Adaptive Resonance Theory

Michael Byrd

Neural Networks

Fall 2008

Motivation

How can we create a machine that can act and navigate in a world that is constantly changing? Such a machine would have to:

- Learn what objects are and where they are located.
- Determine how the environment has changed if need be.
- Handle unexpected events.
- Be able to learn unsupervised.

Known as the "Stability – Plasticity Dilemma": How can a system be adaptive enough to handle significant events while stable enough to handle irrelevant events?

Stability – Plasticity Dilemma

More generally, the Stability – Plasticity Dilemma asks:

How can a system retain its previously learned knowledge while incorporating new information?

Real world example:

Suppose you grew up in New York and moved to California for several years later in life. Upon your return to New York, you find that familiar streets and avenues have changed due to progress and construction. To arrive at your specific destination, you need to incorporate this new information with your existing (if not outdated) knowledge of how to navigate throughout New York. How would you do this?

Adaptive Resonance Theory

Gail Carpenter and Stephen Grossberg (Boston University) developed the *Adaptive Resonance* learning model to answer this question.

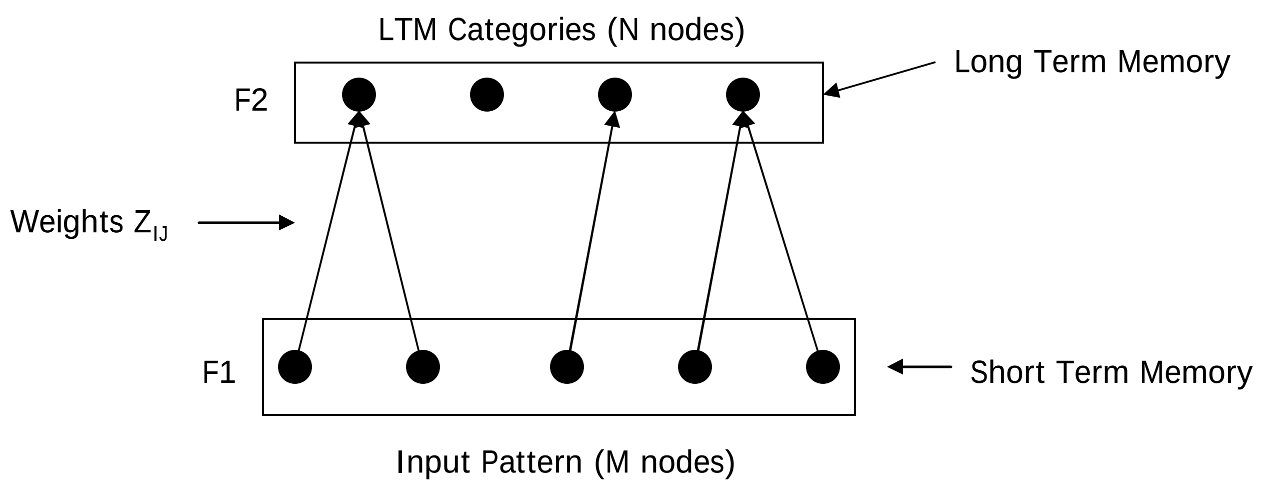
Essentially, ART (Adaptive Resonance Theory) models incorporate new data by checking for similarity between this new data and data already learned; "memory". If there is a close enough match, the new data is learned. Otherwise, this new data is stored as a "new memory".

Some models of Adaptive Resonance Theory are:

- ART1 – Discrete input.
- ART2 – Continuous input.
- ARTMAP – Using two input vectors, transforms the unsupervised ART model into a supervised one.
- Various others: Fuzzy ART, Fuzzy ARTMAP (FARTMAP), etc...

Competitive Learning Models

ART Models were developed out of *Competitive Learning Models*.



Let:

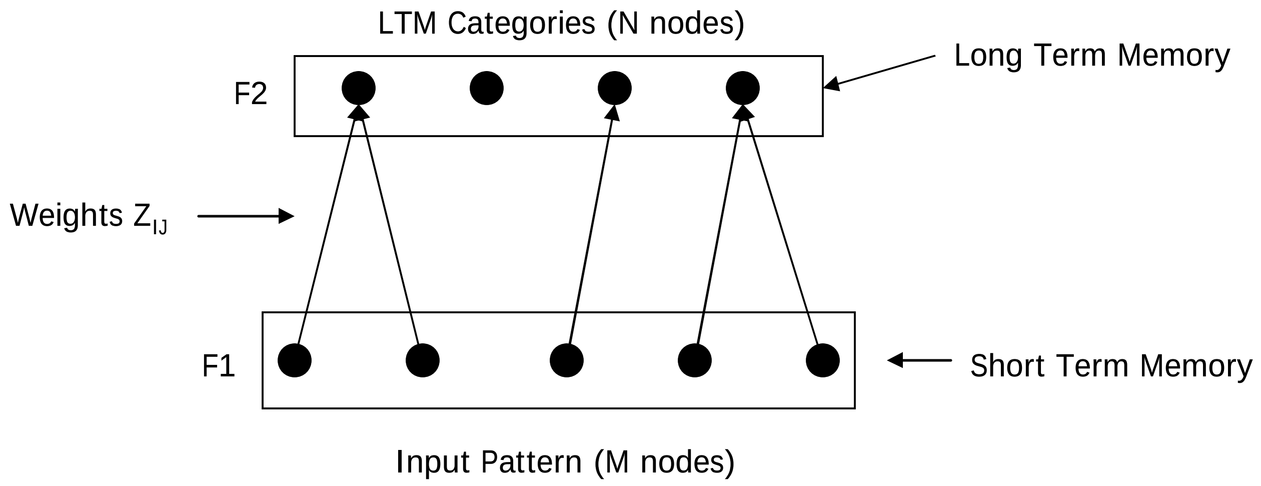
I = 1 ... M, J = 1 ... N

X_I = normalized input for node "I"

Z_{IJ} = weight from input node I to LTM category J

Competitive Learning Models (2)

Competitive Learning models follow a "winner take all" approach in that it searches for a LTM node that will determine how Z_{IJ} is modified.



Algorithm:

1. Normalize the input value for each node "I": $$X_i = \frac{I_i}{\sum_{k=1}^{M} I_k}$$
2. For each node "J", determine the total weight assigned to it from the input: $$T_j = \sum_{i=1}^{M} X_i Z_{ij}$$
3. Determine which LTM node most closely relates to the input: $$V_j = \begin{cases} 1 : T_j > \max(T_k : k \neq j) \\ 0 : T_j < \max(T_k : k \neq j) \end{cases}$$

Competitive Learning Models (3)

Once the appropriate LTM node has been chosen the weight vector is updated based on the memory contained within the “winning” node. One such practice is to replace the existing weight vector with the difference between itself and the normalized input values. That is:

Let:

$Z_j = <z_{1j}, z_{2j}, \ldots, z_{Mj}>$ be the weight vector for LTM node “J”

$X_j = <x_1, x_2, \ldots, x_M>$ be the vector representing the normalized input values for each input node.

