MONASH University

Information Technology

FIT5186 Intelligent Systems

Lecture 7

Unsupervised Learning with Adaptive Resonance Theory

Learning Objectives

- Understand
  - clustering applications using self-organising feature maps
  - the stability-plasticity dilemma problem with neural network learning
  - the adaptive resonance theory algorithm

Unsupervised Learning (Self-Organisation)

- In the last lecture, we looked at unsupervised learning and self-organisation
  - What it is (concept of competition versus cooperation)
  - Winner-take-all & counter-propagation networks
  - Feature mapping
  - SOFM algorithm
- This lecture
  - Clustering with SOFM
  - Adaptive Resonance Theory (ART)

Neural Clustering via SOFM

- The neural network approach to clustering is via the SOFM.
- While the effect is similar to the K-means algorithm, the procedure is quite different:
  - Neighbourhood structure
  - Weight updates
  - Global competition + local cooperation
- Both K-means and SOFM algorithms achieve clustering of data sets, and are useful.

Example

- Clustering of Real Estate data to find natural structures
  - See NeuroShell 2 example problem.
  - Useful to see if the asking price is reasonable based on characteristics of the house.
  - House price is not included as an input, but the NN classifies the data itself.
  - The resulting clusters tend to fall into the categories of low, middle and high price houses.

Example (continued)

- Open example REALTY.DSC.
- Click on advanced neural network.
- Click on data entry to see existing data file
  - Contains inputs like acres, floor area (sq. ft), # bedrooms, bathrooms, kitchen/eating area, other rooms and the price (although this will not be used).
- Click on Define Inputs and Outputs
  - Choose all inputs except Price.
  - Compute mins and maxs for data scaling.

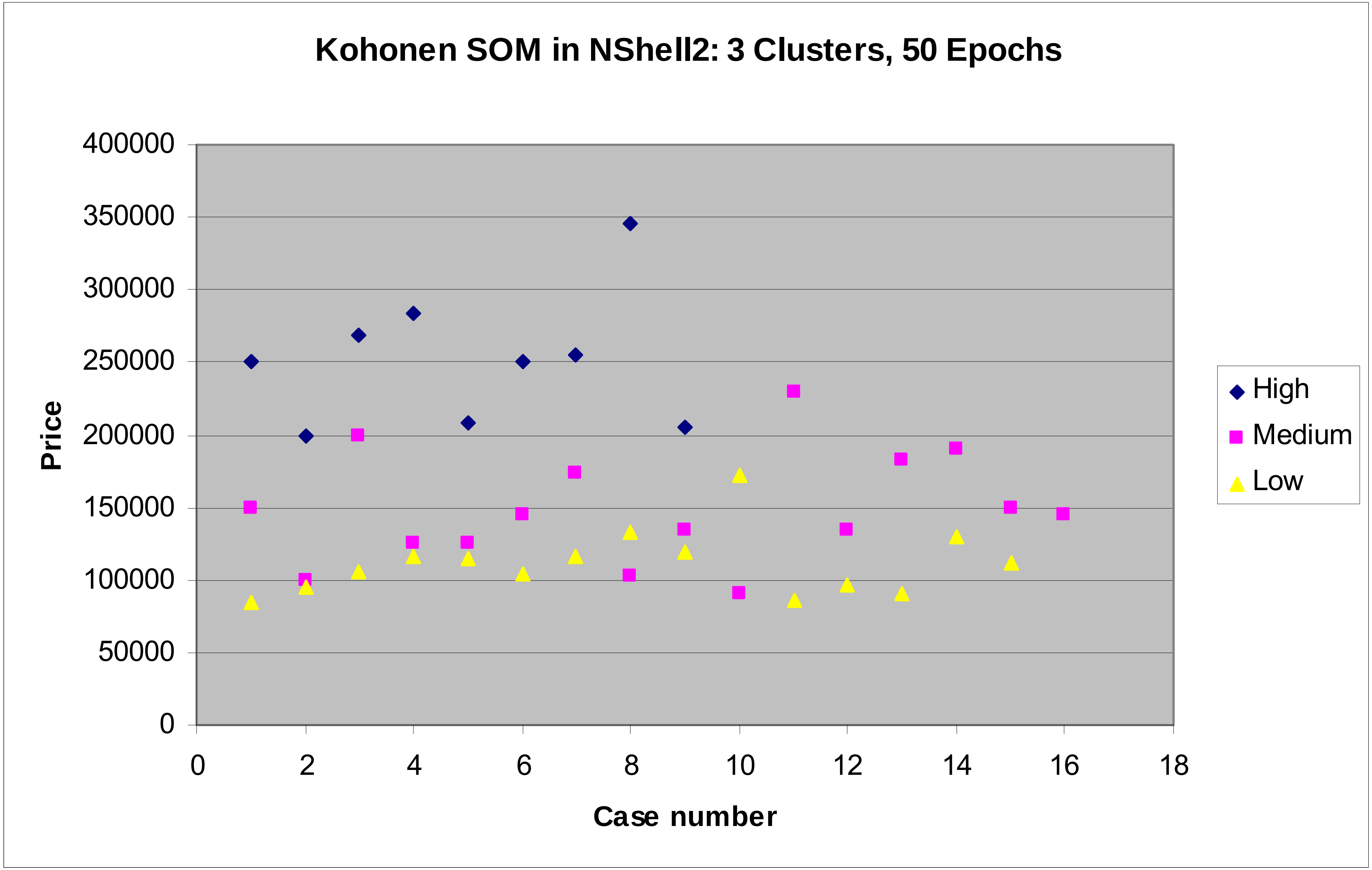
Example (continued)

