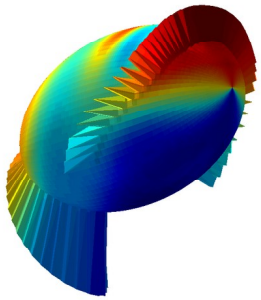
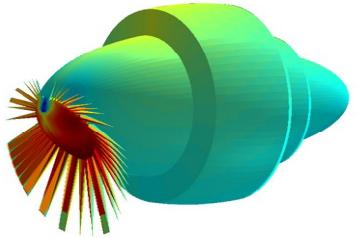
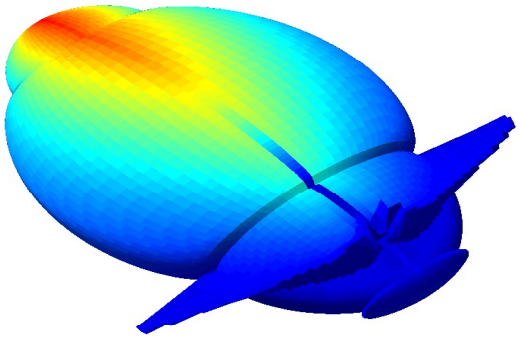
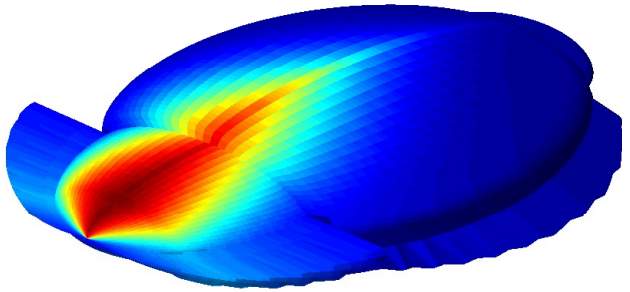




Lecture 11
Unsupervised Learning





Unsupervised Learning

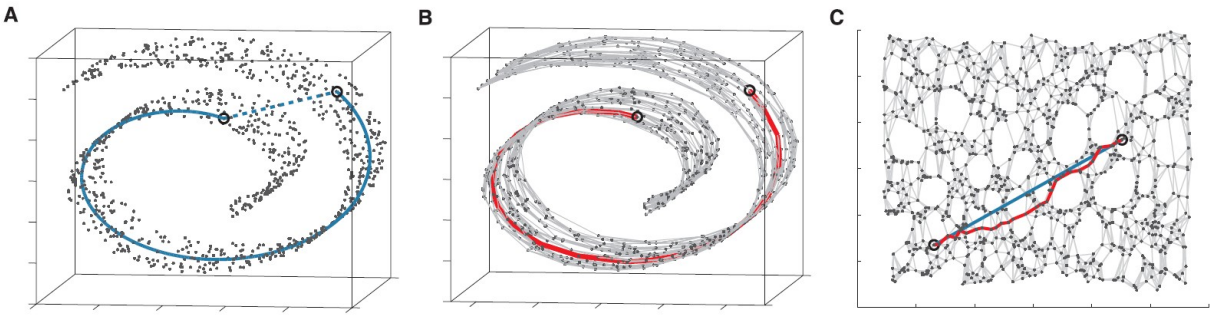
Unlike the supervised case, there is no "desired response" for any stimulus.

Instead, the system tries to ***infer*** some characteristic structural aspect of the data, e.g. clustering or lower-dimensional representation.

An Important Observation:

Unsupervised learning is mainly useful when the data/stimulus has ***redundancy***, i.e., the actual information it carries is less than the "representational volume" of the signal.

Example: ISOMAP



2-dimensional data embedded in 3-dimensional space.

Tenenbaum, J.B., de Silva, V. and Langford, J.C. (2000) A global geometric framework for nonlinear dimensionality reduction, *Science* **290**: 2319-2323.

What Unsupervised Learning Can Do

Similarity Detection: Deciding whether the current stimulus similar to something seen (and learned) before? And how similar?

Associative Memory: Recalling classes or objects based on similarity.

Clustering: Grouping data into clusters based on feature similarity.

Prototyping: Extracting prototypical representations of classes obtained by clustering.

Principal Component Extraction: Extracting the principal components of a dataset.

Dimensionality Reduction: Mapping high-dimensional data to fewer dimensions.

Recoding: Generating more concise representations of complex data.

Feature Mapping: Creating a topographic map of the input based on feature similarity.

We will focus mainly on ***clustering*** and ***feature mapping***.

Clustering: Essential Elements

Feature space: The space in which the data lives

Similarity measure: A function which quantifies the similarity between two data points.

e.g. $$s(x^1, x^2) = \frac{x^{1^T} x^2}{\|x^1\| \|x^2\|}$$

- Direction cosine
- Cosine similarity

where $\|\|$ is the Euclidean norm.

$s(x^1, x^2)$ measures the cosine of the angle between x^1 and x^2

and $s(x^1, x^2) = 1$ if x^1 and x^2 are collinear, 0 if orthogonal.

Distance measure: A norm measuring the distance between data points, e.g. Euclidean norm.

Cluster center: An archetypal "center" or "mean" vector in feature space typifying a given cluster (sometimes called a centroid, but that is less general)

K-Means Clustering

Given data $\{x^q\}$ $x^q \in X \leq \Re^n$

and a pre-specified set of clusters = C^k $k = 1,2, \ldots, K$

1. Choose K points from $\{x^q\}$ and make each one a cluster center α^r, $r = 1, \ldots, K$

2. For all remaining points in $\{x^q\}$, assign each one to the cluster with whose center it has the greatest similarity (or minimum distance)

3. Compute the centroid of each cluster and make each centroid the center for its cluster

$$\alpha^r = \frac{1}{N_r} \sum_{x^q \in C^r} x^q$$

4. Reassign all points with respect to new cluster centers

5. If no reassignments occur, stop. else repeat from step 3

K-Means Clustering Critique

+ Easy to implement

+ Often works well.

- Must know *K* a priori
(often impractical, since finding *K* is usually the key problem)

- Gets stuck in sub optimal configuration

One alternative ⟹ Self-Organized Feature Maps.

Assessing Cluster Quality

The Silhouette Metric

P.J. Rousseeuw (1987). "Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis". *Computational and Applied Mathematics*. 20: 53-65.

