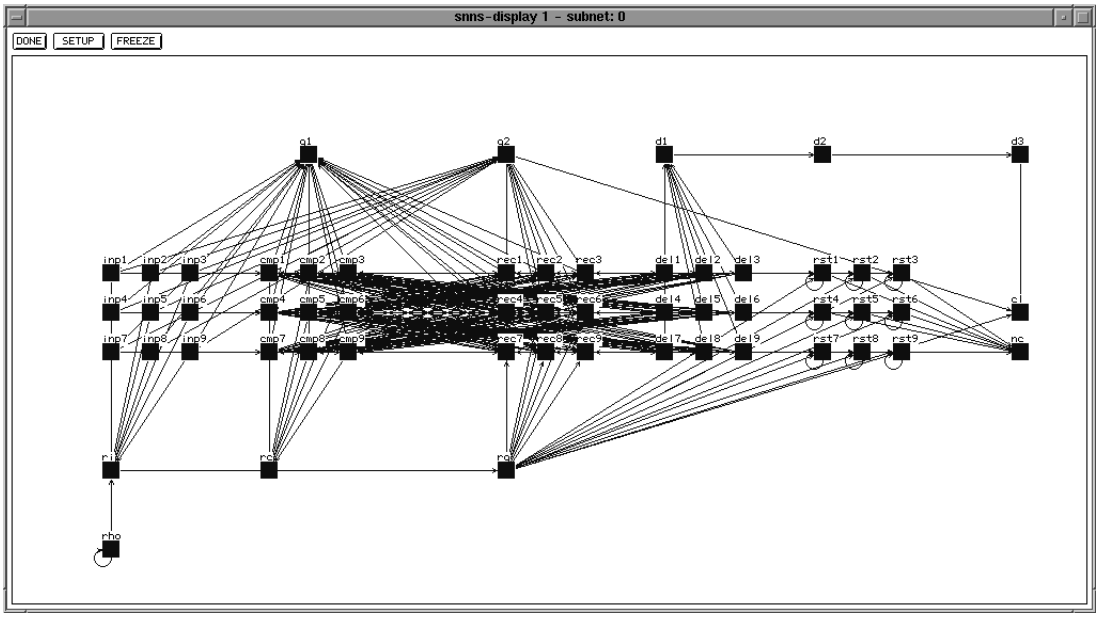
1-1-1995

Adaptive Resonance Theory (ART): An Introduction

Lucien G. Heins

Daniel R. Tauritz
Missouri University of Science and Technology, tauritzd@mst.edu

Adaptive Resonance Theory (ART): An Introduction
by
L.G. Heins & D.R. Tauritz
May/June 1995



ART-1 network produced by SNNS v3.3 - 9 input nodes, 9 output nodes

Index

Paragraph 1 - Introduction

The purpose of this paper is to provide an introduction to Adaptive Resonance Theory (ART) by examining ART-1, the first member of the family of ART neural networks. The only prerequisite knowledge in the area of neural networks necessary for understanding this paper is backpropagation [Hinton86]. For an easy introduction to neural networks see [Freeman91], for a more in depth overview of the field see [Hertz91].

Many interesting problems concern the classification of data. For example, say we want to classify animals according to certain characteristics described by a set of parameters. We might have a dog, a cat and an owl. Some characteristics might be "number of legs", "can fly", "has fur" and "is a carnivore". With these characteristics we would hope that the cat and the dog are classified together and the owl separately. In this paper an algorithm which performs this mapping is called a *clustering algorithm.* A clustering algorithm takes as input a set of input vectors and gives as output a set of clusters and a mapping of each input vector to a cluster. Input vectors which are close to each other according to a specific similarity measure should be mapped to the same cluster. Clusters can be labelled to indicate a particular semantic meaning pertaining to all input vectors mapped to that cluster. The cat and the dog might be classified in a cluster labelled "mammals" and the owl in "birds". However one could also choose "pets" as label for the cluster with the cat and the dog and "winged animal" for the other. Clusters are usually internally represented using *prototype vectors* which are vectors indicating a certain similarity between the input vectors which are mapped to a cluster. In the above example the first cluster might have prototype vector (4 legs,can't fly,has fur,is a carnivore) and the second might have prototype vector (2 legs,can fly,doesn't have fur,is a carnivore).

