Lecture 2

Adaptive Resonance Theory
Neural Model - I
Contents

I. Introduction
II. Characteristics
III. Architecture
IV. ART Model
V. Procedures
VI. Other ARTs
Adaptive Resonance Theory (ART) was introduced as a theory of human cognitive information processing (Grossberg, 1976, 1980).

- The theory has since led to an evolving series of real-time NN models for unsupervised category learning and pattern recognition.
  - ARTs are capable of learning stable recognition categories in response to arbitrary input sequences with either fast or slow learning.
There is no guarantee that, as more inputs are applied to a NN, the weight matrix will eventually converge. The same problem exists for $k$-means or vector quantitation.

To overcome such a Stability-Plasticity Dilemma

- ART
  - Modified type of competitive learning

ART networks are NNs that self-organize stable recognition codes in real time in response to arbitrary sequences of input patterns.

ART networks are able to grow additional neurons if a new input cannot be categorized appropriately with the existing neurons.
ART networks consist of an input layer and an output layer.
- Bottom-up weights are used to determine output-layer candidates that may best match the current input.
- Top-down weights represent the ‘prototype’ for the cluster defined by each output neuron.

A close match between input and prototype is necessary for categorizing the input.

Finding this match can require multiple signal exchanges between the two layers in both directions until ‘resonance’ is established or a new neuron is added.

A vigilance parameter $\rho$ determines the tolerance of this matching process.
Motivations

Previous methods have the following problems:

1. The number of class nodes is pre-determined and fixed.
   - Under- and over-classification may result from training
   - Some nodes may have empty classes.
   - No control of the degree of similarity of inputs grouped in one class.
2. Training is non-incremental:
   - With a fixed set of samples
   - Adding new samples often requires re-train the network with the enlarged training set until a new stable state is reached.

ART-1: Stably learn to categorize binary input patterns presented in an arbitrary order (Carpenter & Grossberg, 1987a)
Characteristics

- A self-organizing NN architecture with unsupervised learning

- ART remains open to new learnings without washing away previously learned code

- ART maintains the plasticity required to learn new patterns, while preventing the modification of pattern that have been learned previously
ART is a vector classifier, which accepts input vector and categories depending upon which of stored pattern it matches most or resembles the most.

- If input does not match to any pattern, then a new category is created by storing this new pattern.

- No stored pattern is modified if input is not matched with any pattern.
Characteristics

- ART networks tackle the stability-plasticity dilemma

- Plasticity
  - They can always adapt to unknown inputs (by creating a new cluster with a new weight vector) if the given input cannot be classified by existing clusters.

- Stability
  - Existing clusters are not deleted by the introduction of new inputs (New clusters will just be created in addition to the old ones).

- Problem
  - Clusters are of fixed size, depending on vigilance parameter $\rho$. 
Adaptive Resonance Theory (ART)

- Self-organizing systems – *Unsupervised Learning*
  - Designed to handle Stability-Plasticity dilemma
Architecture

- ART architecture

\[ x: \text{input (input vectors)} \]
\[ y: \text{output (classes)} \]
\[ b_{j,i}: \text{bottom up weights from } x_i \text{ to } y_j \text{ (real values)} \]
\[ t_{i,j}: \text{top down weights from } y_j \text{ to } x_i \text{ (binary)} \]
\[ \rho: \text{vigilance parameter for similarity comparison (} 0 < \rho < 1 \) \]
There are two basic layers of computing units:
- Input layer and output layer

Input layer receives binary input vectors (for ART-1) from the input sites.

As soon as an input vector arrives, it is passed to input layer and from there to output layer.

Output layer contains elements which fire according to the ‘winner-takes-all’ method:
- Only the element receiving the maximal scalar product of its weight vector and input vector fires.
**ART Model**

- **Ideas of ART model:**
  - Suppose the input samples have been appropriately classified into $k$ clusters.
  - Each weight vector $b_j$ is a representative (average) of all samples in that cluster.
  - When a new input vector $x$ arrives
    1. Find the winner $j^*$ among all $k$ cluster nodes
    2. Compare $b_{j^*}$ with $x$
        if they are sufficiently similar ($x$ resonates with class $j^*$),
        then update $b_{j^*}$ based on $|x - b_{j^*}|$
        else
        create a free class node and make $x$ as its first member.
To achieve these, we need:

- A mechanism for testing and determining (dis)similarity between $x$ and $b_j^*$.  
- A control for creating new class nodes.  
- Need to have all operations implemented by units of local computation.

