MONASH University

Information Technology

FIT5186 Intelligent Systems

Lecture 6

Unsupervised Learning – Clustering with Self-Organisation

Learning Objectives

- Understand
  - the principles of unsupervised learning and self-organisation
  - the basic clustering techniques
  - the self-organising feature map algorithm
  - competitive learning and cooperative learning

This Lecture

- Unsupervised Learning and Self-Organisation
- Clustering
- Winner-Take-All Learning
- Counter-Propagation
- Kohonen's Self-Organisation Feature Map
- Learning Vector Quantisation

What is Unsupervised Learning?

- The process of discovering relationships between inputs without any feedback from the environment during learning.
  - i.e. self-organisation
- The relationships discovered will be translated into outputs.
- We can use such networks to discover patterns, features, regularities or categories without a “teacher” (e.g. without even knowing what the initial categories were).

Unsupervised Learning

- These networks can learn by examining the inputs against some “similarity” measures.
- An important application in data processing is **clustering**.
  - The identification of groups or sections of the input data which are “similar” according to some criteria (attributes).
- Examples:
  - Group by similarity of number;
  - Group by similarity of colour;
  - Group by closeness in space;
  - Combine the above criteria (attributes).

Clustering

- Clustering serves several purposes:
  - Allows us to find unusual patterns in large data sets;
  - Pre-processing data to remove outliers;
  - Provides an alternative view for the data by allowing natural data structures and divisions to form;
  - Supervised learning can also be applied to data within a cluster, so that clustering effectively reduces the size of the training set.
    - e.g. using an MFNN model for each cluster.

Similarity Measures

- No information is available about the correct cluster for objects so use similarity measures.
- Distance is one important similarity measure.
  - the Euclidean distance between two points ***x*** (x_1,x_2) and ***y*** (y_1,y_2) is defined as:

$$\| \mathbf{x} - \mathbf{y} \| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

(Based on the Pythagoras Theorem)

  - In *N* dimensions ***x*** $(x_1, x_2, x_3, \ldots, x_n)$ and ***y*** $(y_1, y_2, y_3, \ldots, y_n)$:

$$\| \mathbf{x} - \mathbf{y} \| = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2}$$

Clustering Algorithms

(a) Traditional (Statistical)

- *e.g.* **K-means**: a statistical technique which tries to allocate data to K typical (or average) points – i.e. it forces K clusters.
- Each data point belongs to the cluster to which it is closest (in distance).

(b) Neural Networks

- e.g. **Self-Organising neural networks**:

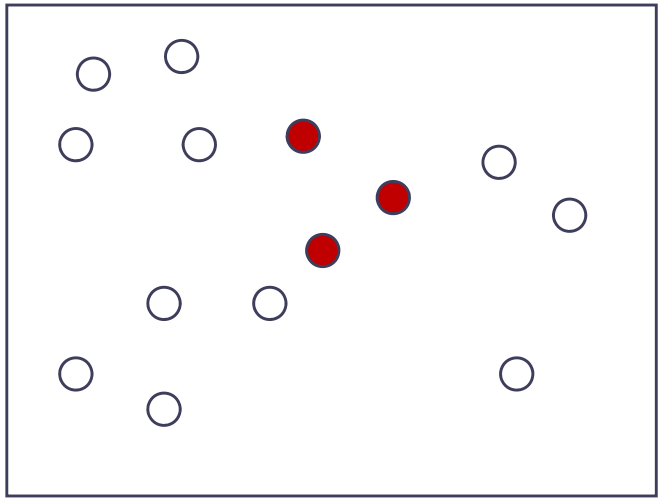
  Grouping similar points close together while placing dissimilar points as far apart as possible.

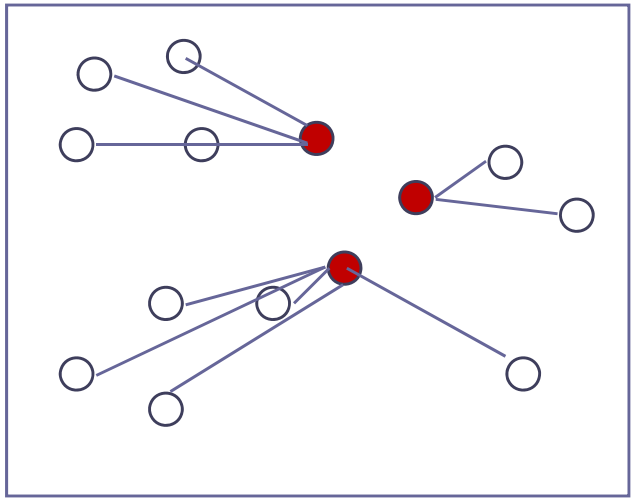
K-Means Algorithm

- Choose the number of clusters **K**, and their centres (randomly).
- Assign all data points to their closest cluster.
- Recalculate the centre of each cluster as the mean of all data points in that cluster (hence K-**Means**).
- Repeat until the new cluster centres and the mean of data points are sufficiently similar.
  - i.e. the new cluster centres are the same as or very similar to the previous ones.

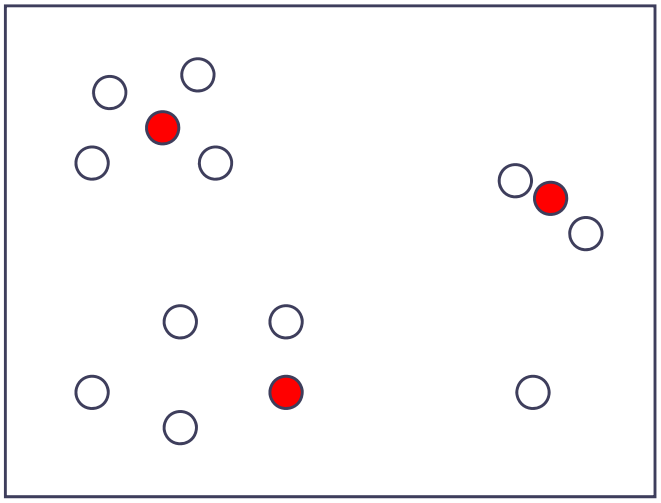
K-Means (for K=3 Clusters)