- Click on Design, Architectures and Parameters, Unsupervised Kohonen.
  - This is the self-organising network.
- The number of inputs is 6 (based on data file), and the number of outputs is 3 (the number of clusters we are trying to find). This can be changed.
- Exit and open the Learning module.
- Start training and experiment with the initial neighbourhood size (=2) and learning rate (=0.5).

Example (continued)

- Once the network has learnt to cluster the data, open Apply to File.
- This passes all of the data through the network and produces a cluster number.
  - Set the maximum output to 1.
- Examine Data module
  - Comparing the cluster number to the original price of each house, it is clear that cluster 1 is high, cluster 2 is middle, and cluster 3 is low price.

Result of 1-Dimensional SOFM



Viscovery SOMine

- SOFM with great visualisation of results.
- You will be using this software (Trial Version) in Week 9 tutorial.
- Useful for understanding meaning of clusters.
- Good for explaining to non-technical people.

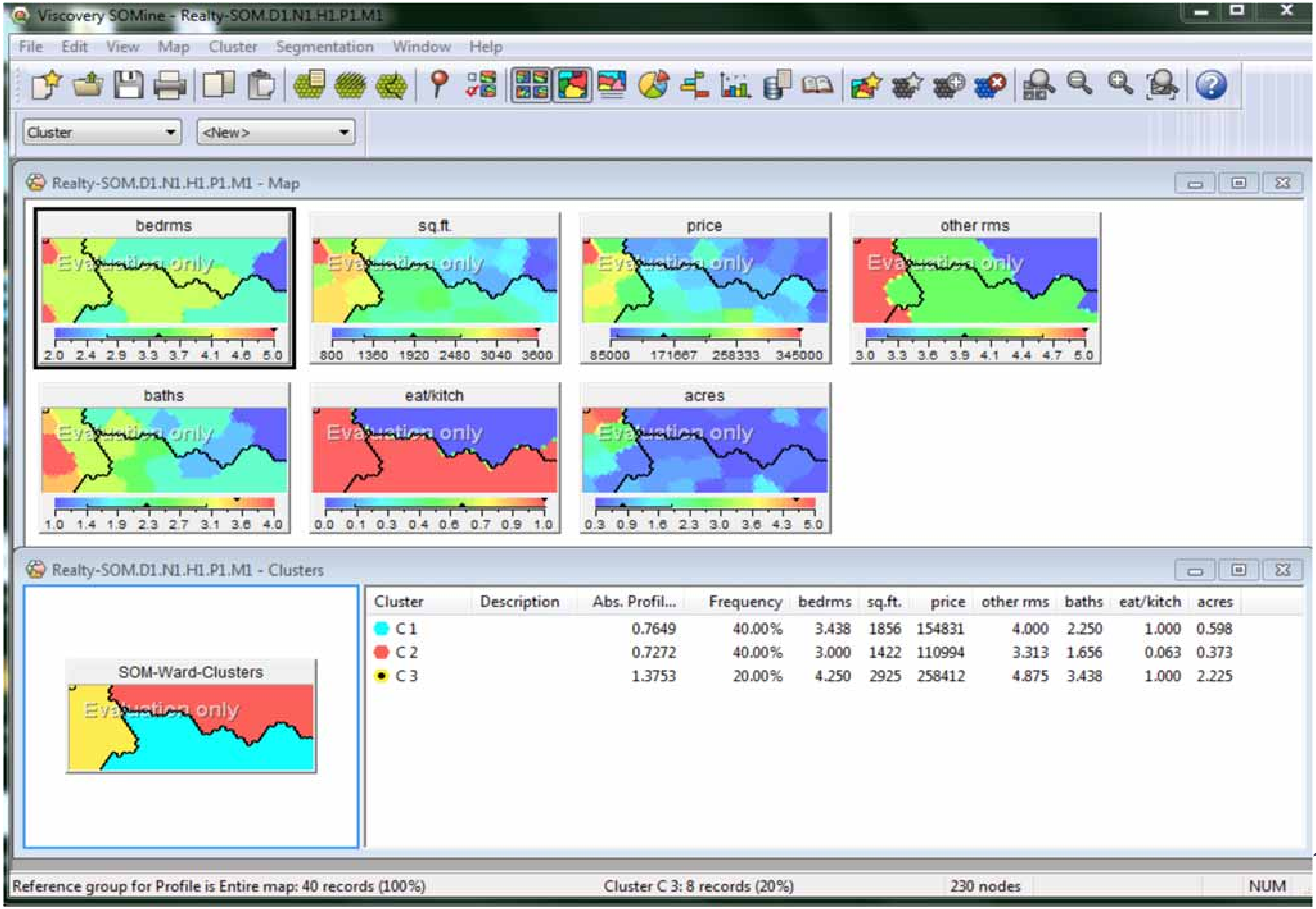
Realty Example in Viscovery SOMine

- Save data in the correct format (e.g. Excel)
- Open Viscovery SOMine/New problem.
- Point to Realty.xls.
- Select all inputs.
- Review preprocessing information and change where necessary.
- Start processing.

Realty Example in Viscovery SOMine
(cont'd)

- Window shows the progress of training.
- Map opens automatically.
- Show components (e.g. a separate map for each variable).
- Look for relationships between maps.
- High priced houses have more rooms, bathrooms, acres, etc.

Realty Example in Viscovery SOMine (cont'd)



Problem with NN Learning

- One of the good things about human memory is its ability to learn many new things without necessarily forgetting things learned in the past.
- The neural networks we have looked at so far do not have this ability.
  - They are extremely sensitive to new inputs, and often overwrite past learning.

Problem with NN Learning (continued)

- Consider a feedforward NN trained using backpropagation to classify upper case letters of the alphabet.