Then the new weight vector is simply:

$$Z_j^{new} = X_j - Z_j^{old} = \left\langle |x_1 - z_{1j}|, |x_2 - z_{2j}|, \ldots, |x_M - z_{Mj}| \right\rangle$$

Changing this weight vector constitutes the “learning process”.

Problems

- If the input pattern is deformed somehow, the weight vector will become erroneously changed.
- Passing in the same input pattern consecutively can yield unstable learning.
- If the weight vector is changed considerably enough, it could discount previously learned information, and would have to be retrained. That is, it does *not* solve the Stability – Plasticity Dilemma.

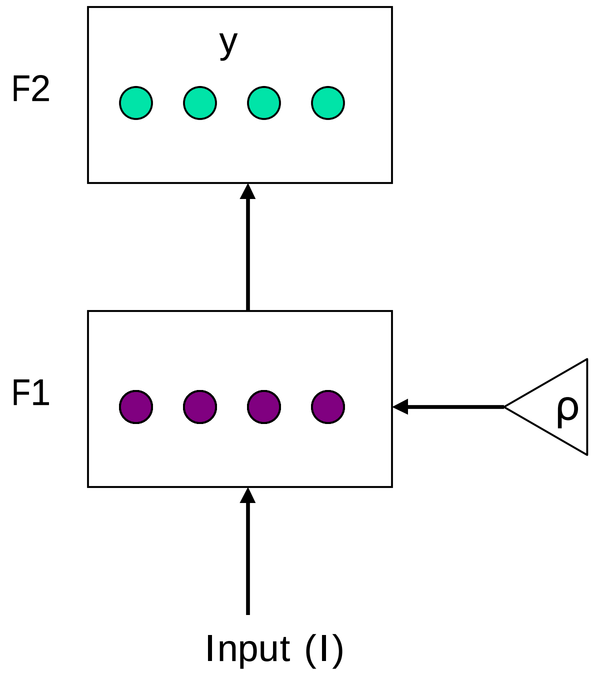
Adaptive Resonance Model

The basic ART model, ART1, is comprised of the following components:

1. The short term memory layer: F1 – Short term memory.
2. The recognition layer: F2 – Contains the long term memory of the system.
3. Vigilance Parameter: ρ – A parameter that controls the generality of the memory. Larger ρ means more detailed memories, smaller ρ produces more general memories.

Training an ART1 model basically consists of four steps.

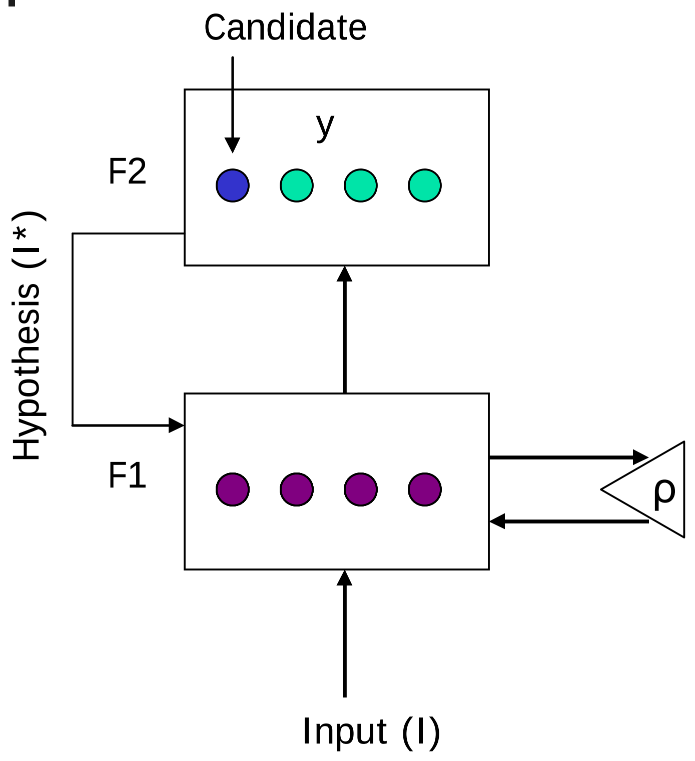
Adaptive Resonance Model (2)



Step 1: Send input from the F1 layer to F2 layer for processing. The first node within the F2 layer is chosen as the closest match to the input and a hypothesis is formed. This hypothesis represents what the node will look like after learning has occurred, assuming it is the correct node to be updated.

F1 (short term memory) contains a vector of size M, and there are N nodes within F2. Each node within F2 is a vector of size M. The set of nodes within F2 is referred to as “y”.

Adaptive Resonance Model (3)

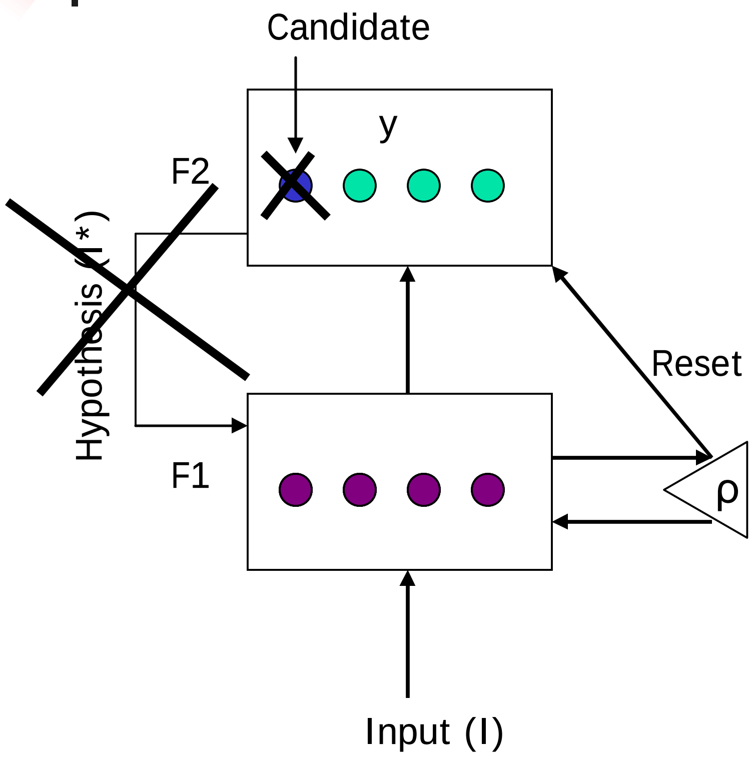


Step 2: Once the hypothesis has been formed, it is sent back to the F1 layer for matching. Let $T_j(I^*)$ represent the level of matching between I and I* for node j. Then:

$$T_j(I^*) = \frac{I \wedge I^*}{I} \quad \text{where} \quad A \wedge B = \min(A, B)$$