Only the basic ideas are presented:

- Simplified from the original ART model  
- Some of the control mechanisms realized by various specialized neurons are done by logic statements of the algorithm.
Procedures

- Procedure of ART
  - 3 phases after each input vector \( x \) is applied

1) Recognition phase

Determine the winner cluster for \( x \)

- Using bottom-up weights \( b \)
- Winner \( j^* \) with \( y_{j^*} = \max T(b_j, x) = b_{j^*} \cdot x \)
  - \( T(a, b) \) is a choice function
- \( x \) is tentatively classified to cluster \( j^* \)
- The winner may be far away from \( x \) (e.g., \( |t_{j^*} - x| \) is unacceptably large)
2) Comparison phase

- Compute similarity using top-down weights $t$
  Define vector $s^* = (s_1^*, \ldots, s_n^*)$
  with $s_i^* = M(t_{l,j^*}, x_l)$
  \[ t_{l,j^*} \cap x_l = \begin{cases} 1 & \text{if both } t_{l,j^*} \text{ and } x_l \text{ are 1} \\ 0 & \text{otherwise} \end{cases} \]
  where $M(\cdot)$ is a match function

- If $(\# \text{ of 1’s in } s^*)/(\# \text{ of 1’s in } x) > \rho \rightarrow \text{Resonance occurs}$
  - Accept the classification, update $b_{j^*}$ and $t_{j^*}$

- else
  - Remove $j^*$ from further consideration
  - Look for other potential winner or create a new node with $x$ as its first pattern.
3) Weight update/adaptive phase

- Initial weights for \( n \) input neurons: no bias such that
  bottom up: \( b_{j,l}(0) = 1/(n+1) \), and top down: \( t_{l,j}(0) = 1 \)
- When a resonance occurs with node \( j^* \), update \( b_{j^*} \) and \( t_{j^*} \)

\[
b_{j^*,l}(\text{new}) = \frac{s^*_l}{\varepsilon + \sum_{l=1}^{n}s^*_l} = \frac{t_{l,j^*}(\text{old}) x_l}{\varepsilon + \sum_{l=1}^{n} t_{l,j^*}(\text{old}) x_l}
\]

\[
t_{l,j^*}(\text{new}) = s^*_l = t_{l,j^*}(\text{old}) x_l
\]

- If \( m \) sample patterns are clustered to node \( j \), then
  \( t_j = \) pattern whose 1’s are common to all these \( m \) samples

  \[
t_j(1) = t_j(0) \land x(1) \land x(2) \ldots \land x(m) = x(1) \land x(2) \ldots \land x(m),
\]

\( b_{j,l}(\text{new}) \neq 0 \) iff \( s_l \neq 0 \) only if \( x_l(i) \neq 0 \),

\( b_j \) is a normalized \( t_j \)
Initialize each $t_{l,j}(0) = 1$, $b_{j,l}(0) = \frac{1}{n+1}$

While the network has not stabilized, do

1. Let $A$ contain all output nodes;
2. For a randomly chosen input vector $x$,
   Compute $y_j = b_j \cdot x$ for each $j \in A$.
3. Repeat (Until $A$ is empty or $x$ is associated with some node)
   (a) Let $j^*$ be a node in $A$ with largest $y_j$.
   (b) Compute $s^* = (s_1^*, ..., s_n^*)$ where $s_j^* = t_{l,j^*}x_l$
   (c) If $\frac{|s^*|}{||x||} \leq \rho$ then remove $j^*$ from set $A$
   else associate $x$ with node $j^*$ and update weights:
      \[
      b_{j^*,l}(\text{new}) = \frac{t_{l,j^*}(\text{old})x_l}{\varepsilon + ||t_{j^*}(\text{old}) \cap x||}
      \]
      \[
      t_{l,j^*}(\text{new}) = t_{l,j^*}(\text{old}) x_l
      \]
4. If $A$ is empty, create a new node with a weight vector $t = x$
End-while.
Assume that an ART-1 network with 7 input neurons \((n = 7)\) and initially one output neuron \((N = 1)\).