Choose K=3 random centres

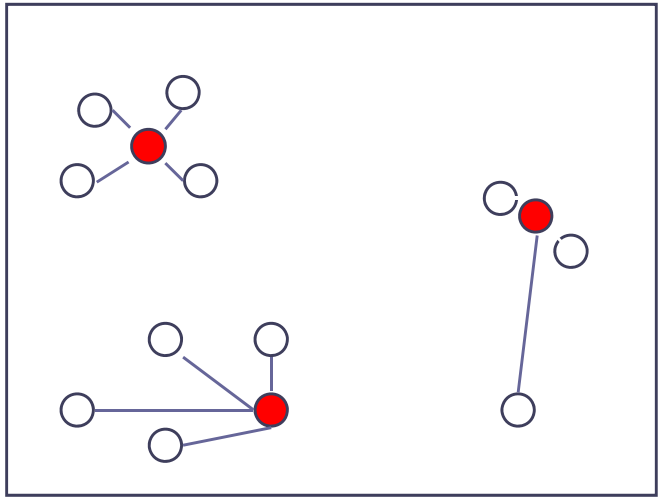




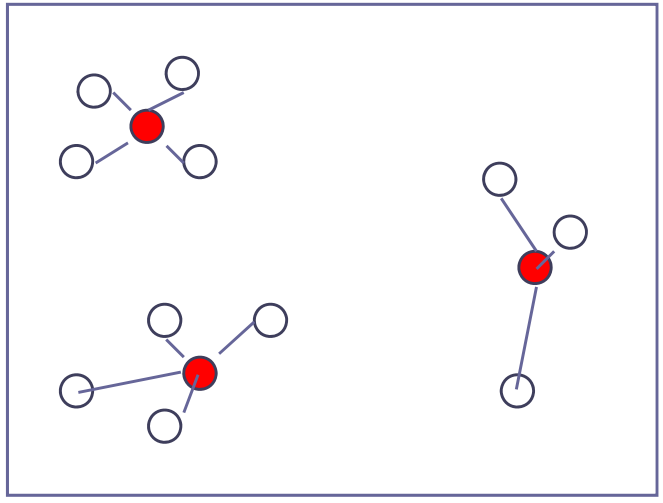
Assign data to centres



Move centres to mean of data



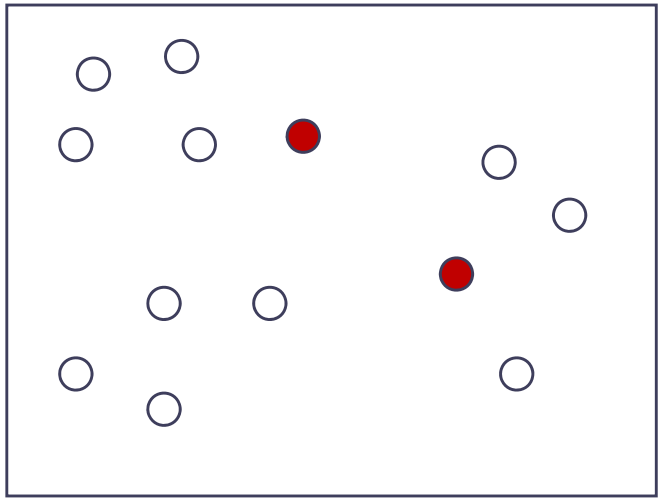
Re-assign data to new centres

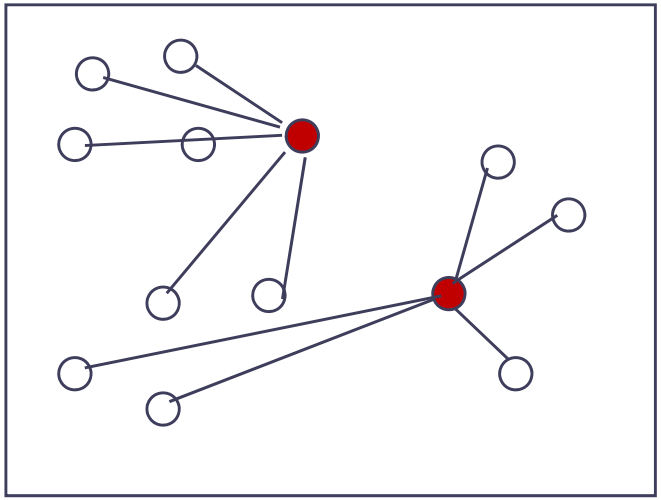


Move centres to mean of data

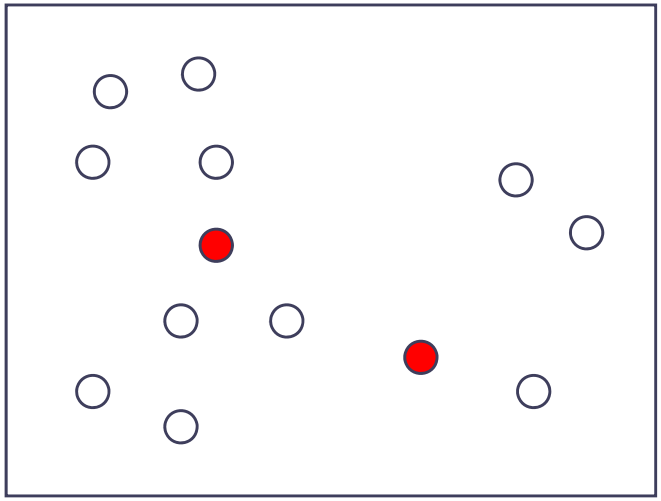
K-Means (for K=2 Clusters)

