1-2000

Supervised adaptive resonance theory and rules

Ah-hwee TAN
Singapore Management University, ahtan@smu.edu.sg

SUPERVISED ADAPTIVE RESONANCE THEORY AND RULES

A.-H. Tan
Kent Ridge Digital Labs
National University of Singapore
Singapore

Supervised Adaptive Resonance Theory is a family of neural networks that performs incremental supervised learning of recognition categories (pattern classes) and multidimensional maps of both binary and analog patterns. This chapter highlights that the supervised ART architecture is compatible with IF-THEN rule-based symbolic representation. Specifically, the knowledge learned by a supervised ART system can be readily translated into rules for interpretation. Similarly, *a priori* domain knowledge in the form of IF-THEN rules can be converted into a supervised ART architecture. Not only does initializing networks with prior knowledge improve predictive accuracy and learning efficiency, the inserted symbolic knowledge can also be refined and enhanced by the supervised ART learning algorithm. By preserving symbolic rule form during learning, the rules extracted from a supervised ART system can be compared directly with the originally inserted rules.

1 Introduction

Supervised Adaptive Resonance Theory is an extension of Adaptive Resonance Theory (ART) to perform incremental supervised learning of recognition categories (pattern classes) and multidimensional maps of both binary and analog patterns. Two classical examples of supervised ART systems are ARTMAP [3, 4] and its bidirectional compressed variant, known as the Adaptive Resonance Associative Map

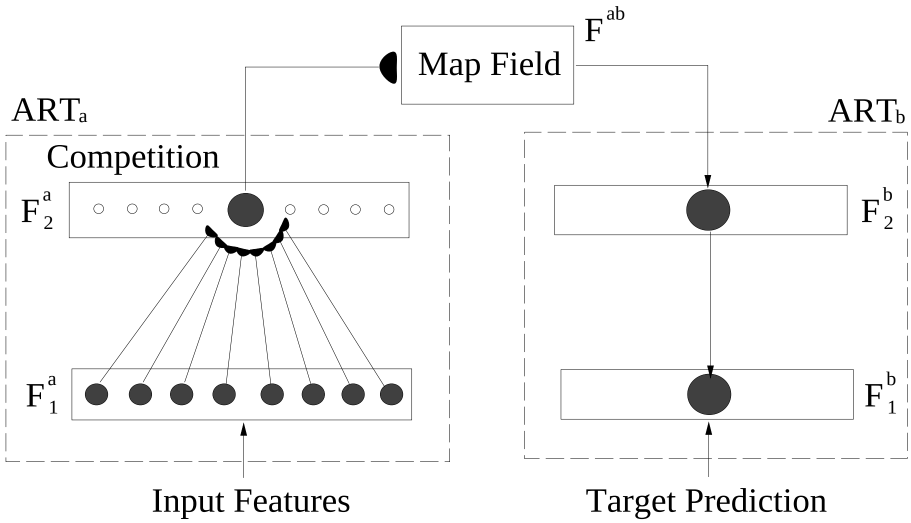

Figure 1: Rules in ARTMAP. Each node in the F_2^a field represents a recognition category of ART$_a$ input patterns. Through the inter-ART map field, each such node is associated to an ART$_b$ category in the F_2^b field, which in turn encodes a prediction. Learned weight vectors, one for each F_2^a node, constitute a set of rules that link antecedents to consequents. The number of rules equals the number of F_2^a nodes that become active during learning.

(ARAM) [15]. Although both ARTMAP and ARAM has architectures that are compatible with rule-based representation, to facilitate discussion, I have used ARTMAP (in particular fuzzy ARTMAP) to illustrate the linkages between supervised ART systems and symbolic knowledge processing.

When performing classification tasks, ARTMAP formulates recognition categories of input patterns, and associates each category with its respective prediction. The knowledge that ARTMAP discovers during learning, is compatible with IF-THEN rule-based representation. Specifically, the recognition categories learned by the F_2^a category nodes are compatible with rules that link antecedents to consequents (Figure 1). At any point during the incremental learning process, the system architecture can be translated into a compact set of rules analyzable by human experts [6].

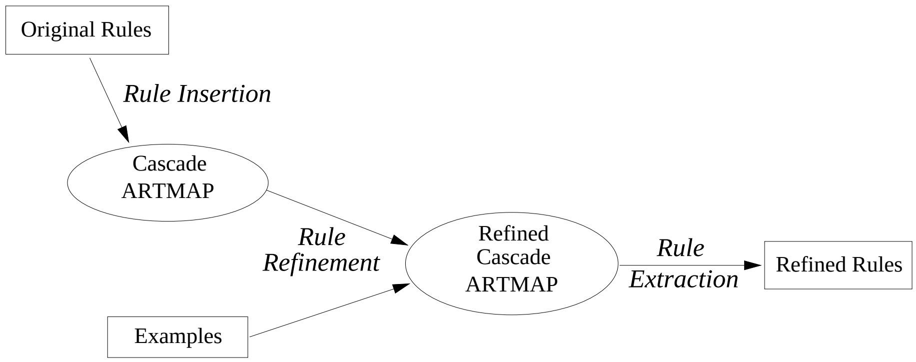

Figure 2: Cascade ARTMAP for symbolic knowledge refinement and evaluation.

On the other hand, rules can be readily inserted into an ARTMAP network that can then be refined by learning from examples. To handle intermediate attributes, Cascade ARTMAP, a generalization of ARTMAP, represents rule cascades of rule-based knowledge explicitly [16]. During ARTMAP learning, new recognition categories (rules) are created dynamically to cover the deficiency of the domain theory. By the self-stabilizing property, learning in Cascade ARTMAP does not wash away existing knowledge and the meanings of units do not shift. This allows preservation of symbolic rules during neural network learning. Using a generalized ARTMAP rule extraction procedure, the final system states can be converted back to a compact set of rules. This enables direct comparison of the original knowledge with the refined rules. Also, each extracted rule is associated with a confidence factor that indicates its importance or usefulness. This allows ranking and evaluation of the extracted knowledge. In all, the Cascade ARTMAP rule insertion, refinement, and extraction procedures form a paradigm for symbolic knowledge refinement and evaluation (Figure 2).

The remaining sections of this chapter are organized as follows. To make this chapter self-contained, section 2 provides a summary of fuzzy ARTMAP. Section 3 motivates the generalization of fuzzy ARTMAP to Cascade ARTMAP and presents the Cascade ARTMAP rule insertion, rule chaining, rule refinement, and rule extraction algorithms. Section 4 illustrates Cascade ARTMAP's performance on a DNA promoter recognition problem. The final section states concluding remarks and highlights future applications.

2 Fuzzy ARTMAP

Fuzzy ARTMAP [3], a generalization of binary ARTMAP [4], learns to classify inputs by a pattern of fuzzy membership values between 0 and 1 indicating the extent to which each feature is present. This generalization is accomplished by replacing the ART 1 modules [2] of the binary ARTMAP system with fuzzy ART modules [5]. Each ARTMAP system includes a pair of Adaptive Resonance Theory modules (ART_a and ART_b) that create stable recognition categories in response to arbitrary sequences of input patterns (Figure 3). An associative learning network and an internal controller link these modules to make the ARTMAP system operate in real time.

2.1 Fuzzy ART

Fuzzy ART [5] incorporates computations from fuzzy set theory into ART systems. By replacing the crisp (nonfuzzy) intersection operator ($\cap$) that describes ART 1 dynamics [2] by the fuzzy AND operator ($\wedge$) of fuzzy set theory, fuzzy ART can learn stable categories in response to either analog or binary patterns.

ART field activity vectors: Each ART system includes a field F_0 of nodes that represent a current input vector; a field F_1 that receives both bottom-up input from F_0 and top-down input from a field F_2 that represents the active code, or category. Vector $\mathbf{I}$ denotes F_0 activity; vector $\mathbf{x}$ denotes F_1 activity; and vector $\mathbf{y}$ denotes F_2 activity.

Weight vector: Associated with each F_2 category node j is a vector $\mathbf{w}_j$ of adaptive weights, or long-term memory (LTM) traces. Initially

$$w_{j1}(0) = \ldots = w_{jM}(0) = 1; \tag{1}$$

then each category is *uncommitted*. After a category codes its first input, it becomes *committed*.

