**Abstract** - The default ARTMAP algorithm and its parameter values specified here define a ready-to-use general-purpose neural network system for supervised learning and recognition.

I. INTRODUCTION: ART TECHNOLOGY TRANSFER

Adaptive Resonance Theory (ART) neural networks model real-time prediction, search, learning, and recognition. ART networks function both as models of human cognitive information processing [1,2,3] and as neural systems for technology transfer [4]. A neural computation central to both the scientific and the technological analyses is the `ART matching rule` [5], which models the interaction between top-down expectation and bottom-up input, thereby creating a focus of attention which, in turn, determines the nature of coded memories.

Sites of early and ongoing transfer of ART-based technologies include industrial venues such as the Boeing Corporation [6] and government venues such as MIT Lincoln Laboratory [7]. A recent report on industrial uses of neural networks [8] states:

[The] Boeing ... Neural Information Retrieval System is probably still the largest-scale manufacturing application of neural networks. It uses [ART] to cluster binary templates of aeroplane parts in a complex hierarchical network that covers over 100,000 items, grouped into thousands of self-organised clusters. Claimed savings in manufacturing costs are in millions of dollars per annum. (p. 4)

At Lincoln Lab, a team led by Waxman developed an image mining system which incorporates several models of vision and recognition developed in the Boston University Department of Cognitive and Neural Systems (BU/CNS). Over the years a dozen CNS graduates (Aguilar, Baloch, Baxter, Bomberger, Cunningham, Fay, Gove, Ivey, Mehanian, Ross, Rubin, Streilein) have contributed to this effort, which is now located at Alphatech, Inc.

Customers for BU/CNS neural network technologies have attributed their selection of ART over alternative systems to the model’s defining design principles. In listing the advantages of its THOT® technology, for example, American Heuristics Corporation (AHC) cites several characteristic computational capabilities of this family of neural models, including fast on-line (one-pass) learning, “vigilant” detection of novel patterns, retention of rare patterns, improvement with experience, “weights [which] are understandable in real world terms,” and scalability (www.heuristics.com).

Design principles derived from scientific analyses and design constraints imposed by targeted applications have jointly guided the development of many variants of the basic networks, including fuzzy ARTMAP [9], ART-EMAP [10], ARTMAP-IC [11], Gaussian ARTMAP [12], and distributed ARTMAP [3,13]. Comparative analysis of these systems has led to the identification of a default ARTMAP network, which features simplicity of design and robust performance in many application domains [4]. Selection of one particular ARTMAP algorithm, specified here with a complete set of default parameter settings, is intended to facilitate ongoing technology transfer. A user may start with this version of the system, then, if necessary, adjust parameters to suit individual applications.

II. WINNER-TAKE-ALL VS. DISTRIBUTED CODE REPRESENTATIONS

The default ARTMAP algorithm (Section IV) outlines a procedure for labeling an arbitrary number of output classes in a supervised learning problem. A critical aspect of this algorithm is the distributed nature of its internal code representation, which produces continuous-valued test set predictions distributed across output classes.

The character of their code representations, distributed vs. winner-take-all, is, in fact, a primary factor differentiating various ARTMAP networks. The original models [9,14] employ winner-take-all coding during training and testing, as do many subsequent variations and the majority of ART systems that have been transferred to technology. Default ARTMAP is the same as fuzzy...
ARTMAP during training, but uses a distributed code representation during testing. ARTMAP-IC [11] equals default ARTMAP plus instance counting, which biases a category node's test set output by the number of training set inputs coded by that node. Distributed ARTMAP (dARTMAP) employs a distributed code (and instance counting) during both training and testing [3,13]. Versions of these networks [4] form a nested sequence:

fuzzy ARTMAP \( \Box \) default ARTMAP \( \Box \)
ARTMAP-IC \( \Box \) distributed ARTMAP

That is, distributed ARTMAP reduces to ARTMAP-IC when coding is set to winner-take-all during training; ARTMAP-IC reduces to default ARTMAP when counting weights are set equal to 1; and default ARTMAP reduces to fuzzy ARTMAP when coding is set to winner-take-all during testing as well as training.

ARTMAP variants with winner-take-all (WTA) coding and discrete target class predictions have shown consistent relative deficits in labeling accuracy and post-processing adjustment capabilities.

