In *WCNN'93 World Congress on Neural Networks*, vol. I, pp. 613-616, 1993.

A Generalizing Adaptive Discriminant Network

Tony Martinez, Cory Barker and Christophe Giraud-Carrier
Computer Science Department, Brigham Young University, Provo, Utah 84602

To appear in *Proceedings of the 1993 World Congress on Neural Networks*

Abstract

This paper overviews the AA1 (Adaptive Algorithm 1) model of ASOCS the (Adaptive Self-Organizing Concurrent Systems) approach. It also presents promising empirical generalization results of AA1 with actual data. AA1 is a topologically dynamic network which grows to fit the problem being learned. AA1 generalizes in a self-organizing fashion to a network which seeks to find features which discriminate between concepts. Convergence to a training set is both guaranteed and bounded linearly in time.

Introduction

ASOCS (Adaptive Self-Organizing Concurrent Systems) models support efficient computation through self-organized learning and parallel execution. Learning is done through the incremental presentation of rules and/or examples. The model incorporates rules into an adaptive logic network in a parallel and self organizing fashion. The system itself resolves inconsistencies and generalizes as the rules are presented.

In processing mode, ASOCS supports fully parallel execution on actual inputs according to the learned rules. The adaptive logic network acts as a parallel hardware circuit during execution, mapping *n*-input boolean vectors into *m*-output boolean vectors, in a combinatoric fashion.

The overall philosophy of ASOCS follows the high level goals of current neural network models. However, the mechanisms of learning and execution vary significantly. The ASOCS logic network is topologically dynamic with the network growing to efficiently fit the specific application. Current ASOCS models are based on digital nodes. ASOCS also supports use of symbolic learning mechanisms, thus combining the parallelism and distributed nature of connectionist computing, with the potential power of AI symbolic learning. A proof of concept ASOCS chip has been developed [1].

ASOCS models have two significant advantages over other learning models; a) they are guaranteed to learn any arbitrary set of legal rules, and b) learning time is both fast and bounded (linear with the depth of the network, and logarithmic with the number of nodes). Current research seeks to improve the generalization ability of ASOCS models.

A number of formal ASOCS models have been proposed to date [3,4,5]. This paper overviews the AA1 (Adaptive Algorithm 1) ASOCS model and presents results of recent empirical studies. See [2,3] for more detailed and formal presentations of the AA1 model.

AA1 Learning Algorithm

In ASOCS, the knowledge base to be learned is represented by a set of rules called *instances*. Instances are Boolean rules or examples defining what a given output should be when confronted with a given Boolean input vector. The following are examples of instances:

AB -> Z (i)
B'C -> Z (ii)
AC' -> Z' (iii)

Instance (i) says that if A and B are both true, then Z should be set to true; instance (iii) says that if A is true and C is false, then Z should be set to false. If the output of an instance is negated it is called a *negative* instance; otherwise it is a *positive* instance. This is known as the *polarity* of an instance. Instances (i) and (ii) are positive and instance (iii) is negative. Two instances with the same polarity are said to be *concordant* with respect to each other, while two instances of opposite polarity are *discordant.* Instances (i) and (ii) are concordant; instances (ii) and (iii) are discordant.

AA1 is an incremental learner. Precedence is given to newer instances. The idea is that general rules are learned first, and then followed by exceptions. (However, results in this paper show that the model can support non-ordered training set data as well). As new instances are learned, they are maintained as an instance set which is maintained consistent and minimal. Consistency means that no two discordant instances of the instance set can both be matched by some state of the environment. Due to its complexity, complete minimality is not a reasonable goal. However, within computational constraints, simplification is sought both for parsimony and generalization.

The ASOCS is said to *fulfill* the instance set if, when any positive instance is matched, the network outputs a positive value (or 1), and when any negative instance is matched, the network outputs a negative value (or 0). In this paper, only single-output ASOCS are considered. The system is naturally extended to multiple-outputs [3].

The two main components of ASOCS AA1 are the *adaption unit (AU)* and the logic network of nodes. Figure 1 shows the overall system structure.