Parameters: A choice parameter $\alpha > 0$, a learning rate parameter $\beta \in [0, 1]$, and a vigilance parameter $\rho \in [0, 1]$ determine fuzzy ART

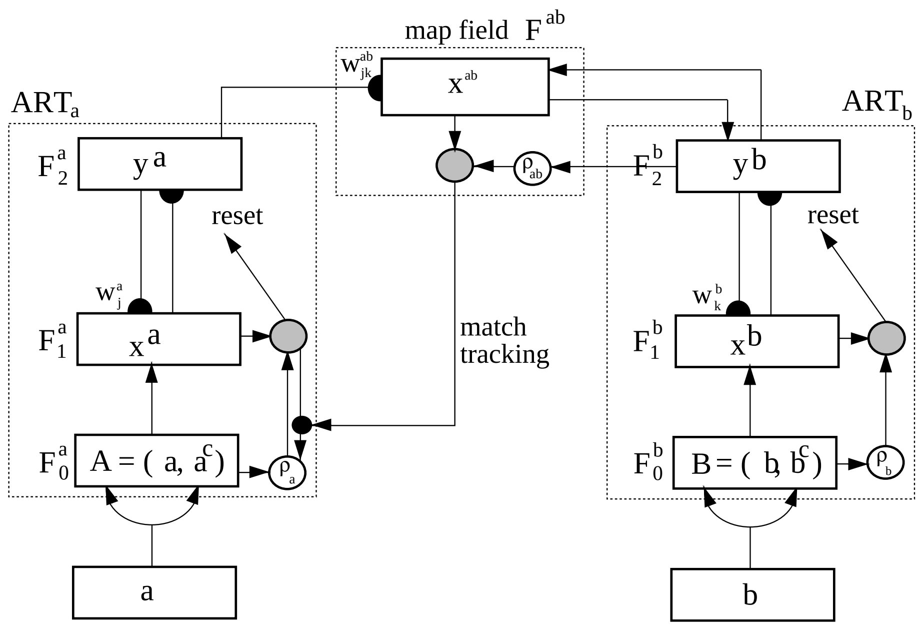

Figure 3: Fuzzy ARTMAP architecture. The ART_a complement coding preprocessor transforms the M_a-vector $\mathbf{a}$ into the $2M_a$-vector $\mathbf{A} = (\mathbf{a}, \mathbf{a}^c)$ at the ART_a field F_0^a. $\mathbf{A}$ is the input vector to the ART_a field F_1^a. Similarly, the input to F_1^b is the $2M_b$-vector $(\mathbf{b}, \mathbf{b}^c)$. When ART_b disconfirms a prediction of ART_a, map field inhibition induces the match tracking process. Match tracking raises the ART_a vigilance (ρ_a) to just above the F_1^a-to-F_0^a match ratio $|\mathbf{x}^a|/|\mathbf{A}|$. This triggers an ART_a search which leads to activation of either an ART_a category that correctly predicts $\mathbf{b}$ or to a previously uncommitted ART_a category node.

dynamics.

Category choice: For each input $\mathbf{I}$ and F_2 node j, the *choice function* T_j is defined by

$$T_j(\mathbf{I}) = \frac{|\mathbf{I} \wedge \mathbf{w}_j|}{\alpha + |\mathbf{w}_j|}, \tag{2}$$

where the fuzzy intersection $\wedge$ (Zadeh, 1965) is defined by

$$(\mathbf{p} \wedge \mathbf{q})_i \equiv \min(p_i, q_i) \tag{3}$$

and where the norm $|\cdot|$ is defined by

$$|\mathbf{p}| \equiv \sum_{i=1}^{M} |p_i|. \tag{4}$$

The system makes a *category choice* when at most one F_2 node can become active at a given time. The index J denotes the chosen category, where

$$T_J = \max\{T_j : j = 1 \ldots N\}. \tag{5}$$

When the J^{th} category is chosen, $y_J = 1$; and $y_j = 0$ for $j \neq J$.

Resonance or reset: *Resonance* occurs if the *match function* $|\mathbf{I} \wedge \mathbf{w}_J|/|\mathbf{I}|$ of the chosen category meets the vigilance criterion:

$$\frac{|\mathbf{I} \wedge \mathbf{w}_J|}{|\mathbf{I}|} \geq \rho. \tag{6}$$

Learning then ensues, as defined below. Otherwise *Mismatch reset* occurs, where the value of the choice function T_J is set to 0 for the duration of the input presentation . The search process continues until a chosen category J satisfies the matching criterion (6).

Learning: Once search ends, the weight vector $\mathbf{w}_J$ learns according to the equation

$$\mathbf{w}_J^{(\text{new})} = \beta(\mathbf{I} \wedge \mathbf{w}_J^{(\text{old})}) + (1 - \beta)\mathbf{w}_J^{(\text{old})}. \tag{7}$$

Fast learning corresponds to setting $\beta = 1$. Using the fast learning and slow recoding option, we set $\beta = 1$ when J is an uncommitted node and take $\beta < 1$ after the category is committed.

Normalization by complement coding: Normalization of fuzzy ART inputs prevents category proliferation. The complement coded $F_0 \rightarrow F_1$ input $\mathbf{I}$ is the 2M-dimensional vector

$$\mathbf{I} = (\mathbf{a}, \mathbf{a}^c) \equiv (a_1, \ldots, a_M, a_1^c, \ldots, a_M^c), \tag{8}$$

where

$$a_i^c \equiv 1 - a_i. \tag{9}$$

A complement coded input is automatically normalized, because

$$|\mathbf{I}| = |(\mathbf{a}, \mathbf{a}^c)| = \sum_{i=1}^{M} a_i + (M - \sum_{i=1}^{M} a_i) = M. \tag{10}$$

With complement coding, the initial condition

$$w_{j1}(0) = \ldots = w_{j,2M}(0) = 1, \tag{11}$$

replaces the fuzzy ART initial condition (1).

2.2 ARTMAP Prediction and Search

Fuzzy ARTMAP incorporates two fuzzy ART modules ART_a and ART_b that are linked together via an inter-ART map field F^{ab}. The map field forms predictive associations between categories and realizes the ARTMAP *match tracking rule*.

ART_a and ART_b: Inputs to ART_a and ART_b are complement coded. For ART_a, $\mathbf{I} = \mathbf{A} = (\mathbf{a}, \mathbf{a}^c)$; and for ART_b, $\mathbf{I} = \mathbf{B} = (\mathbf{b}, \mathbf{b}^c)$ (Figure 3). For ART_a, $\mathbf{x}^a$ denotes the F_1^a output vector; $\mathbf{y}^a$ denotes the F_2^a output vector; and $\mathbf{w}_j^a$ denotes the j^{th} ART_a weight vector. For ART_b, $\mathbf{x}^b$ denotes the F_1^b output vector; $\mathbf{y}^b$ denotes the F_2^b output vector; and $\mathbf{w}_k^b$ denotes the k^{th} ART_b weight vector. For the map field, $\mathbf{x}^{ab}$ denotes the F^{ab} output vector, and $\mathbf{w}_j^{ab}$ denotes the weight vector from the j^{th} F_2^a node to F^{ab}.

Map field activation: The map field F^{ab} receives input from either or both of the ART_a or ART_b category fields. A chosen F_2^a node J sends input to the map field F^{ab} via the weights $\mathbf{w}_J^{ab}$. An active F_2^b node K sends input to F^{ab} via one–to-one pathways between F_2^b and F^{ab}. If both ART_a and ART_b are active, then F^{ab} remains active only if ART$_a$ predicts the same category as ART$_b$. The F^{ab} output vector $\mathbf{x}^{ab}$ obeys:

$$\mathbf{x}^{ab} = \begin{cases} \mathbf{y}^b \wedge \mathbf{w}_J^{ab} & \text{if the J}^{th}\ F_2^a \text{ node is active and } F_2^b \text{ is active} \\ \mathbf{w}_J^{ab} & \text{if the J}^{th}\ F_2^a \text{ node is active and } F_2^b \text{ is inactive} \\ \mathbf{y}^b & \text{if } F_2^a \text{ is inactive and } F_2^b \text{ is active} \\ \mathbf{0} & \text{if } F_2^a \text{ is inactive and } F_2^b \text{ is inactive.} \end{cases} \tag{12}$$

By (12), $\mathbf{x}^{ab} = \mathbf{0}$ if $\mathbf{y}^b$ fails to confirm the map field prediction made by $\mathbf{w}_J^{ab}$. Such a mismatch event triggers an ART$_a$ search for a better category, as follows.

Match tracking: At the start of each input presentation ART$_a$ vigilance ρ_a equals a baseline vigilance parameter $\overline{\rho_a}$. When a predictive error occurs, match tracking raises ART$_a$ vigilance just enough to trigger a search for a new F_2^a coding node. ARTMAP detects a predictive error when

$$|\mathbf{x}^{ab}| < \rho_{ab}|\mathbf{y}^b|, \tag{13}$$

where ρ_{ab} is the map field vigilance parameter. A signal from the map field to the ART$_a$ orienting subsystem causes ρ_a to "track the F_1^a match." That is, ρ_a increases until it is slightly higher than the F_1^a match value $|\mathbf{A} \wedge \mathbf{w}_J^a||\mathbf{A}|^{-1}$. Then, since ART$_a$ fails to meet the matching criterion, the search for another F_2^a node begins.