Suppose a clustering $C = \{C_1, \ldots C_m\}$ has been done with point $x^q \rightarrow$ cluster C_k

We calculate a silhouette value for x^q as:

$$\sigma(x^q) = \frac{b(x^q) - a(x^q)}{\max(a(x^q), b(x^q))}$$

where $a(x^q) \equiv$ mean distance of x^q from all other points in C_k

$b(x^q) \equiv$ mean minimum distance of x^q from all clusters other than C_k

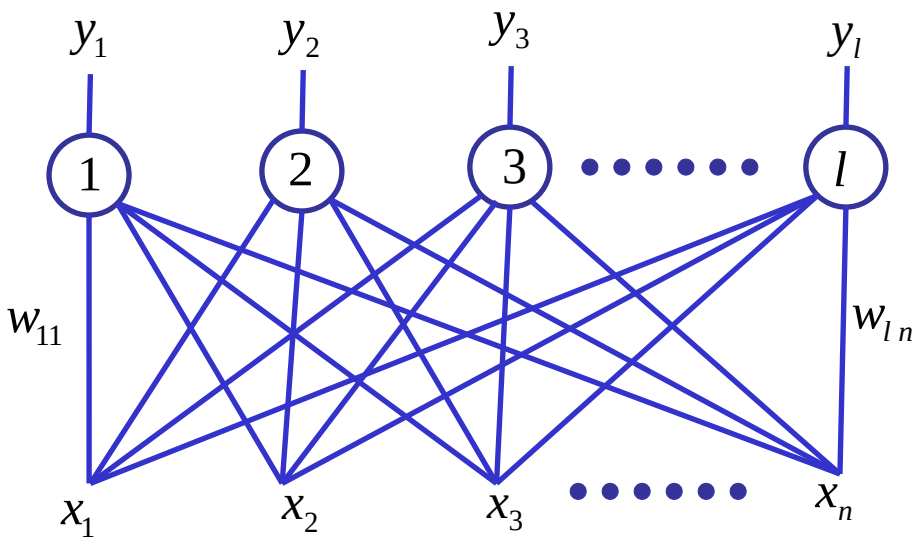
$-1 \leq \sigma(x^q) \leq 1$ where a score near 1 implies that $a(x^q) \ll b(x^q)$

Silhouette score of the clustering: $S(C) = \frac{1}{N} \sum_q \sigma(x^q)$ N is the number of points

K can be chosen to maximize the silhouette score.

Competitive Learning Networks

We have a 1-layer network of linear units, and inputs x_j



$$s_i = \sum_{j=1}^{n} w_{ij} x_j = w_i^T x$$

The weight vectors are all normalized, i.e.

$$\|w_i\| = \sqrt{\sum_j w_{ij}^2} = 1$$

Network Operation

1. Apply input vector x

2. Determine $s_i = w_i^T x \qquad \forall\ i = 1,\ldots,l$

3. Determine the winner i^*, the unit with the largest s_i

$$i^* = \arg\max(s_i)$$

4. Set $y_{i^*} = 1$

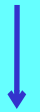
$$y_i = 0 \qquad \forall\ i \neq i^*$$

Winner take all (WTA)

5. Update weights $\Delta w_{ij} = \eta\, y_i \left(x_j - w_{ij} \right)$

This learning rule works best if the x vectors are also normalized. Otherwise, we can use

$$\Delta w_{ij} = \eta\, y_i \left(\frac{x_j}{\|x\|} - w_{ij} \right)$$

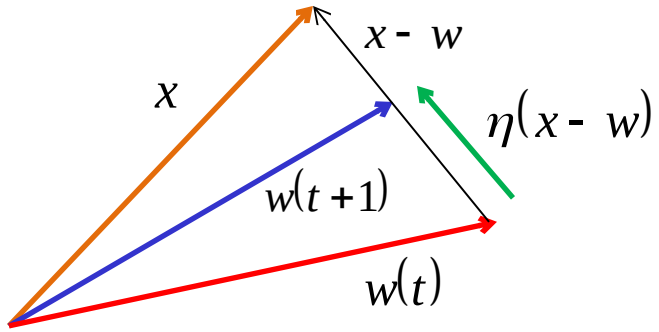


This helps keep the ws normalized because w_i is attracted to $\bar{x}/\|\bar{x}\|$ not $\bar{x}$

(Note that only the winners weights get updated by this rule)

What does this algorithm do?

WTA → find the neuron whose weight vector is most "similar to" the input vector. i.e. $w_{i^*}^T x > w_i^T x \quad \forall i \neq i^*$



Learning Rule → Move the winning neuron's weight vector "closer to" x.

→ Each neuron's weight vector converges to the centroid of all input patterns for which it wins = ***cluster center***

To see that the learning moves w_i towards x, consider:

$$w_{ij}(t+1) = w_{ij}(t) + \eta\left(x_j - w_{ij}(t)\right) \quad \text{for the winning unit } i$$

$$s_i(t+1) = \sum_j w_{ij}(t+1)\, x_j$$

$$= \sum_j \left(w_{ij}(t) + \eta\, x_j - \eta\, w_{ij}(t)\right) x_j$$

$$= (1-\eta)\, s_i(t) + \eta \sum_j x_j^2 \quad = s_i(t) + \eta\left(x^T x - w_i^T(t) x\right)$$

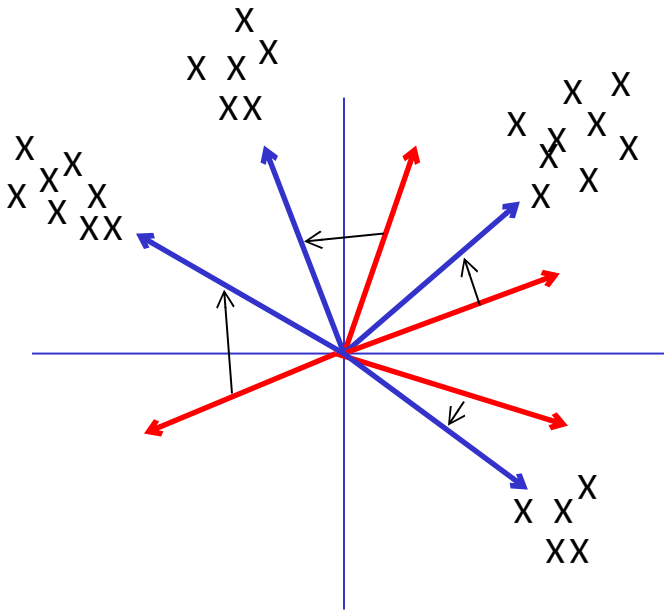
since $s_i(t) = w_i^T(t)\, x$ and $\sum_j x_j^2 = x^T x$

$$\therefore \quad s_i(t+1) = s_i(t) + \eta\underbrace{\left[x^T x - w_i^T(t)\, x\right]}_{\geq 0 \text{ (if } x \text{ and } w_i \text{ are normalized)}} \geq s_i(t)$$

