AA1*: A DYNAMIC INCREMENTAL NETWORK THAT LEARNS BY DISCRIMINATION

Christophe Giraud-Carrier
Department of Computer Science
University of Bristol
Bristol, BS8 1TR, U.K.

Tony Martinez
Department of Computer Science
Brigham Young University
Provo, UT 84602, U.S.A.

Abstract

An incremental learning algorithm for a special class of self-organising, dynamic networks is presented. Learning is effected by adapting both the function performed by the nodes and the overall network topology, so that the network grows (or shrinks) over time to fit the problem. Convergence is guaranteed on any arbitrary Boolean dataset and empirical generalisation results demonstrate promise.

Introduction

Most connectionist models are based on static networks of computing nodes, where learning is effected by changing the weights of the connections between nodes and/or by modifying the nodes' functions. Adaptive Self-Organising Concurrent Systems (ASOCS) [5] are a class of dynamic networks that learn by adapting not only the function performed by their nodes, but also their overall topology. Hence, these networks grow (or shrink) over time to fit the problem.

ASOCS have two basic modes of operation: learning and execution. In execution mode, the system functions as a logic network. It receives inputs and produces an output as the data flow asynchronously and in parallel through the network. In learning mode, the system is incrementally presented a set of Boolean instances. As each instance is presented, the network first executes and compares its computed output with that of the training instance. If the two are identical, no change is made and the next training instance is presented, otherwise, the network adapts to learn the new instance, while preserving consistency with previously acquired knowledge.

Several ASOCS learning algorithms have been developed. They include Adaptive Algorithm (AA) 1 [10], AA2 [8] and AA3 [9]. AA2 inductively constructs a logic circuit consistent with the training instances. Though its convergence is evident, AA2 does not generalise to previously unseen instances. AA3 inductively constructs a binary decision tree. A formal proof of AA3's convergence is in [1]. However, AA3's generalisation is little more than memorisation in that the predicted output of any previously unseen instance is that of the first training instance. In most cases, this bias is too strong, dependent upon ordering of the training instances, and thus scarcely better than merely guessing. AA1 learns by discrimination and constructs a network in which knowledge is distributed across all of the nodes. AA1 has also been shown to converge on any arbitrary Boolean instance

$$\begin{array}{llcll} A & B & \rightarrow & Z & (1) \\ B' & C & \rightarrow & Z & (2) \\ A & C' & \rightarrow & Z' & (3) \end{array}$$

Figure 1: Sample Instances

set [3]. However, unlike AA2 and AA3, empirical studies show that AA1 also meaningfully generalises to previously unseen instances [7].

This paper presents AA1*, an extension to AA1 grounded in the *Occam's Razor Principle*. AA1* focuses on minimising network's growth to achieve greater predictive accuracy with smaller networks. The memory requirement at each node is also reduced by one third, and first-order features are available for discrimination during Node Selection. Results on several datasets demonstrate the validity of the extensions.

AA1* Learning

Only an overview of AA1* is possible here. Emphasis is placed on the differences with AA1. Further details and background on AA1 may be found in [3, 5, 10].

Instances

An *instance* is a conjunction of Boolean input values together with the (also Boolean) output value it implies. By convention, V is used to denote the fact that "(variable) V is true or positive", and V' to denote "V is false or negative". Fig. 1 contains three sample instances. The left-hand side of an instance needs not contain an input value for each input variable. Missing input variables are treated as though their value was *don't-care* and can thus be replaced by either true or false without affecting the output value. Formally, if I is an instance in which no value is specified for some input variable V, then I represents both instances obtained from I by adding V and V', respectively to the left-hand side of I. In Fig. 1, if A, B and C are the only three input variables, then instance (1) represents both instances $ABC \rightarrow Z$ and $ABC' \rightarrow Z$. In other words, instance (1) means that when A and B are both true, then Z must become true, regardless of the values of any other input variables. Hence, instances can be viewed as if-then rules.

The *polarity* of an instance is the value of its implied output. Instances (1) and (2) of Fig. 1 are positive instances, while instance (3) is a negative instance. If two instances have the same polarity, then they are said to be *concordant* with respect to each other. Otherwise, they are *discordant*.

Nodes

Nodes are the basic architectural units of AA1*. However, their structure and connectivity differ from that of classical connectionist models. In AA1*,

- Each node has exactly two inputs and a node's output is the AND or OR of its (possibly inverted) inputs. Logical functions are extended in a natural way to ac-

P	N
?	1
0	0
	1

Figure 2: Sample NT

commodate missing input values denoted by ? (e.g., 0 AND ? implies 0, 0 OR ? implies ?).