  - Once the network has been trained, suppose you decide that lower case letters should also be included, e.g. an "a" should be classified in the same group as an "A".
  - You cannot just present all of the lower case characters to the network (it will forget the decision boundaries learned for the upper case characters).
  - You have to present both the upper and lower case characters together.

Problem with NN Learning (continued)

- Consider a SOFM used to determine clusters for some data.
  - Once the weights have converged, you cannot add new data points to the set and train the network on those points alone, without the network forgetting the previous data points.
- There is no way for the network to know if it has or has not seen an input before.
- These networks are very sensitive to the last thing they are presented with.
- Retraining the network each time new inputs are needed is very time consuming.

Problem with NN Learning (continued)

- Under self-organising learning schemes, neurons compete with each other based on some criteria, and the winner is said to classify the input.
- Instabilities can arise when different neurons respond to the same input on different occasions.
- Later learning can wash away earlier learning if the statistical distribution of inputs is not stationary, or if new inputs arise.

Stability-Plasticity Dilemma

- This problem is known as the **stability-plasticity dilemma**.
  - How can a learning system remain adaptive (plastic) in response to significant inputs, yet remain stable in response to irrelevant inputs?
  - How does the system know to switch between its plastic and stable modes?
  - How does the system retain previously learned information while continuing to learn new things?

Stability-Plasticity Dilemma (continued)

- **Stability**: the system doesn’t change its behaviour in relation to irrelevant inputs.
- **Plasticity**: the system adapts its behaviour according to significant inputs.
- **Dilemma**: how to achieve stability without rigidity and plasticity without chaos?
  - Preservation of learned knowledge.
  - Ongoing learning capability.