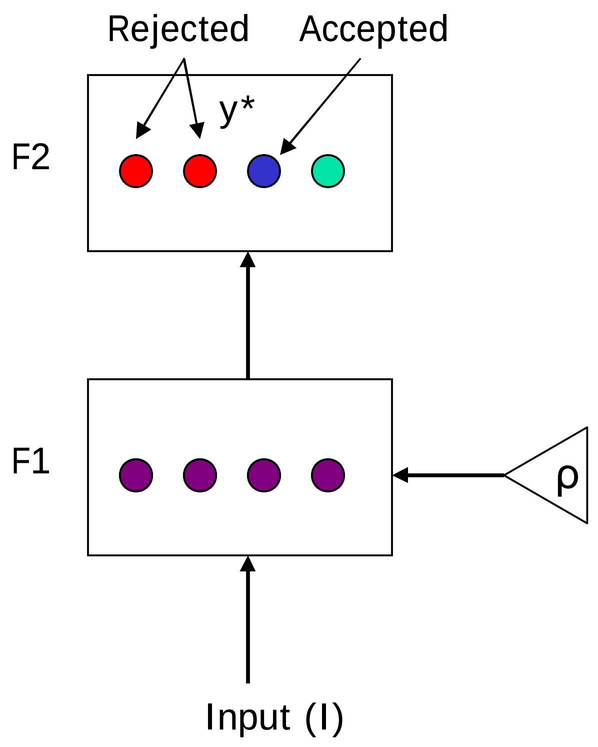
If $T_j(I^*) \geq \rho$ then the hypothesis is accepted and assigned to that node. Otherwise, the process moves on to Step 3.

Adaptive Resonance Model (4)



Step 3: If the hypothesis is rejected, a "reset" command is sent back to the F2 layer. In this situation, the jth node within F2 is no longer a candidate so the process repeats for node j+1.

Adaptive Resonance Model (5)



Step 4:

1. If the hypothesis was accepted, the winning node assigns its values to it.
2. If none of the nodes accepted the hypothesis, a new node is created within F2. As a result, the system forms a new memory.

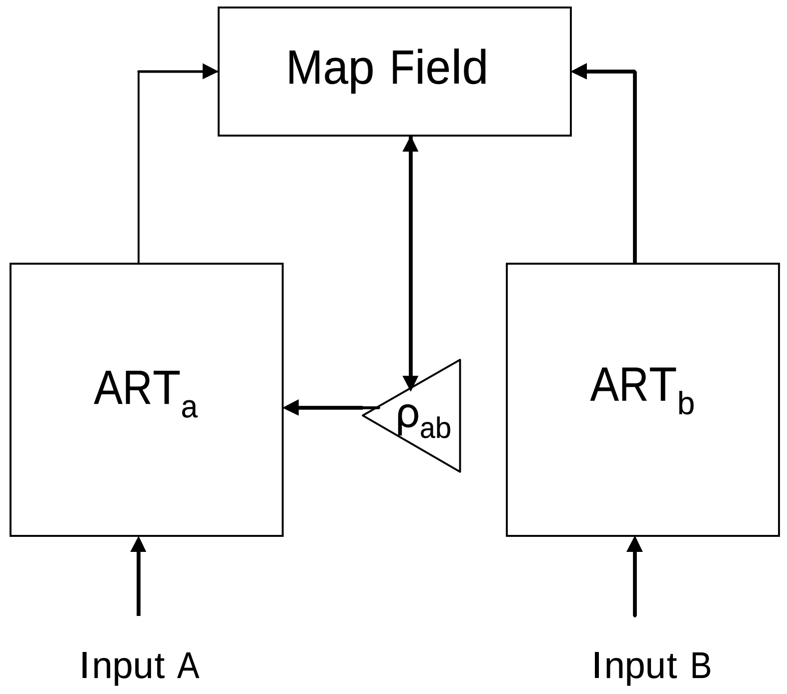
In either case, the vigilance parameter ensures that the new information does not cause older knowledge to be forgotten.

ART Extensions

As mentioned previously, Adaptive Resonance Theory has can be categorized into the following:

1. ART1 – Default ART architecture. Can handle discrete (binary) input.
2. ART2 – An extension of ART1. Can handle continuous input.
3. Fuzzy ART – Introduces fuzzy logic when forming the hypothesis.
4. ARTMAP – An ART network where one ART module attempts to learn based off of another ART module. In a sense, this is supervised learning.
5. FARTMAP – An ARTMAP architecture with Fuzzy logic included.