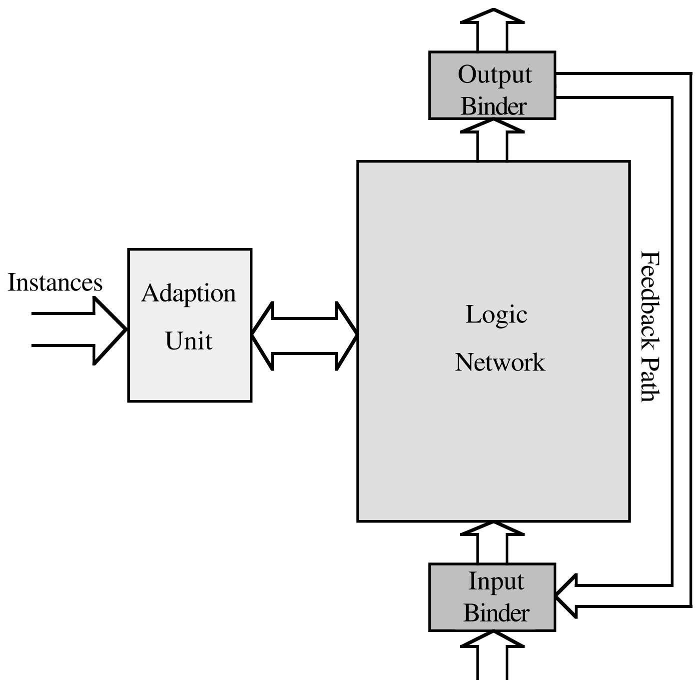

Figure 1 — Overall System Structure

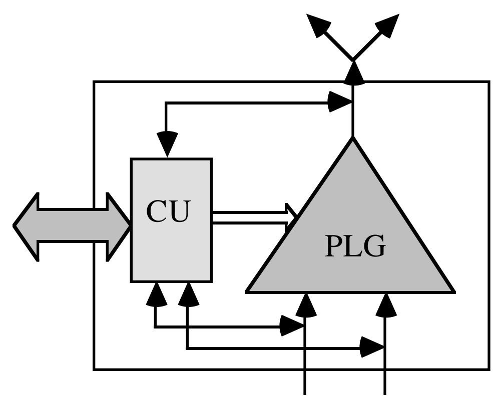

Figure 2 — Network Node Details

The AU guides the logic network through adaptation by broadcasting commands to the network nodes. Each node (Figure 2) has two basic parts: the control unit and a two-input programmable logic gate (PLG). In execution mode the PLG is the only active element and the network functions like a hardwired logic circuit; in learning mode the control unit changes the function of the PLG, sends messages to and receives messages from neighbor nodes, and changes its interconnections to neighbor nodes. The PLG can be programmed to compute any one of the 16 boolean functions of 2 inputs.

When a new instance (NI) is input to the system it is stored and maintained in the *instance table (IT)* of the AU. The IT is a two-column table, column P stores positive instances and N stores negative instances. Each node contains a node table (NT) which corresponds to the IT stored in the AU, having both a positive and a negative column. The NI is then *presented* (together with the corresponding IT cell location) to the network by placing the values of the NI at the network inputs, including don't care values. Each PLG does the 3-state version of its current logic function and thus each node outputs either 0, 1, or "?". This value is then stored in the NT at the cell corresponding to where the NI was placed in the IT.

The network learns by discriminating positive instances from negative instances. If a cell in the node table is asserted (1 or 0), that node *discriminates* the instance represented by that cell from all discordant instances which have the opposite assertion. For example, Let *K* be a node whose NT is given by the leftmost node table of Figure 3. Consider the first cell of the P column whose value is 0, and let *I* be the corresponding instance. If *K* outputs 0, there is no way to tell if instance *I* has been matched or if it is the instance corresponding to the first cell of the N column. However, if *K* outputs 1, clearly *I* is not matched. Hence, since *K* outputs 1 for the instances corresponding to the second and third cells of the N column, *K* is able to discriminate between *I* and these two instances. Discrimination is the key factor in AA1.

There are four classes of nodes depending on the state of their node tables. Examples are shown from left to right in figure 3.

• *discriminant node*: a node that discriminates at least one positive instance from one negative instance.

• *non-discriminant node*: a node that does not discriminate at least one positive instance from one negative instance

• *complete discriminant node*: a node that discriminates every positive instance from every negative instance.

• *one-sided discriminant node*: a discriminant node that always asserts one value for either all positive or all negative instances, and the opposite value for at least one discordant instance.

P	N	P	N	P	N	P	N
0	0	1	1	1	0	?	1
1	1		1	1	0	1	1
?	1				0	0	1

Figure 3 — The Four Classes of Nodes in AA1

Note that for the network to fulfill the instance set, the top node (output of the network) must be a complete discriminant node. The goal in AA1 is to always maintain a complete discriminant node as the network's top node. However, when a new instance is input, it may contradict previous information, and when broadcast to the network, the top node will no longer be complete discriminant, thus requiring network modification.