Let input vectors are

\[
\{(1, 1, 0, 0, 0, 0, 1), \\
(0, 0, 1, 1, 1, 1, 0), \\
(1, 0, 1, 1, 1, 1, 0), \\
(0, 0, 0, 1, 1, 1, 0), \\
(1, 1, 0, 1, 1, 1, 0)\}
\]

and the vigilance parameter \(\rho = 0.7\) and \(\varepsilon = 0.5\).

Initially, all top-down weights are set to \(t_{l,1}(0) = 1\), and all bottom-up weights are set to \(b_{1,l}(0) = 1/8\).
In the 1st epoch, for the 1st input vector, \((1, 1, 0, 0, 0, 0, 1)\), we get

\[
y_1 = \frac{1}{8} \cdot 1 + \frac{1}{8} \cdot 1 + \frac{1}{8} \cdot 0 + \frac{1}{8} \cdot 0 + \frac{1}{8} \cdot 0 + \frac{1}{8} \cdot 0 + \frac{1}{8} \cdot 1 = \frac{3}{8}
\]

Clearly, \(y_1\) is the winner as there is only one output node (no competitors).

Since we have

\[
\frac{\sum_{l=1}^{7} t_{l,1} x_l}{\sum_{l=1}^{7} x_l} = \frac{3}{3} = 1 > 0.7,
\]

the vigilance condition is satisfied and we get the following new weights:

\[
b_{1,1}(1) = b_{1,2}(1) = b_{1,7}(1) = \frac{1}{0.5 + 3} = \frac{1}{3.5} \\
b_{1,3}(1) = b_{1,4}(1) = b_{1,5}(1) = b_{l,6}(1) = 0
\]

\[\rightarrow B(1) = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 3.5 & 3.5 & 0 & 0 & 0 & 0 & 3.5 \end{bmatrix}^T\]

Also,

\[
t_{l,1}(1) = t_{l,1}(0) x_l, \quad l = 1, \ldots, 7
\]

\[\rightarrow T(1) = [1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1]^T\]
For the 2\textsuperscript{nd} input vector, \((0, 0, 1, 1, 1, 1, 0)\), we get
\[
y_1 = \frac{1}{3.5} \cdot 0 + \frac{1}{3.5} \cdot 0 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + \frac{1}{3.5} \cdot 0 = 0
\]
\(y_1\) is still the winner.
However, this time this does not reach the vigilance threshold:
\[
\frac{\sum_{i=1}^{7} t_{l,1} x_i}{\sum_{i=1}^{7} x_i} = \frac{0}{4} = 0 < 0.7.
\]
This means that we have to generate a 2\textsuperscript{nd} node in the output layer to represent the current input.
The top-down weights of the new node are identical to the current input vector because we set the initial value \(t_{l,2}(1) = 1\).
Then, we get the following updated weight matrices:
\[
B(2) = \begin{bmatrix}
\frac{1}{3.5} & \frac{1}{3.5} & 0 & 0 & 0 & 0 & \frac{1}{3.5} \\
0 & 0 & \frac{1}{4.5} & \frac{1}{4.5} & \frac{1}{4.5} & \frac{1}{4.5} & 0
\end{bmatrix}^T
\]
\[
T(2) = \begin{bmatrix}1 & 1 & 0 & 0 & 0 & 0 & 1\end{bmatrix}^T
\]
\[
0 & 0 & 1 & 1 & 1 & 1 & 0
\]
For the 3\textsuperscript{rd} input vector, \((1, 0, 1, 1, 1, 1, 0)\), we get

\[ y_1 = \frac{1}{3.5}; \quad y_2 = \frac{4}{4.5} \]

Here, \(y_2\) is the clear winner.

\(y_2\) satisfies the vigilance threshold:

\[
\frac{\sum_{l=1}^{7} t_{l,2} x_l}{\sum_{l=1}^{7} x_l} = \frac{4}{5} = 0.8 > 0.7.
\]

Therefore, we try the update of the 2\textsuperscript{nd} node’s weights.