Random at first

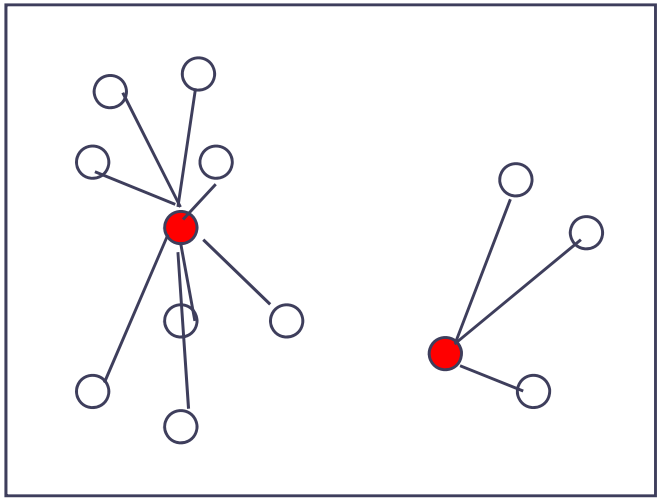




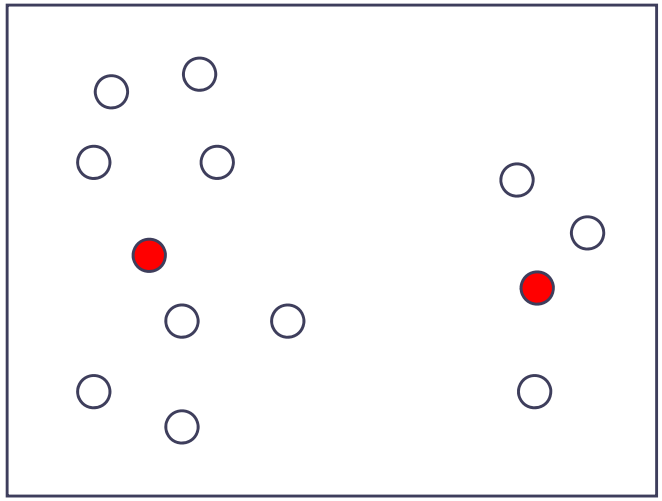
Assign data to clusters



Move clusters to centroids



Re-assign data to clusters



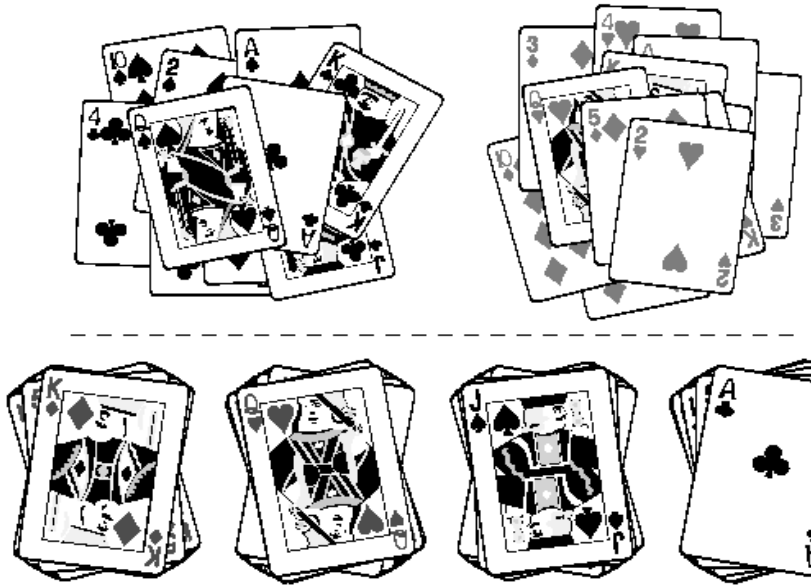
Move clusters to centroids

Implementing K-Means Algorithm

- The number of clusters K is chosen before stating the algorithm.
- The clusters can be positioned anywhere in the N-dimensional space, not necessarily at a data point.
- Experiments are often needed for finding the best number of clusters K, as well as the choice of initial K clusters.

Implementing K-Means Algorithm (continued)

- Clusters describe the underlying structure in data.
- Structures in data can exist at many different levels.
- In many cases, there is no a priori reason to select a particular value for K.
- The data set is tested for several K's to choose an appropriate value.
- Different values of K may lead to different clusters which are equally valid.



K = 2

K = 4

Neural Network Approaches

- Competitive clustering with unsupervised networks
  - Architecturally similar to supervised neural networks like MFNNs (where weights are updated to correct a known error in the outputs).
  - Weights are updated only to those neurons declared “winners” in a competition between output neurons based on which fire the most.
  - Different approaches exist, using only one neuron as “one winner” or a few neurons as “winners”.

Neural Network Approaches (continued)

- Competitive unsupervised networks
  - Winner-Take-All networks (WTA)
  - Counter-Propagation Networks (WTA + Backpropagation)
  - Self-Organising Feature Maps (SOFM/SOM)
  - Learning Vector Quantization (LVQ)
  - Adaptive Resonance Theory (ART) → Lecture 7

Winner-Take-All Networks

- A single winning neuron takes all of the learning.
- Assumes that the number of clusters is known (say *p*).
- Architecture is a single-layer feedforward neural network with continuous activation functions.
  - The number of inputs = the dimension of input patterns
  - The number of outputs = *p*
- Outputs are calculated, and the output neuron with the largest value is declared as the “winner”, denoted as neuron *m* (for maximum).

Winner-Take-All Networks (continued)

- The weights connecting each input dimension *i* to this winning neuron are then updated according to

$$w_{mi}(t+1) = w_{mi}(t) + c(x_i - w_{mi})$$

learning rate

  - All other weights are unchanged.
  - The winning neuron takes all the learning, hence "winner-take-all".
- It is not necessary to calculate the actual output, since **the winning neuron will also have the largest net input**; so just calculate the net inputs and take the largest as the winner. This is because the activation function is monotonically increasing.
- This process is known as **competitive learning**, since the neurons compete with each other to be declared the winner and receive the learning through weight adaptation.

Winner-Take-All Networks (continued)

- Weights should be initialised to random values.
- See "**Examples from Lecture 6**", p. 1.
- During learning, the weights of the *p* output neurons gradually move towards the centres of the clusters.
  - No learning if $w = x$ (i.e. the weights w of the winning neuron m is the same as the current input vector x).
  - The learning keeps moving the weights of a wining neuron towards the current input pattern.
- Because of the single layer architecture, the WTA network cannot cluster linearly non-separable data (the same reason as single layer Perceptrons).