III. THE DEFAULT ARTMAP SYSTEM

Default ARTMAP codes the current input as a winner-take-all activation pattern during training and as a distributed activation pattern during testing. For distributed coding, the transformation of the filtered bottom-up input to an activation pattern across a field of nodes is defined by the increased-gradient CAM rule [13]. The default network also implements the MT- search algorithm [11] and sets the baseline vigilance parameter equal to zero, for maximal code compression. Other design choices for default ARTMAP include fast learning, whereby weights converge to asymptote on each learning trial; single-epoch training, which emulates on-line learning; a choice-by-difference signal function [15] from the input field to the coding field; and four-fold cross-validation.

ARTMAP’s capacity for fast learning implies that the system can incorporate information from examples that are important but infrequent and can be trained incrementally. Fast learning also causes each network’s memory to vary with the order of input presentation during training. Voting across several networks trained with different orderings of a given input set takes advantage of this feature, typically improving performance and reducing variability as well as providing a measure of confidence in each prediction [9]. While the number of voting systems is, in general, a free parameter, five voters have proven to be sufficient for many applications. Default ARTMAP thus trains five voting networks for each training set combination.

Even with the number of voters fixed, other design choices appear in systems where output activations may be distributed. In particular, default ARTMAP, which produces a continuous-valued distribution \( \Box_k \) across output classes \( k \) for each test set item, presents options for combining weighted predictions across voters to make a final class choice. One strategy sums the \( \Box_k \) values of individual networks to produce a net distributed output pattern, which is then used to determine the predicted class. An alternative strategy first lets each voting network choose its own winning output class, then assigns the test set inputs on the basis of these individual votes. In most applications, the first of these two voting strategies produces better results.

IV. DEFAULT ARTMAP ALGORITHM

Fig. 1 and Table I summarize default ARTMAP notation. Table II lists default parameter values. A user who wishes to explore network variations might begin by varying the baseline vigilance, \( \Box \). In some cases, higher values of \( \Box \) increase predictive accuracy but may decrease code compression.

![Diagram](image-url)
**TABLE I**

DEFAULT ARTMAP NOTATION

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>input component index</td>
</tr>
<tr>
<td>$j$</td>
<td>coding node index</td>
</tr>
<tr>
<td>$k$</td>
<td>output class index</td>
</tr>
<tr>
<td>$M$</td>
<td>number of input features</td>
</tr>
<tr>
<td>$a$</td>
<td>feature vector $(a_i), \ 0 \leq a_i \leq 1$</td>
</tr>
<tr>
<td>$A$</td>
<td>complement coded input vector: $A = (a, a^c)$</td>
</tr>
<tr>
<td>$K$</td>
<td>actual output class of training input</td>
</tr>
<tr>
<td>$y$</td>
<td>coding field activation pattern (CAM): $(y_j)$</td>
</tr>
<tr>
<td>$J$</td>
<td>chosen coding node (winner-take-all)</td>
</tr>
<tr>
<td>$C$</td>
<td>number of committed coding nodes</td>
</tr>
<tr>
<td>$\cdot$</td>
<td>committed node subsets</td>
</tr>
<tr>
<td>$T_j$</td>
<td>signal from input field to coding node $j$</td>
</tr>
<tr>
<td>$k$</td>
<td>signal from coding field to output node $k$</td>
</tr>
<tr>
<td>$w_j$</td>
<td>coding node weight vector $j$: $(w_{ij})$</td>
</tr>
<tr>
<td>$W_k$</td>
<td>output class weight vector $k$: $(W_{jk})$</td>
</tr>
<tr>
<td>$\parallel$</td>
<td>vigilance variable</td>
</tr>
<tr>
<td>$\parallel$</td>
<td>component-wise minimum (fuzzy intersection): $(p \parallel q)_i = \min(p_i, q_i)$</td>
</tr>
<tr>
<td>$\parallel$</td>
<td>vector size ($L_1$-norm): $</td>
</tr>
<tr>
<td>$p^c$</td>
<td>vector complement: $(p^c)_i = 1 - p_i$</td>
</tr>
</tbody>
</table>