In paragraph 2 the argument will be made that many popular neural networks such as backpropagation have drawbacks making them less suitable to solving these kinds of classification problems. This will be the motivation for introducing ART.
In paragraph 3 the sequential algorithm underlying the ART-1 network is given along with another sequential clustering algorithm with which it is compared.
Paragraph 4 will introduce competitive networks, show how to extend them to ART networks, and examine the ART architecture in detail.
Paragraph 5 will mention some other members of the ART family including many references for further reading.

Paragraph 2 - Motivation

For example, say we want to categorize the vectors within a certain input environment. At a certain point in time we start training a backpropagation network with N vectors. When training is completed these N vectors will be correctly classified and hopefully other vectors within this input environment will also be because of generalization. However, as the input environment changes in time the accuracy of the backpropagation network will rapidly decrease because the weights are fixed thus preventing the network from adapting to the changing environment. This algorithm is not *plastic*. An algorithm is plastic if it retains the potential to adapt to new input vectors indefinitely.

To overcome this problem the network can be retrained on the new input vector (or the last few). The network will adapt to any changes in the input environment (remain plastic) but this

will cause a rapid decrease in the accuracy with which it categorizes the old input vectors because old information is lost. This algorithm is not *stable*. An algorithm is stable if it preserves previously learned knowledge.

This conflict between stability and plasticity is called the stability-plasticity dilemma [Carpenter87a]. The problem can be posed as follows:

- How can a learning system be designed to remain plastic, or adaptive, in response to significant events and yet remain stable in response to irrelevant events?
- How does the system know how to switch between its stable and its plastic modes to achieve stability without rigidity and plasticity without chaos?
- In particular, how can it preserve its previously learned knowledge while continuing to learn new things?
- And, what prevents the new learning from washing away the memories of prior learning?

Most existing algorithms are either stable but not capable of forming new clusters, or plastic but unstable.

The above method using backpropagation could be adapted by retraining the network on the entire set of input vectors each time a new input vector is presented. This however would be extremely inefficient and thus its use would be precluded in any practical application. The problem with this method is that it is not incremental.

What we need is a network which itself is incremental thus making it unnecessary to retrain the network on the entire set of input vectors. As we will see in an example in paragraph 3 some incremental networks are unstable.

ART was specifically designed to overcome the stability-plasticity dilemma [Grossberg76b]. The ART-1 neural network was designed to overcome this dilemma for binary input vectors [Carpenter87a], ART-2 for continuous ones as well [Carpenter87b]. In this paper we will further confine ourselves to discussing ART-1.

ART-1 is an unsupervised neural network. It is unsupervised in the sense that it establishes the clusters without external interference.

Paragraph 3 - Concepts

We can study some properties of a neural network by examining its sequential counterpart without being distracted by its architecture. To gain insight into what ART-1 does, as opposed to how it does it, an algorithmic description will be presented in this chapter.

First of all let us clarify what is meant by an incremental clustering algorithm by presenting an algorithm shell for this purpose.

CLUSTER - A clustering algorithm shell with incremental update of prototype vectors and a variable number of clusters

Step 1 - Initialisation

- Start with no cluster prototype vectors

Step 2 - Apply new input vector

- Let I := [next input vector]

Step 3 - Find the closest cluster prototype vector (if any)
- Let P := [closest prototype vector]

Step 4 - Check if P is too far from I
- If P is too far from I, or if there are no cluster prototype vectors yet, then create a new cluster, with prototype vector equal to I; output the index of this cluster; goto step 2

Step 5 - Update the matched prototype vector
- Update P by moving it closer to I
- Output P's index
- Goto step 2

To obtain an actual algorithm it is necessary to define "closest", "too far" and "move closer to". A possible instantiation of CLUSTER is one using the Euclidean distance measure.