Map field learning: Weights w_{jk}^{ab} in $F_2^a \to F^{ab}$ paths initially satisfy

$$w_{jk}^{ab}(0) = 1. \tag{14}$$

During resonance with the ART$_a$ category J active, $\mathbf{w}_J^{ab}$ approaches the map field vector $\mathbf{x}^{ab}$ as in (7). With fast learning, once J learns to predict the ART$_b$ category K, that association is permanent; i.e., $w_{JK}^{ab} = 1$ for all time.

3 Cascade ARTMAP

Prior knowledge of a problem domain could help a neural network in learning to solve the problem. Specifically, pre-existing symbolic rules can be used to initialize a neural network architecture before learning. Not only can rule insertion improve network learning efficiency, it also serves to provide knowledge that is not captured by training cases or that cannot be easily learned by a neural network, and thus improves the system predictive performance. In addition, incomplete or partially correct symbolic knowledge can be refined or enhanced through neural network learning algorithms. Rule insertion and refinement in neural networks therefore automates symbolic knowledge enhancement and repair.

A popular approach to rule insertion and refinement uses rules to initialize the architecture of a multi-layer neural network and refines the network using a backpropagation algorithm [8, 9, 20]. One major problem of using backpropagation networks (BP) to refine rule-based knowledge is the preservation of symbolic knowledge. Under the weight tuning process of a backpropagation algorithm, symbolic rules quickly lose their original meanings. In fact, large shifts in the meanings of hidden units can occur as a result of training [19].

Another major limitation of the BP approach is that the initial rule base has to be roughly complete, or else the initial network architecture created may not be sufficiently rich for dealing with the problem domain. As the standard backpropagation algorithm is not able to create additional nodes or connections dynamically during learning, a network initialized by a small set of rules may even have a lower chance of eventually learning the task. This problem was noted and partially solved by Lacher *et. al.*, who used virtual rules to create potential connections for learning [9]. However, in general, it is difficult to decide beforehand the virtual rules or connections desired. Tresp, Hollatz, and Ahmad [21] employed a learning algorithm that allowed creation of basis functions during learning. However, as their model only represents rules associating input attributes to output predictions, the network is not general enough to deal with rule-based domain theories involving intermediate attributes and rule chaining.

3.1 Rule Cascade Representation

Note that ARTMAP also handles a class of IF-THEN rules that map a set of input attributes directly to a disjoint set of output attributes. Symbolic rule-based knowledge, on the other hand, often involves *rule cascades* and *intermediate attributes*. A set of rules is said to form a *rule cascade* when a consequent of a rule also serves as an antecedent of another rule. Such attributes that have dual roles are usually called *intermediate attributes* in contrast to *input attributes* that only serve as antecedents and *output attributes* that only serve as consequents. Through intermediate attributes, firing of a rule may lead to the firing of another rule at a later stage of an inferencing process. Intermediate attributes and rule cascades are useful for feature abstraction and task decomposition so that only a small set of simple rules is needed at each level of the cascade.

The proposed solution to representing rule cascades here is Cascade ARTMAP that uses ARTMAP architecture but generalizes one-step prediction to multi-step inferencing. Cascade ARTMAP unifies the ARTMAP input attribute field F_1^a and output attribute field F_1^b in the sense that both F_1^a and F_1^b contain the input, output, and intermediate attributes. Consider the two rules below that form a simple two-level rule cascade:

Rule 1: IF A and B THEN C
Rule 2: IF C and D THEN E

where A, B, and D are input attributes; C is an intermediate attribute; and E is an output attribute. All attributes (A, B, C, D, and E) are represented in both F_1^a and F_1^b (Figure 4). For Rule 1, an F_2^a category node is used to encode A and B, and is associated to an F_2^b node that predicts C. Likewise for Rule 2, an F_2^a node is used to encode C and D, and is associated to an F_2^b node predicting E. By unifying the input field (F_1^a) and the output field (F_1^b) of ARTMAP, several desired features of symbolic processing are obtained. Besides that rule insertion can be readily performed in Cascade ARTMAP, rule chaining and inferencing can also be performed as in production systems.

3.2 Rule Insertion

As the knowledge structure of Cascade ARTMAP is compatible with rule-based knowledge representation, IF-THEN rules can be readily translated into the recognition categories of a Cascade ARTMAP system. Initializing a Cascade ARTMAP network with pre-existing rules before learning serves to set up the global solution structure. This

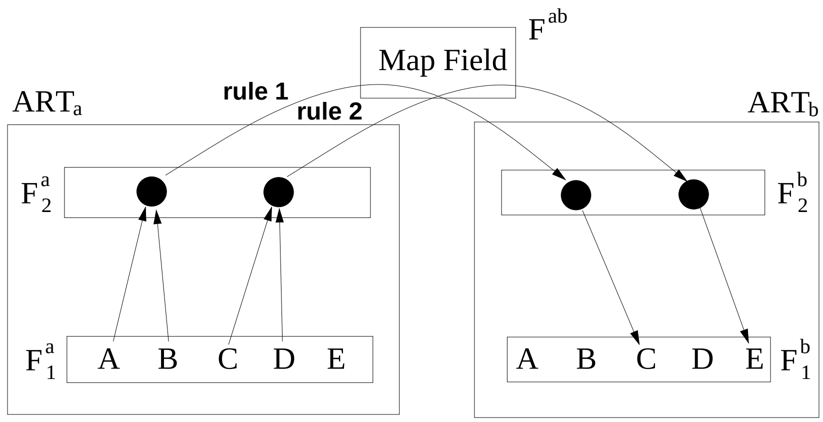

Figure 4: Cascade ARTMAP representation of the sample rule cascade.

helps to improve learning efficiency and predictive accuracy. Without rule insertion, Cascade ARTMAP performance reduces to that of fuzzy ARTMAP.

Rule insertion proceeds in two phases. The first phase parses all rules for attribute names to set up a *symbol table* in which each attribute in the rules has a unique entry. Based on the symbol table, the second phase translates each rule into two 2M-dimensional vectors $\mathbf{A}$ and $\mathbf{B}$, where M is the total number of attributes in the symbol table, as inputs to the Cascade ARTMAP ART$_a$ and ART$_b$ modules. Given a rule of the following format,

IF	$x_1, x_2, \ldots, x_m, \neg\bar{x}_1, \neg\bar{x}_2, \ldots, \neg\bar{x}_{\bar{m}}$
THEN	$y_1, y_2, \ldots, y_n, \neg\bar{y}_1, \neg\bar{y}_2, \ldots, \neg\bar{y}_{\bar{n}}$

where $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$ are positive attributes, and $\bar{x}_1, \ldots, \bar{x}_{\bar{m}}$ and $\bar{y}_1, \ldots, \bar{y}_{\bar{n}}$ preceded by the logical NOT operator $\neg$ are negative attributes, the algorithm derives the pair of vectors

$$\mathbf{A} = (\mathbf{a}, \mathbf{a}^c) \quad \text{and} \quad \mathbf{B} = (\mathbf{b}, \mathbf{b}^c) \tag{15}$$

such that for each index $j = 1, \ldots, M$,

$$(a_j, a_j^c) = \begin{cases} (1,0) & \text{if } e_j = x_i \text{ for some } i \in \{1, \ldots, m\} \\ (0,1) & \text{if } e_j = \bar{x}_i \text{ for some } i \in \{1, \ldots, \bar{m}\} \\ (0,0) & \text{otherwise} \end{cases} \tag{16}$$

and

$$(b_j, b_j^c) = \begin{cases} (1,0) & \text{if } e_j = y_i \text{ for some } i \in \{1,\ldots,n\} \\ (0,1) & \text{if } e_j = \bar{y}_i \text{ for some } i \in \{1,\ldots,\bar{n}\} \\ (0,0) & \text{otherwise} \end{cases} \tag{17}$$

where e_j is the jth attribute in the symbol table. Note that complement coding is employed above for encoding both the positive and negative attributes. If the rules contain no negative attribute, the complement vectors $\mathbf{a}^c$ and $\mathbf{b}^c$ may be eliminated.

The vector pairs derived from the rules are then used as training patterns to initialize a Cascade ARTMAP network. The network learning and inferencing algorithms will be described in subsequent sections. It suffices to note at this point that each distinct vector $\mathbf{A}$ is translated into a recognition category in ART_a and likewise each distinct vector $\mathbf{B}$ is translated into a recognition category in ART_b. Given a pair of vectors $\mathbf{A}$ and $\mathbf{B}$ derived from a rule, their respective recognition categories are associated through the map field. During network initialization, the vigilance parameters ρ_a and ρ_b are each set to 1 to ensure that only identical attribute vectors are grouped into one recognition category. Contradictory symbolic rules are detected during rule insertion when identical input attribute vectors are associated with distinct output attribute vectors. The detection is achieved through a *perfect mismatch* phenomenon, in which the system tries to raise ART_a vigilance ρ_a above 1 in response to a mismatch in ART_b.