In this case, the updates do not result in any weight changes at all:

\[
B(3) = \begin{bmatrix}
\frac{1}{3.5} & \frac{1}{3.5} & 0 & 0 & 0 & 0 & 0 & \frac{1}{3.5} \\
0 & 0 & \frac{1}{4.5} & \frac{1}{4.5} & \frac{1}{4.5} & \frac{1}{4.5} & 0 & 0
\end{bmatrix}^T
\]

\[
T(3) = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 0
\end{bmatrix}^T
\]
For the 4th input vector, \((0, 0, 0, 1, 1, 1, 0)\), we get
\[
y_1 = 0; \quad y_2 = \frac{3}{4.5}
\]

Here, \(y_2\) is the winner.

The vigilance test succeeds because
\[
\frac{\sum_{l=1}^{7} t_{l,2} x_l}{\sum_{l=1}^{7} x_l} = \frac{3}{3} = 1 > 0.7.
\]

Therefore, we update the 2nd node’s weights.

This gives the following new weight matrices:

\[
B(4) = \begin{bmatrix}
\frac{1}{3.5} & \frac{1}{3.5} & 0 & 0 & 0 & 0 & \frac{1}{3.5} \\
0 & 0 & 0 & \frac{1}{3.5} & \frac{1}{3.5} & \frac{1}{3.5} & 0
\end{bmatrix}^T
\]

\[
T(4) = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0
\end{bmatrix}^T
\]
Finally, the 5th input vector, \((1, 1, 0, 1, 1, 1, 0)\), gives 

\[ y_1 = \frac{2}{3.5}, \quad y_2 = \frac{3}{3.5} \]

Once again, \(y_2\) is the winner.

The vigilance test fails this time because

\[
\frac{\sum_{i=1}^{7} t_{i,2} x_i}{\sum_{i=1}^{7} x_i} = \frac{3}{5} = 0.6 < 0.7.
\]

This means that the active set \(A\) is reduced to contain the 1st node only, which becomes the uncontested winner.

The vigilance test fails for the 1st node as well:

\[
\frac{\sum_{i=1}^{7} t_{i,1} x_i}{\sum_{i=1}^{7} x_i} = \frac{2}{5} = 0.4 < 0.7.
\]
We thus have to create a 3\textsuperscript{rd} output neuron, which gives the weight matrices:

\[
b_{3,1}(5) = b_{3,2}(5) = b_{3,4}(5) = b_{3,5}(5) = b_{3,6}(5) = \frac{1}{0.5 + 5} = \frac{1}{5.5}
\]

\[
b_{3,3}(5) = b_{3,7}(5) = 0
\]

Also,

\[
t_{l,3}(5) = t_{l,3}(4)x_l = [1 \ 1 \ 1 \ 1 \ 1 \ 1] \in [1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0]
\]

Thus,

\[
B(5) = \begin{bmatrix}
\frac{1}{3.5} & \frac{1}{3.5} & 0 & 0 & 0 & 0 & 0 & \frac{1}{3.5} \\
0 & 0 & 0 & \frac{1}{3.5} & \frac{1}{3.5} & \frac{1}{3.5} & 0 \\
\frac{1}{5.5} & \frac{1}{5.5} & 0 & \frac{1}{5.5} & \frac{1}{5.5} & \frac{1}{5.5} & 0
\end{bmatrix}^T
\]

\[
T(5) = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 1 & 0
\end{bmatrix}^T
\]
In the 2\textsuperscript{nd} epoch, the 1\textsuperscript{st} input vector, \((1, 1, 0, 0, 0, 0, 1)\), gives

\[ y_1 = \frac{3}{3.5}; \quad y_2 = 0; \quad y_3 = \frac{2}{5.5} \]

Here, \(y_1\) is the winner, and the vigilance test succeeds:

\[
\frac{\sum_{l=1}^{7} t_{l,1} x_l}{\sum_{l=1}^{7} x_l} = \frac{3}{3} = 1 > 0.7.
\]

The 2\textsuperscript{nd} input vector, \((0, 0, 1, 1, 1, 1, 0)\), results in

\[ y_1 = 0; \quad y_2 = \frac{3}{3.5}; \quad y_3 = \frac{3}{5.5} \]

Here, \(y_2\) is the winner, and the vigilance test succeeds as follows:

\[
\frac{\sum_{l=1}^{7} t_{l,2} x_l}{\sum_{l=1}^{7} x_l} = \frac{3}{3} = 1 > 0.7.
\]

No update happens to the winners’ weights.
The 3rd input vector, \((1, 0, 1, 1, 1, 1, 0)\), gives

\[ y_1 = \frac{1}{3.5}; \quad y_2 = \frac{3}{3.5}; \quad y_3 = \frac{4}{5.5} \]