Counter-Propagation Networks

- To overcome the nonlinear separability problem, counter-propagation networks (proposed by Hecht-Nielsen, 1987) introduce a hidden layer into the architecture.
- The hidden layer is a winner-take-all layer.
- The winning (hidden) neuron outputs a value of 1; all other hidden neurons output a value of zero.
- The output layer uses supervised learning with discrete activation functions.

Counter-Propagation Networks (continued)

- The output layer learning rule is supervised
  - Outstar learning rule (don't worry about details).
  - Just know that the Counter-Propagation network combines unsupervised and supervised learning to enable clustering of linearly non-separable data to be detected.
- Sometimes, the relationships cannot be measured in distance alone.
- Need to be able to extract similar features from input patterns.

Kohonen's Self-Organising Feature Maps

- When clustering complex data on a large scale, it is often better to cluster on the similarity of general features (i.e. capturing the overall structure of the data), rather than clustering based on the similarity of each component of the input pattern.
- Feature extraction has become an important and recognised application of self-organisation.
- Kohonen's Self-Organising Feature Maps (**SOFM**s) or Self-Organising Maps (**SOM**s) have become the principle neural technique for capturing the basic similarity of features within large data sets.

Feature Mapping

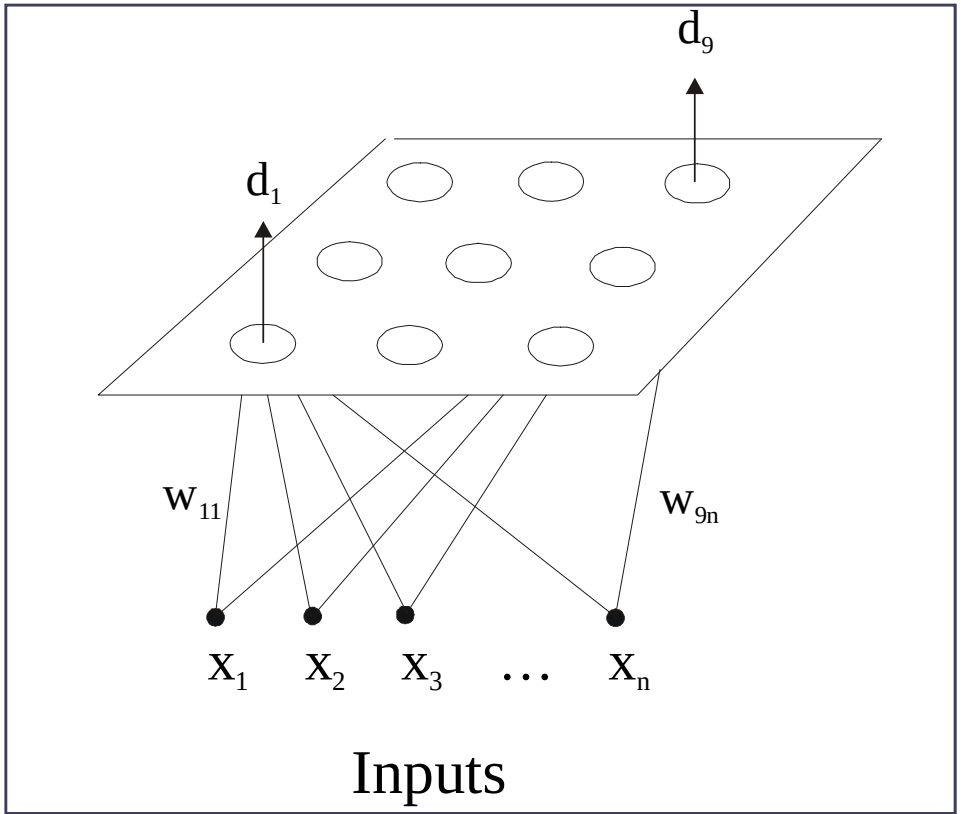
- Often patterns in the real world consist of many variables, but we are not interested in all of these.
  - There may be certain features though that we are hoping to detect or classify.
  - e.g. when examining our genes, we are all unique (so we would all represent a unique point in multidimensional “gene-space”), but we can mostly be classified into “races” based upon features like skin, hair and eye colour, etc.

Feature Mapping (continued)

- Patterns in multi-dimensional space may have a very complicated structure, but they often have a very simple structure when clustered in 1, 2 or 3-dimensional feature space.
- The goal of self-organising feature maps is to find a mapping from multi-dimensional input space to low-dimensional feature space, so that we can see the clusters and decision boundaries which form.

Self-Organising Feature Maps

- The input vectors are connected to an array of neurons (usually 1 dimensional (a row) or 2 dimensional (a rectangular lattice).
- A SOFM architecture with *n* inputs and a square array of 9 neurons:



Self-Organising Feature Maps (continued)

- When an input pattern is presented to the SOFM, certain regions of the array will “fire”, and the weights connecting the inputs to those regions will be strengthened.
- Once learning is complete, “similar” input patterns will fire the same regions.
- In this way, similar input patterns can be identified and grouped together or clustered.

The Importance of Ordering

- In previous neural networks we have studied such as MFNNs, the weights or connections between neurons has played a key role.
- In MFNNs, the location (physical arrangement) of each neuron within each layer has played no role for the out-going or in-coming connections.
- With SOFMs, the ordering and physical arrangement of the neurons are important since we are hoping to have regions of the array respond to certain inputs.

Biological Justification

- The cerebral cortex in the human brain is organised into regions which are responsible for different tasks.
  - When you read a book, different neurons fire compared to when you listen to music.
  - This structure makes the brain extremely fault tolerant (it has to be, since neurons are dying continually)
  - Each part of the brain performs feature extraction to some degree (we don't need all of the inputs we receive).