- Each node records output information about previous training instances in its *node table* (NT).
- Each node's output is sent to 0 or more other nodes' input on weightless connections.
- The overall network structure is a rooted graph and the output of the root or *top node* is the network's output.

The NTs are ignored during execution. During learning, they are used to control discrimination as discussed below. An NT has two columns, P and N, holding the node's output values for each positive and negative training instance, respectively. There is, of course, a consistent one-to-one correspondence between training instances and NT cells. Fig. 2 shows a sample NT.

In AA1, input variables are generally not nodes. Rather, they combine in pairs and become inputs to nodes. Only when an odd number of input variables are recruited, does AA1 produce a node for one of the input variables. That node is simply constructed by having both inputs be the selected variable and setting the function to AND. Unlike AA1, AA1* produces one such node for each input variable. The motivation for this extension is given later.

Discrimination

If a cell in a node's NT contains the value 0 or 1, then that node *discriminates* the corresponding instance from all discordant instances whose cells contain the opposite value. For example, assume that the NT of Fig. 2 belongs to some node K and consider the second cell of the P column, whose corresponding instance is I. If K outputs 0, there is no way to tell whether it is I that is matched or the instance corresponding to the second cell of the N column. However, if K outputs 1, then clearly I is not matched. But K outputs 1 for the instances corresponding to the first and third cells of the N column. Hence, K discriminates I from these two instances. Clearly, cells that contain ? cannot participate in discrimination.

A node that discriminates every positive instance from every negative instance is a *complete discriminant node*. A complete discriminant node thus asserts one value (0 or 1) for all negative instances and the opposite value for all positive instances. A node that asserts one value for either all positive or all negative instances, and the opposite value for at least one discordant instance is a *one-sided discriminant node*. Given an instance I and a set S of discordant instances from which I must be discriminated, a node N's

```
(1) Compute add-list & delete-list by
    maintaining current instance set
    consistent and minimal
(2) For each instance in delete-list
      Empty corresponding cell in NTs
(3) For each instance in add-list
   (a) Do Execution with NT update
   (b) If (output != target value)
      (i) Do Node Selection
     (ii) If (more nodes are needed)
            Do Node Addition
    (iii) Do Node Combination
     (iv) Perform Self-Deletion
```

Figure 3: AA1* Learning Algorithm

discriminant count (with respect to I and S), denoted by $DC_N(I, S)$, is the number of instances in S that N discriminates from I.

In AA1, information about which instances a node discriminates from another given instance is stored in the NT as a binary D column. AA1* computes D on-the-fly, thereby reducing the memory requirement at each node by one third, while not increasing computation time.

Learning Algorithm

As training instances are presented to the network, AA1* incrementally learns to discriminate each new training instance from all previously observed discordant instances. Essentially, the network constantly adapts so that its top node is complete discriminant with value 1 in the P column. Pseudo-code for the learning algorithm is given in Fig. 3. It is executed as each new training instance is presented.

An instance set is *inconsistent* if it contains two discordant instances whose conjunctive parts could be true simultaneously. Step (1) remedies such inconsistencies by removing one of the offending instances or adding variables sufficient to discriminate between them. Details of which instance to remove and how to add discriminant variables are in [5, 6]. Informally, an instance set is *minimal* if it can not be made smaller (i.e., less instances and/or variables) while retaining its functionality. Complete minimisation is not practical, but step (1) achieves partial minimisation through pairwise comparison of the training instance and instances in the current instance set [5, 6]. Step (1) produces a delete-list and an add-list containing the instances to remove from and add to the current instance set to keep it consistent and somewhat minimal.

Step (2) processes the delete-list by removing instances from the current instance set and causing each node to empty the corresponding cells in its NT. Step (3) adds the instances of the add-list to the current instance set and presents each in turn to the network. Note that the maintenance of a *current instance set* in steps (2) and (3) is necessary only to step

(1). The stored instance set is ignored during execution and network modification.

Step (3) is the heart of the learning algorithm. It is performed for each instance in the add-list. In step(3)(a), the network executes based on the values of the instance's input variables. Each node applies its stored function to its two inputs and passes the result to all of its successors. Since execution is with update, each node also stores its output value for the new instance in a corresponding cell of its NT. The output of the top node is then checked in step (3)(b). If it is concordant with that of the new instance, then no changes need be made and the next instance is processed. If, on the other hand, it is different from that of the new instance (i.e., either discordant or ?), then the current top node is no longer complete discriminant. Adjustments are made to the network in steps (3)(b)(i)-(iii) to create a new complete discriminant top node. Since the current top node is complete discriminant for all instances except the new one, the idea is to construct a one-sided discriminant node that discriminates the new instance from all discordant instances and to combine it with the current top node to obtain the new desired complete discriminant top node. Steps (3)(b)(i)-(iii) are detailed in the next section. In step (3)(b)(iv), the network undergoes a phase of self-deletion in which nodes that are no longer needed (e.g., non-discriminant nodes) are removed from the network. Self-deletion increases parsimony. Details of the various self-deletion mechanisms are beyond the scope of this paper. They may be found in [5, 10].

