machines to operate with incomplete or without explicit information. In some engineering applications for example, computers are used as controllers to carry out the necessary decision making. Current computers can perform well if the decision process is well understood. Yet in many cases explicit description of a decision process is not available because the relation between the pattern of the environmental variables (input) and the required action (output) is very complex or it is not well understood. Of interest in such cases are machines which can deduce descriptions of the input-output relation from an abstract specification such as a typical set of input-output examples.

Neural nets are machine models which have been developed in the effort to meet this challenge. These models are able to automatically develop internal representations of domain information which is presented to them in terms of domain examples, thus exhibiting true learning by example capabilities. Neural nets operate in two phases. In an initial "training phase" they are presented with factual information consisting of input-output examples typical of a certain desired behavior. During this phase the network's function is adapted so that it becomes consistent with the examples. In the second phase - the normal processing phase - the network produces responses for inputs on the basis of its adapted function or in other words on the basis of its experience. Responses are produced even for inputs which the network may have never encountered before.

There are two fundamental problems associated with some of the most widely known neural net models [Hopf82] [Rume86]. The quality of the internal representations which can be developed with a given net depends on the degree of nonlinearity inherent with the net. The precision of the adaptation therefore depends on the nets size and topology which is an apriori choice. There is no known systematic way to go about this choice. Also these models assume a tremendous number of interconnections which make their implementation with existing VLSI technology very difficult. We have developed an alternative model named Athena which does not face these problems. The network as suggested with this model expands during training and therefore an apriori choice of topology is not required. The structure is tree-like feed-forward and its simplicity allows implementation with conventional VLSI technology. The analysis and the foundations of Athena have been described in [Kout88]. The purpose of this discussion is to present machine architectures considered for implementation. The model is briefly discussed here for reasons of cohesiveness. A machine architecture based on Data Flow models is then analyzed. The proposed machine can be implemented with conventional technology. Another advantage of Athena is that the required computations are such that existing optical processing structures can be efficiently employed. It is explained here how such optical processing structures can be embedded in the original architecture for a significant improvement in speed and throughput.

2. The basis of the internal representations

The subject of this work is the class of stimulus-response relations which can be formulated as a mapping M: $V \rightarrow Z$ where V is a discrete set and Z is a finite set. The elements of Z can be viewed as the classes into which the elements of V can be classified, and M is by definition (and without loss of generality) a many-to-one mapping. Given such a mapping M : $V \rightarrow Z$, an equivalence relation θ can be defined on V as follows: For any pair of distinct objects X and Y (elements of V), we will say that X relates to Y ($X \theta Y$) if and only if M(X)=M(Y). The relation θ therefore defines a partition of the objects (elements of V) into a number of object classes (equivalence classes) C_i, i=1,2,..k. Since the object classes uniquely identify M, they can be used as a representation for M. In turn the object classes are uniquely defined by a generalized hypersurface which consists of the envelopes of the classes as shown in figure 1a. In [Kout88] we describe a feed forward network model consisting

A mapping M: V->Z clusters V into classes. Lines correspond to a hypersurface on the basis of which M and the classes can be reconstructed.

Figure 1a Figure 1b

of hard thresholding elements which can internally represent an approximation of a hypersurface by means of hyperplane segments. The function of the model can adapt to any given mapping M by internally representing the corresponding hypersurface which partitions V into the object classes defined by M. The approximation is exact if the input space V is discrete (figure 1b). The hypersurface is incrementally formed on the basis of the available factual information consisting of a collection of objects (elements of V) of known classification. This incremental process is guided by an entropy measure. The hypersurface partitions V into an expectedly minimal number of convex regions (sets) each of which contains objects of a single class. Given an object X of unknown classification, that is one which was not encountered in the training set, assume that it belongs to a certain convex region S_k of the final partition. If all the instances of the training set which fall within S_k belong to the same object class, let's say C_k, then within S_k the classification decision is predominantly C_k and therefore with a high degree of confidence X's classification should also be C_k. For the reasons of completeness and reference the model is also briefly discussed in the following.