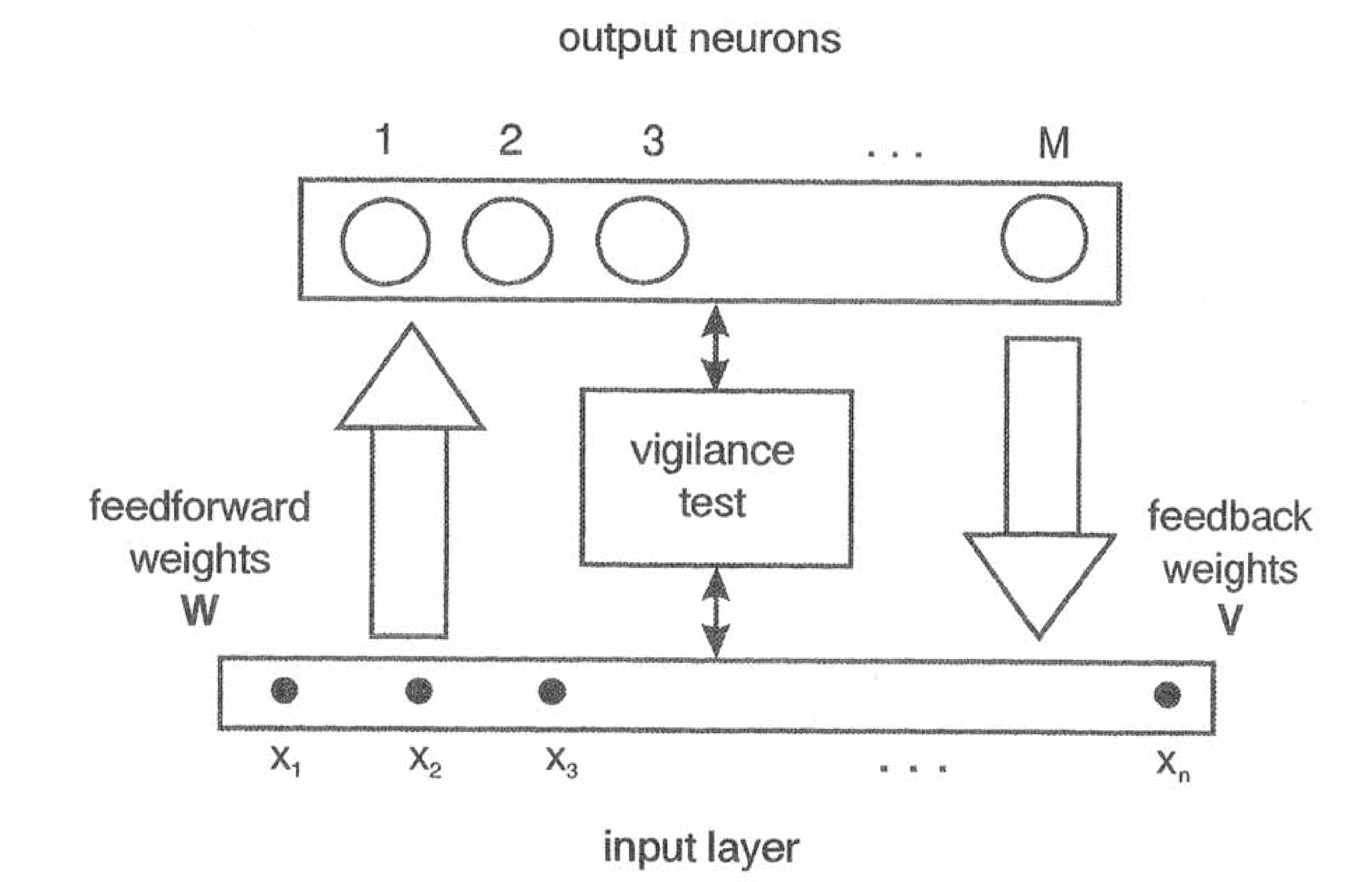
Solution to the Stability-Plasticity Dilemma

- Adaptive Resonance Theory (ART) provides the solution.
- It was developed by Grossberg and Carpenter in 1987.
- It is an extension of some of the unsupervised competitive learning schemes (self-organisation).
- It uses an adaptive form of self-organisation and introduces a mechanism for specifying how stable and plastic the neural network learning is.
- There are three different versions:
  - ART1: classification of binary input patterns.
  - ART2: classification of continuous input patterns.
  - ART3: more biologically real.

ART1 Network

- ART1 resolves the *stability-plasticity dilemma* by adding a feedback mechanism to the feedforward one between the input layer and the output layer.
  - See the next slide for architecture.
  - Its architecture consists of input layer (of dimension N), a (dynamically growing) output layer (of dimension M, where M is the number of clusters detected; M =1 initially).
  - There are weights connecting input to output (bottom-up weights **W** - continuous), and weights connecting output back to input (top-down weights **V** - binary).