Biological Justification (continued)

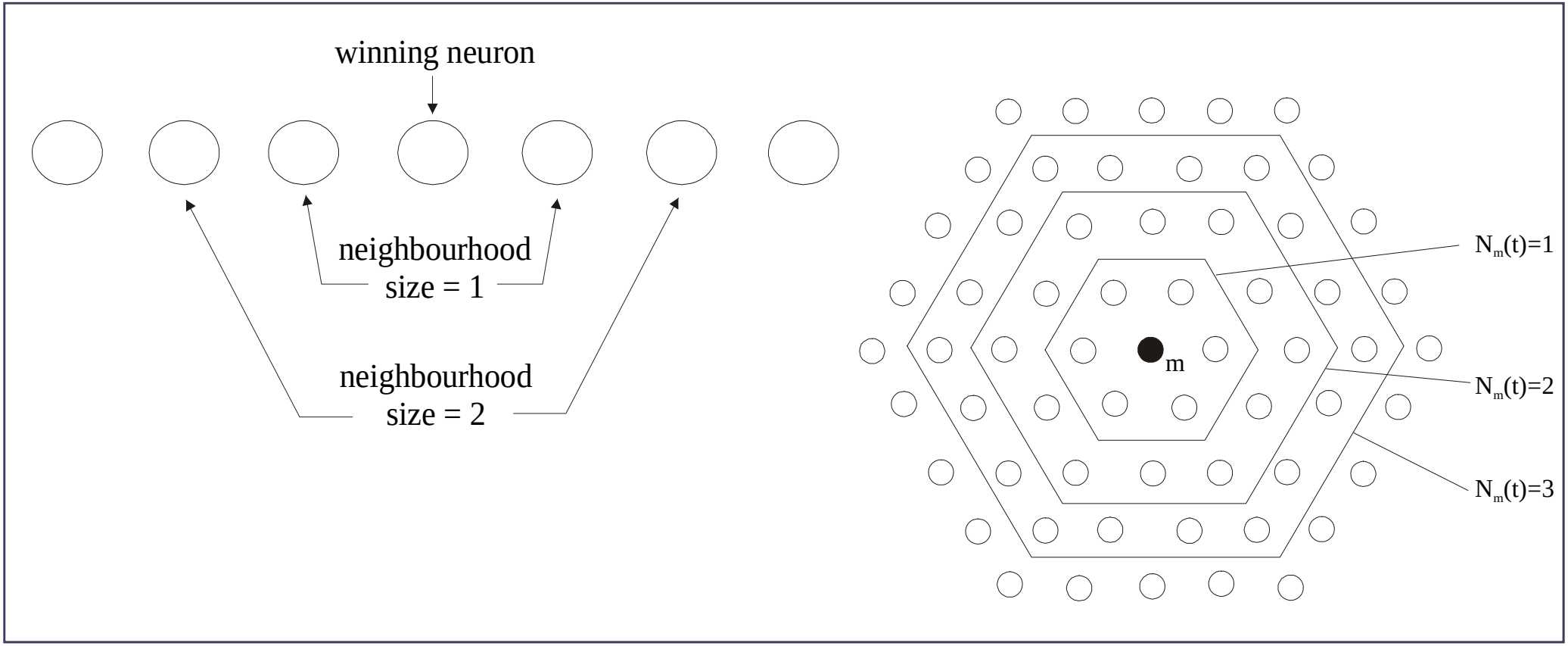
- When a part of the brain is stimulated by an input, an entire region of neurons usually “fire” (rather than just one neuron as we have considered thus far).
- The amount by which the neurons fire is not uniform.
  - There is usually a clear “winner” which fires the most, but the surrounding neurons also get affected by this, and the entire region becomes active.
- Similar ideas are used in SOFMs, and so they are said to be more biologically sound.

Neighbourhood

- In order to replicate this kind of response in a SOFM, during learning
  - The weights connecting the input space to the winning neuron are strengthened.
  - The weights of neurons in the “neighbourhood” of the winning neuron are also strengthened (although not as much).
- The structure of the neighbourhood depends on the structure of the array of neurons.

Neighbourhood (continued)

- For a linear array of neurons (left) or a hexagonal array of neurons (right), then the neighbourhood size can be defined as:



Neighbourhood (continued)

- Initially the neighbourhood size is defined to be a large region around the winning neuron, but as learning proceeds, the neighbourhood size is slowly decreased.
  - This results in more localised responses.
- The localised response is needed to help clearly differentiate distinct input patterns.
  - This is enforced by varying the amount of learning received by each neuron within the winning neighbourhood.
- The winning neuron receives the most learning at any stage, with its neighbouring neurons receiving less the further away they are from the winning neuron.

Neighbourhood (continued)

- The size of the neighbourhood around winning neuron *m* at time *t* is denoted by $N_m(t)$.
- The amount of learning that every neuron *i* within the neighbourhood of *m* received is determined by the learning factor *c*:

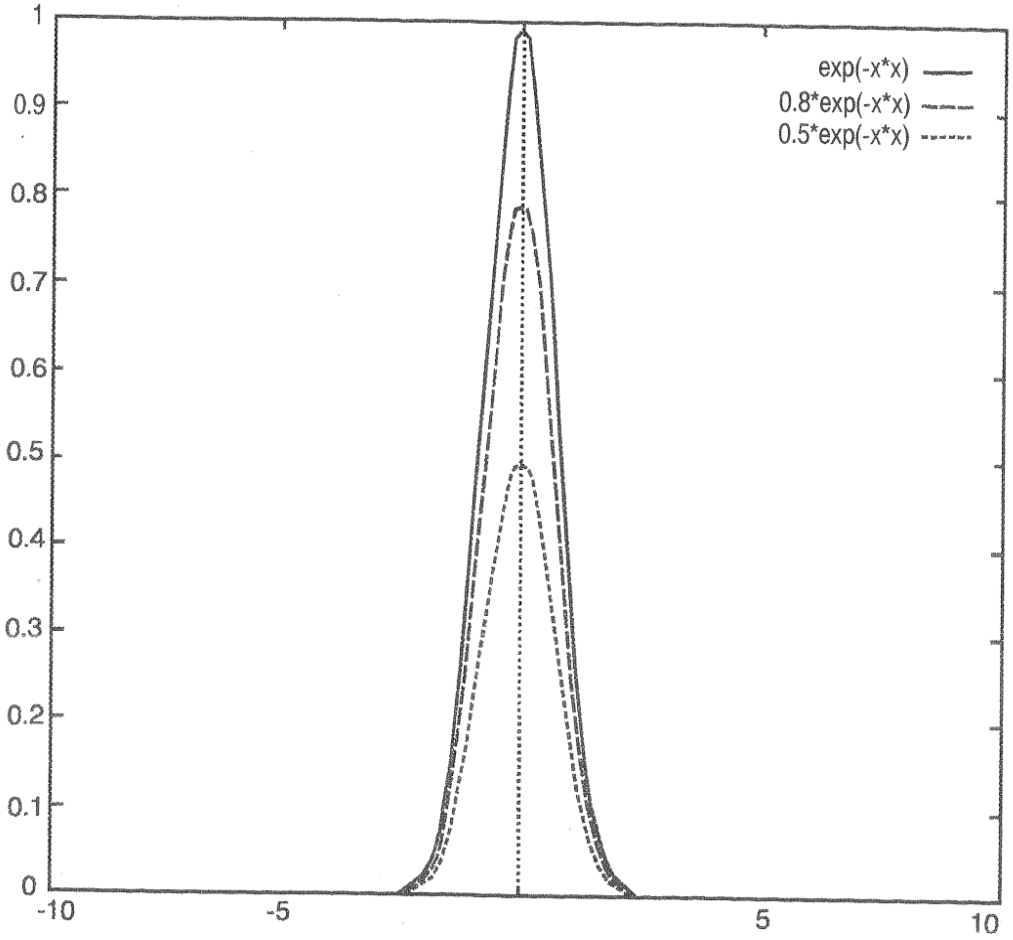
$$c(N_i(t)) = \alpha(t)\exp(-\| r_i - r_m \| / \sigma^2(t))$$