3.3 Rule Chaining and Inferencing

A symbolic production rule system consists of three main components: a *working memory* component, a *rule* or *production* component, and an external inference engine or *interpreter*. The interpreter repeatedly performs a three-phase cycle, consisting of a *match* phase, a *select* phase, and an *execute* phase. In the match phase, the interpreter compares the antecedent set of each rule against the working memory content. Rules with completely matched antecedents are included into a *conflict* set. In the select phase, a single rule is selected from the conflict set using some strategies. If the conflict set is empty, the system halts. Otherwise, in the execute phase, the interpreter adds the consequent(s) of the selected rule to the working memory.

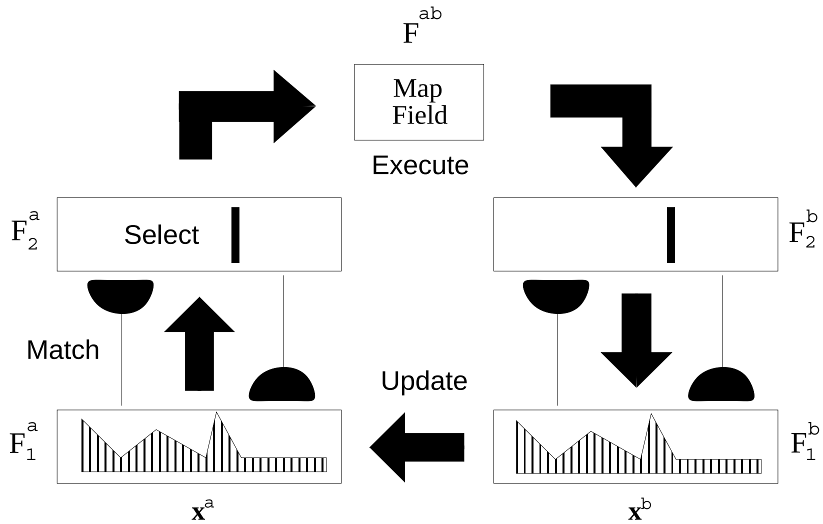

Figure 5: Cascade ARTMAP rule chaining and inferencing: $\mathbf{x}^a$ represents the system's memory state and accumulates attribute values during multi-step inferencing.

In Cascade ARTMAP, the attribute fields F_1^a and F_1^b can be identified as the working memory component (Figure 5). F_1^a maintains the current memory state $\mathbf{x}^a$ and provides the antecedents for condition matching and rule firing. F_1^b stores the next memory state $\mathbf{x}^b$ derived through a rule firing. The rule component is implemented by the two category fields F_2^a and F_2^b, the map field F^{ab}, and their interconnections. The match, select, and execute three-phase cycle is performed without an external interpreter. In the match phase, a choice function T_j^a is computed for each F_2^a category node (rule) based on the memory state $\mathbf{x}^a$. With parallel implementation, the match phase can be performed in a single activation process. The select phase is realized by a winner-take-all interaction among all F_2^a nodes in which the F_2^a node with the largest choice function T_j^a is identified. If the selected node (rule) does not satisfy the ART$_a$ vigilance constraint, the system goes through another round of memory search to select another F_2^a node that satisfies the ART$_a$ vigilance criterion. If no such node exists, the system halts. Otherwise, in the execute phase, the consequent(s) of the selected rule is(are) read out into F_1^b. Note that exact match is not required for a rule to be fired as long as it satisfies the

ART$_a$ vigilance criterion. At the end of the cycle, the new memory state $\mathbf{x}^b$ is used to update $\mathbf{x}^a$ in F_1^a to prepare for the next inferencing cycle. For the sample rule cascade (Figure 4), the input attribute set {A,B,D} activates F_2^a node J_1 that infers C. Through the memory update process, C is fed back from F_1^b to F_1^a. The memory state $\mathbf{x}^b$ which contains {A,B,C,D} then activates J_2 in the next inferencing cycle that derives E.

3.4 Learning and Rule Refinement

Learning in Cascade ARTMAP is more complicated than that in fuzzy ARTMAP as a chain of rule firing is involved in making a prediction. The proposed solution is a backtracking algorithm that identifies all rules (F_2^a nodes) responsible for making a prediction by tracing from the last rule fired. Specifically, if J is the last F_2^a node selected which makes the prediction, the algorithm identifies a *precursor* set $\Psi(J)$ that contains node J and all F_2^a nodes that result in the firing of node J. The backtracking occurs in the direction of $F_2^a \rightarrow F_1^a \rightarrow F_1^b \rightarrow F_2^b \rightarrow F^{ab} \rightarrow F_2^a$. For example in Figure 4, the backtracking algorithm traces from J_2 in F_2^a to its antecedents {C,D} in F_1^a. It then checks that C in F_1^b is an intermediate attribute activated by K_1 in F_2^b, and finally traces to J_1 in F_2^a. The backtracking stops at J_1 as its antecedents are all input attributes. The precursor set of the F_2^a node J_2, $\Psi(J_2)$ is thus evaluated to be $\{J_1, J_2\}$.

If the prediction made by node J is correct, for each F_2^a node (rule) j in the precursor set $\Psi(J)$, the weight vector (antecedent set) $\mathbf{w}_j^a$ is reduced towards its fuzzy intersection with the F_1^a activity vector $\mathbf{x}^a$. In the binary pattern and fast learning case, a fired rule learns to ignore those features that are absent in the current input. This results in a generalization by reducing the number of features the rule attends to.

A more complicated situation occurs when a prediction error is encountered. With a long chain of rule firing, blame assignment can be difficult as it is unclear which rule in the inferencing path causes the error. To handle prediction mismatch, a *mini-match tracking* process raises the ART$_a$ vigilance ρ_a by slightly more than the *minimum* match achieved by the fired rules. Mini-match tracking is equivalent to the parallel match tracking mechanism used in Fusion ARTMAP [1]. This method inhibits the F_2^a node with the minimum match from firing again for the current input. The assumption is that the rule with the worst match is most likely to be the one which causes the prediction error. The system then goes through another round of memory search and inferencing with a higher vigilance until a resonance is achieved.

3.5 Cascade ARTMAP Algorithm

As an on-line real-time system, Cascade ARTMAP needs not separate learning and performance phases, i.e., the system functions in response to the current input environment. Given an F_1^a input vector, Cascade ARTMAP undergoes a series of prediction loops until either an uncommitted F_2^a node is selected (which means no prediction), or a correct prediction is made by a committed F_2^a node (as in fuzzy ARTMAP). In each prediction loop, Cascade ARTMAP accumulates intermediate attribute values through a series of inferencing cycles until one or more output attributes are derived. The Cascade ARTMAP dynamics, as illustrated in Figures 6 and 7, are formalized below. An *A then B* paradigm is used in which the ART$_a$ input vector $\mathbf{A}$ is processed before the ART$_b$ input vector $\mathbf{B}$.

Activity vectors: Let $\mathbf{A}$ and $\mathbf{B}$ denote the F_1^a and F_1^b input vectors respectively. Let

$$\mathbf{x}^a \equiv (\mathbf{x}_i^a, \mathbf{x}_h^a, \mathbf{x}_o^a, \mathbf{x}_i^{ac}, \mathbf{x}_h^{ac}, \mathbf{x}_o^{ac}) \tag{18}$$

and

$$\mathbf{x}^b \equiv (\mathbf{x}_i^b, \mathbf{x}_h^b, \mathbf{x}_o^b, \mathbf{x}_i^{bc}, \mathbf{x}_h^{bc}, \mathbf{x}_o^{bc}) \tag{19}$$

denote the 2M-dimensional F_1^a and F_1^b activity vectors respectively, where $\mathbf{x}_i^a$ and $\mathbf{x}_i^b$ denote the M$_i$-dimensional input attribute vectors; $\mathbf{x}_h^a$ and $\mathbf{x}_h^b$ denote the M$_h$-dimensional intermediate attribute vectors; $\mathbf{x}_o^a$ and $\mathbf{x}_o^b$ denote the M$_o$-dimensional output attribute vectors; and $\mathbf{x}_i^{ac}$, $\mathbf{x}_i^{bc}$, $\mathbf{x}_h^{ac}$, $\mathbf{x}_h^{bc}$, $\mathbf{x}_o^{ac}$, and $\mathbf{x}_o^{bc}$ denote the respective complement attribute vectors. $\mathbf{x}^a$ and $\mathbf{x}^b$ are also known as memory state vectors. Let $\mathbf{y}^a$ and $\mathbf{y}^b$ denote the F_2^a and F_2^b activity vectors respectively. Let $\mathbf{x}^{ab}$ denote the map field F^{ab} activity vector.