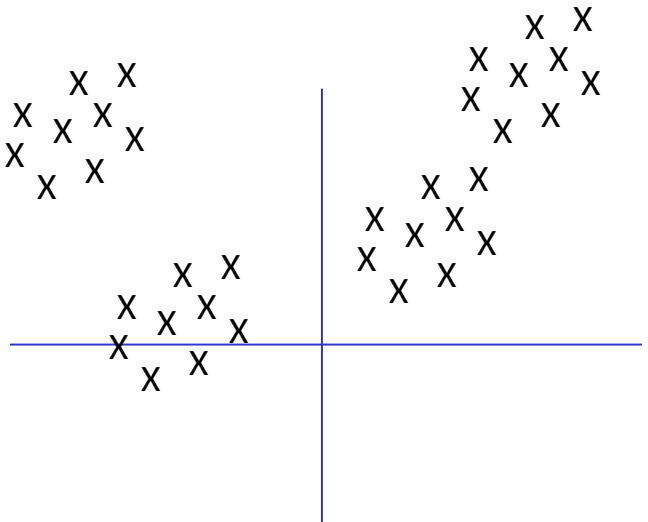
when $w_i = x$, $s_i(t+1) = s_i(t) \rightarrow$ convergence

Non-Radial Clusters

Using $s_i = w_i^T x$ to choose the winner works when the clusters are "radial"



But what if the clusters are not radial?



Use ***non-normalized*** weights with

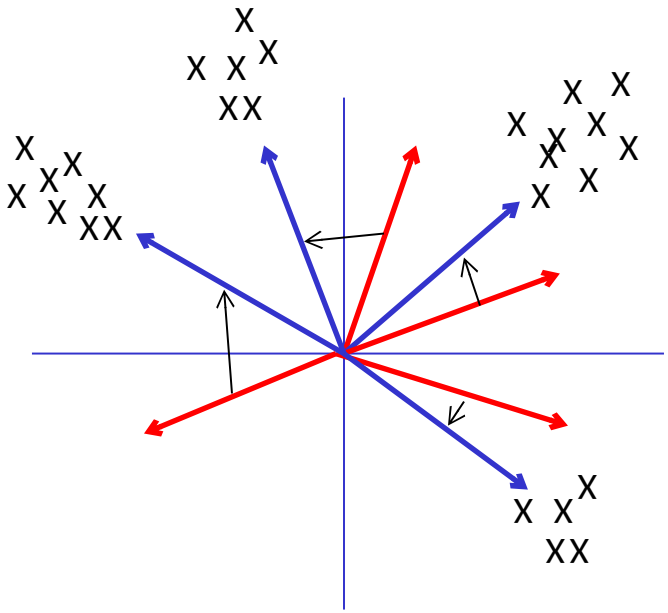
$$s_i = \| x - w_i \| \quad \text{for}$$

$$\text{WTA} \rightarrow i^* = \arg\min(s_i)$$

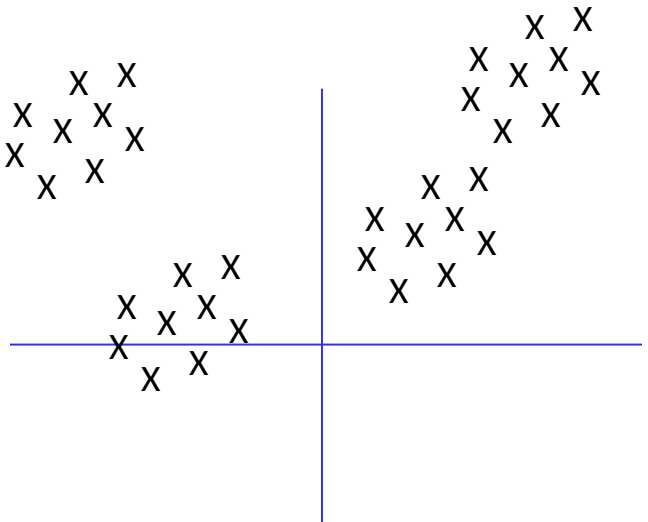
$$\Delta w_{ij} = \eta\, y_i (x_j - w_{ij})$$

Non-Radial Clusters

Using $s_i = w_i^T x$ to choose the winner works when the clusters are "radial"



But what if the clusters are not radial?