CLUSTER-EUCLIDEAN - An instantiation of CLUSTER using a Euclidean distance measure

Step 1 - Initialisation
- Start with no cluster prototype vectors

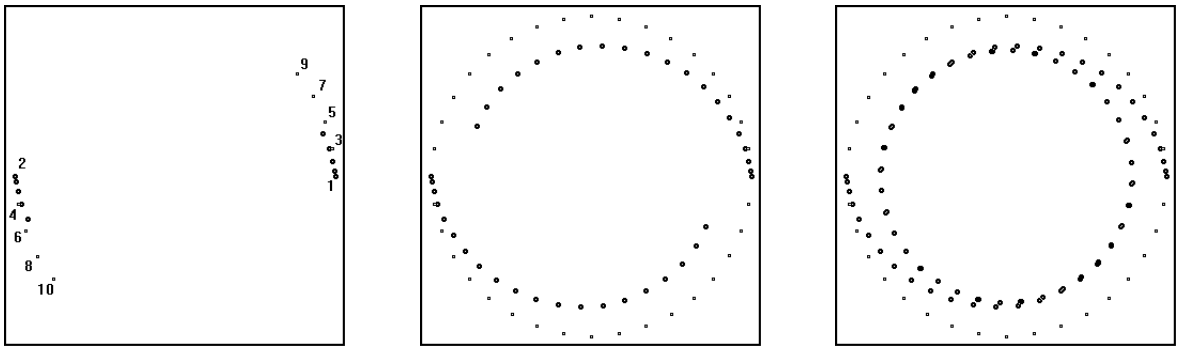
Step 2 - Apply new input vector
- Let I = $(I_1, \ldots, I_n)$:= [next input vector]

Step 3 - Find the closest cluster prototype vector (if any)
- Find the P = $(P_1, \ldots, P_n)$ to minimize d(P,I) = $\sqrt{\sum_{x=1}^{n} (\quad)^2}$

Step 4 - Check if P is too far from I
- If d(P,I) > θ, or if there are no cluster prototype vectors yet, then create a new cluster, with prototype vector equal to I; output the index of this cluster; goto step 2

Step 5 - Update the matched prototype vector
- Let P := $(1 - \lambda) \cdot P + \lambda \cdot I$
- Output P's index
- Goto step 2

This instantiation is, however, unstable in the sense that the prototype vectors can cycle indefinitely during repetitive presentation of a finite sequence of input vectors [see fig.1]. Also, different prototype vectors may have infinitesimal differences. Both problems are solved in the ART-1 algorithm.



a b c

Fig.1 Snapshots of Cluster-Euclidean (see Appendix) for θ=1, λ=0.2 (radius of circle of input vectors is 1) after a) 10 input vectors b) 40 input vectors and c) 200 input vectors. The squares represent the input vectors (numbered in their order of presentation in a), the circles the traces of the prototype vectors. After the second input vector has been presented there are (and continue to be) precisely two prototype vectors, moving counter-clockwise, and eventually reaching a limit-cycle.

ART-1 Clustering Algorithm

Note: $v \cap w$ = bitwise AND of vectors v and w; $\|u\|$ = [magnitude of u] = # of 1's in u

Step 1 - Initialisation

- Initialise the vigilance parameter ρ so $0 < \rho \leq 1$
- Initialise the set P of prototype vectors to {}

Step 2 - Apply new input vector

- Let I:=[next input vector]
- Let P':=P be the set of candidate prototype vectors

Step 3 - Find the closest prototype vector from P'

- Find the i which maximizes $\dfrac{\|p_i \cap I\|}{\beta + \|p_i\|}$

Step 3' - Check if I is closer to p_i or to (1,1, ..., 1)

- If $\dfrac{\|p_i \cap I\|}{\beta + \|p_i\|} < \dfrac{\|I\|}{\beta + n}$ then create a new cluster p_j equal to I; $P = P \cup \{p_j\}$; output j ; goto step 2

Step 4 - Check if p_i is too far from I

- If $\dfrac{\|p_i \cap I\|}{\|I\|} < \rho$ then $P' = P' - p_i$; if P' is empty goto step 2 otherwise goto step 3

Step 5 - Update the matched prototype vector

- Let $p_i = p_i \cap I$; output i ; goto step 2

The β acts as a tie-breaker, favouring larger magnitude prototype vectors when multiple prototype vectors are subsets of the input vector. This compensates for the fact that prototype

vectors can only move in one direction. The vigilance parameter defines the class sizes. When it is small it produces large classes. As it gets larger, the vigilance of the network increases, and finer classes are the result. When equal to one, the prototype vectors have to match the input vectors perfectly. In this situation every input vector produces a new class equal to itself. Also notice that in step 4 a form of contrast enhancement is performed. This means that clusters represented by a smaller magnitude prototype vector have a smaller variance of the vectors mapped to that cluster.