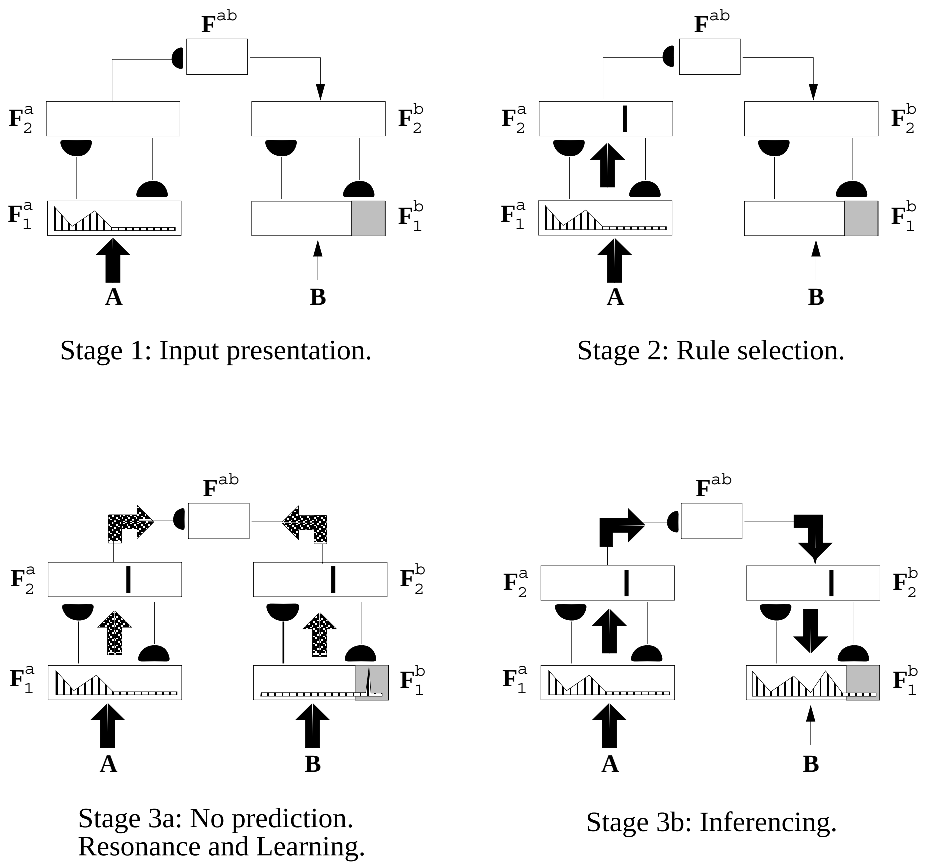

Figure 6: Cascade ARTMAP algorithm stages 1, 2, and 3. The shaded sub-fields of F_1^b represent output attributes.

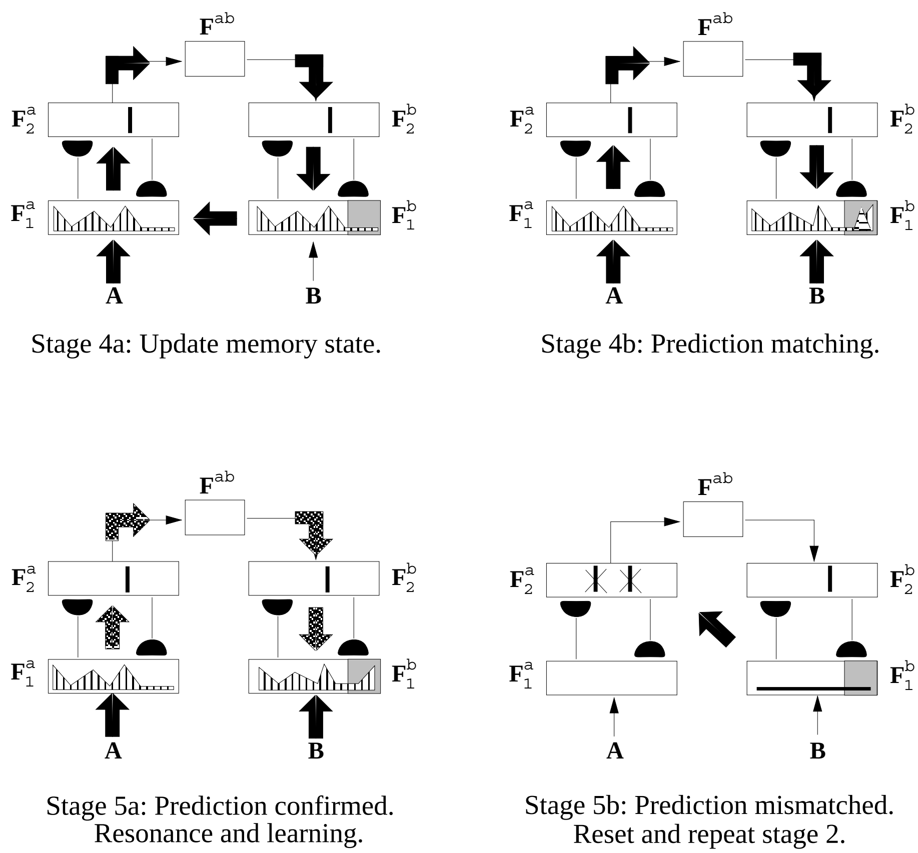

Figure 7: Cascade ARTMAP algorithm stages 4 and 5. The shaded subfields of F_1^b represent output attributes.

Weight vectors: Let $\mathbf{w}_j^a$ and $\mathbf{w}_j^b$ denote the 2M-dimensional weight vectors of the j^{th} category node in F_2^a and F_2^b respectively. Let $\mathbf{w}_j^{ab}$ denote the weight vector from the jth F_2^a node to F^{ab}. Initially, the weight vectors contain all 1's. This implies that all category nodes are uncommitted and all F_2^a nodes are not associated with any prediction.

Scope vectors: Let $\mathbf{S}_j$ denote the 2M-dimensional scope vector of the j^{th} category node in F_2^a. A scope vector identifies the attributes relevant to an F_2^a node and allows a more accurate computation of its match function. For an uncommitted F_2^a node j, the scope vector $\mathbf{S}_j \equiv (\mathbf{s}_j, \mathbf{s}_j)$ is defined by

$$s_{ji} = \begin{cases} 1 & \text{if } i \text{ indexes an input attribute} \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

For an F_2^a node j created by an inserted rule, the scope vector $\mathbf{S}_j \equiv (\mathbf{s}_j, \mathbf{s}_j)$ is defined by

$$s_{ji} = \begin{cases} 1 & \text{if } i \text{ indexes an attribute at a level previous to} \\ & \text{the rule's consequent(s) in the rule cascade} \\ 0 & \text{otherwise.} \end{cases} \tag{21}$$

Parameters: Cascade ARTMAP dynamics are determined by the choice parameters $\alpha_a > 0$ and $\alpha_b > 0$; the learning rates $\beta_a \in [0, 1]$ and $\beta_b \in [0, 1]$; and the vigilance parameters $\rho_a \in [0, 1]$ and $\rho_b \in [0, 1]$. During network initialization, the network learns the patterns derived from rules using $\rho_a = \rho_b = 1$ such that each distinct attribute vector creates a category. During network refinement, the system learns example patterns using $\rho_b = 1$ for output classification and $\rho_a < 1$ to allow input generalization.

Stage 1 (Input presentation): At the beginning of an input presentation, the ART$_a$ vigilance ρ_a equals a baseline vigilance value $\bar{\rho_a}$. F_1^a contains the input vector $\mathbf{A}$:

$$\mathbf{x}^a = \mathbf{A}. \tag{22}$$

Stage 2 (Rule selection): Given the memory state vector $\mathbf{x}^a$, for each F_2^a node j, the choice function T_j^a is defined by

$$T_j^a = \frac{|\mathbf{x}^a \wedge \mathbf{w}_j^a|}{\alpha_a + |\mathbf{w}_j^a|}, \tag{23}$$

where the fuzzy AND operator $\wedge$ is defined by

$$(\mathbf{p} \wedge \mathbf{q})_i \equiv min(p_i, q_i), \tag{24}$$

and the norm $|.|$ is defined by

$$|\mathbf{p}| = \sum_i |p_i| \tag{25}$$

for vectors $\mathbf{p}$ and $\mathbf{q}$. The system is said to make a *category choice* when at most one F_2^a node can become active at a given time. The category choice is indexed at J, where

$$T_J^a = \max\{T_j^a : \text{for all } F_2^a \text{ node } j\}. \tag{26}$$

If more than one T_j^a is maximal, the F_2^a category node j with the smallest index is chosen. In particular, nodes become committed in order $j = 1, 2, 3, \ldots$. When the J^{th} category is chosen, $y_J^a = 1$; and $y_j^a = 0$ for $j \neq J$.