Network Modification

In steps (3)(b)(i)-(iii), AA1* recruits nodes that together suffice to produce a one-sided discriminant node that discriminates the new instance, NI, from the set, D, of all discordant instances. If N and M are nodes that discriminate NI from $D_1 \subseteq D$ and $D_2 \subseteq D$, respectively, then it is a result of [3] that the node obtained by combining N and M discriminates NI from $D_1 \cup D_2$. This result extends naturally to show that, given an appropriate set of nodes, step (3)(b)(iii) produces a new network whose top node is complete discriminant. Hence, the goal of the system is to compute a set $\{N_1, N_2, ..., N_n\}$ of nodes such that, if for each $1 \leq i \leq n$, N_i discriminates NI from $D_i \subseteq D$, then $\bigcup_{i=1}^{n} D_i = D$.

The set $\{N_i\}$ is naturally constructed by selecting existing nodes (step (3)(b)(i)), and adding new nodes if necessary (step (3)(b)(ii)). In AA1, Node Selection successively recruits the node that discriminates the new instance from the largest number of remaining, non-discriminated, discordant instances. That is, the ith node selected, N_i, is such that $DC_{N_i}(NI, D - \bigcup_{k=1}^{i-1} D_k) = \max_{N \in Net}\{DC_N(NI, D - \bigcup_{k=1}^{i-1} D_k)\}$. Selection goes on until the value of i such that $\max_{N \in Net}\{DC_N(NI, D - \bigcup_{k=1}^{i-1} D_k)\} = 0$, is reached (i.e., no further discrimination can be achieved by existing nodes). The set of selected nodes produced by this procedure is clearly not guaranteed to be minimal. Hence, the resulting network may be larger than it needs be. Following the *Occam's Razor Principle*, AA1* extends Node Selection as follows. Let W_1 be the first node selected by AA1 and S_1 be the set of subsequently selected nodes. Let W_2 be the first node selected by AA1 when W_1 is left out of the competition and S_2 be the set of subsequently selected nodes. In general, let W_i be the first node selected by AA1 when W_1, ...,W_{i-1} are left out of the competition and S_i be the set of subsequently selected nodes. AA1* chooses the smallest of the S_i. In the current implementation, AA1* computes only S_1 through S_5 as empirical studies show

Table 1: Simulation Results

Application	AA1		AA1*		
	PA	**IR**	**PA**	**IR**	**HC**
Monk1	95.5	0.63	98.4	0.43	1.8
Monk2	73.4	1.52	71.4	1.45	9
Monk3	92.0	0.64	91.3	0.61	1.5
hepatitis	72.8	0.88	71.8	0.88	2.1
shuttle-l-c	100	1.33	100	1.40	0
shuttle-exp	98.7	0.18	98.0	0.17	0.3
Averages	88.7	0.86	88.5	0.82	2.5

that in most cases S_2 or S_3 are the smallest. This extension increases learning time only by a constant multiplicative factor. Execution time is reduced as the networks are smaller.

Another way in which AA1* improves on AA1 is by creating individual nodes for input variables, as mentioned above. These nodes represent first-order features that may be helpful in early discrimination. In AA1, candidate nodes for selection are combinations of input variables, i.e., higher-order features. Recruitment of single variables is delayed until Node Addition. However, given a new instance which did not cause the combination of two inputs, the logical combination of these inputs is not guaranteed to have a better discriminant count than one of the input alone. Hence, smaller networks may be obtained by recruiting existing individual input variables during Node Selection.

Once selection is completed, if some instances in D remain to be discriminated from, then Node Addition takes place. A set of candidate input variables is computed and a sufficient subset is extracted by applying the aforementioned procedure used for Node Selection by AA1. The process is completed by Node Combination. Details are in [5, 10].

Simulation Results

Several datasets from [11] were used to test AA1 and AA1*'s generalisation. In both cases, generalisation is possible due to the bias towards parsimony, manifest in three ways. First, if the current network correctly predicts the value of a training instance, no unnecessary changes are made. Second, during node selection, the system uses heuristics to minimise the number of recruited nodes. And third, after a new instance has caused network modification, unnecessary nodes are deleted. Empirical results confirm these findings as shown in Table 1. The PA column shows predictive accuracy on the test set. The IR column gives the ratio of the number of nodes in the final network to the number of training instances and serves as a measure of parsimony. The HC column under AA1* shows the average number of times, in one pass over the training set, that a smaller set of nodes was found using the above extension.