Architecture for ART1 Network



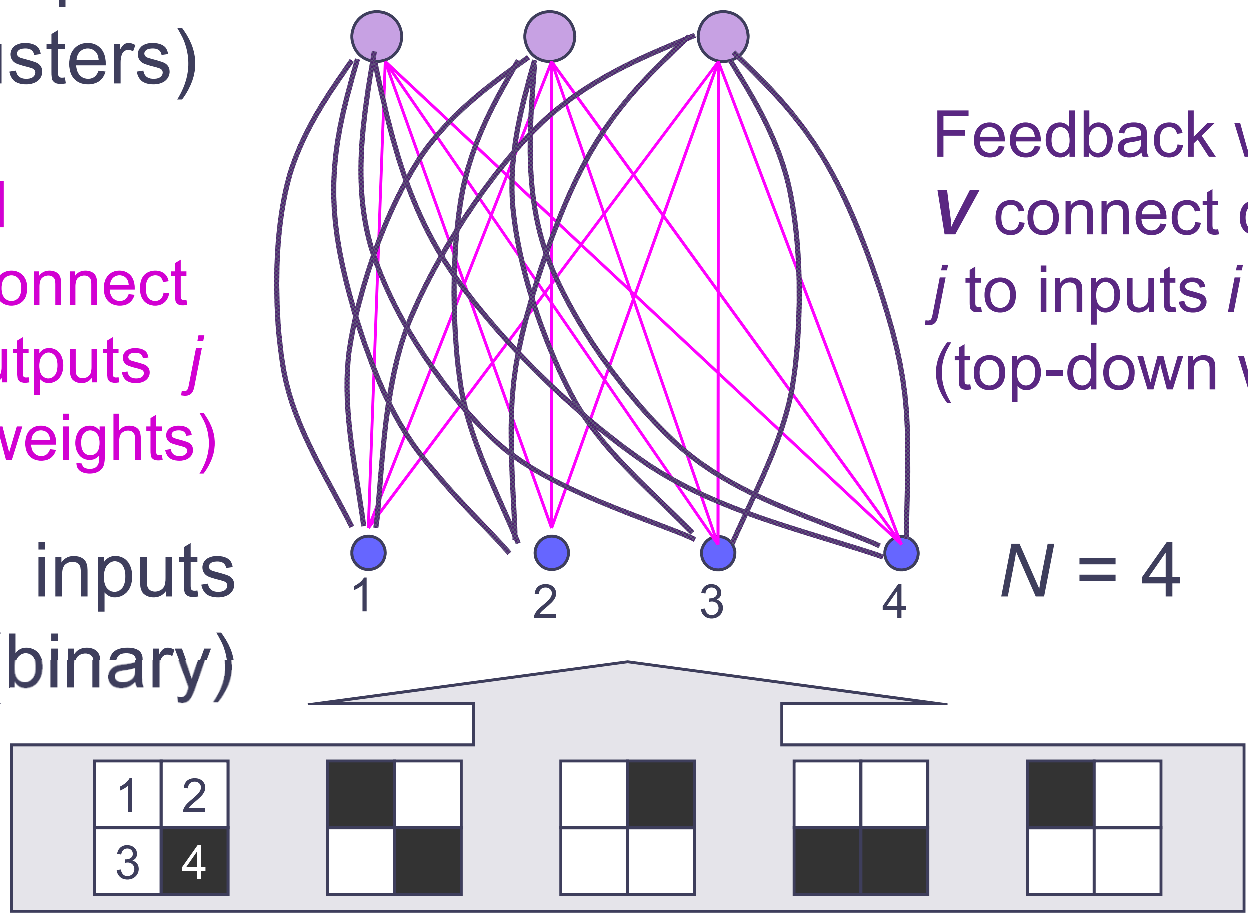
ART Architecture

j outputs
(clusters)

M = 3

Feedforward weights ***W*** connect inputs *i* to outputs *j* (bottom-up weights)

Feedback weights ***V*** connect outputs *j* to inputs *i* (top-down weights)



i inputs
(binary)

N = 4

ART1

- ART1 aims to discover clusters for binary input patterns in a controlled manner.
  - The network originates the first cluster with the first input.
  - It creates a second cluster only if the distance between the 2nd input and the first cluster exceeds a certain threshold; otherwise the 2nd pattern is clustered with the first cluster.
- This process of pattern inspection followed by either a new cluster creation or acceptance of the pattern to an old cluster is the main step of the ART1 algorithm.

ART1 (continued)

- The details of the cluster decision involve computing a "matching score" reflecting the degree of similarity of the present input to the previously established clusters.

- A winning cluster is identified using the bottom-up weights, which store the "average" (prototype) pattern for each cluster.