When implementing this algorithm it is necessary to deal with the restriction of limited memory resources. The following algorithm allocates a fixed amount of memory to work on, assuming that this will be enough. This corresponds to how the actual ART-1 network works. It has two major drawbacks. First of all one may not always know beforehand the maximum number of different clusters. And secondly, if this maximum is known, but very high, one may not want to allocate all the memory resources before they are really needed. To overcome both problems it is possible to begin with a small fixed amount of memory and whenever there is a shortage of unused prototype vectors to allocate another portion of memory.

ART-1 Network Algorithm

Step 1 - Initialisation

- Initialise N to the total number of clusters
- Initialise the vigilance parameter ρ so $0 < \rho \leq 1$
- Let p_i = (,) $\forall i \in$ [,]
- Initialise the set P of prototype vectors to { | }

Step 2 - Apply new input vector

- Let I:=[next input vector]
- Let P':=P be the set of candidate prototype vectors

Step 3 - Find the closest prototype vector from P'

- Find the i which maximizes $\frac{\|p_i \cap I\|}{\beta + \|p_i\|}$

Step 4 - Check if p_i is too far from I

- If $\frac{\|p_i \cap I\|}{\|I\|} < \rho$ then P′= P′ – p_i ; if P' is empty goto step 2 otherwise goto step 3

Step 5 - Update the matched prototype vector

- Let $p_i = p_i \cap I$; output i ; goto step 2

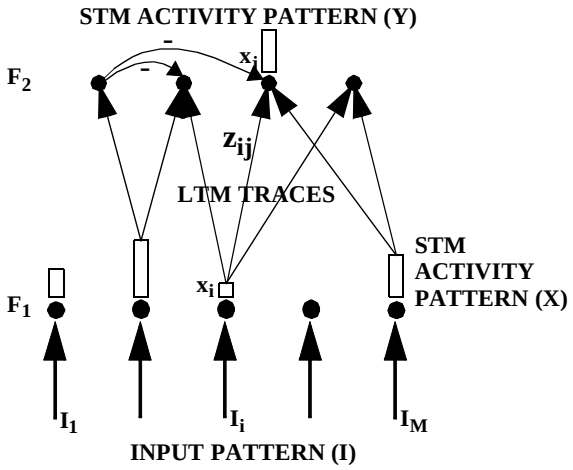
Though ART-1 is unsupervised it can sometimes be useful to add a limited amount of supervision by allowing the vigilance parameter to be changed externally. When for example the granularity of the clusters is not fine enough one can dynamically increase the vigilance parameter.

Paragraph 4 - Mechanics

The network algorithm presented in the previous chapter describes the dynamic behaviour of the ART-1 neural network. The different steps correspond with phases that can be distinguished when examining the behaviour of the network. In this chapter a closer look at these phases will be taken. Two inherent aspects of neural networks are that they are continuous and parallel: continuous in the sense that the activations of the nodes and the weights of the connections change continuously in time, parallel in the sense that these changes occur concurrently.

A class of neural networks often used for clustering is the class of competitive networks. First a particular competitive network will be described. After that the ART-1 network, which is in essence an extension of this competitive network, will be introduced.