**TABLE II**

DEFAULT PARAMETER VALUES

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameter</th>
<th>Range</th>
<th>Default Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta$</td>
<td>signal rule parameter</td>
<td>$[0,\infty)$</td>
<td>0.01</td>
<td>$\parallel = 0^*$ maximizes code compression</td>
</tr>
<tr>
<td>$b$</td>
<td>learning fraction</td>
<td>$[0,1]$</td>
<td>1.0</td>
<td>$\parallel = 1$ implements fast learning</td>
</tr>
<tr>
<td>$e$</td>
<td>match tracking</td>
<td>$(-1,1)$</td>
<td>$-0.001$</td>
<td>$\parallel &lt; 0$ (MT−) codes inconsistent cases</td>
</tr>
<tr>
<td>$r$</td>
<td>baseline vigilance</td>
<td>$[0,1]$</td>
<td>0.0</td>
<td>$\parallel = 0$ maximizes code compression</td>
</tr>
<tr>
<td>$p$</td>
<td>CAM rule power</td>
<td>$[0,\infty)$</td>
<td>1.0</td>
<td>$p = 1$ Increased Gradient (IG) CAM rule converges to WTA as $P \parallel$</td>
</tr>
<tr>
<td>$E$</td>
<td>training epochs</td>
<td>$\geq 1$</td>
<td>1</td>
<td>$E=1$ simulates on-line learning</td>
</tr>
<tr>
<td>$F$</td>
<td># data subsets</td>
<td>$\geq 3$</td>
<td>4</td>
<td>$F$-fold cross-validation</td>
</tr>
<tr>
<td>$V$</td>
<td># voting systems</td>
<td>$\geq 1$</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
A. Classification Methodology

This section outlines a canonical classification procedure for training and evaluating supervised learning systems, including ARTMAP.

A.1. List output classes for the supervised learning problem.

A.2. If possible, estimate an a priori distribution of output classes.

A.3. If not provided, create a ground truth set for each class by assigning output labels to a designated set of input vectors.

A.4. Divide the ground truth set into F disjoint subsets.

A.5. In each of the F subsets, designate either all ground truth inputs in that set; or P randomly chosen labeled inputs for each output class (or all inputs in a given class if fewer than P have been labeled). Fix random orderings of designated inputs in each subset.

A.6. Choose one subset for validation, one for testing, and the rest for training.

A.7. Train V systems (voters), each with E presentations of input vectors from one of the ordered training sets (Section IV.B).

A.8. For each voter, choose parameters by validation (if parameter choice is required).

A.9. Present to each voter all test set inputs. Produce an output class prediction \( \hat{O}_k \) for each test input (Section IV.C).

A.10. Sum the distributed output class predictions across the V voters.

A.11. Label inputs by one of three methods (breaking ties by random choice):

A.11.a. Baseline: Assign the input to the output class \( k \) with the largest summed prediction.

A.11.b. Prior probabilities: Select an output class at random according to the estimated a priori distribution in the data set. Assign that class label to the still-unlabeled input with the largest summed prediction for this class.

A.11.c. Validation: Bias the summed output class distribution, evaluating performance on the validation set. One such method [4] selects decision thresholds for each output class, with an upper bound of 10% set for each false alarm rate. Alternatively, the distributed prediction of each voter (or of the sum) could be weighted by a steepest descent algorithm. Use the biased summed distribution to label the input by the baseline or prior probabilities method.

A.12. Post-training output class adjustments:

A.12.a. Standard post-processing methods: Mapping tasks, for example, may benefit from local image smoothing. Post-processing for speckle removal may be implemented as a simple voting filter which assigns to each pixel the label originally assigned to a majority of its eight neighbors plus three copies of itself.

A.12.b. Class distribution adjustment: Starting with the output class predictions produced by any method (Step A.11), target distribution percentages may be adjusted up or down (e.g., based on inspection of resulting classes), and class labels recomputed by the prior probabilities method.

A.12.c. False alarm rate adjustment: A decision threshold for an over-represented class may be increased to reduce the validation set false alarm rate.

A.13. Classifier evaluation: Compute average performance statistics across all combinations of training subsets (each with \( V \) voters). Classifier evaluation measures include test set output class distributions, hit and false alarm rates for each class, overall accuracy on the test set, performance variability between tasks, product appearance (e.g., for mapping, overall and by overlays for each class), and degree of improvement by post-processing.