Use ***non-normalized*** weights with

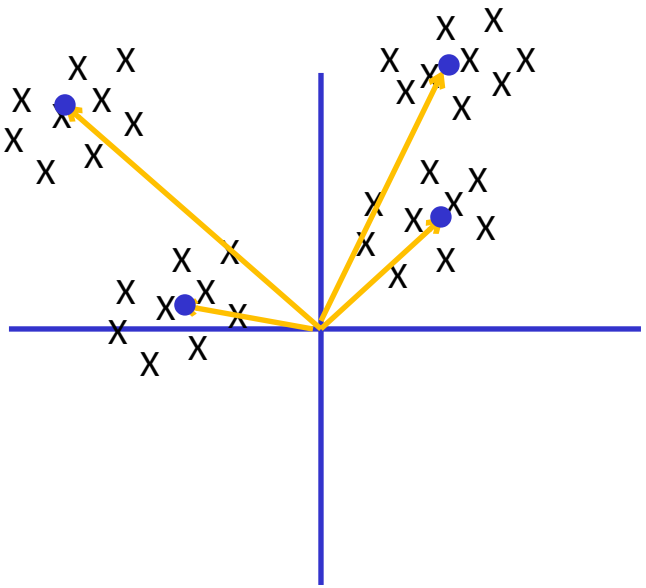
$$s_i = \|x - w_i\| \quad \text{for}$$

$$\text{WTA} \rightarrow i^* = \arg\min(s_i)$$

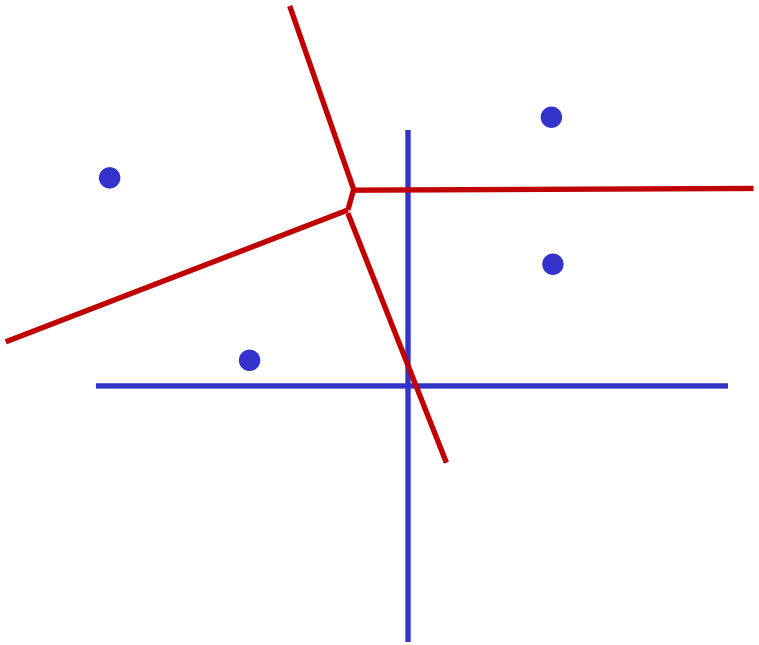
$$\Delta w_{ij} = \eta\, y_i (x_j - w_{ij})$$

Note

In this case, the weight vectors converge to cluster centers



The algorithm performs a Voronoi tessellation around the cluster centers



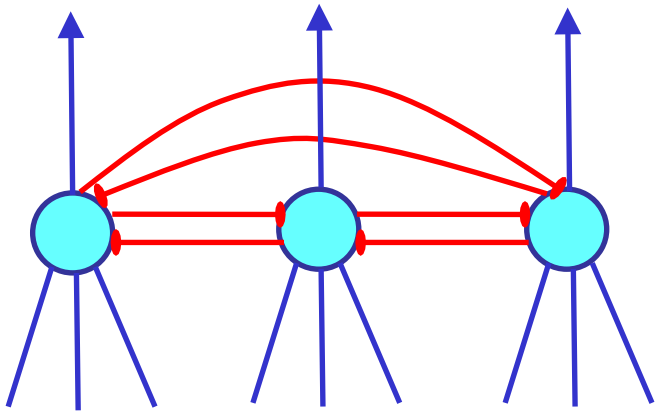
The cluster centers can be regarded as *prototype vectors* for each cluster

For normalized x, $w_{i,}$ using $w_i^T x$ to determine the winner is equivalent to using $\|x - w_i\|$

$$\|x - w_i\| = \sqrt{(x - w_i)^T (x - w_i)}$$
$$= \sqrt{x^T x + w_i^T w_i - 2 w_i^T x}$$
$$= \sqrt{\text{constant} - 2 w_i^T x}$$

$\rightarrow$ Largest $w_i^T x \Rightarrow$ smallest $\|x - w_i\|$

In hardware (or even in simulation) WTA can be achieved by lateral inhibition



Whoever reaches firing threshold first, inhibits all others.

In fact, competitive learning corresponds exactly to the K-means clustering method (its on-line variant).

Competitive learning can create dead units: neurons that never win, and therefore never learn.

To prevent these, we can

- Initialize weight vectors using actual input patterns

or

- Use leaky learning, where losing neurons also learn, but at a much lower rate

→ Their weight vectors drift closer to where the data is.

or

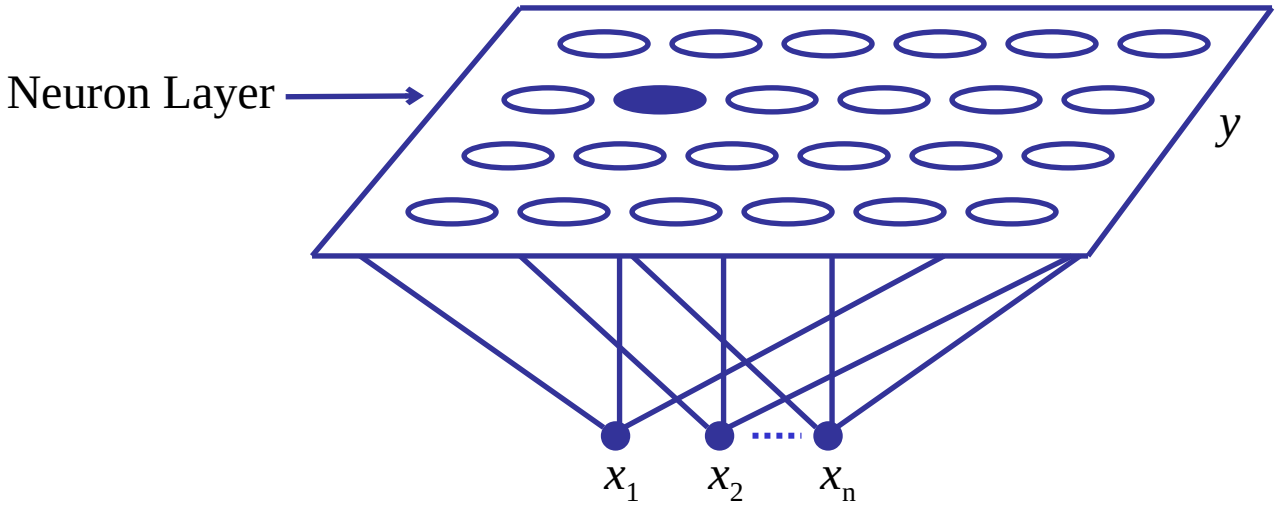
- Use a conscience scheme, where no unit is allowed to win more than a fraction of the time. (This is hard to stabilize in some cases)

Self-Organized Feature Map

Self-Organized Feature Maps (SOFM)

Kohonen, 1982

Self-organized feature maps use competitive learning in a network whose neurons are arranged in a metric space, e.g., a 2-dimensional grid.



r_i = position of unit i on the grid.