Competitive Network



1. $x_i = \dfrac{I_i}{\sum_{k=1}^{M} I_k} \qquad (i = 1, ..., M)$

2. $T_j = \sum_{i=1}^{M} x_i z_{ij} \qquad (j = M+1, ..., N)$

3. $\dfrac{d}{dt} z_{ij} = \varepsilon x_j(-z_{ij} + x_i)$

A competitive network consists of two layers of nodes, the input layer F_1 and the output layer F_2. F_1 is fully connected with F_2 via weighted bottom-up connections called pathways. The set of pathways with corresponding weights is called an adaptive filter, adaptive because the weights can be changed dynamically to adapt to new input vectors. Patterns of activation of F_1 and F_2 nodes are called short term memory (STM) traces because they only exist during a single presentation of an input vector. The weights in the adaptive filter encode the long term memory (LTM) traces. LTM traces are equivalent to the prototype vectors in the previously discussed clustering algorithms.

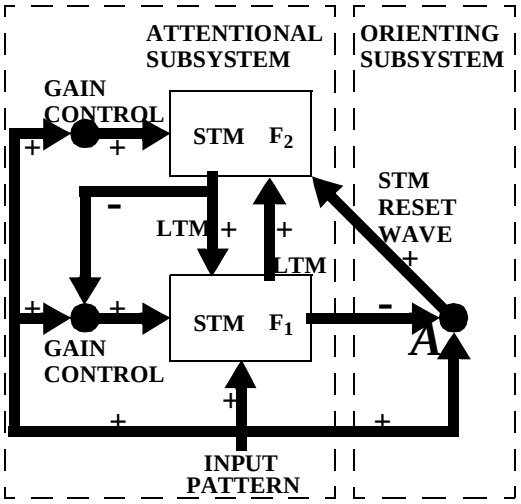
An input pattern that is presented to the network generates an activity pattern X at the F_1 layer. The F_1 activity pattern X is the normalized input pattern (eq.1), see [Grossberg76a] for how this can be implemented. This pattern is transformed by the weights in the pathways from F_1 to F_2. Each F_2 node receives as input pattern X multiplied by the weights in the pathways to that node (eq.2) which comprise the prototype vector corresponding to that node. The output node for which the dot product of the input vector and the prototype is largest represents the cluster which best matches the input vector. The F_2 layer is a competitive layer. Every node in this layer has inhibiting connections to the other nodes. As a result only the node with the largest input has an output. Finally the weights in the pathways are changed to accommodate the new input vector (eq.3). ε is a parameter which determines the speed of learning.

It can be shown, through explicit counterexamples, that this network is not stable, see [Carpenter87a]. This network thus clearly does not overcome the stability-plasticity dilemma. As mentioned in paragraph 2, ART was specifically designed in response to this problem.

A competitive network similar to the one just described can be augmented to obtain ART-1. A top-down adaptive filter and various components which modulate the working of the network are added.

ART-1 Network

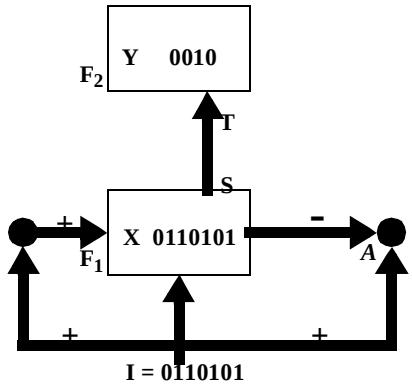
The ART-1 network self-organizes and self-stabilizes its recognition codes in response to arbitrary orderings of arbitrarily many and arbitrarily complex binary input patterns. In this paragraph a description of the ART-1 network will be given by following the phases which can be distinguished during the processing of a specific input pattern.



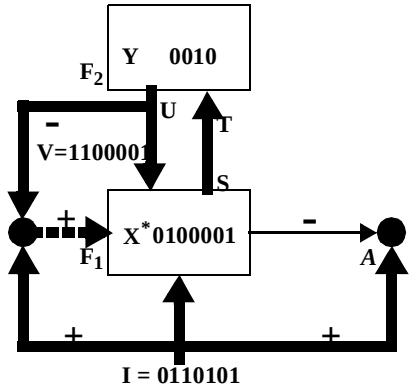
Two layers, F_1 and F_2, of the attentional subsystem encode patterns of activation in the STM traces. Bottom-up and top-down pathways between F_1 and F_2 contain LTM traces which multiply the signals in these pathways. The remainder of the circuit modulates these STM and LTM processes.