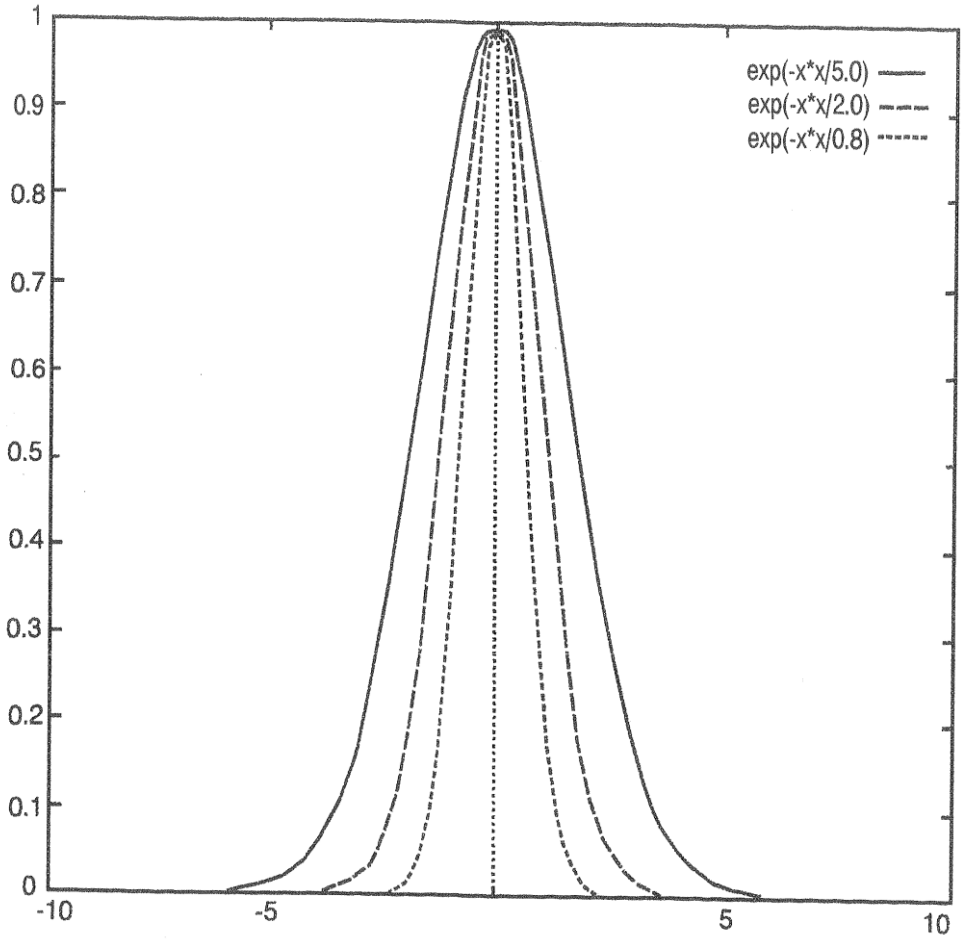
where $||r_i - r_m||$ is the physical distance (number of neurons) between neuron *i* and the winning neuron *m*.

- Two functions $\alpha(t)$ and $\sigma^2(t)$ are used to control the amount of learning each neuron *i* receives at time *t*.

Neighbourhood (continued)

- The following figure shows the effect of the two functions $\alpha(t)$ (left) and $\sigma^2(t)$ (right) on learning, where zero on the horizontal axis denotes the position of the winning neuron.





Neighbourhood (continued)

- The two functions $\alpha(t)$ and $\sigma^2(t)$, can be slowly decreased over time, with an initial value and rate of decrease.
- The amount of learning (determined by the learning factor) is the greatest at the winning neuron (where $i = m$ and $r_i = r_m$), and decreases the further away a neuron is from this winning neuron (resulting from the exponential function).
- Neurons outside the neighbourhood of the winning neuron receive no learning. The effect is enhanced by the two functions.

Learning in SOFMs

- Basic (common) learning steps (same as other NN models)
  - Presenting input patterns.
  - Calculating neuron outputs.
  - Updating weights.
- Special learning features of SOFMs
  - Using similarity (distance) measure to calculate the neuron output.
  - Using the concept of a neighbourhood of weight updates.