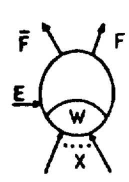
3. Structure and training

The building element of this model is a new type of threshold unit which is described in figure 2. Each unit has a set X of n "data" input lines, a control or enable input E, and two outputs F and F'. With each unit a weight vector W and threshold T are associated, representing some hyperplane $P = \{X | X \in R^n \text{ and } W^t X = T\}$. Then the functions F and F' represent the upper P^u and lower P^L half-spaces respectively. While the E input of a unit is not activated (E=0), both the outputs F and F' of that unit are inactive (F=F=0) regardless of the input X. When a unit is fed with an input vector $X \in R^n$ and its E input is activated, then F and F' are complementary assuming the values :

$$\begin{aligned} F=1 \text{ and } F'=0 \text{ if } W^tX \geq T \\ F=0 \text{ and } F=1 \text{ if } W^tX < T \end{aligned} \qquad (1)$$

where W^tX is the scalar product of W and X. For an enabled unit the meaning of the above equations (1), is that the unit's F output is active if the input X is on a certain side of the hyperplane that unit represents. If X is on the other side of the hyperplane then F is active.

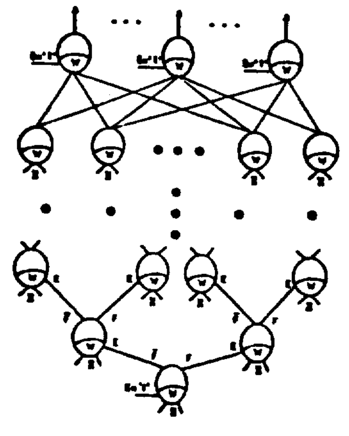
Each of the outputs of every unit is connected to the E input of another unit in the next higher layer. In this way, a set of units is connected to form a D_tree hereafter referred to as a D_tree. The enable input of the unit at the root of the D_tree is set to be always active. Any input vector presented to the network, is routed to the data inputs of every unit in the D_tree. During the presentation of an input vector X, the following operation takes place in the D_tree. The input X is broadcasted to all the units and for each unit at most one output is activated. Thus, in the whole network at most one path is activated for each input vector. If, given the input X, unit u_k is enabled (its E input is activated), then necessarily all of its ancestors are also enabled. If the ancestors of u_k represent the hyperplanes $P_1, P_2, \ldots P_j$, then u_k is enabled when X lies within one of the convex sets into which $P_1, P_2, \ldots P_j$ subdivide the input space V. Thus, a certain unit u_k is enabled only when the network's input lies within a certain



$$F = \begin{cases} E \text{ and } (W,X) \geq T \\ 0 \quad \text{Otherwise} \end{cases}$$

$$\bar{F} = \begin{cases} E \text{ and } (W,X) < T \\ 0 \quad \text{Otherwise} \end{cases}$$

A single unit
Figure 2



The network structure
Figure 3

bounded subset of the input space, hereafter referred to as the space of activity of u_k, and further, u_k divides that subset in two parts. In like manner, each output F_i of a unit at a leaf of the D_tree represents a convex subset V_i of the input space.

A final layer of output units of the same type completes the network. Each output unit corresponds to a single object class. The purpose of each output unit is to perform the logical OR function among a selected number of the tree's outputs. This operation is conceptually equivalent to the formation of the union of a selected number of convex sets V_i of those corresponding to the outputs of the D_tree. A selected set of outputs from the leaf units of the D_tree are used as data inputs for an output unit. The output unit corresponding to the class C_i collects those outputs of the D_tree representing the convex sets which form C_i. The E input of the output unit is set to be always active. The OR function is performed by an output unit by setting its weight and threshold values as follows. The threshold value is set to $1 \in R$. The components of the weight vector which correspond to the selected outputs of the D_tree are set to 1 and all others are set to 0. The complete network structure is illustrated in figure 3.

Adjustment of the network's weights (W_i's) and thresholds (T_i's) takes place incrementally starting from the lowest layer (consisting of only the root of the D_tree) and proceeding layer by layer towards the leaf units of the D_tree. This adjustment is based on a collection of input-output examples (the training set) which is a typical sample set of a target mapping M as earlier explained. The training set is presented a number of times and at the n-th iteration the W's and T's of all the units at the n-th layer are determined in parallel but independently of each other. The rationale of this adaptation process can be briefly explained as follows.