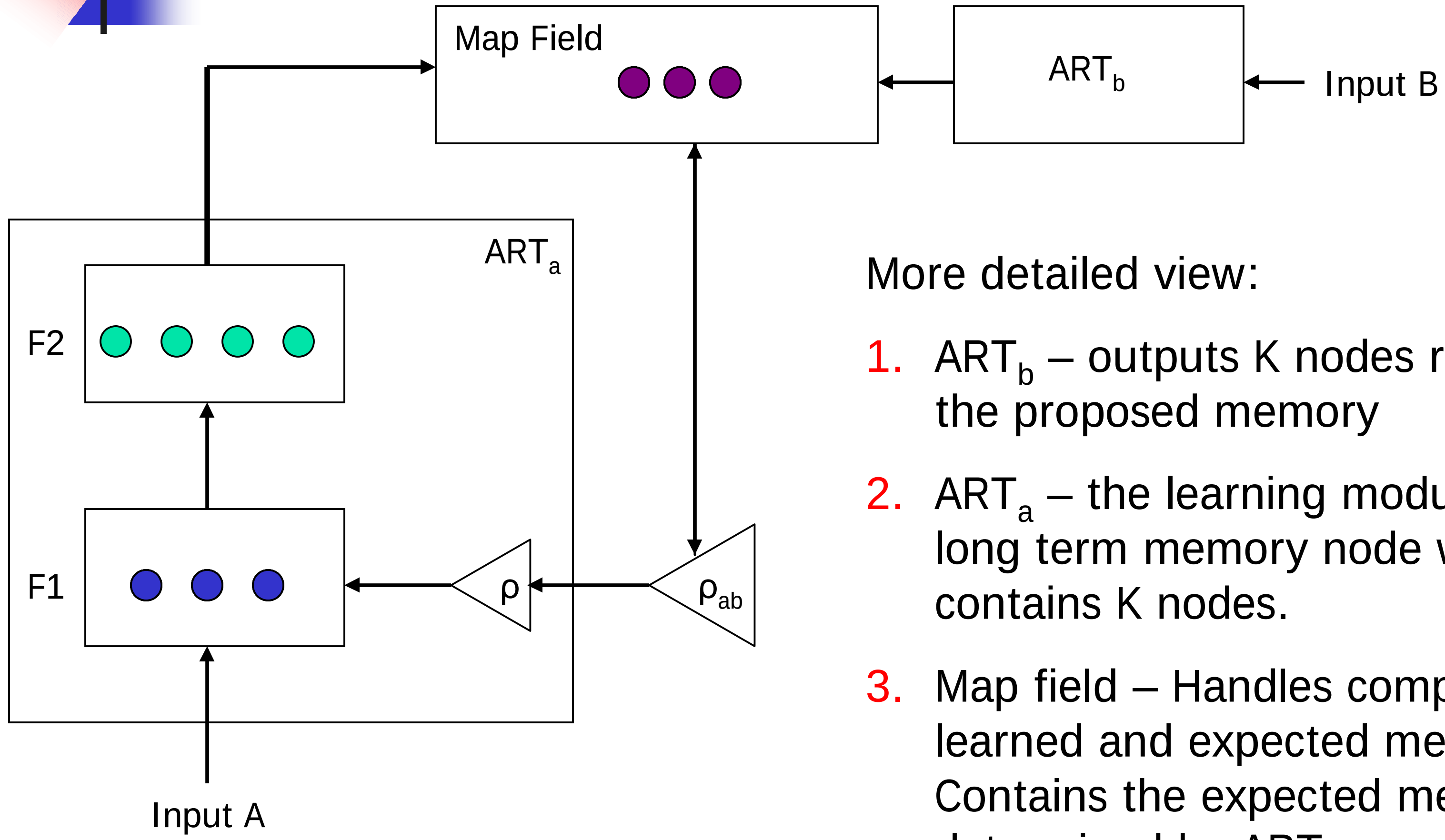
ARTMAP



Description of Components:

1. ART$_a$ – Outputs a proposed hypothesis to be matched against ART$_b$.
2. ART$_b$ – Outputs a prototype vector that ART$_a$ should be close to.
3. ρ_{ab} – Vigilance parameter to match the outputs of ART$_a$ and ART$_b$.
4. Map Field – Responsible for comparing the outputs. Can modify ρ_{ab} to help match ART$_a$ to ART$_b$.

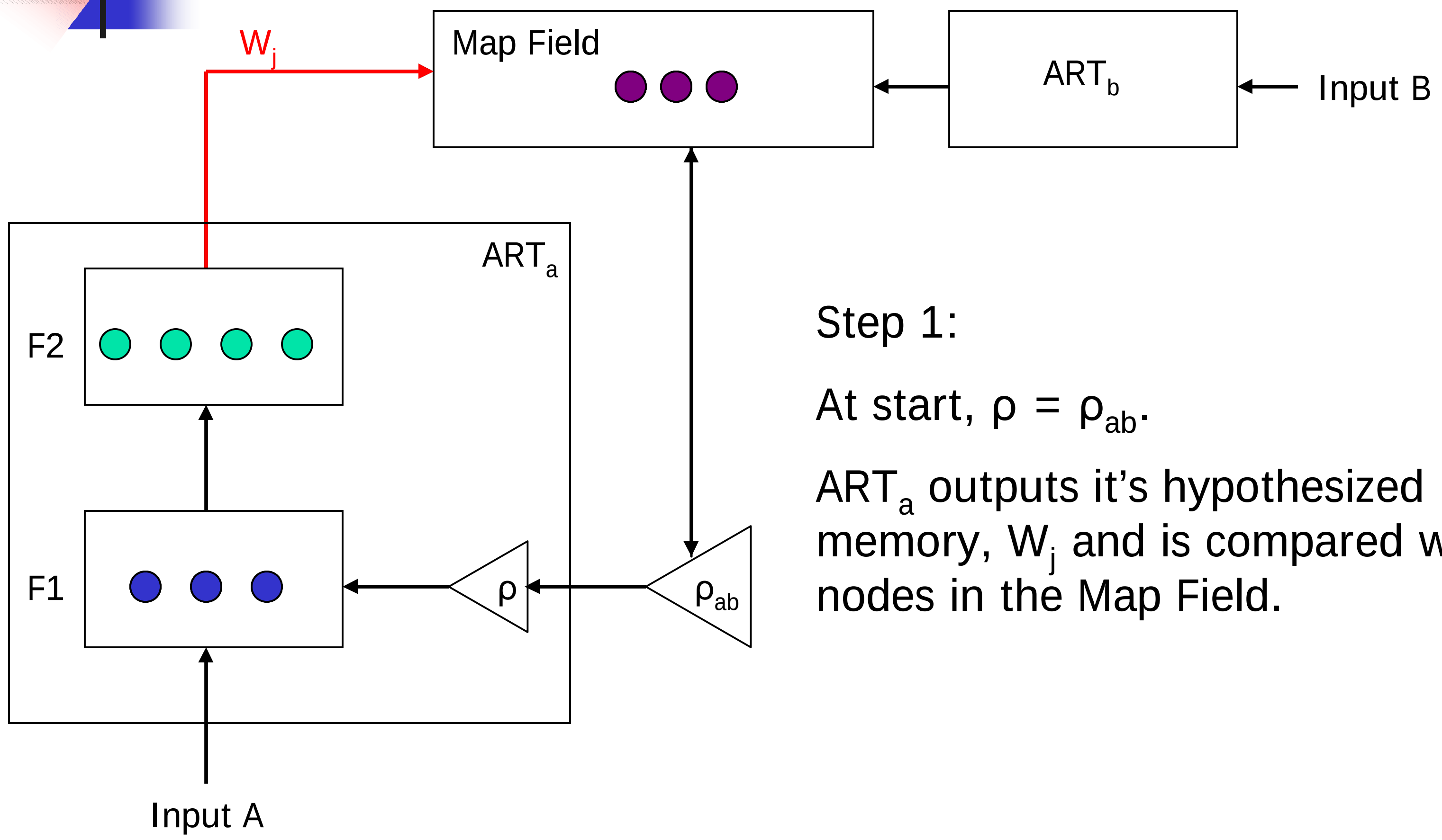
ARTMAP (2)



More detailed view:

1. ART_b – outputs K nodes representing the proposed memory
2. ART_a – the learning module. Each long term memory node within F2 contains K nodes.
3. Map field – Handles comparison of learned and expected memory. Contains the expected memory determined by ART_b.