Choose neuron m such that

$$y_m = \max(y_j); \text{ where } \quad y_j = \sum_{i=1}^{N} w_{ij}(t)\, x_i$$

ART1 (continued)

- The winning neuron now passes its stored "exemplar" class pattern (stored in top-down weights **V**) back to the input layer.
  - We are going to see if the input pattern is "close enough" to this best matching cluster.
  - Having the average right is one thing; getting a close match with the 0-1 patterns is another.
- The winning cluster is said to be "good enough" if

pre-defined vigilance factor

dot product

$$\boxed{\frac{\|\mathbf{v}_j . \mathbf{x}\|}{\|\mathbf{x}\|} > \rho} \qquad (0 < \rho < 1)$$

ART1 (continued)

- By definition: $$\|\mathbf{x}\| = \sum_{i=1}^{N} |x_i|$$

  Remember that **X** is a vector of 0's and 1's (so just add up the number of 1's).

- The dot product of two vectors is:

  $$||\mathbf{V} \bullet \mathbf{X}|| = (v_1 x_1 + v_2 x_2 + \ldots + v_n x_n)$$

- ρ is called the **vigilance factor**.
  - A high value (say 0.99) means that the input pattern must match the exemplar pattern for the cluster almost exactly to join the cluster.
  - A low value (say 0.01) means that almost any input pattern is likely to pass the test.

ART1 (continued)

- If the test fails, then
  - Neuron *m* is disabled as the winner.
  - The next closest neuron (cluster) is calculated as the winner and the test is repeated.
  - If there are no existing clusters which pass the vigilance test, a new cluster is created (a new neuron is added to the output layer, with bottom-up and top-down weights).
- If the test is passed, then
  - Neuron *m* is considered “close enough” according to the specified vigilance level.
  - The weights (both bottom-up and top-down) are adapted to reflect the addition of the input to cluster *m.*

ART1 (continued)

- Weights are adapted according to:

$$w_{ij}(t+1) = \frac{v_{ij}(t)\,x_i}{0.5 + \sum_{i=1}^{N} v_{ij}(t)\,x_i}$$

$$v_{ij}(t+1) = x_i\, v_{ij}(t)$$

  - The weight subscripts are read as "input to output" for both **W** and **V**. ***i*** ***j***

- The weights **W** do not depend on their previous values, but depend on the **V**'s.
- The weights **V** act as the ***short term memory*** of each cluster, while weights **W** represent the ***long term memory*** (prototypes) of each cluster.

ART1 Algorithm

- Step 1: Choose a vigilance factor $0 < \rho < 1$, and initialise the weights according to:

$$\boxed{w_{ij} = \frac{1}{1+N} \quad \text{and } v_{ij} = 1}$$

for all $i = 1, 2, \ldots, N$ and $j = 1$.

- Step 2: Present a binary input vector **X** to the network.
- Step 3: Calculate matching scores and determine the winning neuron *m* which has the maximum net input:

$$\boxed{y_m = \max(y_j); \text{ where } \quad y_j = \sum_{i=1}^{N} w_{ij}(t) x_i}$$

ART1 Algorithm (continued)

- Step 4: Perform the vigilance test on the winning neuron:

$$\text{Is} \quad \frac{\| \mathbf{v}_m . \mathbf{x} \|}{\| \mathbf{x} \|} > \rho \ ?$$

  - If the test is passed, then go to Step 5.
  - If the test is failed, then disable neuron *m* as the winner and return to Step 3 (excluding neuron *m* from the competition).
  - If there are no more neurons left to consider in the Step 3 competition, create a new neuron and initialise as in Step 1, then go to Step 5.

ART1 Algorithm (continued)

- Step 5: Update the weight matrices **W** and **V** for the weights connecting the input layer to neuron (cluster) *m* (the neuron which either passed the vigilance test, or the newly created neuron):