F_1 nodes are *supraliminally* activated (that is, sufficiently activated to generate output) if they receive a signal from at least two out of three possible input sources. The three are bottom-up input, top-down input and attentional gain control input. If a F_1 node receives input from only one of these sources it is *subliminally* activated. This is called the *2/3 rule*.

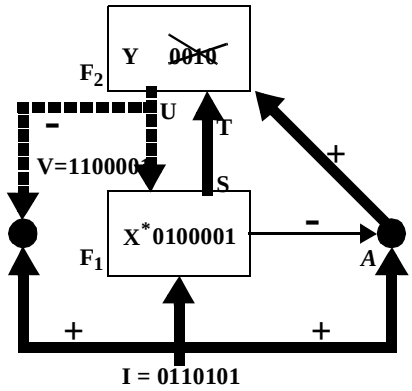
After the presentation of an input vector a parallel search is initiated. This is called the hypothesis testing cycle:



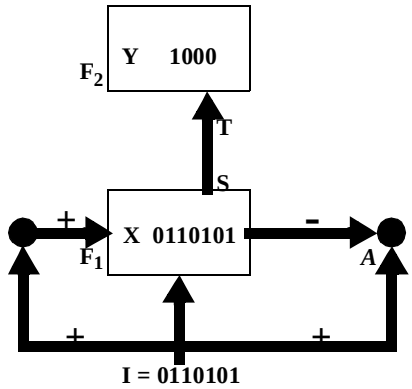
1. Input pattern *I* generates the STM activity pattern *X* at F_1 and activates both F_1's gain control and the orienting subsystem *A*. Pattern *X* both inhibits *A* and generates the bottom-up signal pattern *S* which is transformed by the adaptive filter into the input pattern *T*. F_2 is designed as a competitive network, only the node which receives the largest total input is activated ("winner-take-all"). This is step 3 of the network algorithm.



2. Pattern *Y* at F_2 generates the top-down signal pattern *U* which is transformed by the top-down adaptive filter into the expectation pattern *V*. Pattern *Y* also inhibits F_1's gain control. As a result only those F_1 nodes that represent bits in the intersection of the input pattern *I* and the expectation pattern *V* remain supraliminally activated. If *V* mismatches *I* this results in a decrease in the total inhibition from F_1 to *A*.



3. If the mismatch is severe enough (step 4 of the network algorithm) *A* can no longer be prevented from releasing a nonspecific arousal wave to F_2. This resets the active node at F_2. The vigilance parameter ρ determines how much mismatch will be tolerated.



4. After the F_2 node is inhibited its top-down expectation is eliminated and *X* can be reinstated at F_1. The cycle then begins again. *X* once again generates input pattern *T* to F_2, but a different node is activated. The previously chosen F_2 node remains inhibited until F_2's gain control is disengaged by removal of the input pattern.

The parallel search, or hypothesis testing cycle, repeats automatically at a very fast rate until one of three possibilities occurs: (1) a F_2 node is chosen whose top-down expectation approximately matches input *I*; (2) a previously uncommitted F_2 node is selected; or (3) the entire capacity of the system is used and input *I* cannot be accommodated. Until one of these outcomes prevails, essentially no learning occurs because all the STM computations of the hypothesis testing cycle proceed so quickly that the more slowly varying LTM traces in the bottom-up and top-down adaptive filters cannot change in response to them. Significant learning (step 5 of the network algorithm) in response to an input pattern occurs only after the cycle that it generates comes to an end and the system is in a *resonant* state.

The above description does not tell us *how* the components work. There are various ways to implement these. Guidelines in the form of mathematical equations are to be found in

[Carpenter87a]. A possible implementation is the one used in SNNS, the Stuttgart Neural Network Simulator (available via FTP from the Internet) which is further described in [Herrmann92].

Paragraph 5 - Adaptations

Since the introduction of ART-1 many adaptations have been made by Grossberg and Carpenter and more recently by various other researchers in the field of neural networks. About one year after ART-1 Grossberg and Carpenter introduced a variant which could handle continuous input vectors which they called ART-2. Since then they have introduced adaptations such as: ART-3 [Carpenter90], ART-2a [Carpenter91a], ARTMAP [Carpenter91b], Fuzzy ART [Carpenter91c] and Fuzzy ARTMAP [Carpenter92].