ARTMAP (3)

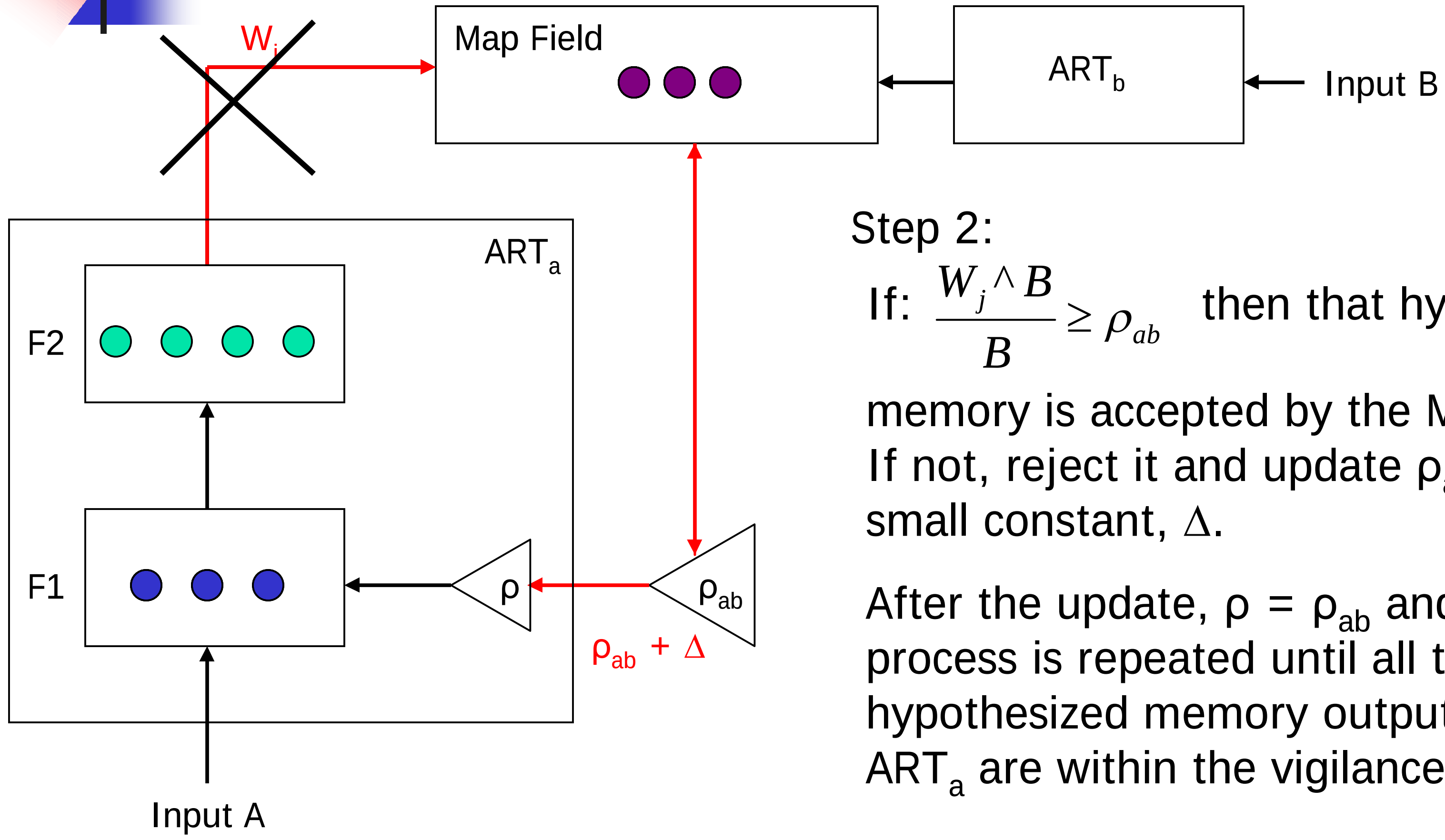


Step 1:

At start, $\rho = \rho_{ab}$.

ART_a outputs it's hypothesized long term memory, W_j and is compared with the nodes in the Map Field.

ARTMAP (4)



Step 2:

If: $\frac{W_j \wedge B}{B} \geq \rho_{ab}$ then that hypothesized memory is accepted by the Map Field. If not, reject it and update ρ_{ab} by a small constant, Δ.

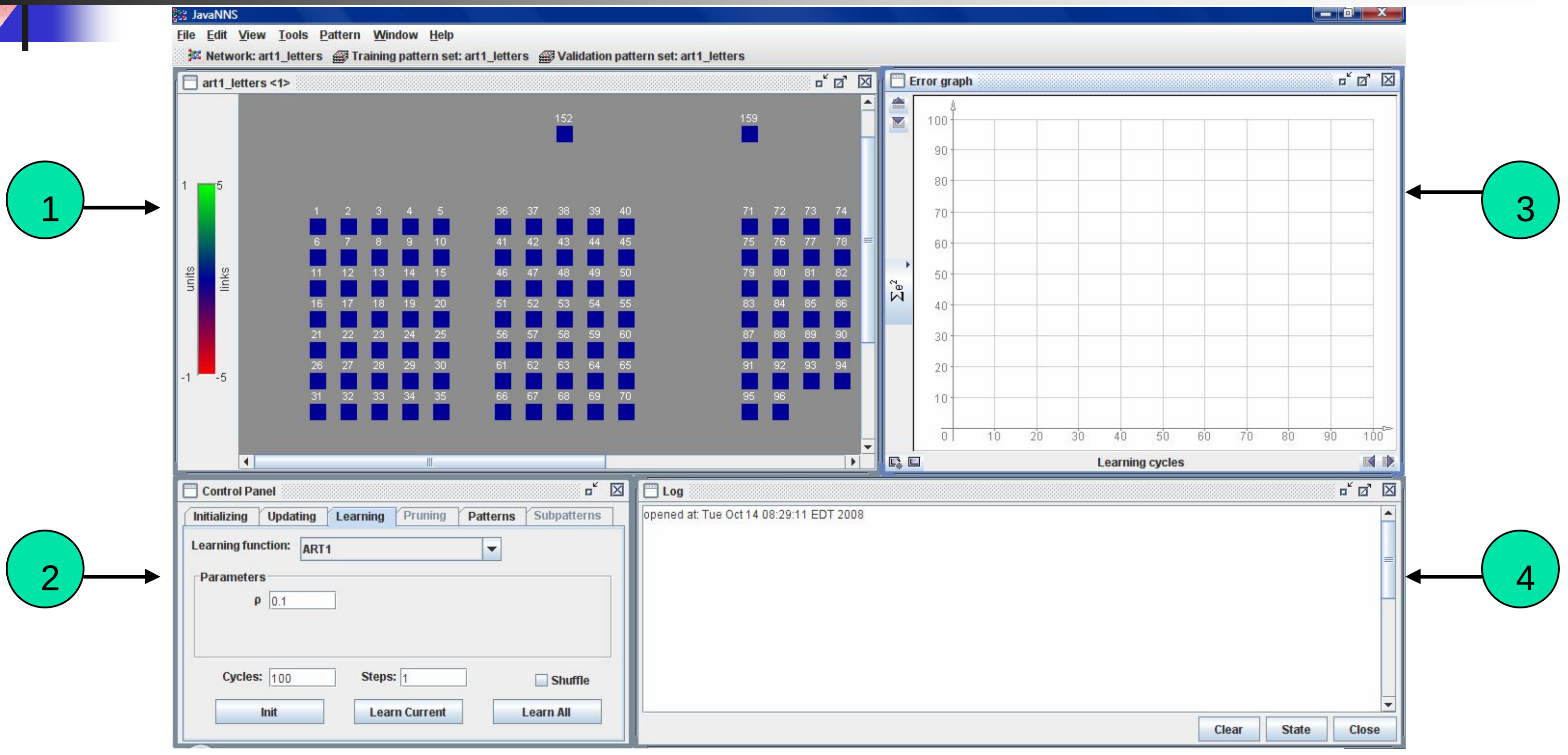
After the update, $\rho = \rho_{ab}$ and the process is repeated until all the hypothesized memory outputs from ART$_a$ are within the vigilance level.