Once again, \(y_2\) is the winner, but this time the vigilance test fails:

\[
\frac{\sum_{l=1}^{7} t_{l,2}x_l}{\sum_{l=1}^{7} x_l} = \frac{3}{5} = 0.6 < 0.7.
\]

This means that the active set is reduced to \(A = \{1, 3\}\). Since \(y_3 > y_1\), the 3rd node becomes the new winner.

\[
\frac{\sum_{l=1}^{7} t_{l,3}x_l}{\sum_{l=1}^{7} x_l} = \frac{4}{5} = 0.8 > 0.7.
\]
Then, we get the following updated weight matrices:

\[
B(5) = \begin{bmatrix}
\frac{1}{3.5} & \frac{1}{3.5} & 0 & 0 & 0 & 0 & \frac{1}{3.5} \\
0 & 0 & 0 & \frac{1}{3.5} & \frac{1}{3.5} & \frac{1}{3.5} & 0 \\
\frac{1}{4.5} & 0 & 0 & \frac{1}{4.5} & \frac{1}{4.5} & \frac{1}{4.5} & 0
\end{bmatrix}^T
\]

\[
T(5) = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & 0
\end{bmatrix}^T
\]

For the 4th vector, \((0, 0, 0, 1, 1, 1, 0)\), the 2nd node wins, passes the vigilance test, but no weight changes occur.

The 5th vector, \((1, 1, 0, 1, 1, 1, 0)\), makes the 2nd unit win, which fails the vigilance test.

The new winner is the 3rd output neuron, which passes the vigilance test but does not lead to any weight modifications.

Further presentation of the five sample vectors leads to no weight changes; the network has thus stabilized.
Event encoding using ART network
- Encode five events each consisting of information on time, location and target object

Input vectors
- Time: 8:00 ~ 20:00
- Location: Living room, Balcony, Bedroom, Study room, Kitchen
- Target object: Flowerpot, Drawer, Food materials, Toys, Toaster, Plate

Events
- Event 1: Watering flowerpot
- Event 2: Bringing an object in the drawer
- Event 3: Taking a food material on the plate
- Event 4: Arranging toys
- Event 5: Toasting bread in a toaster
Input encoding

- Each node has a value of ‘0’ or ‘1’.
- $x_1, \ldots, x_{12}$: current time
  - $x_1$: 8:00~9:00, $x_2$: 9:00~10:00, ..., $x_{12}$: 19:00~20:00
- $x_{13}, \ldots, x_{17}$: current location
  - $x_{13}$: Living room, $x_{14}$: Balcony, $x_{15}$: Bedroom, $x_{16}$: Study room, $x_{17}$: Kitchen
- $x_{18}, \ldots, x_{23}$: detected target object
  - $x_{18}$: Flowerpot, $x_{19}$: Drawer, $x_{20}$: Food material, $x_{21}$: Toys, $x_{22}$: Toaster, $x_{23}$: Plate
ART Application Example

- Generated events

<table>
<thead>
<tr>
<th></th>
<th>Event 1</th>
<th>Event 2</th>
<th>Event 3</th>
<th>Event 4</th>
<th>Event 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>8:00~9:00</td>
<td>10:00~12:00</td>
<td>18:00~19:00</td>
<td>18:00~20:00</td>
<td>8:00~9:00</td>
</tr>
<tr>
<td>Location</td>
<td>Balcony</td>
<td>Living room, Study room</td>
<td>Kitchen</td>
<td>Living room, Study room</td>
<td>Kitchen</td>
</tr>
<tr>
<td>Target object</td>
<td>Flowerpot</td>
<td>Drawer</td>
<td>Food materials, Plate</td>
<td>Toy, Drawer</td>
<td>Toaster, Plate</td>
</tr>
</tbody>
</table>

- 100 events are randomly generated based on ground-truth events with error rate $p = 5\%$ (perturbation).
- ART network has incrementally learned using the 100 randomly generated events.
ART Application Example

- ART network for encoding events

Output layer
(Event)

Input layer

- Clustering result with 5 output neurons ($\rho = 0.4$ and $\varepsilon = 0.5$)

<table>
<thead>
<tr>
<th>Event</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Not decided</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of events</td>
<td>14</td>
<td>23</td>
<td>15</td>
<td>22</td>
<td>26</td>
<td>-</td>
</tr>
<tr>
<td>Clustering result</td>
<td>13</td>
<td>22</td>
<td>14</td>
<td>22</td>
<td>23</td>
<td>6</td>
</tr>
</tbody>
</table>
Notes

- Classification as a search process.