More recently various other researchers in the field of neural networks have introduced adaptations. For example, in 1994 Bartfai introduced a variant on ARTMAP which he called SMART [Bartfai94] which stands for Self-consistent Modular ART and is capable of bi-level clustering using two different vigilance parameters. At the moment he is working on HART [Bartfai95] which stands for Hierarchical ART and is capable of multi-level clustering. Another neural network architecture inspired by ART is CALM [Murre89] which stands for Categorizing And Learning Model.

A continually updated list of references to ART related publications can be found on the ART WWW site at http://www.wi.leidenuniv.nl/art/.

References

Bartfai, Guszti (1994) "Hierarchical Clustering with ART Neural Networks", Technical Report CS-TR-94/1, Department of Computer Science, Victoria University of Wellington, New Zealand, January 1994

Bartfai, Guszti (1995) "An ART-based Modular Architecture for Learning Hierarchical Clusterings", Technical Report CS-TR-95/3, Department of Computer Science, Victoria University of Wellington, New Zealand, February 1995

Carpenter, Gail, Grossberg, Stephen (1987a) "A Massively Parallel Architecture for a Self Organizing Neural Pattern Recognition Machine", Computer Vision, Graphics, and Image Processing, 1987, Volume 37, pp.54-115

Carpenter, Gail, Grossberg, Stephen (1987b) "ART 2: Self-organization of stable category recognition codes for analog input patterns", Applied Optics 26(23):4919-4930, December 1987.

Carpenter, Gail, Grossberg, Stephen (1990) "ART3: Hierarchical Search Using Chemical Transmitters in Self-Organizing Pattern Recognition Architectures", Neural Networks, 1990, Vol.3, pp.129-152

Carpenter, Gail, Grossberg, Stephen, Rosen, D.B. (1991a) "ART2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition", Neural Networks, 1991, Vol.4, pp.493-504

Carpenter, Gail, Grossberg, Stephen, Reynolds, J.H. (1991b) "ARTMAP: Supervised Real-Time Learning and Classification of Nonstationary Data by a Self-Organizing Neural Network", Neural Networks, Vol.4, 1991, pp.565-588

Carpenter, Gail, Grossberg, Stephen, Rosen, D.B. (1991c) "Fuzzy ART: Fast Stable Learning and Categorization of Analog Patterns by an Adaptive Resonance System", Neural Networks, Vol.4, 1991, pp.759-771

Carpenter, Gail, Grossberg, Stephen, Markuzon, N., Reynolds, J.H., Rosen, D.B. (1992) "Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps", IEEE Transactions on Neural Networks, Vol.3.No.5, pp.698-713, September 1992

Grossberg, Stephen (1976a) "Adaptive pattern classification and universal recoding. I. Parallel development and coding of neural feature detectors", Biol. Cybernet 23, 1976, pp. 121-134

Grossberg, Stephen (1976b) "Adaptive pattern classification and universal recoding. II. Feedback, expectation, olfacation, and illusions", Biol. Cybernet. 23, 1976, pp.187-202

Freeman, James A., Skapura, David M. (1991) "Neural Networks: Algorithms, Applications, and Programming Techniques", Chapter 8

Herrmann, K.-U. (1992) "ART - Adaptive Resonance Theory - Architekturen, Implementierung und Anwendung", diplomarbeit 929, IPVR, Universitat Stuttgart, 1992

Hertz, John, Krogh, Anders, Palmer, Richard G. (1991) "Introduction to the theory of neural computation", Paragraph 9.3

Hinton, G.E., Rumelhart, D.E., Williams, R.J. (1986) "Learning Internal Representations by Error Propagation", in Parallel Distributed Processing, Vol.1, McClelland, J.L., Rumelhart, D.E., Eds., pp.318-362, MIT Press