Resonance occurs if the match function m_J^a of the selected node J meets the vigilance criterion:

$$m_J^a = \frac{|\mathbf{x}^a \wedge \mathbf{w}_J^a \wedge \mathbf{S}_J|}{|\mathbf{x}^a \wedge \mathbf{S}_J|} \geq \rho_a, \tag{27}$$

where the generalized fuzzy AND operation $\wedge$ is defined by

$$(\wedge_{j=1}^N \mathbf{p}_j)_i \equiv \min(p_{1i}, \ldots, p_{Ni}), \tag{28}$$

for vectors $\mathbf{p}_1, \ldots, \mathbf{p}_N$, and the norm $|.|$ is defined as in (25). Otherwise mismatch reset occurs in which the value of the choice function T_J^a is set to 0 for the duration of the input presentation to prevent persistent selection of the same category during search. Stage 2 then repeats to select another new index J.

Stage 3a (No prediction): If the selected F_2^a node J has no prediction, i.e.,

$$w_{Jk}^{ab} = 1 \quad \text{for all } F^{ab} \text{ node } k, \tag{29}$$

each F_2^a node j in the precursor set $\Psi(J)$ learns the F_1^a activity pattern according to the equation

$$\mathbf{w}_j^{a(\text{new})} = (1 - \beta_a)\mathbf{w}_j^{a(\text{old})} + \beta_a(\mathbf{x}^a \wedge \mathbf{w}_j^{a(\text{old})}). \tag{30}$$

If the input vector $\mathbf{B}$ is present, a category node is selected in F_2^b as in stage 2. The selected F_2^b node K learns the F_1^b pattern according to the equation

$$\mathbf{w}_K^{b(\text{new})} = (1 - \beta_b)\mathbf{w}_K^{b(\text{old})} + \beta_b(\mathbf{x}^b \wedge \mathbf{w}_K^{b(\text{old})}). \tag{31}$$

The F_2^a category node J is then associated to the F_2^b node K through the inter-ART map field:

$$w_{Jk}^{ab} = \begin{cases} 1 & \text{if } k = K \\ 0 & \text{otherwise.} \end{cases} \tag{32}$$

After which, the system halts.

Stage 3b (Inferencing): If the selected F_2^a node J has learned to make a prediction, i.e., (29) does not hold, its weight vector $\mathbf{w}_J^{ab}$ activates F^{ab}. The F^{ab} activity vector $\mathbf{x}^{ab}$ is defined by

$$\mathbf{x}^{ab} = \mathbf{w}_J^{ab}. \tag{33}$$

Once the map field is active, F_2^b is activated through the 1-to-1 pathway between F^{ab} and F_2^b . For each F_2^b node k, the choice function T_k^b is defined by

$$T_k^b = x_k^{ab}. \tag{34}$$

The system again makes a category choice indexed at K where

$$T_K^b = \max\{T_k^b\text{: for all } F_2^b \text{ node } k\}. \tag{35}$$

When the K^{th} category is chosen, $y_K^b = 1$; and $y_k^b = 0$ for $k \neq K$. The activated F_2^b node K then performs a top-down priming on F_1^b:

$$\mathbf{x}^b = \mathbf{w}_K^b. \tag{36}$$

When F_1^b is activated by a category choice in F_2^b, the termination condition is checked by computing a goal signal g:

$$g = \sum_{i=1}^{M_o}(x_{oi}^b + x_{oi}^{bc}). \tag{37}$$

A conclusion is reached whenever any output attribute is made known, i.e., $g > 0$.

Stage 4a (Update memory state): If a conclusion is not reached, i.e., $g = 0$, the memory state vector $\mathbf{x}^a$ is updated with $\mathbf{x}^b$ by the equation

$$\mathbf{x}^{a(\text{new})} = \mathbf{x}^{a(\text{old})} \vee \mathbf{x}^{b(\text{old})}, \tag{38}$$

where the fuzzy OR operation $\vee$ is defined by

$$(\mathbf{p} \vee \mathbf{q})_i \equiv max(p_i, q_i) \tag{39}$$

for vectors $\mathbf{p}$ and $\mathbf{q}$. The inferencing cycle then repeats from stage 2.

Stage 4b (Prediction matching): If a conclusion is reached, i.e., $g > 0$, the match function m_K^b of the prediction $\mathbf{x}^b$ and the F_1^b input vector $\mathbf{B}$ is computed by

$$m_K^b = \frac{|\mathbf{B} \wedge \mathbf{x}^b|}{|\mathbf{B}|}. \tag{40}$$

Stage 5a (Resonance): If the prediction match satisfies the ART$_b$ vigilance criterion ($m_K^b \geq \rho_b$), resonance occurs. The activated F_2^a and F_2^b nodes learn the template patterns in their respective modules as in (30) and (31) respectively. After learning, the system halts.

Stage 5b (Match tracking): A prediction mismatch triggers a match tracking process. Using mini-match tracking, a node j is identified which has the minimum match function value among all nodes in $\Psi(J)$. The choice function T_j^a of the node j is set to zero during the input presentation. The ART$_a$ vigilance ρ_a is raised to slightly greater than the match achieved by the node j_m:

$$\rho_a^{(\text{new})} = \max\{\rho_a^{(\text{old})}, \min\{m_j^a | j \in \Psi(J)\} + \epsilon\}. \tag{41}$$

Perfect mismatch occurs when the system attempts to increase ρ_a above 1. A perfect match in ART$_a$ ($\rho_a = 1$) with a ART$_b$ mismatch indicates the existence of contradictory knowledge where identical antecedent sets are associated with different consequents. After match tracking, a new prediction loop then repeats from stage 2.

3.6 Rule Extraction

Rules can be derived more readily from an ARTMAP network than from a backpropagation network, in which the roles of hidden units are usually not explicit. In an ARTMAP network, each node in the F_2^a field represents a recognition category of ART_a input patterns. Through the inter-ART map field, each such node is associated to an ART_b category in the F_2^b field, which in turn encodes a prediction. Learned weight vectors, one for each F_2^a node, constitute a set of rules that link antecedents to consequences (Figure 1). The number of rules equals the number of F_2^a nodes that become active during learning.

As large databases typically cause ARTMAP to generate too many rules to be of practical use. The goal of the rule extraction task is to select a small set of highly predictive category nodes and to describe them in a comprehensible form. To evaluate a category node, a *confidence factor* that measures both *usage* and *accuracy* is computed. Removal of low confidence recognition categories created by atypical examples produces smaller networks. Removal of redundant weights in a category node's weight vector reduces the number of antecedents in the corresponding rule.

3.6.1 Rule Pruning

The rule pruning algorithm derives a confidence factor for each F_2^a category node in terms of its usage frequency in a *training* set and its predictive accuracy on a *predicting* set. As Cascade ARTMAP generalizes ARTMAP one-step prediction process to multi-step inferencing, an input pattern makes use of a set of F_2^a category nodes in Cascade ARTMAP in contrast to a single F_2^a node in fuzzy ARTMAP. For evaluating *usage* and *accuracy*, each F_2^a category node j maintains three counters: an encoding counter c_j, that records the number of training set patterns encoded by node j; a predicting counter p_j, that records the number of predicting set patterns predicted by node j; and a success counter s_j, that records the number of predicting set patterns predicted correctly by node j.

For each training set pattern, the encoding counter (c_j) of each F_2^a node j in the precursor set $\Psi(J)$, where J is the last F_2^a node (rule) fired that makes the prediction, is increased by 1. For each predicting set pattern, the predicting counter (p_j) of each F_2^a node j in the precursor set $\Psi(J)$ is increased by 1. If the prediction is correct, the success counter (s_j) of each F_2^a node j in the precursor set $\Psi(J)$ is increased by 1. Based on the encoding, predicting, and success counter values, the *usage* (U_j) and the accuracy (A_j) of an F_2^a node j are computed by

$$U_j = c_j / \max\{c_k\text{: for all } F_2^a \text{ node } k\} \tag{42}$$

and

$$A_j = P_j / \max\{P_k\text{: for all } F_2^a \text{ node } k\}, \tag{43}$$

where P_j, the percent of the predicting set pattern predicted correctly by node j, is computed by

$$P_j = s_j / p_j. \tag{44}$$

U_j and A_j are then used to compute the confidence factor of node j by the equation

$$CF_j = \gamma U_j + (1 - \gamma) A_j, \tag{45}$$

where $\gamma \in [0, 1]$ is a weighting factor. After confidence factors are determined, recognition categories can be pruned from the network using one of following strategies.

Threshold Pruning - This is the simplest type of pruning where the F_2^a nodes with confidence factors below a given threshold τ are removed from the network. A typical setting for τ is 0.5. This method is fast and provides a first cut elimination of unwanted nodes. To avoid over-pruning, it is sometimes useful to specify a minimum number of recognition categories to be preserved in the system.