B. Default ARTMAP Training (Winner-Take-All Code)

The default ARTMAP algorithm specified here is a special case of the distributed ARTMAP (dARTMAP) algorithm described in [13].

B.1. Complement code \( M \)-dimensional training set feature vectors \( a \) to produce \( 2M \)-dimensional input vectors \( A \):

\[
A = (a, a^c) \quad \text{and} \quad |A| = M
\]

B.2. Set initial values: \( w_{ij} = 1, \quad W_{jk} = 0, \quad C = 1 \)

B.3. Select the first input vector \( A \), with associated actual output class \( K \)

B.4. Set initial weights for the newly committed coding node \( j = C \):

\[
w_C = A
\]

\[
W_{CK} = 1
\]
B.5. Set vigilance $\mathcal{L}$ to its baseline value:

\[ \mathcal{L} = \bar{\mathcal{L}} \]

and reset the code:

\[ y = 0 \]

B.6. Select the next input vector $A$, with associated actual output class $K$ (until the last input of the last training epoch)

B.7. Calculate signals to committed coding nodes $j = 1 \ldots C$:

\[ T_j = |A \cdot w_j| + (1 \cdot \mathcal{L})(M \cdot w_j) \]

B.8. Search order: Sort the committed coding nodes with $T_j > \mathcal{L}M$ in order of $T_j$ values (max to min)

B.9. Search for a coding node $J$ that meets the matching criterion and predicts the correct output class $K$, as follows:

B.9.a. Code: For the next sorted coding node $\{j = J\}$ that meets the matching criterion

\[ |A \cdot w_j| \geq \mathcal{L}M, \text{ set } y_j = 1 \text{ (WTA)} \]

B.9.b. Output class prediction:

\[ \mathcal{L}_K = \sum_{j=1}^{C} W_{jk} y_j = W_{jk} \]

B.9.c Correct prediction: If the active code $J$ predicts the actual output class $K$ ($\mathcal{L}_K = W_{JK} = 1$), go to Step B.11 (learning)

B.9.d Match tracking: If the active code $J$ fails to predict the correct output class ($\mathcal{L}_K = 0$), raise vigilance:

\[ \mathcal{L} = |A \cdot w_j| + \mathcal{L} \]

Return to Step B.9.a (continue search).

B.10. After unsuccessfully searching the sorted list, increase $C$ by 1 (add a committed node).

Return to Step B.4

B.11. Learning: Update coding weights:

\[ w_{j}^{\text{new}} = f|A \cdot w_{j}^{\text{old}}| + (1 \cdot \mathcal{L})(M \cdot w_{j}^{\text{old}}) \]

Return to Step B.5 (next input).

C. Default ARTMAP Testing (Distributed Code)

C.1. Complement code $M$-dimensional test set feature vectors $a$ to produce $2M$-dimensional input vectors $A$

C.2. Select the next input vector $A$, with associated actual output class $K$

C.3. Reset the code: $y = 0$

C.4. Calculate signals to committed coding nodes $j = 1 \ldots C$:

\[ T_j = |A \cdot w_j| + (1 \cdot \mathcal{L})(M \cdot w_j) \]

C.5. Let $\{j = 1 \ldots C: T_j > \mathcal{L}M\}$ and $\{j = 1 \ldots C: T_j = \mathcal{L}M\} = \{j = 1 \ldots C: w_j = A\}$

C.6 Increased Gradient (IG) CAM Rule:

C.6.a. Point box case: If $\exists \neq \{\}$ (i.e., $w_j = A$ for some $j$), set $y_j = \frac{1}{|A|}$ for each $j \neq \{\}$

C.6.b. If $\exists \neq \{\}$, set

\[ y_j = \frac{1}{|M \cdot T_j|} \]

for each $j \neq \{\}$

C.7. Calculate distributed output class predictions:

\[ \mathcal{L}_K = \sum_{j=1}^{C} W_{jk} y_j \]

C.8. Until the last test input, return to Step C.2

C.9. Predict output classes from $\mathcal{L}_K$ values, according to the chosen labeling method (see Step A.11)
REFERENCES