Simulation

Several simulation and software packages exist for ART systems. The one chosen for this project is the Java Neural Network Simulator (abbreviated: Java NNS): http://ww9w.ra.cs.uni-tuebingen.de/software/JavaNNS/welcome_e.html

- Based off of the Stuttgart Neural Network Simulator (SNNS) package.
    - Written in C\C++.
- Developed at the University of Tübingen by CS students/faculty.
- Able to simulate multiple neural network architectures including:
    - Backpropagation systems
    - Radial Basis Functions
    - Spiking Neural Networks
    - ART1, ART2 and ARTMAP networks

Java Neural Network Simulator


JavaNNS
File Edit View Tools Pattern Window Help
Network: art1_letters
Training pattern set: art1_letters
Validation pattern set: art1_letters
art1_letters <1>
units
links
Error graph
Learning cycles
Control Panel
Initializing Updating Learning Pruning Patterns Subpatterns
Log
Init
Learn Current
Learn All
Clear
State
Close
1
2
3
4

Learning function: ART1

Parameters

ρ 0.1

Cycles: 100 Steps: 1 Shuffle [ ]

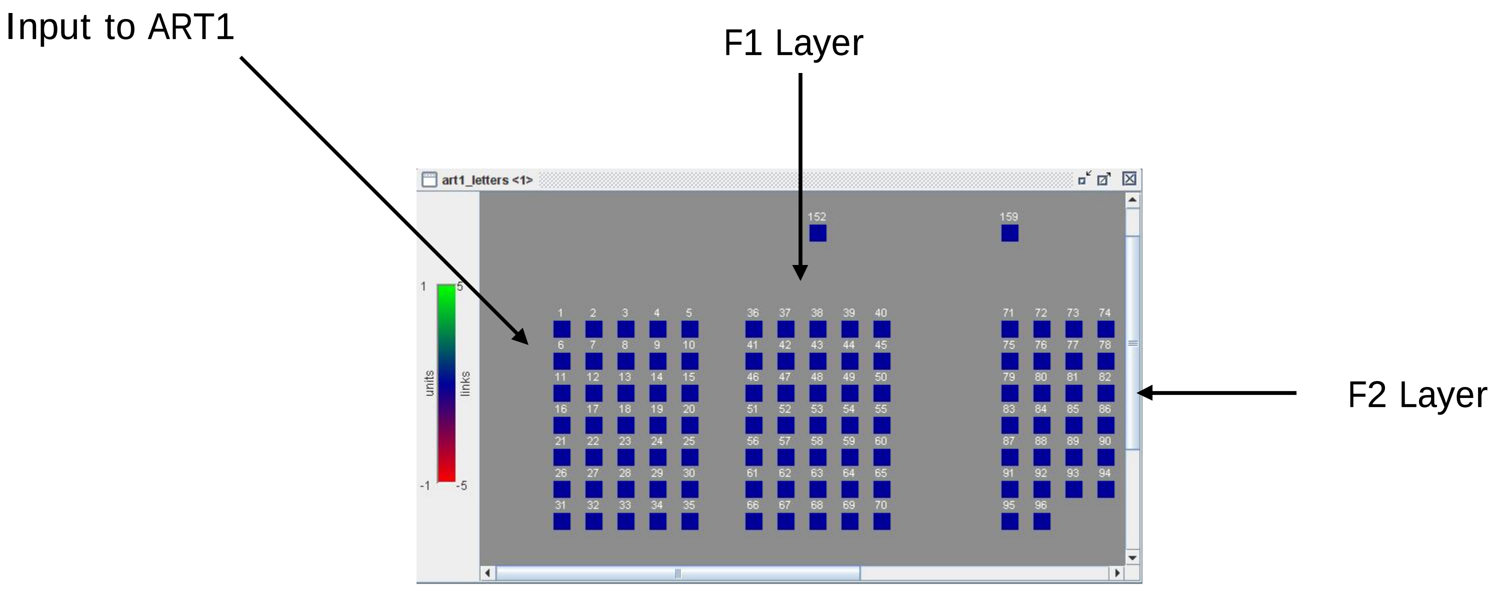
```
opened at: Tue Oct 14 08:29:11 EDT 2008
```

There are many parts to the simulator, but four are shown here:

1. ART Network
2. Control Panel
3. Error graph (not always used)
4. Log (optional)

Java Neural Network Simulator (2)

For this specific simulation, the input is a 5x7 grid of nodes that represent an upper-case letter in the English alphabet. There are 35 (5x7) nodes in the F1 layer representing the best hypothesized letter learned so far, and 26 nodes within the F2 layer, each representing a possible letter in the alphabet.