$w_i = [w_{i1} \quad w_{i2} \quad . \; . \; . \quad w_{in}]^T$

Learning in the SOFM

Training Algorithm

Given training data points, $X = \{x^k\}$

1. Present vector $x^q \in X$
2. Determine winning unit i^*

$$i^* = \arg\min \left\| w_i - x^q \right\|$$

3. Update all weights by

$$\Delta w_{ij} = \eta \Lambda(i, i^*, t)(x_j^q - w_{ij})$$

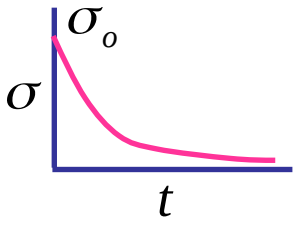
4. Repeat from 1

$\Lambda(i, i^*, t)$ is a time dependent neighborhood function with a peak at $i = i^*$

A typical choice is

$$\Lambda(i, i^*, t) = \exp\left[\frac{- \left\| r_i - r_{i^*} \right\|^2}{2\sigma^2(t)} \right]$$

$$\sigma(t) = \sigma_o \exp\left(- \frac{t}{\tau_N} \right)$$



where σ_o and τ_N are fixed parameters.

Effect of Training

- The weight vector, w_{i*}, of the winning neuron moves closer to x^q
- The weight vectors of neurons physically close to i^* move towards x^q by smaller amounts that depend monotonically on their distance from i^*.
- The weight vectors of neurons far from i^* do not move much at all.

Over the course of learning, the activity of neurons on the 2-d (or even 1-d) surface comes to represent the topological/statistical relationships of data in the input feature space.

→ similar inputs have similar output representations but are represented by activity on a lower-dim manifold.

This is useful for

- Clustering
- Visualization
- Removal of redundancy (data compression)
- Organization of information onto spatial maps (useful in the brain)
- Inference of structure in the input space.

Network Functionality:

Over the course of learning, the activity of neurons on the 2-d (or even 1-d) surface comes to represent the topological/statistical relationships of data in the input feature space.

→ similar inputs have similar output representations

but are represented by activity on a lower-dim manifold.

This is useful for

- Clustering
- Visualization
- Removal of redundancy (data compression)
- Organization of information onto spatial maps (useful in the brain)
- Inference of structure in the input space.

Two-Phase Learning

Experience has shown that it is often beneficial to divide the learning of SOFM into two phases with learning rate varied over time by

$$\eta(t) = \eta_o \exp\left(-\frac{t}{\tau_L}\right)$$

The Self-Organizing Phase:

This is where the weights go from their initial random organization to one reflecting the structure of the data:

- Start with $\eta_o = 0.1$, $\tau_L = 1000$.
- Set the neighborhood function $\Lambda(i, i^*, 0)$ to cover almost the whole network. This can be done by setting $\sigma_0 \approx 0.5\, r_{max}$
- Shrink the neighborhood function slowly over 1000 or so iterations. This can be done by setting $\tau_N = 1000 / \ln\sigma_0$ so that $\sigma(1000) = 1$

The Convergence Phase:

This refines the learned weights.

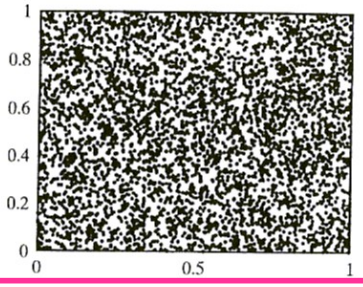
- Maintain $\eta(t) \approx 0.01$ and $\Lambda(i, i^*, t) \approx 1$ (only the nearest neighbors)
- Train for about 500 times the number of data points.

Example 1: Density Mapping

Data → 2-dimensional

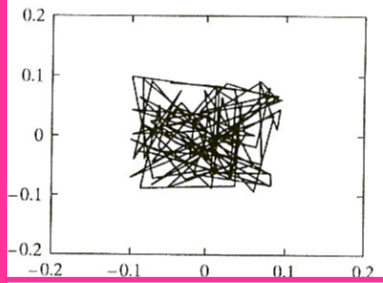
Neural layer → 2-dimensional

Given set of data points in feature space (note uniform distribution)

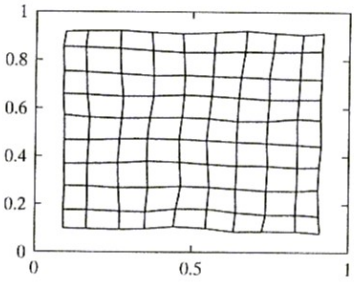


(a)

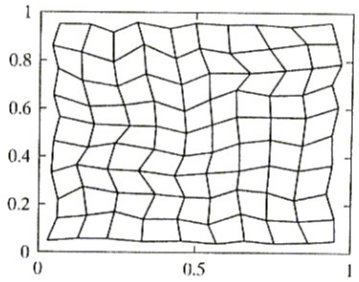
Neighborhood relationships between initial weight vectors (each is shown connected to its 4 neighbors)



(b)



Neighborhood relationships of weight vectors after self-organization phase.



Final weight vector neighborhood relationships after the convergence phase.

The distribution of weight vectors comes to reflect the distribution of data

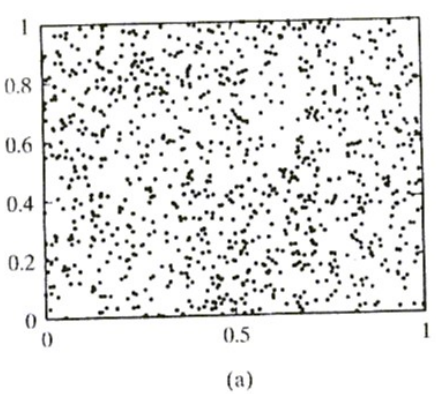
Example 2: Dimensionality Reduction

Data → 2-dimensional

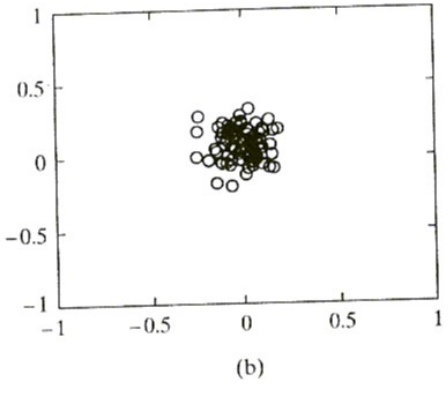
Neural layer → 1-dimensional

Goal: Finding a lower-dimension map of the data that preserves neighborhood relationships as far as possible.