Local Pruning - Local pruning removes recognition categories one at a time from an ARTMAP network. The baseline system performance on the training and the predicting sets is first determined. Then the algorithm deletes the recognition category with the lowest confidence factor. The category is replaced, however, if its removal degrades system performance on the training and predicting sets.

A variant of the local pruning strategy updates baseline performance each time a category is removed. This option, called *hill-climbing*, gives slightly larger rule sets but better predictive accuracy. A hybrid strategy first prunes ARTMAP using threshold pruning and then applies local pruning on the remaining smaller set of rules.

3.6.2 Antecedent Pruning

During rule extraction, a non-zero weight to an F_2^a category node translates into an antecedent in the corresponding rule. The antecedent pruning procedure calculates an error factor for each antecedent in each rule based on its performance on the training and predicting sets. When a rule (F_2^a node) J makes a prediction error, for each F_2^a node j in the precursor set $\Psi(J)$, each antecedent of the rule j that also appears in the current memory state has its error factor increased in proportion to the smaller of its magnitudes in the rule and in the memory state vector $\mathbf{x}^a$. After the error factor for each antecedent is determined, a local pruning strategy, similar to the one for rules, removes redundant antecedents.

4 DNA Promoter Experiments

Promoters are short nucleotide sequences that occur before genes and serve as binding sites for the enzyme RNA polymerase during gene transcription. Identifying promoters is thus an important step in locating genes in DNA sequences. One major approach to DNA matching or sequence comparison concerns with the alignment of DNA sequences. Sequence alignment is usually performed by computing a match function which rewards matches and penalizes mismatches, insertions, and deletions [22]. This can be done by dynamic programming which can be computationally expensive for multiple sequences. Consensus sequence analysis solves the problem of aligning multiple sequences by identifying functionally important sequence features that are conserved in the DNA sequences. For example, consensus patterns of promoter sequences can be identified at the protein binding sites. Besides statistical methods reported in the biological literature, machine learning and information theoretic techniques are also being used for DNA matching and recognition [7, 10].

The promoter data set [11] used in the Cascade ARTMAP experiments consists of 106 patterns, half of which are positive instances (promoters). Although larger sets of promoter data are available, this version of the promoter data set is used here to allow a direct comparison with the results of the others. Each DNA pattern represents

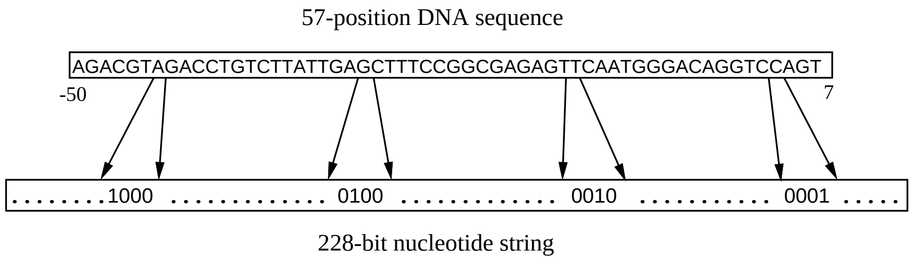

Figure 8: A 57-position DNA sequence. Each position takes one of the four nucleotide values {A,G,T,C}. Using local representation, each DNA sequence is expanded into a 228-bit nucleotide string. This version of 106-case promoter data set, obtained from the UCI machine learning database repository, contains no missing value.

a 57-position window, with the leftmost 50 window positions labeled -50 to -1 and the rightmost seven labeled 1 to 7 (Figure 8). Each position is a nominal feature which takes one of the four nucleotide values {A, G, T, C}. There is no missing feature value. Using local representation, each 57-position pattern is expanded into a 228-bit nucleotide-position string.

The promoter data set and an imperfect domain theory have been used to evaluate a hybrid learning system called Knowledge Based Artificial Neural Network (KBANN) [20]. The imperfect domain theory (Table 1), if requires exact match, only classifies half of the 106 cases correctly. The KBANN theory refinement procedure translates the imperfect theory into a feedforward network, adds links to make the network fully connected between layers, and trains the network using a backpropagation algorithm. Simulation results showed that by incorporating the domain theory, KBANN outperformed many learning/recognition systems, including consensus sequence analysis [13], K Nearest Neighbor (KNN), ID-3 symbolic learning algorithm [14], and backpropagation network trained purely from examples [20] (Table 2).

In Cascade ARTMAP experiments, the first two rules of the domain theory are combined into a single rule:

promoter :- conformation, minus_35, minus_10.

Besides providing a slight improvement in system predictive accuracy, the elimination of attribute *contact* reduces Cascade ARTMAP net-

Table 1: A rule-based theory for classifying promoters. It consists of 14 rules and a total of 83 antecedents. The antecedent notation T@-36 indicates the nucleotide value T in position -36.

promoter	:-	conformation, contact.
contact	:-	minus_35, minus_10.
minus_35	:-	C@-37, T@-36, T@-35, G@-34, A@-33, C@-32.
minus_35	:-	T@-36, T@-35, G@-34, C@-32, A@-31.
minus_35	:-	T@-36, T@-35, G@-34, A@-33, C@-32, A@-31.
minus_35	:-	T@-36, T@-35, G@-34, A@-33, C@-32.
minus_10	:-	T@-14, A@-13, T@-12, A@-11, A@-10, T@-9.
minus_10	:-	T@-13, A@-12, A@-10, T@-8.
minus_10	:-	T@-13, A@-12, T@-11, A@-10, A@-9, T@-8.
minus_10	:-	T@-12, A@-11, T@-7.
conformation	:-	C@-47, A@-46, A@-45, T@-43, T@-42, A@-40, C@-39, G@-22, T@-18, C@-16, G@-8, C@-7, G@-6, C@-5, C@-4, C@-2, C@-1.
conformation	:-	A@-45, A@-44, A@-41.
conformation	:-	A@-49, T@-44, T@-27, A@-22, T@-18, T@-16, G@-15, A@-1.
conformation	:-	A@-45, A@-41, T@-28, T@-27, T@-23, A@-21, A@-20, T@-17, T@-15, T@-4.

Table 2: Performance of fuzzy ARTMAP, Cascade ARTMAP, and Cascade ARTMAP rules on the promoter data set comparing with the symbolic learning algorithm ID-3, the KNN system, consensus sequence analysis, the backpropagation network, the KBANN system, and the NofM rules.

Systems	# Nodes/ Rules	# Antecedent	Error (%)
ID-3	-	-	17.9
KNN (K=3)	105	-	12.3
Consensus Sequences	-	-	11.3
Backpropagation Network	16	-	7.5
Fuzzy ARTMAP	20.6	-	6.5
KBANN	16	-	2.9
Cascade ARTMAP	13+15.9	-	2.0
NofM rules	12	100	3.8
Cascade ARTMAP rules	19.5	53.1	3.0

work complexity and produces simpler rule sets.

Cascade ARTMAP simulation is performed with parameter values $\alpha_a = \alpha_b = 2$ and $\beta_a = \beta_b = 1$, determined empirically. The input patterns are not complement coded as they already have a uniform norm of 57. In each simulation, Cascade ARTMAP is initialized with the domain theory, trained on 96 patterns selected randomly, and tested on the remaining 10 patterns. To use a voting strategy, Cascade ARTMAP is trained in several simulation runs using different orderings of the training set. For each test case, voting across 20 runs yields a final prediction. An averaging technique similar to voting was also used in the KBANN system [20].

Table 2 compares the performance of fuzzy ARTMAP and Cascade ARTMAP, averaged over 20 simulations, with other alternative systems. Among the systems that do not incorporate *a priori* symbolic knowledge, fuzzy ARTMAP (Cascade ARTMAP without rule insertion) achieves the lowest error rate. While the KBANN system and Cascade ARTMAP both obtain significant improvement in predictive performance by incorporating rules, Cascade ARTMAP produces a lower error rate than KBANN. In addition to the 13 inserted rules, an average of 15.9 recognition nodes (rules) are created.

In each simulation, rules are also extracted from the trained Cascade ARTMAP network. Due to the small data set size, confidence factors are computed solely based on *usage*. Threshold pruning with threshold $\tau = 0.01$ is applied, followed by the rule and antecedent pruning procedures using the local pruning strategy. Comparing predictive performance, rules extracted from Cascade ARTMAP are still slightly more accurate than the NofM rules extracted from KBANN [18, 19]. While the Cascade ARTMAP rule sets contain more rules than the NofM rule sets, the number of antecedents is almost half of that of the NofM rule sets.

The promoter rules formulated by Cascade ARTMAP are similar in form to the consensus sequences derived by conventional statistical methods. However, whereas consensus sequences are used with an exact match condition, Cascade ARTMAP rules are based on competitive activation and do not require exact match in antecedents. Through the approximate matching property, the number of nucleotides used to identify a promoter is usually small (at most four in this case). By contrast, the consensus sequences, obtained by noting the positions with the same base in greater than 50% of the promoter patterns [12], used a minimum of twelve nucleotides.