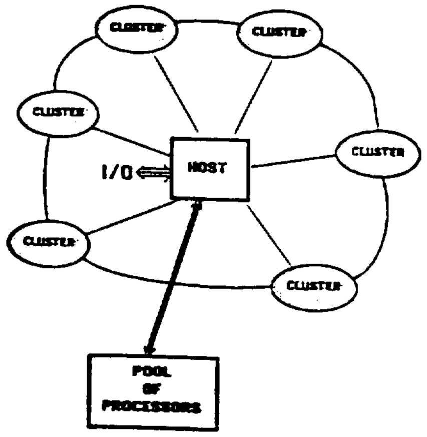
After the training set has been presented n-1 times all units from the root of the D_tree up to (and including) layer n-1 have been assigned W and T values defining the hyperplanes of an intermediate partition. Thus the activity space of each unit in the n-th layer is determined. During the n-th presentation each unit in the n-th layer is allowed to "observe" only those instances of the training set (that is those of the presented examples) which are pertinent to its own activity space. On the basis of the "observed set" then each unit determines a hyperplane which further partitions the region corresponding to its activity space in two parts. This hyperplane must be the one which is "most useful" in discriminating among instances of different classes. The estimator used in this model to measure the goodness of a hyperplane is the entropy. In [Kout88] we describe in detail a constrained optimization process for the hyperplane selection which is based on the optimization of entropy. The exact optimization of the entropy is a laborious process which is plaqued by the dimensionality of the input space and for this reason the constrained optimization process further employs heuristics based on discriminant analysis techniques. The greediest computation of those required in the constrained optimization process is the inversion of a matrix and therefore its complexity is that of the matrix inversion.

The hyperplanes are the means by which decision information is internally represented in the network. In keeping the system's internal representations as simple as possible (as few hyperplanes as possible), it is expected that these representations capture the structure or regularities which are possibly exhibited by the input space. The reason simply is that if the system operates properly by having acquired a minimal amount of information, then necessarily that information must be of high quality, reflecting the essential characteristics (e.g. structure, regularities) of the input space.

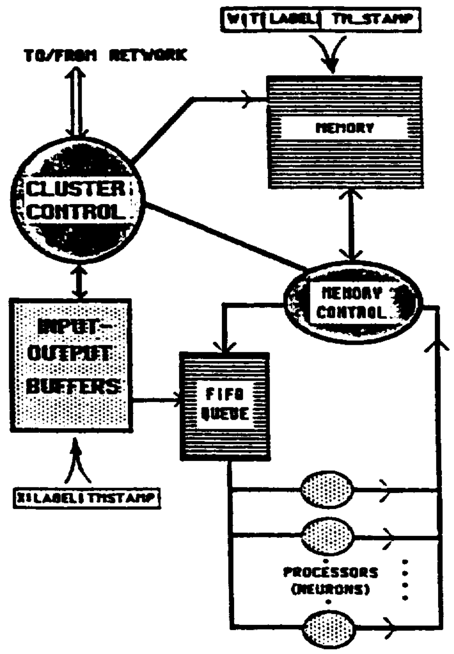
4. The model's implementation

In the following we outline our approach towards the development of a machine architecture. The net model (Athena) we have developed has a particularly simple structure. In contrast with other models the number of the required interconnections is small and therefore it is possible to implement with current VLSI technology. The acyclic nature of the model makes the use of parallel Data Flow architectures [Wats82], [Denn80], [Arv83], particularly efficient for its implementation. Such an architecture is presented in this section.

The overall architecture consists of a host I/O processor and a number of clusters interconnected in a ring structure as shown in figure 4. This machine is intended to simulate concurrently a large number of networks of the Athena type described earlier in this paper. We will hereafter refer to the stored representation of an Athena type of



Overal structure
Figure 4



Architecture of a single cluster
Figure 5

network simulated on this machine as a *virtual network*. The D_tree of a virtual network is implemented in one or more of the clusters. The functions of the units of the output layer of a virtual network are performed by the host. All inputs are entered in the system through the host. When an input intended for a particular virtual network is entered, the host attaches to the input a timestamp and a label identifying the virtual network and distributes it to the clusters.