- Different ordering of sample input presentations may result in different classification.

- Increase of $\rho$ increases the number of classes learned, and decreases the average class size.

- Classification may shift during search, which will reach stability eventually.

- There are different versions of ART-1 with minor variations.
General Procedure of ART-1

Initialize each $T(0), B(0)$
While the network has not stabilized, do
1. Let $A$ contain all output nodes;
2. Compute $y_j = T(x, b_j)$ for each $j \in A$
   
   // $T(\cdot)$ is a choice function
3. Repeat (Until $A$ is empty or $x$ is associated with some node)
   (a) Let $j^*$ be a node in $A$ with largest $y_j$.
   (b) Compute $M(x, t_{j^*})$ for $j^* \in A$
   
   // $M(\cdot)$ is a match function
   (c) If $M(x, t_{j^*}) \leq \rho$ then remove $j^*$ from set $A$
   else associate $x$ with node $j^*$ and update weights:
      $b_{j^*,l}(\text{new}) = U_b(x, t_{j^*})$
      $t_{l,j^*}(\text{new}) = U_t(x, t_{j^*})$
   
   // $U_b(\cdot), U_t(\cdot)$ is an update function
4. If $A$ is empty, create a new node with a weight vector $x$
End-while.

Note that $T(\cdot), M(\cdot), U_b(\cdot), U_t(\cdot)$ have some variations.
General Procedure of ART-1

- ART-1 using bottom-up weight $b_{j,l}$ and top-down weight $t_{l,j}$:
  - $T(x, b_j) = b_j \cdot x$
  - $M(x, t_{j^*}) = \frac{||s^*||}{||x||}$, where $s_j^* = t_{l,j^*} \cdot x_l$ and $|| \cdot ||$: the number of ones in the vector
  - $U_b(x, t_{j^*}) = b_{j^*,l} (\text{new}) = \frac{t_{l,j^*}(\text{old})x_l}{\varepsilon + ||t_{j^*}(\text{old}) \cdot x||}$
  - $U_t(x, t_{j^*}) = t_{l,j^*} (\text{new}) = t_{l,j^*}(\text{old}) \cdot x_l$

- Equivalent to ART-1 using single weight $w_{l,j}$ [Baraldi & Parmiggiani]:
  - $T(x, w_j) = \frac{||w_j \cap x||}{\alpha + ||w_j||}$ where $\cap$ is a bitwise AND operator
  - $M(x, w_{j^*}) = \frac{||w_j \cap x||}{||x||}$
  - $U(x, w_{j^*}) = w_{j^*} (\text{new}) = w_{j^*}(\text{old}) \cap x$

- Mathematically proved that the bottom-up and top-down attentional module is equivalent to an attentional system with only forward connections
Other ARTs

- **ART-2** (Carpenter & Grossberg, 1987b)
  - Can classify the analog pattern
  - Preprocessing in the input layer:
    - Normalization, contrast enhancement, noise reduction
  - Comparison with input in the input layer: similar as ART-1
  - Dynamics of model is governed by differential equation
  - Applications
    - Recognition within the pictures
    - Recognition objects from the radar
    - Speech recognition

- **ART 3** (Carpenter & Grossberg, 1990a, 1990b)
  - To carry out parallel search, or hypothesis testing, of distributed recognition codes in a multilevel network hierarchy

- **ART 2-A and ARTEX**
Other ARTs

- **ARTMAP** (Carpenter, Grossberg, & Reynolds, 1991a,b)
  - A hierarchical network architecture designed by using ART-1, which can rapidly self-organize stable categorical mappings between \( m \)-dimensional input vectors and \( n \)-dimensional output vectors

- **Fuzzy ART** (Carpenter, Grossberg, & Rosen, 1991b)
  - Generalizes ART-1 to be capable of learning stable recognition categories in response to both analog and binary input patterns

- **Fusion ART**

- **EM ART**

- **Deep ART** (Park and Kim, 2016)