Table 3 shows a sample set of refined promoter rules extracted from Cascade ARTMAP. *Conformation* has been dropped as a condition for promoters, so are the four rules defining it. All the minus_35 and minus_10 rules are preserved, but have been refined to refer to only two salient nucleotide bases. Two new rules for identifying promoters are created, which contain features of *minus_35* and *conformation*. These two rules are believed to compensate for the elimination of *conformation*. Eight non-promoter rules are created. They are slightly more irregular due to the randomness of non-promoters.

The confidence factor attached to each ARTMAP rule provides another dimension for interpreting the rule. By having a confidence factor of 1, the first promoter rule is very frequently used and thus important. It is activated by different combinations of minus_35 and minus_10 rules, each individually does not have a high usage. The two new promoter rules are roughly of equal importance but are not as

Table 3: A set of promoter rules extracted from Cascade ARTMAP. The set consists of 19 rules and a total of 46 antecedents. The real number associated with each rule represents the rule's confidence factor.

promoter (1.00)	:-	minus_35, minus_10.
promoter (0.31)	:-	A@-45, G@-34.
promoter (0.22)	:-	G@-34, T@-25, T@-18.
minus_35 (0.41)	:-	G@-34, C@-32.
minus_35 (0.34)	:-	T@-36, T@-35.
minus_35 (0.22)	:-	A@-33, C@-32.
minus_35 (0.03)	:-	T@-36, C@-32.
minus_10 (0.44)	:-	A@-12, T@-8.
minus_10 (0.31)	:-	A@-13, T@-9.
minus_10 (0.19)	:-	A@-11, T@-7.
minus_10 (0.06)	:-	A@-9, T@-8.
non-promoter (0.19)	:-	A@5.
non-promoter (0.16)	:-	A@-49, C@6, G@.7
non-promoter (0.16)	:-	A@7.
non-promoter (0.16)	:-	T@-23.
non-promoter (0.12)	:-	A@-15, T@1.
non-promoter (0.12)	:-	C@-46, G@-26.
non-promoter (0.06)	:-	T@-34, T@-33, C@-27, T@-26, G@5.
non-promoter (0.03)	:-	A@-45, T@-44, G@-42, T@-29, A@-24, T@-7, A@6, G@7.

heavily used as the first promoter rule. The first three minus_35 rules are more highly utilized than the last minus_35 rule. A similar pattern is observed for the minus_10 rules. The non-promoter rules have lower and less contrasting confidence values. The first four non-promoter rules nevertheless seem slightly more important. The last two non-promoter rules have the least confidences, and could be dropped with little degradation of overall performance.

In contrast, the promoter rules extracted by the NofM algorithm consists of only 9 rules but contains 83 countable antecedents [19]. Moreover, the rules make use of several complex constructs, including NofM, a counting function "nt", addition, subtraction, multiplication, and comparison of real numbers. Also, the NofM rules involve seven nucleotide ambiguity codes, and have already employed a compressed format for representing adjacent nucleotide bases to simplify rules. Comparing complexity, ARTMAP rules are much cleaner and easier to interpret. More importantly, by preserving the symbolic rule form during learning, the extracted rules are identical in form and can be compared directly with the original rules. Furthermore, the use of confidence factors enables ranking of rules. This is particularly important to human experts in analyzing the rules.

5 Conclusion

This chapter has presented supervised Adaptive Resonance Theory systems in the perspective of symbolic knowledge processing. The inherent characteristics of the supervised ART systems, most notably, the fast and incremental learning capabilities and the compatibility with rule-based knowledge, give rise to a computing paradigm fundamentally different from those of other machine learning systems, specifically the backpropagation neural networks and the C4.5 symbolic induction algorithm. With its unique features, supervised ART has offered an interesting alternative approach to many real-world problems. Two applications are described below.

In personalized information systems, supervised ART systems can be used to model users' profile so that only the information most relevant to a user is identified and presented [17]. Each user profile is represented by a set of recognition categories, each associating a set of conjunctive features of a piece of information to a relevance factor. As the network structure is compatible with rule-based knowledge, user-defined rules can be readily translated into the recognition categories of a supervised ART system. In addition, subsequent user feedback on individual pieces of information can be used to refine the network. Through the refinement process, the network learns interest terms that are not explicitly mentioned by the user. As both user-defined and system-learned knowledge are represented in a single system, any inherent conflict or inconsistency can be detected and resolved readily.

Another potential domain is that of knowledge discovery and interpretation. Traditional data mining tools do not incorporate users' domain knowledge in the knowledge discovery process. As a result, the discovered knowledge can be very different from the users' perspectives and difficult to interpret. Supervised ART, on the other hand, provides a mechanism to incorporate users' knowledge. By building upon a user's prior knowledge, the final result is expected to be more interpretable to the user.

Acknowledgements

This chapter is based on the author's dissertation thesis submitted to the Department of Cognitive and Neural Systems, Boston University. The author would like to acknowledge Gail A. Carpenter, Stephen Grossberg, and Michael Cohen for their guidance and support. Thanks also go to Victor Chew who gave valuable comments on a previous version of the manuscript.

References

[1] Y. R. Asfour, G. A. Carpenter, S. Grossberg, and G. W. Lesher. Fusion ARTMAP: A neural network architecture for multi-channel data fusion and classification. In *World Congress on Neural Network, Portland, OR*, volume II, pages 210–215. Hillsdale, NJ: Lawrence Erlbaum Associates, July 1993.

[2] G. A. Carpenter and S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115, 1987.

[3] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen. Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3:698–713, 1992.

[4] G. A. Carpenter, S. Grossberg, and J. H. Reynolds. ARTMAP: Supervised real time learning and classification by a self-organizing neural network. *Neural Networks*, 4:565–588, 1991.

[5] G. A. Carpenter, S. Grossberg, and D. B. Rosen. Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural Networks*, 4:759–771, 1991.

[6] G. A. Carpenter and A.-H. Tan. Rule extraction: From neural architecture to symbolic representation. *Connection Science*, 7(1):3–27, 1995.

[7] R. Farber, A. Lapedes, and K Sirotkin. Determination of eucaryotic protein coding regions using neural networks and information theory. *Journal of Molecular Biology*, 226:471–479, 1992.

[8] L. M. Fu and L. C. Fu. Mapping rule-based knowledge into neural architecture. *Knowledge-Based Systems*, 3:48–56, 1990.

[9] R. C. Lacher, S. I. Hruska, and D. C. Kuncicky. Backpropagation learning in expert networks. *IEEE Transactions on Neural Networks*, 3:62–72, 1992.

[10] A. Lapedes, C. Barnes, C. Burks, R. Farber, and K Sirotkin. Application of neural networks and other machine learning algorithms to dna sequence analysis. In *Computers and DNA, SFI Studies in the sciences of complexity, vol VII*, pages 265–281. Reading, MA: Addison-Wesley, 1990.

[11] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases [machine-readable data repository]. Irvine, CA:

University of California, Department of Information and Computer Science, 1992.

[12] M. O'Neill. Escherichia coli promoters: I. Consensus as it relates to spacing class, specificity, repeat substructure, and three dimensional organization. *Journal of Biological Chemistry*, 264:5522–5530, 1989.

[13] M. O'Neill. Escherichia coli promoters: II. A spacing class-dependent promoter search protocol. *Journal of Biological Chemistry*, 264:5531–5534, 1989.

[14] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[15] A.-H. Tan. Adaptive Resonance Associative Map. *Neural Networks*, 8(3):437–446, 1995.

[16] A.-H. Tan. Cascade ARTMAP: Integrating neural computation and symbolic knowledge processing. *IEEE Transactions on Neural Networks*, 8(2):237–250, 1997.

[17] A.-H. Tan and C. Teo. Learning user profiles for personalized information dissemination. In *Proceedings, 1998 IEEE International Joint Conference on Neural Networks, Alaska*, pages 183–188, 1998.

[18] G. G. Towell and J. W. Shavlik. Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules. In *Advances in Neural Information Processing Systems 4*, pages 977–984. San Mateo, CA: Morgan Kaufmann, 1992.

[19] G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, 1993.

[20] G. G. Towell, J. W. Shavlik, and M. O. Noordewier. Refinement of approximately correct domain theories by knowledge-based neural networks. In *Proceedings, 8th National Conference on AI, Boston, MA*, pages 861–866. AAAI Press/The MIT Press, 1990.

[21] V. Tresp, J. Hollatz, and S. Ahmad. Network structuring and training using rule-based knowledge. In *Advances in Neural Information Processing Systems 5*, pages 977–984. San Mateo, CA: Morgan Kaufmann, 1993.

[22] M. S. Waterman. *Mathematical Methods for DNA sequences.* Boca Raton, FL: CRC Press, 1989.