The basic architecture of a single cluster is shown in figure 5. It consists of a memory module which is accessible by a memory controller. The controller communicates with a pool of independent (but identical) processors through a unidirectional loop link. These processors are dedicated to the cluster and will be called hereafter loop processors. The memory stores representations of many networks of the Athena type. A network is represented in the memory as a linked list of records. Each record represents a unit of a network. Each record therefore consists of a number of fields containing information about the weight vector and threshold associated with the corresponding unit, as well two "destination fields" identifying the records corresponding to the children units. Additionally, there is a label field and a timestamp field associated with each record. The label field identifies the virtual network to which the record belongs. The use of the latter two fields will be explained in the following.

The loop processors of a cluster are all identical and can operate independently of each other in parallel. Their purpose is to simulate the functions of the units in an Athena network at the normal processing mode. Therefore these processors are only required to perform the scalar product of two vectors and the threshold function.

An independent pool of processors is utilized exclusively for matrix and other computations needed during the training phase. These processors are not associated with any particular cluster. These processors are used to run the training algorithm by which the weight vector and threshold are determined for each unit of a virtual network. The computed weights and thresholds are then communicated to the appropriate cluster where they are assigned as values to the appropriate fields of the corresponding records. These computations are considerably more sophisticated compared to the scalar product and threshold function needed during the normal processing mode. Furthermore, training takes place only once for each virtual net. For this reason it would be inefficient to require that the loop processors within each cluster be sophisticated enough to carry out the computations required by the training algorithm. The simple threshold test is the only function needed during the normal processing phase of a virtual net and it is therefore the most frequently executed one.

* In this machine the development of a virtual network is carried out as follows:

During the first presentation of the training set, the weight vector and threshold of the root of the virtual tree must be determined. One processor of the pool is assigned to gather the presented examples and compute the W and T values. These values are then sent over to a cluster's memory controller which forms a record and stores it in the memory. The destination fields of this record contain the addresses of two other records with null values for their weight, threshold and destination fields. The labels of the new records are set to the same value as that of their parent.

A partially developed virtual net is recursively expanded as follows. During a new presentation of the training set each example input is processed by the existing partial virtual net. In this way it is determined how the training set should be partitioned for the computation of the weights and thresholds of the leafs (records with null fields) of the current virtual net. Each subset of the partitioned training set is then assigned to a processor and the computed W and T values are subsequently passed on to the corresponding records (in the appropriate cluster). If a certain subset contains examples of only one class then no W or T values are computed and the corresponding record's fields remain null except that the record is marked with a label identifying that class and further, no children are linked to it.

As shown in figure 4 the basic architecture consists of a number of clusters which are interconnected by a unidirectional ring communication network. All clusters are architecturally the same having the structure shown in figure 5. In this architecture a virtual network can expand over more than one clusters. If the available capacity of the memory module in a cluster does not allow completion of the development of a virtual network in that cluster, then free space is seeked in other clusters and development of the virtual network continues there. The process of developing the virtual network within a cluster is carried out as described above. Suppose now that the W and T values for a record are computed but no children can be linked to this record due to unavailability of memory space. Then the W and T values just computed, are not assigned to this record. Instead, a request seeking free space in another cluster is submitted over the communication network interconnecting the clusters. When one is found, a new record is formed in the new cluster, the computed W and T values are assigned to it and the record in the original cluster obtains a pointer address (link) to it. The original record maintains null W and T fields and it is only used for binding purposes. Such records used for binding the parts of a decision D_tree which is distributed over a number of clusters will be called "dummy" records. The function of dummy records is not any part of determining an output, it rather is to designate the fact that the signal they receive must be communicated outside the environment of the cluster.

* During the normal processing mode the function of a virtual net is simulated as follows :

Activation of a unit is simulated by "firing" its corresponding record in the following way. The complete information contained in the record representing that unit is duplicated in a *packet* which is then sent to the processing FIFO queue. If there is an idle processing unit, it gets the packet removing it from the queue. This processing unit then computes the scalar product and performs the threshold comparison. The result of this computation determines which of the "destination" units should be activated. The processing unit finds the address of the corresponding destination record from the destination fields of the packet it has acquired, and simply sends this address to the memory controller. When the controller receives the address, it fires the appropriate destination record, that is, it forms a new packet corresponding to the destination record and sends it to the processing queue repeating the above cycle.

Firing of a dummy record does not produce a packet to be send to the processing queue. An address is only obtained and sent either to the host or to another cluster via the cluster's local control which interfaces the cluster with the (ring) communication network. If the dummy packet represents an output of the corresponding D_tree (exit), then the address is sent to the host. If it represents binding (continuation of the D_tree) to another cluster, then the address is sent to that cluster. In the destination cluster then, the binded root will be fired continuing the process there.