SOFM Algorithm

- Step 1: Initialise
  - the weights to small random values.
  - the neighbourhood size $N_m(0)$ to be large (but less than the number of neurons in the array).
  - the parameter functions $\alpha(t)$ and $\sigma^2(t)$ to be between 0 and 1.
- Step 2: Present an input pattern *x* through the input layer and calculate the "similarity" (distance or closeness) of this input to the weights of each neuron *j*:

$$d_j = \| \mathbf{x} - \mathbf{w}_j \| = \sqrt{\sum_{i=1}^{N} (x_i - w_{ij})^2}$$

SOFM Algorithm (continued)

- Step 3: Select the "winning neuron" (m) which has the maximum similarity (minimum distance)

$$s_m = \min(d_j) \quad \text{for all neurons } j \text{ in the array}$$

- Step 4: Update the weights connecting the input layer to the winning neuron, and all neurons within the neighbourhood of m based on the learning rule:

$$w_{ji}(t+1) = w_{ji}(t) + c(N_j(t))[x_i - w_{ji}(t)]$$

$$c(N_j(t)) = \alpha(t)\exp(-\|r_j - r_m\| / \sigma^2(t)) \text{ for all neurons } j \text{ in } N_m(t)$$

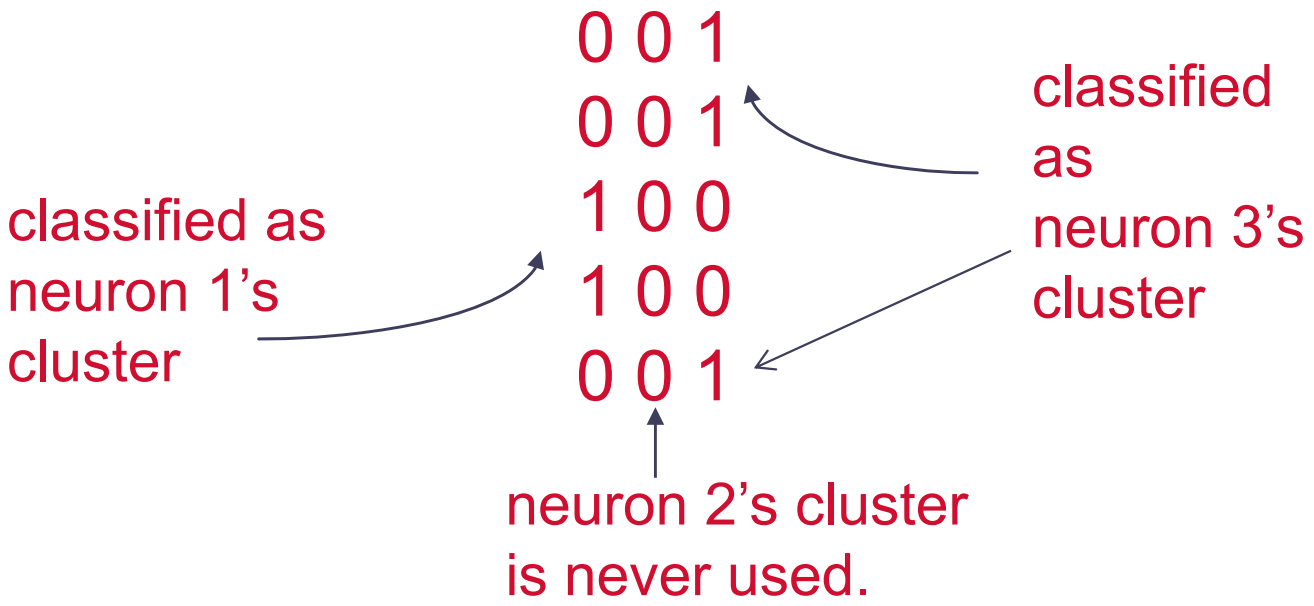
- Step 5: Continue to present inputs (from Step 2) for Ω epochs; then decrease the neighbourhood size $N_m(t)$, $\alpha(t)$ and $\sigma^2(t)$: Repeat until weights have stabilised.

Example

- Consider again the clustering example used for the WTA algorithm ("**Examples from Lecture 6**", p. 1)
  - What if we don't know the number of clusters (*p*)?
  - Let's use a one dimensional array of 3 neurons (each connected to the 2 dimensional input vector).
  - Of course, much larger problems are more interesting - but you need to know how to apply the algorithm by hand (hence small examples).
  - See "**Examples from Lecture 6**", p. 2.

Example – Using NeuroShell 2

- NeuroShell 2 can also perform SOFM (in advanced / architectures), but is restricted to a linear (1 dimensional) array.
- Solving this same problem using NeuroShell 2, gives a (winner-take-all) output for each of the 5 inputs (the same order as in the example) as:



i.e. The SOFM has detected $p = 2$ clusters.

Competition versus Cooperation

- The SOFM demonstrates ***competitive*** learning since the maximum similarity (or minimum distance) is used as the criterion for selecting a winning neuron (every neuron competes to be the winning neuron).
- The SOFM also shows ***cooperative*** learning because the winning neuron shares its learning with those neurons in its surrounding neighbourhood.
- So: **global competition + local cooperation**
  - A distinctive property as compared to traditional clustering techniques. (co-opetition)

Other Properties of the SOFM Algorithm

- The SOFM (once it has converged) provides a good *approximation* to the original input space.
  - It extracts the main features, and finds “prototype points” which represent many similar input patterns.
- The mapping is *topologically ordered* in the sense that the location of a neuron in the array corresponds to a particular feature from the input patterns.

Other Properties of SOFM (continued)