All non-Boolean inputs have their values translated into their equivalent binary form. Only one form of self-deletion, namely, complete discriminant deletion [10], is implemented in the simulations. The dataset *shuttle-exp* consists of the complete set of 278 instances resulting from expanding the 15 rules of the shuttle-landing-control (*shuttle-l-c*) dataset.

Reported results for *hepatitis* and *shuttle-exp* were gathered using 10-way cross validation. Results for the Monk problems used the provided training and test sets, and results for *shuttle-l-c* were obtained by training on the 15 rules and testing on the 278 corresponding instances. The outcome of the algorithms being order-dependent, several random orderings of the training set were used and averaged.

In all cases (except *shuttle-l-c* which uses only 1 extra node), AA1* produces smaller networks. Surprisingly, predictive accuracy seems to suffer some, except in the case of *Monk1*. These preliminary results demonstrate the potential of the proposed extensions, since on average, network size is reduced by 4.7% and predictive accuracy only by 0.2%. They also show that much can still be done as HC is small for some of the datasets. One would expect AA1* to produce a network of same size for both *shuttle-l-c* and *shuttle-exp*, and given the simple expressions of *Monk1* and *Monk3*, smaller networks could be anticipated. Also, HC is comparatively high for *hepatitis* and *Monk2* while the network's size is only slightly reduced and predictive accuracy not improved. Both *hepatitis* and *Monk2* are known to be difficult problems.

Conclusion

This paper presents an extension to the AA1 learning algorithm for ASOCS. The hierarchical nature of the topology guarantees that execution is essentially O(log$n$), where n is the number of nodes in the network. Convergence on any arbitrary Boolean dataset is guaranteed and empirical generalisation results demonstrate promise. The algorithm's ability to incrementally handle both examples and rules, and to integrate symbolic and connectionist representations, are relevant to current efforts in neural network and machine learning research [2, 4, 12, 13].

AA1* extends the early AA1 [10]. More informed heuristics may be considered, as well as their impact on noisy datasets. Insisting on the instance set being consistent may not always be appropriate as nature is replete with inconsistencies. Moreover, the current handling of inconsistencies introduces chronology as a strong learning bias, and makes the algorithm order-dependent. Ways to remove or appropriately use the order dependency may be considered. Introducing explicit forms of generalisation (e.g., *dropping-condition* rule), is an interesting avenue to pursue. Of particular concern is the interaction between such generalisations and the requirement for consistency. It still remains unclear how to do away with, or effectively replace, the memory-expensive node tables used by AA1*. The algorithm PDLA of [2] is one attempt at overcoming some of AA1*'s limitations.

References

[1] C. Barker and T.R. Martinez. Proof of correctness for ASOCS AA3 networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(3):503-510, 1994.

[2] C. Giraud-Carrier and T. Martinez. Using precepts to augment training set learning. In *Proc. ANNES'93*, 46-51.

[3] C. Giraud-Carrier and T. Martinez. Analysis of the convergence and generalization of AA1. *JPDC*, 1994 (to appear).

[4] L.O. Hall and S.G. Romaniuk. A hybrid connectionist, symbolic learning system. In *Proc. AAAI'90*, 783-788.

[5] T.R. Martinez. *Adaptive Self-Organizing Networks.* PhD thesis, University of California, Los Angeles, 1986.

[6] T.R. Martinez. Consistency and generalization in incrementally trained connectionist networks. In *Proc. International Symposium on Circuits and Systems*, 706-709, 1990.

[7] T.R. Martinez, J.C. Barker, C. Giraud-Carrier. A generalizing adaptive discriminant network. In *Proc. WCNN'93*, I:613-616.

[8] T.R. Martinez and D.M. Campbell. A self-adjusting dynamic logic module. *JPDC*, 11(4):303-313, 1991.

[9] T.R. Martinez and D.M. Campbell. A self-organizing binary decision tree for incrementally defined rule based systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1231-1238, 1991.

[10] T.R. Martinez and J.J. Vidal. Adaptive parallel logic networks. *JPDC*, 5(1):26-58, 1988.

[11] P.M. Murphy and D.W. Aha. UCI repository of machine learning databases. University of California, Irvine, Department of Information and Computer Science, 1992.

[12] D. Ourston and R. Mooney. Changing the rules: A comprehensive approach to theory refinement. In *Proc. AAAI'90*, 815-820.

[13] R. Sun. A connectionist model for commensense reasoning incorporating rules and similarities. *Knowledge Acquisition*, 4:293-321, 1992.