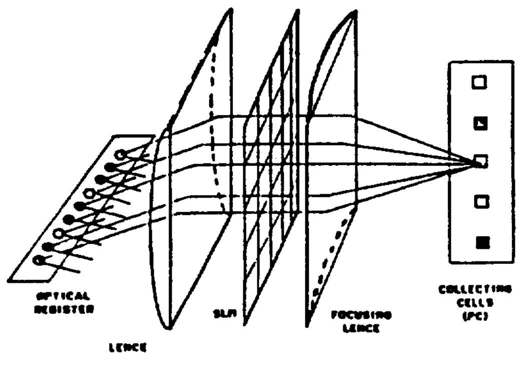
Each packet send to the queue carries a timestamp and a label field identifying the decision tree (D_tree) to which it belongs. Such a label field is needed for the following reasons: In the whole system a lot of virtual networks are stored. When in the normal processing mode the host receives an input intended for a particular virtual network, the host attaches to the input the label of the corresponding decision tree and a timestamp and distributes the labeled input to the clusters. Each cluster containing a part of that decision tree stores a copy of this input in the input buffer. When a processing unit receives a packet, it obtains the input intended for use with this packet, by accessing the input buffer for that input whose label matches that of the received packet. The computations implied by each conceptual decision tree stored in the local memory, are carried out in parallel without affecting each other. Thus parallelism is only limited by the physical delay characteristics and the number of the processing elements. Not only inputs for different decision trees can be processed concurrently, but different inputs intended for the same decision tree can also be processed in parallel as follows:

Each input received by the host is timestamped in addition to being labeled before being distributed to the clusters. The input buffer at each cluster may contain many different inputs intended for the same decision tree, but each input carries a different timestamp. As earlier mentioned, when a processing unit processes an input, the result is a destination address which is sent to the memory controller. The destination address also carries the timestamp of the processed input. The same timestamp is also passed on to the copy of the destination packet which is sent back to the processing units. When the new packet is processed, the input buffer will be accessed for an input whose label and timestamp matches those of the packet being processed. Therefore, if a certain input is used in processing a packet, the same input exactly will be used in the processing of the corresponding destination packet.

The timestamps must also be used when firing a dummy record. If the dummy record represents binding then its timestamp along with the address of the corresponding (binded) root of the virtual network's subtree is obtained and submitted to the appropriate cluster. If the dummy record represents an output leaf unit of the virtual network then its label and timestamp are submitted to the host which furnishes the output.

5. Use of optical processing

We now consider the processing units within a cluster. These units are required to compute the scalar product of two vectors and the threshold function. For this purpose the loop processors can be implemented as simple pipeline processors using conventional VLSI technology. However, the functional requirements for these



Optical processing structure from [Casasent88]
Figure 6

processors are such that advantage can be taken of optical processing structures already proposed. The structure of figure 6 has been proposed by Casasent [Casasent88] for the simultaneous computation of the scalar products of a vector X and a set of vectors Y_j, j=1,2,3, ... The vector X is held in an optical register consisting of an array of laser LED's which is shown in figure 6 positioned on the focus line of the first lens. The vectors Y_j are stored in the spatial light modulator (SLM). The beams on a horizontal plane coming out of the first lens constitute a copy of the optical register. One copy is therefore available for each row of the SLM and so all the partial products X_iY_{ji} are available in parallel, encoded by the intensity of the beams coming out of the SLM. All beams coming out of a single row of SLM cells are focused by the second lens on a single point. An array of photosensitive cells (one for each row of the SLM) is positioned on the focus line of the lens. If r weight vectors can be stored in the SLM, then with this structure r scalar products can be computed concurrently and extremely fast.