$$w_{ij}(t+1) = \frac{v_{ij}(t)x_i}{0.5 + \sum_{i=1}^{N} v_{ij}(t)x_i}$$

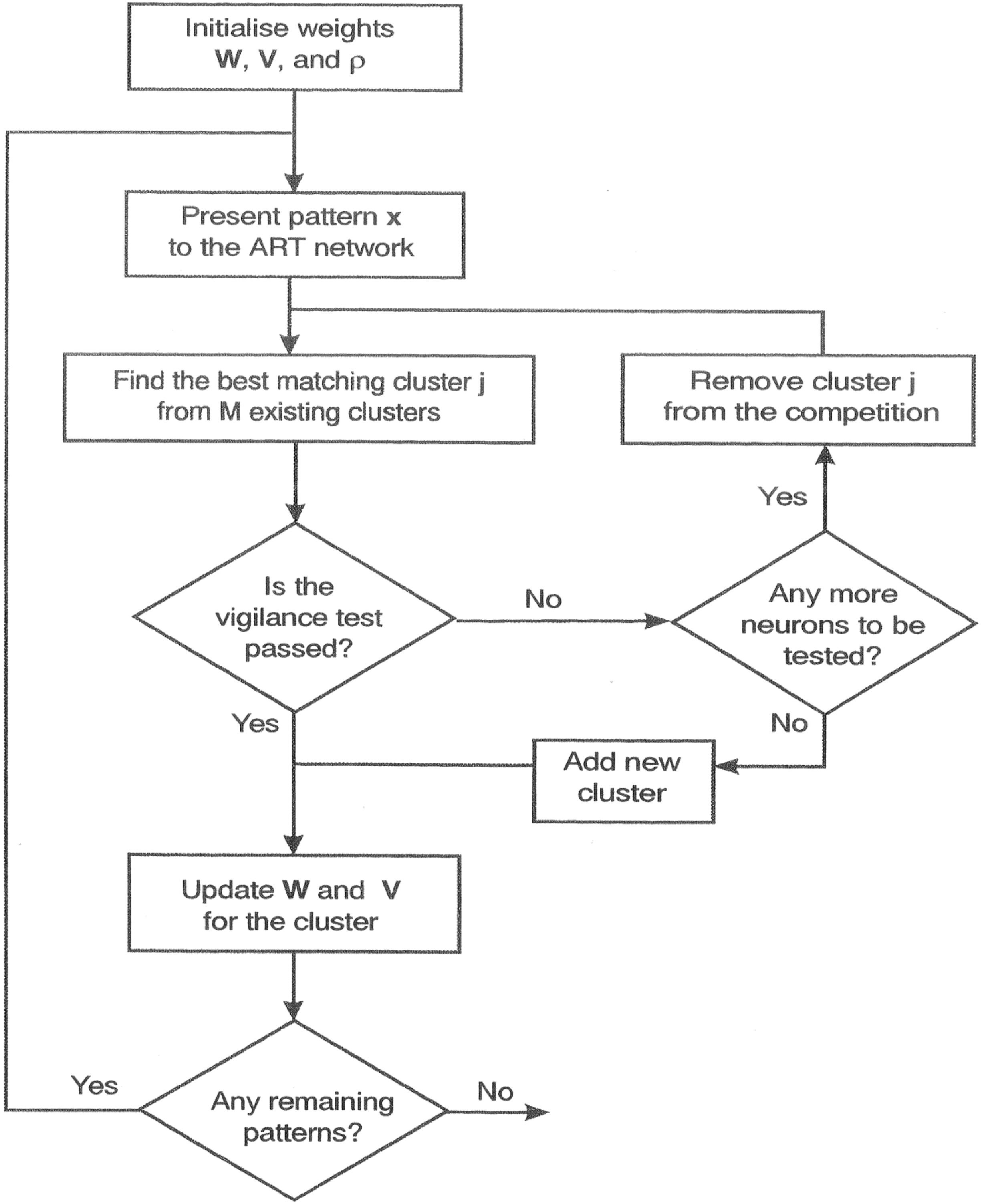
$$v_{ij}(t+1) = x_i v_{ij}(t)$$

If top line (the numerator of a *w*) = 0, then weights do not get adjusted (weights *w* should not be overwritten to zero).

- Step 6: Enable any disabled neurons and repeat from Step 2 with the next input, until all inputs have been presented.

- See the next slide for the flowchart of the algorithm.

Flowchart of ART1 Algorithm



Why Adaptive Resonance?

- In physics, resonance occurs when a small-amplitude vibration of the proper frequency causes a large-amplitude vibration.
- The name of the technique comes from the resonance that the system finds in Step 4 when the pattern possibly resonates with one of the patterns, say of cluster *j*, and reinforces the storage for this cluster.
- During this resonant period, learning or adaptation can occur.
- No learning occurs unless the input “resonates”.