Moore, Barbara (1989) "ART 1 and Pattern Clustering", In proceedings of the 1988 Connectionist Models Summer School edited by Touretzky, D., Hinton, G., Sejnowski, T., Published by Morgan Kaufmann, San Mateo, CA, 1989, pp.174-185

Murre, J.M.J., Phaf, R.H., Wolters G. (1989) "CALM networks: a modular approach to supervised and unsupervised learning", Proceedings of the International Joint Conference on Neural Networks Washington DC, 1. New York: IEEE Press, pp.649-65

Appendix

```
/* Title       : Cluster-Euclidian
   Description: This program implements a simple pattern clustering algorithm for two-dimensional
                vectors using a Euclidian distance measure as described in [Moore89].
   Language   : ANSI-C
*/

#define BMWIDTH 320
#define BMHEIGHT 320
#define SQUARESIZE 300
#define ITER 40
#define THETA 1
#define LAMBDA 0.2
#define PI 3.141592654

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* Linked list containing prototype vectors */
struct proto
{
  double x,y;
  struct proto *next;
} *protos;
unsigned char *bm;

/* Plot a prototype vector */
void plotproto(x,y,last)
double x,y;
int last;
{
  int xx=BMWIDTH/2+(int)(x*(SQUARESIZE/2)),
      yy=BMHEIGHT/2-(int)(y*(SQUARESIZE/2));
  bm[(yy-1)*BMWIDTH+xx]=255;
  bm[yy*BMWIDTH+xx-1]=255;
```

```
  bm[yy*BMWIDTH+xx+1]=255;
  bm[(yy+1)*BMWIDTH+xx]=255;
  if (last)
    bm[yy*BMWIDTH+xx]=255;
}

/* Plot prototype vectors */
void display(last)
int last;
{
  struct proto *p;
  p=protos;
  while (p)
  {
    plotproto(p->x,p->y,last);
    p=p->next;
  }
}

/* Plot an input vector */
void plotinput(x,y)
double x,y;
{
  int xx=BMWIDTH/2+(int)(x*(SQUARESIZE/2)),
      yy=BMHEIGHT/2-(int)(y*(SQUARESIZE/2));
  bm[yy*BMWIDTH+xx]=255;
}

/* Incremental update */
void input(x,y)
double x,y;
{
  struct proto *p,*bestp;
  double dist,bestdist;

  plotinput(x,y);

  p=protos; bestp=NULL;
  while (p)
  {
    dist=sqrt((p->x-x)*(p->x-x)+(p->y-y)*(p->y-y));
    if (!bestp || dist<bestdist)
      { bestp=p; bestdist=dist; }
    p=p->next;
  }

  if (!bestp || bestdist>THETA)
  {
    p=malloc(sizeof(struct proto));
    p->x=x; p->y=y;
    p->next=protos;
    protos=p;
  }
  else
  {
    bestp->x=(1-LAMBDA)*bestp->x+LAMBDA*x;
    bestp->y=(1-LAMBDA)*bestp->y+LAMBDA*y;
  }
```

```
}

void main()
{
  FILE *outfile;
  long i;
  double angle;
  unsigned char *p;

  bm = calloc(BMWIDTH*BMHEIGHT,1);

  protos=NULL;
  for (i=0,angle=0; i<ITER/2; i++,angle+=PI/18)
  {
    input(cos(angle),sin(angle));
    input(cos(angle+PI),sin(angle+PI));
    display(i==ITER/2-1);
  }

  /* Write bitmap to Targa file. */
  outfile = fopen("euclid.tga","wb");
  putc(0,outfile); putc(0,outfile); putc(2,outfile);
  for (i=0;i<9;i++) putc(0,outfile);
  putc(BMWIDTH%256,outfile); putc(BMWIDTH/256,outfile);
  putc(BMHEIGHT%256,outfile); putc(BMHEIGHT/256,outfile);
  putc(24,outfile); putc(32,outfile);
  for (i=0,p=bm;i<BMWIDTH*BMHEIGHT;i++,p++)
  {
    putc(*p^255,outfile);
    putc(*p^255,outfile);
    putc(*p^255,outfile);
  }
  fclose(outfile);
  free(bm);
}
```