Java Neural Network Simulator (3)

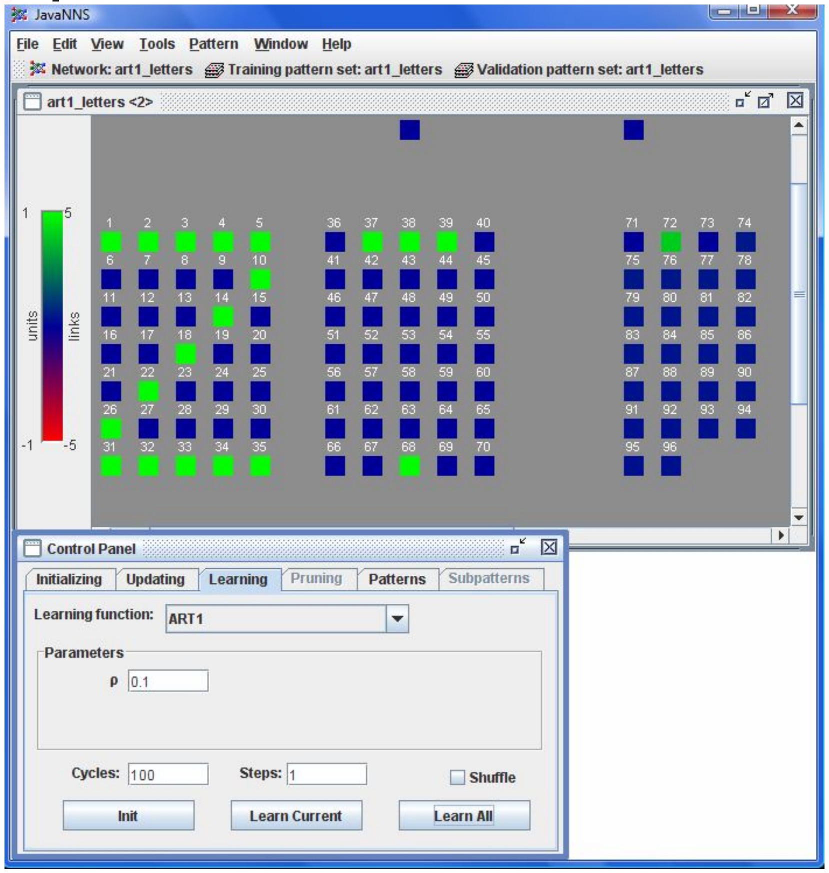
Example: We want the Java NNS to recognize the letter “Z”.

Note: The 26 nodes in the F2 layer are numbered 71 – 96. After looking through the source, I found that “Z” corresponds to node 85. Therefore, this NNS should select that as the winning node as it should be the closest match to any hypothesis made by the F2 layer.

To start:

- Input to the ART is the letter “Z”.
- Each node in F2 represents a letter to be matched against.
- $\rho = 0.1$

Java Neural Network Simulator (4)



Learning function: ART1

Parameters

ρ 0.1

Cycles: 100

Steps: 1

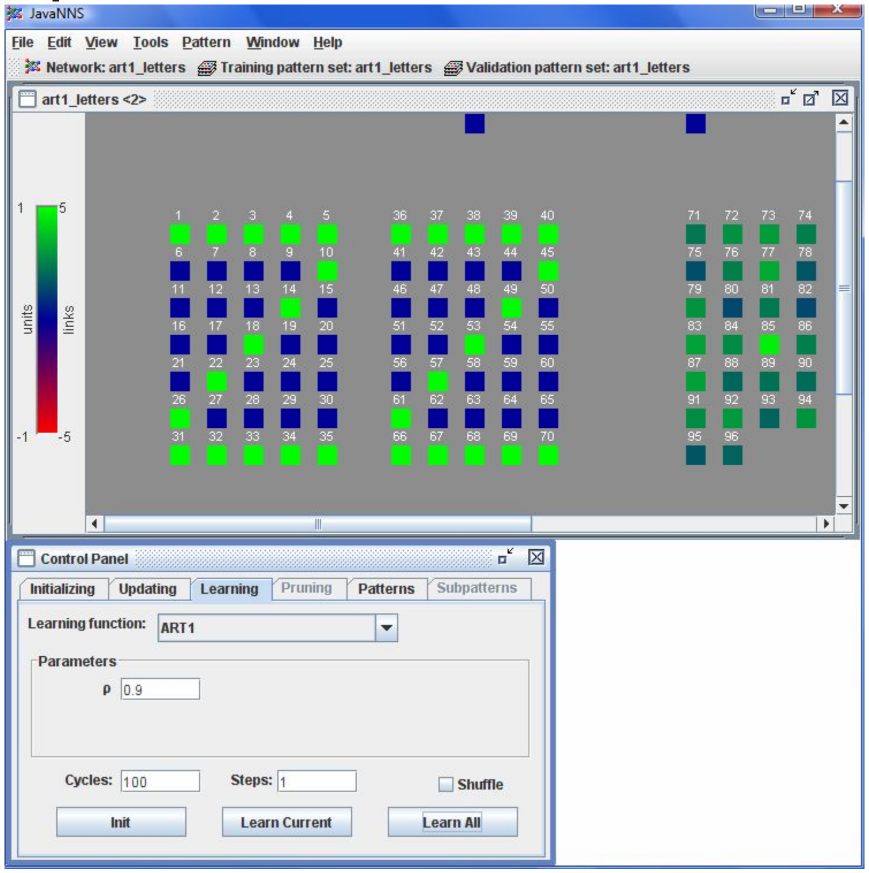
Shuffle

We can see that the ART1 network learned poorly. How do we know this?

1. Input layer shows the input was "Z", but the F1 layer is not even a character.
2. Node 72 in the F2 layer is selected as being the most probable letter. It should be node 85.

What do you think went wrong?

Java Neural Network Simulator (5)



Learning function: ART1

Parameters

ρ 0.9

Cycles: 100 Steps: 1 Shuffle

Answer: The vigilance parameter ρ was too small (ρ=0.1). As a result, very general memories were being formed and the letter was not determined correctly.

By increasing to ρ=0.9, we see the pattern learned in the F1 layer is in fact "Z" and node 85 in the F2 layer is selected as being the most likely letter.

Drawback: it takes a bit longer to learn with a higher vigilance value than with a smaller one.

Java NNS Evaluation

Pros:

1. Very nice and neat graphical user interface.
2. Supports 25+ neural network architectures.
3. Cross platform.
4. Efficient and memory inexpensive.

Cons:

1. Multiple errors and poor error handling (program crashed multiple times).
2. Poor instruction manual (authors keep telling you to refer to other sources).
3. Not all functionality is explained, leaving the user at a loss at times.
4. Adds “temporary” files to the user’s machine without deleting or announcing it.

Application: Clustering Web Users

Consider the case of having a dynamic website consisting of multiple components and each user access each component a certain number of times. How would you redesign the website to provide the most commonly used components to that specific user without being impractical in implementation?

This was the question behind:

Adaptive Neural Network Clustering of Web Users (see references).

Define:

1. M = 200: Number of unique components – F1 layer
2. N: Num. of organizational clusters that users are grouped into – F2 layer
3. H = 70: Number of users

Clustering Web Users (2)

Usage information for each component, per user, can be found in the usage logs recorded on the backend server.