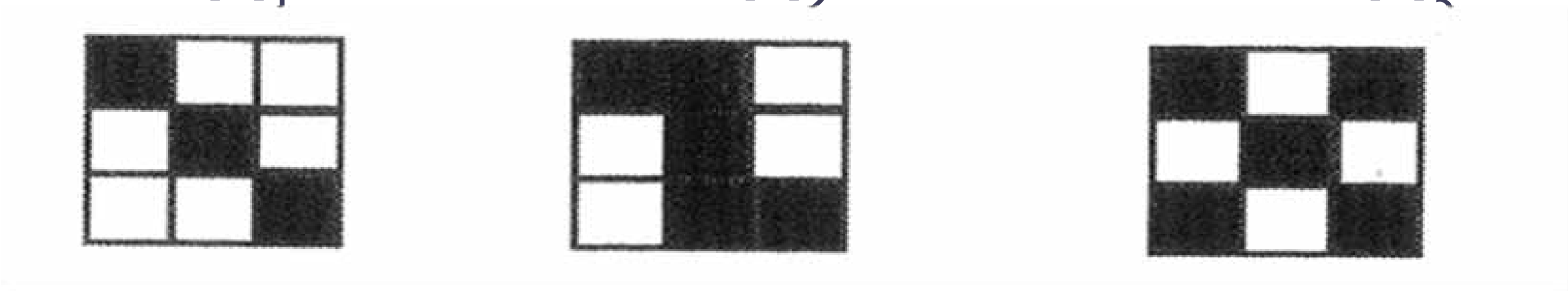
Example

- See "**Examples from Lecture 7**".
- To illustrate ART1, consider the three binary input vectors (1,0,0,0,1,0,0,0,1), (1,1,0,0,1,0,0,1,1) and (1,0,1,0,1,0,1,0,1)
  - See the next slide or "**Examples from Lecture 7 – Input Patterns**".
  - Input space is in 9 dimensions (so N = 9, M = 1).
- Perform the steps of the ART1 algorithm.
  - Select an appropriate vigilance factor so that all three patterns are classified separately when presented in the order above.
  - Compute the final weights **W** and **V**.

Example – Input Patterns

Three input patterns

X_1 X_2 X_3



can be represented as binary vectors using “1” for a black pixel, and “0” for a white pixel, as

$X_1 = (1,0,0,0,1,0,0,0,1)$

$X_2 = (1,1,0,0,1,0,0,1,1)$

$X_3 = (1,0,1,0,1,0,1,0,1)$

The Vigilance Factor in the Example

- The behaviour of ART1 is very dependent on the vigilance factor.
  - with $\rho < 0.6$, the common diagonal black pixels is enough for the network to decide that all 3 patterns are assigned to the same class.
  - with $\rho >= 0.6$, each pattern is examined in much more detail (with more vigilance), and it is determined that they are sufficiently different that each should belong to its own class.

The Vigilance Factor in ART1

- Lower vigilance factor (threshold)
  - Allows for more mismatch patterns.
  - Generates a smaller number of large clusters.
  - Misclassifications more likely.
- Larger vigilance factor (threshold)
  - Causes finer discrimination between classes.
  - Generates a greater number of small clusters.
  - Higher precision.

Summary of ART1

- The weights of ART1 represent the memory traces of the network.
  - The feedforward (bottom-up) weights **W** are the long term memories of the inputs.
  - The feedback (top-down) weights **V** are the short term (transient) states of the network; they store the activity produced during the processes of recognising and comparing an input.
- The vigilance factor is a way to solve the stability-plasticity dilemma.

ART2

- ART1 was designed to be able to classify binary inputs.
- ART2 is a (architectural) modification of ART1 which enables continuous and non-binary inputs to be classified.
  - Examples of non-binary inputs might be gray-scaled images where instead of representing an image as a series of black or white pixels, 255 shades of gray (from white through to black) are used.

ART3

- ART3 is an algorithmic modification of ART2 to make it more biologically plausible.
  - The architecture is the same as ART2 (which is similar to ART1, but has more complicated layer structure to enable real valued inputs).
  - The differential equations for short & long term memory have been modified to model the behaviour of chemical neuro-transmitters.
- ART2 and ART3 are not examinable.

Selected Reading

- Kohonen (1988). The Neural Phonetic Typewritter. *IEEE Computer*, 21, 11-21.
- Carpenter and Grossberg (1988). The ART of Adaptive Pattern Recognition by a Self-Organising Neural Network. *IEEE Computer*, 21, 77-88.
- Smith (1999). Chapter 5.

Week 7 Tutorial

- Using SOFM in NeuroShell 2 - Advanced Neural Networks
  - Example REALTY.DSC (Slides 5 – 8)
- Using SOFM in NeuroShell 2 to cluster data points.
  - Perform one epoch by hand calculations to make sure you understand the algorithm.
  - Use NeuroShell 2 to determine final weights.
  - Experiment with different neighbourhood sizes, epochs, and learning rates.