2-d data



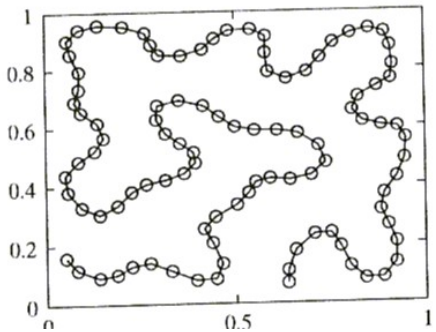
(a)

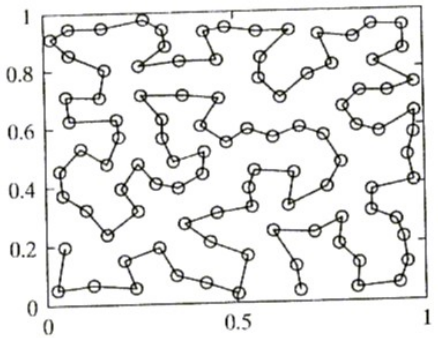


(b)

Initial weights

Weights after learning





Final weight vector distribution

Uniform 2-d data is mapped to a space-filling 1-d curve

From: S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice-Hall, 1999

Example 3: Feature Mapping

From: S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice-Hall, 1999

Given: 16 animals with13 attributes

Each animal is represented by:

- a 13-bit attribute code $x_a^k \in \{0,1\}^{13}$
- a 1-of-16 symbol code $x_s^k \in \{0, c\}^{16}$

c = constant

$$x_s^k = [0 \quad 0 \quad . \quad . \quad 0 \quad c \quad 0 \quad . \quad . \quad 0]^T$$

$$k = 1, 2, \ldots, 16$$

TABLE 9.3 Animal Names and Their Attributes

Animal		Dove	Hen	Duck	Goose	Owl	Hawk	Eagle	Fox	Dog	Wolf	Cat	Tiger	Lion	Horse	Zebra	Cow
is	small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
	medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
has	2 legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	4 legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
	feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
likes to	hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
	run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
	fly	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
	swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Total representation for animal k

$$x^k = \left[x_s^{k^T} \quad x_a^{k^T} \right]^T$$

e.g. Hen $= [0c0000000000000100100010000]^T$

x_s^k x_a^k

Train on x^k for 2000 iterations with a 10 x 10 network

Test with $\hat{x} = \lfloor x_s \; 0 \rfloor$

where x_s is the symbol code for any animal

e.g.

Hen $= [0c0000000000000000000000000]$

Zebra $= [0000000000000c0000000000000]$

Neurons with maximal response to each animal

dog	·	·	fox	·	·	cat	·	·	eagle
·	·	·	·	·	·	·	·	·	·
·	·	·	·	·	·	·	·	·	owl
·	·	·	·	·	·	tiger	·	·	·
wolf	·	·	·	·	·	·	·	·	hawk
·	·	·	lion	·	·	·	·	·	·
·	·	·	·	·	·	·	·	·	dove
horse	·	·	·	·	·	·	hen	·	·
·	·	·	·	cow	·	·	·	·	goose
zebra	·	·	·	·	·	·	duck	·	·

Observations:

- Animals are mapped according to their similarities in attribute space.
- Animals of each class are mapped to a compact cluster of units

Preferred response for each neuron

dog	dog	fox	fox	fox	cat	cat	cat	eagle	eagle
dog	dog	fox	fox	fox	cat	cat	cat	eagle	eagle
wolf	wolf	wolf	fox	cat	tiger	tiger	tiger	owl	owl
wolf	wolf	lion	lion	lion	tiger	tiger	tiger	hawk	hawk
wolf	wolf	lion	lion	lion	tiger	tiger	tiger	hawk	hawk
wolf	wolf	lion	lion	lion	owl	dove	hawk	dove	dove
horse	horse	lion	lion	lion	dove	hen	hen	dove	dove
horse	horse	zebra	cow	cow	cow	hen	hen	dove	dove
zebra	zebra	zebra	cow	cow	cow	hen	hen	duck	goose
zebra	zebra	zebra	cow	cow	cow	duck	duck	duck	goose

Note that classes were *not* given. The system *discovered* them.

Train on x^k for 2000 iterations with a 10 x 10 network

Test with $\hat{x} = |x_s \ 0|$

where x_s is the symbol code for any animal

e.g.

Hen $=[0c0000000000000000000000000]$

Zebra $=[0000000000000c0000000000000]$

Neurons with maximal response to each animal

dog	·	·	fox	·	·	cat	·	·	eagle
·	·	·	·	·	·	·	·	·	·
·	·	·	·	·	·	·	·	·	owl
·	·	·	·	·	·	tiger	·	·	·
wolf	·	·	·	·	·	·	·	·	hawk
·	·	·	lion	·	·	·	·	·	·
·	·	·	·	·	·	·	·	·	dove
horse	·	·	·	·	·	·	hen	·	·
·	·	·	·	cow	·	·	·	·	goose
zebra	·	·	·	·	·	·	duck	·	·

Observations:

- Animals are mapped according to their similarities in attribute space.
- Animals of each class are mapped to a compact cluster of units

Preferred response for each neuron

dog	dog	fox	fox	fox	cat	cat	cat	eagle	eagle
dog	dog	fox	fox	fox	cat	cat	cat	eagle	eagle
wolf	wolf	wolf	fox	cat	tiger	tiger	tiger	owl	owl
wolf	wolf	lion	lion	lion	tiger	tiger	tiger	hawk	hawk
wolf	wolf	lion	lion	lion	tiger	tiger	tiger	hawk	hawk
wolf	wolf	lion	lion	lion	owl	dove	hawk	dove	dove
horse	horse	lion	lion	lion	dove	hen	hen	dove	dove
horse	horse	zebra	cow	cow	cow	hen	hen	dove	dove
zebra	zebra	zebra	cow	cow	cow	hen	hen	duck	goose
zebra	zebra	zebra	cow	cow	cow	duck	duck	duck	goose