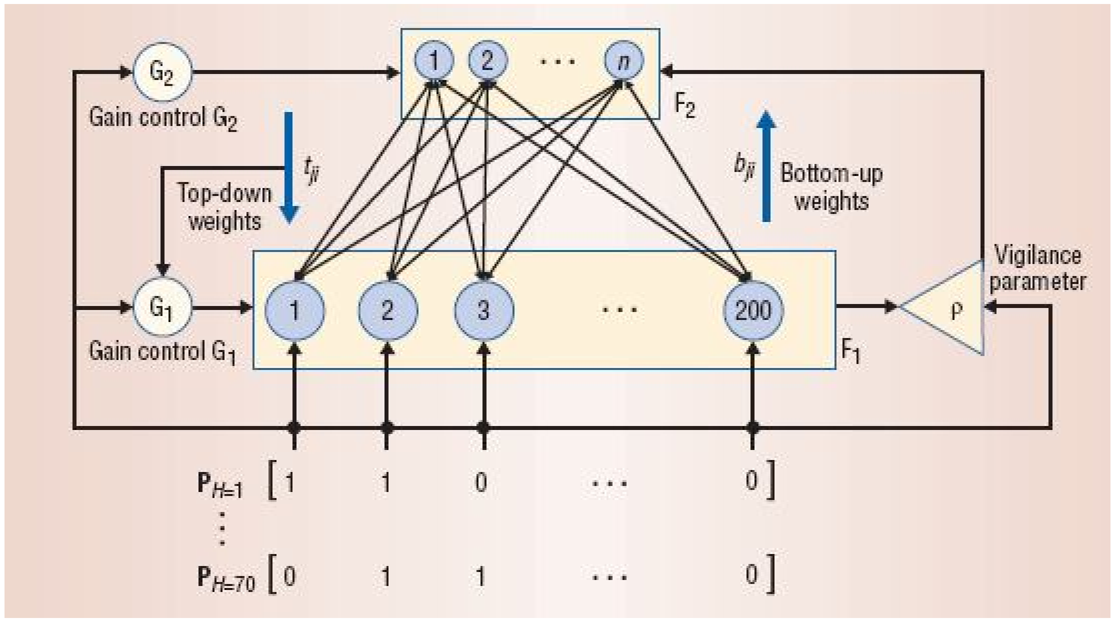
Algorithm:

1. Extract the web logs pertaining to the given user.
2. Extract usage for each possible component using the logs.
3. After assigning weights to each component based on usage, use the ART1 network architecture to determine best possible combination for that user.
4. Repeat this process for each user to achieve the best grouping of users within the F2 layer.

Result is a layer of clusters of organizational types where each user is within a cluster.

Clustering Web Users (3)

ART1 Architecture used:



Note: Control gains G_1 and G_2 are specific to this application only. They were used as additional means of bottom-up and top-down feedback between layers.

Clustering Web Users (4)

Results:

Vigilance parameters of 0.3 through 0.5 were used, producing between 20 and 52 distinct clusters.

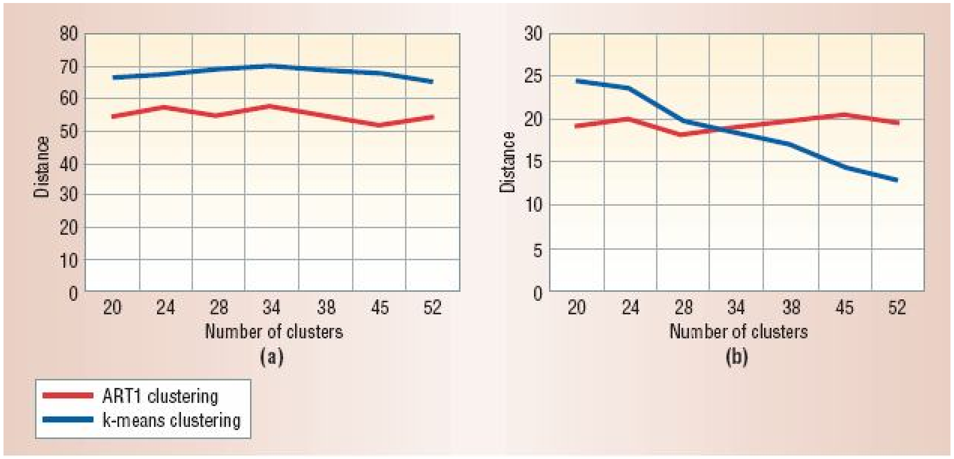


Fig (a) shows the intercluster distance within each cluster formed using ART1 and compared with K-Means Clustering.

Fig (b) shows the intracluster distances between clusters.

Accuracy of clustering for each user ranged from 82% to 97%, with an error producing one user at 12%. Accuracy was determined by checking the usage logs for each user for 13 days after this experiment was conducted and comparing usage.

ART Summary

1. Solves Stability – Plasticity Dilemma.
2. Forms new memories or incorporates new information based on a predefined vigilance parameter.
3. Higher vigilance produces more detailed memories, lower vigilance produces more general memories.
4. Extended to ART2, ARTMAP, Fuzzy ART, etc.

References

1. Santosh K. Rangarajan, Vir V. Phoha, Kiran S. Balagani, Rastko R.Selmic, S.S. Iyengar, "Adaptive Neural Network Clustering of Web Users," **Computer**, Vol. 37, No. 4, pp. 34-40, Apr., 2004
2. Gail A. Carpenter, and Stephen Grossberg, “Adaptive Resonance Theory”, **The Handbook of Brain Theory and Neural Networks**, Ed. 2, Sept., 1998
3. Gail A. Carpenter, “Default ARTMAP”, **Neural Networks**, July., 2003
4. Jianhong Luo, and Dezhao Chen, “An Enhanced ART2 Neural Network for Clustering Analysis”, **Proceedings of the 1st international conference on Forensic applications and techniques in telecommunications, information, and multimedia and workshop**, 2008
5. E.P. Sapozhnikova, V.P. Lunin, "A Modified Search Procedure for the Art Neural Networks," ijcnn,pp.5541, **IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)**-Volume 5, 2000
6. Gail A. Carpenter, and Stephen Grossberg, “The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network”, **Computer**, Vol. 21, No. 3, pp. 77-88, Mar., 1988
7. Robert A. Baxter, “Supervised Adaptive Resonance Networks”, **Proceedings of the conference on Analysis of neural network applications,** pp. 123 – 137, 1991
8. Pui Y. Lee, Siu C. Hui., and Alvis Cheuk Fong, “Neural Networks for Web Content Filtering”, **IEEE Intelligent Systems**, Vol. 17, No. 5, pp. 48-57, Sept., 2002
9. “Adaptive Resonance Theory”, **Wikipedia**, http://en.wikipedia.org/wiki/Adaptive_resonance_theory