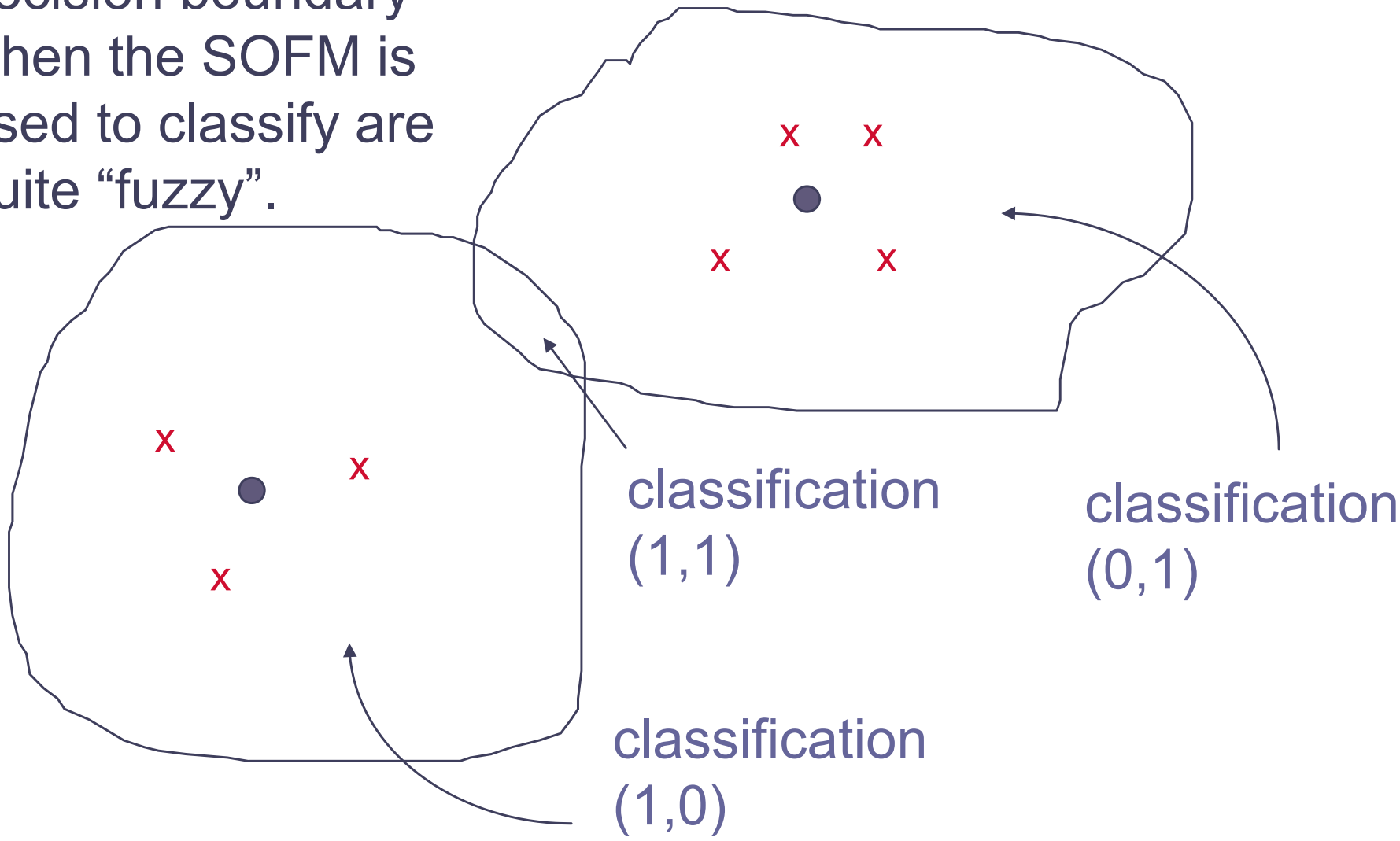
- The SOFM involves a *reduction of dimensionality* (since it can project a point from a high dimensional input space to a 1 or 2 dimensional feature space).
- The mapping reflects variations in the statistical distribution of the inputs.
  - Similar input patterns which are presented often result in a larger region of the feature map being activated.
- Selecting the neuron whose weights are closest in Euclidean distance to the input pattern is equivalent to choosing the maximum net input neuron as the winner.
  - Selecting the winning neuron can be based on either minimum distance or maximum net input.

Learning Vector Quantisation (LVQ)

- Kohonen found that improved classification performance could be achieved if a supervised learning rule is used in conjunction with SOFM.
- This helps make the decision boundaries more clearly defined (“crisper”).
- The SOFM by itself finds the “prototype” points that represent a certain class (e.g. the final weights move to the cluster centre).

Learning Vector Quantisation (LVQ) (continued)

- The edges of the decision boundary when the SOFM is used to classify are quite “fuzzy”.



Learning Vector Quantisation (LVQ) (continued)

- LVQ helps to fine-tune a SOFM by presenting inputs with known classifications, and moving the weights (prototype points) so that the correct responses are learnt.
- The SOFM sorts out the underlying structure by itself (unsupervised), and then LVQ can be used (supervised) to fine-tune the classifications.
- LVQ works by adjusting the weights to the winning neuron only, using a supervised learning rule.
- LVQ is useful when a small set of training data is available.

LVQ Learning Rule

- Step 1: An input pattern (with a known classification) is presented to the SOFM.
- Step 2: A neuron *m* in the array is selected as the winner, using a winner-take-all approach (maximum net input).
- Step 3: If the classification is correct (the winner corresponds to the correct class) then strengthen the weights to *m*

$$w_{mi}(t+1) = w_{mi}(t) + c[x_i(t) - w_{mi}(t)]$$

If the classification is incorrect, then weaken the weights to *m*:

$$w_{mi}(t+1) = w_{mi}(t) - c[x_i(t) - w_{mi}(t)]$$

LVQ Learning Rule (continued)

c is the learning rate.

- Step 4: Repeat this step for all of the known classifications (1 epoch). Repeat for more epochs until weights stabilise (prototype points have finished moving).
- The decision boundaries are now much crisper than before (just after the SOFM).
- Kohonen used the SOFM and LVQ to apply to a "phonetic typewriter".
  - A typewriter that types from dictation (speech).

Applications of SOFM

- World Poverty Map

  http://www.cis.hut.fi/research/som-research/worldmap.html
- Texture segmentation (aerial photographs: detect regions of sand, grass, dirt, concrete, etc.).
- Radar classification of sea ice.
- Clone detection in software.
- Pattern, speech and character recognition.
- Optimisation.
- Clustering in data mining.

Week 6 Tutorial

- Using NeuroShell 2 to solve the loan applicant classification problem discussed in Lecture 5.
  - Download the original and pre-processed data files (credit.txt and credit.xls) from Moodle.
  - Experiment with a single output neuron (predicting a 1 or 2) and two output neurons (1 0 or 0 1).
  - You may try to experiment with different architectures (including different sets of input neurons and different numbers of hidden neurons). These experiments are required if you choose this problem for your assignment.