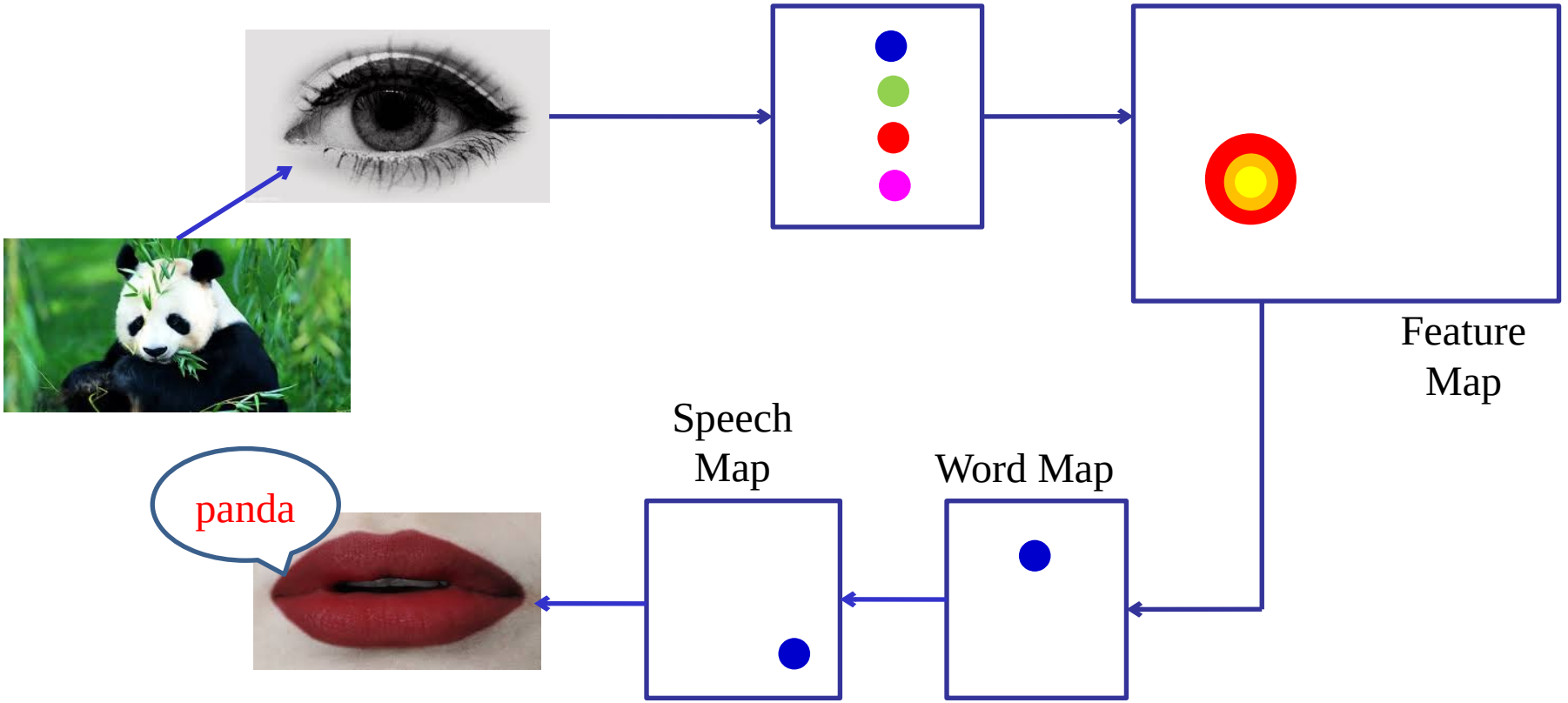
Note that classes were *not* given. The system *discovered* them.

Why is a Feature Map Useful?

Feature maps organize information in ways that are:

- Meaningful.
- Reflect the structure of information.
- Generalize over the information space.
- Act as pointers for high-dimensional representational spaces.

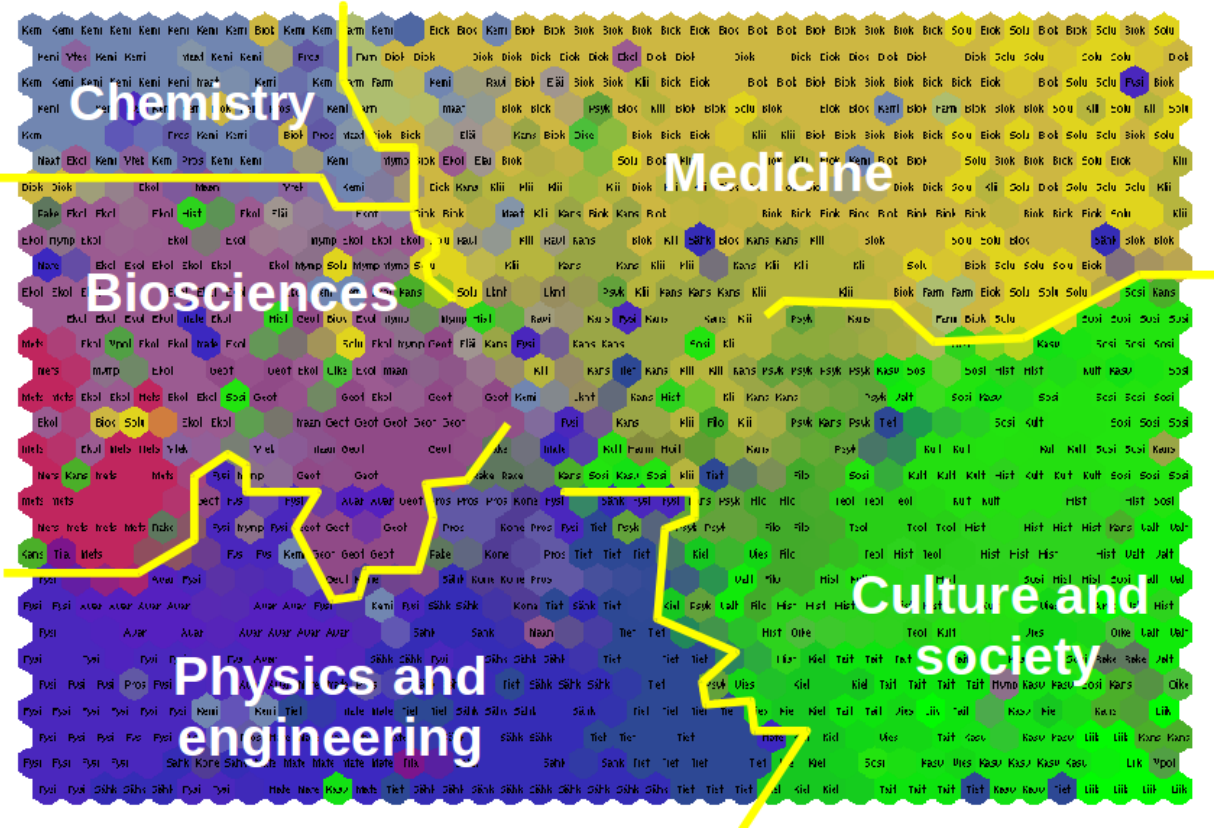
Thus, they can be used to improve decision-making, inference and control.