When a NI is input, the AU first makes the IT consistent and parsimonious. This is a simple bounded algorithm [3] which is not detailed here. The result of this process is a *delete-list* that contains the instances that are to be removed from the current instance set and an *add-list* that contains the instances that have been modified because of inconsistency with the NI. The NI is also a member of the add-list. Either or both of these lists may be empty. The add-list is empty if and only if the current instance set already fulfills the NI. Instances to be deleted are handled simply by broadcasting the cell location of the instance to the network, which is then eradicated from all node tables. Instances on the add-list are each presented to the network with each node updating its node table. At this point, the basic information has been broadcast to the network. However, if the NI was not already fulfilled by the network, the top node will no longer be complete-discriminant and network modification will be required to create a new one.

At this point, the old top node in the network correctly discriminates the instance set with the exception of the NI. If we can create a new node which discriminates the NI from all old instances of the opposite polarity (one-sided discriminant node), this new node can be combined with the old top node to create a new complete discriminant node. This new one-sided discriminant node (NOSD) is created by combining nodes in the network such that their union creates the NOSD. These nodes are gathered through the process of *node selection.*

Since each node table has been updated after broadcast of the delete and add lists, each node can calculate a local metric on its ability to discriminate the NI from the old instances (OI) in its node table. This quantity is called the *discriminant count.* If a node outputs ? for the NI, it does no discrimination (i.e. discriminant count is 0). Otherwise the discriminant count is the total number of discordant OI's (i.e. number of OI cells which are asserted opposite of the assertion of the NI cell). Because of the hierarchical structure of the AA1 network, the best discriminating nodes can be gathered in logarithmic time. These nodes are then combined, with each combination creating a node which discriminates the union of the discriminations done by the nodes from which it was created.

If this union still does not discriminate every OI from the NI, it is necessary to execute *node creation*. Nodes are then created (under direction of the AU) which can discriminate any remaining OI's from the NI. These nodes are combined with the previously selected nodes to create a proper NOSD. This NOSD node is then combined with the old top node and the network is correct.

Finally, any nodes which no longer aid in discrimination (i.e. nodes which are non-discriminant) will delete themselves from the network.

We now summarize the algorithm. If both the add-list and the delete-list are empty, then no changes are required and the next instance to be learned can be presented; otherwise, AA1 proceeds as follows:

1. For each instance *i* in the delete-list
   broadcast the index of *i* in IT to the network and delete *i* from IT in the AU
   each node empties the corresponding cell of its NT
2. For each instance *i* in the add-list
   place *i* in first empty cell of IT
   present *i* to the network
   each node places its output in the corresponding cell of its NT
   if the network's output is discordant or ? then
      do node selection and combination
      if some old instances remain non-discriminated do node addition and combination
3. Combine the old top node with the newly created one-sided discriminant node
4. Do Self-Deletion

Note that only the NI requires full execution of step 2. Modified instances in the add-list are still fulfilled by the network and will therefore never require network modification.

Empirical Results

The AA1 algorithm was tested against a number of common machine learning test databases. In each case a number of learning sequences were done using different training and test sets. AA1 can always converge 100% on a consistent training set. Both consistent and inconsistent (noisy/probabilistic) examples were tried. Congressional voting data contains many missing values. The Hepatitis problem is a typically difficult database due to both inconsistent and missing values. The following table summarizes the generalization results on actual test sets.

Congressional Voting Data	92.4%
Monk 1	98.6%
Monk 2	82.6%
Monk 3 (with noise)	91.2%
Monk 3 (without noise)	99.1%
Hepatitis	70.3%

Conclusion

Empirical testing suggests promise using the basic AA1 learning algorithm. Current research seeks to improve the generalization abilities of AA1. Other research includes overcoming AA1 memory requirements and extending the model to handle multistate and analog input.

Bibliography

1. Chang, J. and Vidal, J. J., Inferencing in Hardware, *Proceedings of the MCC-University Research Symposium*, Austin, TX, (July 1987).
2. Giraud-Carrier, C., and Martinez, T., A Proof of Correctness for AA1, *submitted.*
3. Martinez, T. R. and Vidal, J.J., Adaptive Parallel Logic Networks, *Journal of Parallel and Distributed Computing*, **vol. 5**, 26-58, 1988.
4. Martinez, T. R. and Campbell, D. M., A Self-Adjusting Dynamic Logic Module, *Journal of Parallel and Distributed Computing*, **vol. 11**, no.4, pp. 303-313, 1991.

5. Martinez, T. R. and Campbell, D. M., A Self-Organizing Binary Decision Tree for Incrementally Defined Rule Based Systems, *IEEE Transactions on System, Man, and Cybernetics,* **vol. 21**, No. 5, pp. 1231-1238, 1991.