The output of each PC cell must be compared against a corresponding threshold. This comparison can automatically be performed if a bias proportional to the threshold is used on the PC cell. The reason that full advantage of this structure can be taken for implementing Athena is that it is fit for the kind of computations required in Athena, that is, all the units in a D_tree require the same input vector for the scalar product computations. However, if this structure is employed to implement the processing elements of a cluster, then firing one packet at a time would not be the most efficient way. Rather than firing a single packet at a time, a whole virtual subtree of packets is fired as as follows. When the memory controller receives the address of a packet to be fired, it extracts that packet and r-1 of its successors in a Breadth First manner, where r is the capacity of the SLM in any of the (optical) processing units. All of these packets are sent as a group to the processing queue. An idle processing unit gets this group. The W vectors are stored in the SLM and the corresponding thresholds are used as biases on the PC cells. The appropriate input X is obtained from the input buffer after matching the group's label and timestamp to those of X. All of the outputs of the subtree corresponding to the group are computed in parallel. A binary search through these outputs yields the single output which the subtree produces when processing X and which determines which packet should consequently be fired. The processing unit then sends the address of the packet to be fired along with the virtual network label and the timestamp (these are the same as those of the group processed) to the memory controller. If the packet specified to the controller by this address is a dummy packet, the controller submits the received address to the network via the local control unit. Otherwise a new subtree will then be fired having this packet as a root. The advantage of using this optical processing structure over conventional processors is obvious.

6. Conclusion

In this paper we reviewed the operation of an earlier introduced neural net model. Targets of this model are general classification tasks and more specificaly the automatic development of representations for the necessary classification rules or class descriptions on the basis of example information. The purpose of this paper was to outline a machine architecture that can implement this model. It was shown here that it is possible to efficiently implement the model in current technology using Data Flow architecture principles. For the purposes of real time applications the ability to physically implement is a particular advantage of this model over other existing ones. It was also shown here that advantage can be taken of optical processing structures proposed by other researchers. A particular optical structure which meets the model's computational needs can be effectively embedded to further enhance significantly the efficiency and throughput of the basic machine.

References

Arvind et.al., "The Tagged Token Data Flow Architecture", MIT, August 1983

D. Casasent and E. Baranoski, "Directed graph for adaptive organization and learning of a knowledge base", in Applied Optics Vol.27, No. 3, February 1988

J. B. Dennis, "Data Flow Supercomputers", in Computer Vol.13, No.11, Nov. 1980

R. Duda and P. Hart "Pattern Recognition and Scene Analysis", John Wiley & Sons 1973

R. A. Fisher "The use of multiple measurements in taxonomic problems", Ann.Eugenics, 7, Part II, 179-188 (1936); also in Contributions to Mathematical Statistics (John Wiley, New York, 1950)

J. Ghosh and K. Hwang, "Critical Issues in Mapping Neural Networks on Message-Passing Multicomputers", in Proc. 15th Annual Internat. Symposium on Computer Architecture May 1988

J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", Proc. Natl. Acad. Sci. USA, 79, 2554-2558 1982

D.A. Huffman "The Synthesis of Linear Sequential Coding Networks", Proc. Symposium on Information Theory, London, 1955

D.A. Huffman "A method for the construction of minimum redundancy codes", Proc. IRE, September 1952

T. Kohonen "Self Organization and Associative Memory", Springer-Verlag 1987 (second edition)
C. Koutsougeras and C.A. Papachristou, "A Neural Network Model for Propositional Logic Functions", Computational Intelligence Internat. Conf. Elsevier-North Holland, Amsterdam, September 1988

C. Koutsougeras and C.A. Papachristou, "Training of a Neural Network Model for Pattern Classification Based on an Entropy Measure", in Proc. IEEE Internat. Conf. on Neural Networks (ICNN '88), New York: IEEE, July 1988.

C. Koutsougeras and C.A. Papachristou, "A Neural Network Model for Discrete Mappings", in Proc. IEEE Internat. Conf. on Languages for Automation (LFA '88), New York: IEEE, October 1988.

D. Psaltis and M. Neifeld, "The Emergence of Generalization in Networks with Constrained Representations", in Proc. IEEE Internat. Conf. on Neural Networks (ICNN '88), New York: IEEE, July 1988.

D.E. Rumelhart, G.E. Hinton and R.J. Williams "Learning Internal Representations by Error Propagation", in Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol1: Foundations, D.E. Rumelhart and J.L. McClelland (editors), MIT Press, Cambridge, MA (1986)

I. Watson and J. Gurd, "A Prototype Data Flow Computer with Token Labeling", in AFIPS Conf. Proc. Nat'l Comput. Conf. June 1979

I. Watson and J. Gurd, "A practical Data Flow Computer", in Computer, Vol.15, No.2, Feb. 1982