Inferring Structure in Data

T. Honkela (2013) Multimodally Grounded Translation, *Tralogy II*.

- 3,224 applications submitted to the Academy of Finland were analyzed.
- Each application was represented in terms of extracted keywords.
- The applications were grouped by an SOFM in keyword space.
- Colors are assigned based on actual research area classes.



http://lodel.irevues.inist.fr/tralogy/index.php?id=259

Application to Supervised Learning

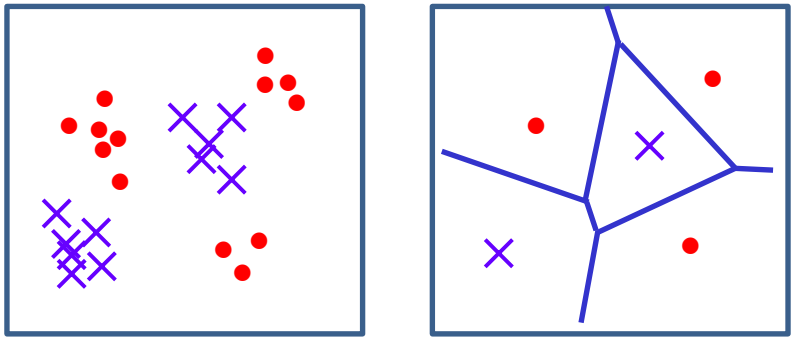
Given a data set with known class labels:

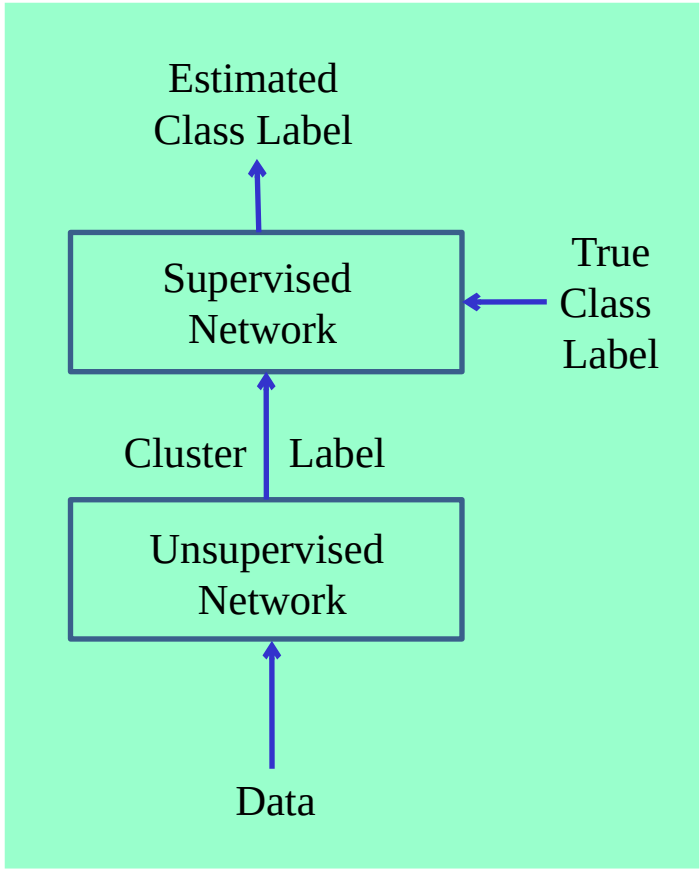
Stage 1: Self-organized clustering

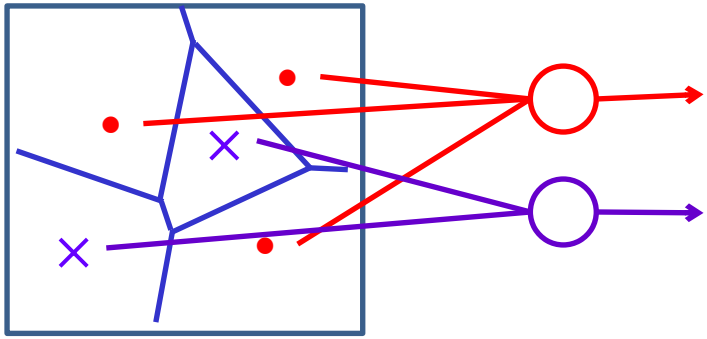
Determine a relatively large number of clusters using competitive learning or SOFM

Stage 2: Supervised aggregation

Combine clusters that belong to the same class using LMS, perceptrons, backprop, etc.







The cluster centers can also be adjusted further through supervised learning to improve the classification → Learning Vector Quantization

Learning Vector Quantization

T. Kohonen (1989) *Self-Organization and Associative Memory*. Berlin: Springer-Verlag

Given a set of labeled data: $\{ x^k , c^k \}$

- Train a self-organized feature map on the data.
- Label each neuron in the SOFM with a class based on its response to data (e.g., majority)
- For N steps
  - Pick a random data point, x^q
  - Find the winning neuron i^* and get its class label $c(i^*)$
  - Find the runner-up neuron i' and get its label $c(i')$
  - If $c(i^*) \neq c^q$ (mismatch)
    and $c(i') = c^q$:
    and x^q lies close to the bisector of w_{i^*} and $w_{i'}$

Does not require SOFM. Can be done with competitive learning

$$\Delta w_{i^* j} = -\eta \left(x_j^q - w_{i^* j} \right)$$

Anti-Hebbian learning: weight moves ***away*** from x^q

$$\Delta w_{i' j} = \eta \left(x_j^q - w_{i' j} \right)$$

Hebbian learning: weight moves ***towards*** x^q

Else: do nothing