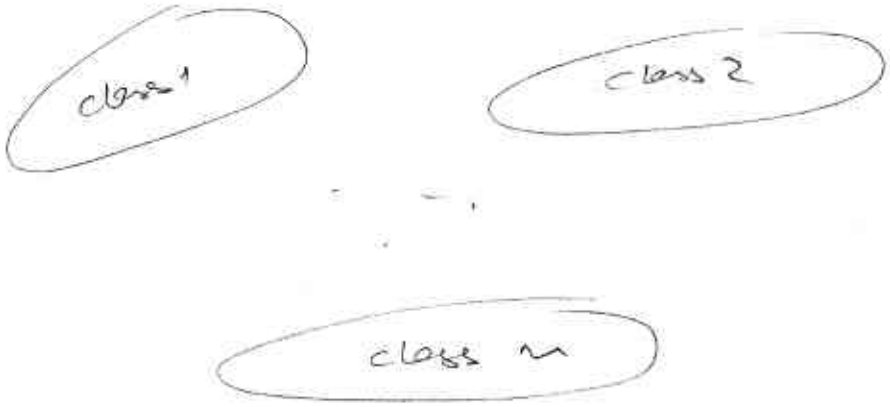
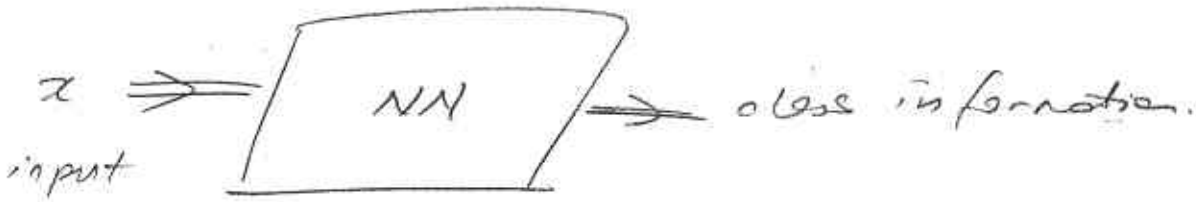
UNSUPERVISED LEARNING

– CLUSTER DETECTION.

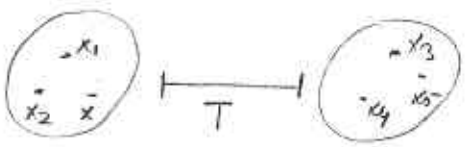


input space



PROBLEM: Given the training set $\{x_1, \ldots, x_N\}$, but without the class information, find a NN architecture and learning rule, which detects the clusters.

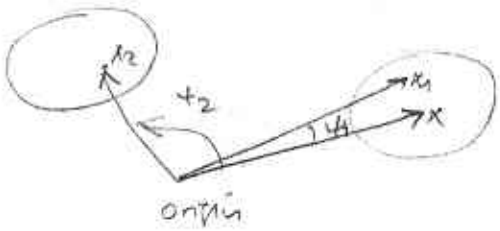
To define a cluster, we need to use a "similarity measure" to check the closeness of patterns within a class. The most common measure is the Euclidean distance between patterns.

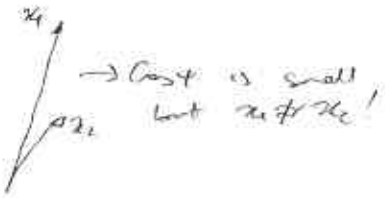


$$\|x - x_i\|^2 = (x - x_i)^t (x - x_i)$$

Here we also determine a smallest distance T to discriminate the clusters.

Another measure might be the cosine of the angle between the vectors.



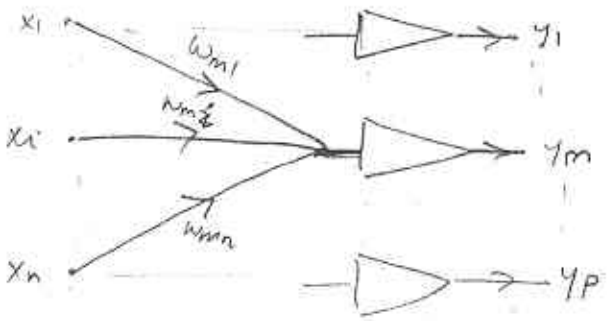


$$x^t x_i = \|x\| \cdot \|x_i\| \cdot \cos\psi_i \qquad \cos\psi_i = \frac{x^t x_i}{\|x\| \cdot \|x_i\|}$$

The smaller the $\cos\psi$, the closer the vectors x and x_i. However, the vectors x and x_i should be comparable.

WINNER-TAKES-ALL LEARNING

$\{x_1, \ldots, x_N\}$ are given as inputs to be classified. The number of distinct classes p is also known. The network is called Kohonen network



$$y = \begin{pmatrix} y_1 \\ y_m \\ y_p \end{pmatrix} = Wx.$$

$$W = \begin{pmatrix} w_1^t \\ w_m^t \\ w_p^t \end{pmatrix} = \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ w_{m1} & \cdots & w_{mn} \\ w_{p1} & \cdots & w_{pn} \end{pmatrix}$$

$$w_m = \begin{pmatrix} w_{m1} \\ w_{mi} \\ w_{mn} \end{pmatrix}$$

In the beginning, all weights are chosen to be random and having unit length:

$$\hat{w}_i = \frac{w_i}{\|w_i\|} \qquad i = 1, \ldots, p$$

When an input pattern x is given, the procedure is to find the output, for which the following is minimum:

$$\|x - \hat{w}_m\| = \min_i \|x - \hat{w}_i\| \Rightarrow$$ (x should be normalized as well)

Hence, the network finds the weight vector which is closest to the input x.

$$\|x - \hat{w}_m\|^2 = (x^t - \hat{w}_m^t)(x - \hat{w}_m) = \|x\|^2 - 2\hat{w}_m^t x + 1$$

($\|w_m\| = 1$). Obviously $\|x - \hat{w}_m\|$ is min. if $w_m^t x$ is max. Hence equivalently, we may look for,

$$w_m^t x = \max_i w_i^t x$$

Hence, when x is present, compute $Wx = \begin{pmatrix} w_1^t x \\ w_m^t x \\ w_p^t x \end{pmatrix}$ and find the output which is maximum. After selecting this output, corresponding weight vector should be adjusted so that it becomes more similar to the input pattern (i.e. $\|x - \hat{w}_m\|$ becomes smaller). A standard technique to guarantee that is to update along the negative gradient of the following function:

$$E = \frac{1}{2}\|x - w_m\|^2 = \frac{1}{2}(\|x\|^2 - 2w_m^t x + w_m^t w_m)$$

$$\frac{\partial E}{\partial w_m} = -(x - w_m)$$

Hence the update rule is:

$$\Delta w_m = \alpha(x - \hat{w}_m) = -\eta \frac{\partial E}{\partial w_m}$$ m: winning neuron
$\Delta w_i = 0$ i: non winning neurons.

$$(w_m(k+1) = w_m(k) + \Delta w_m, \quad w_m(k+1) \leftarrow \frac{w_m(k+1)}{\|w_m(k+1)\|}$$ (normalize!)

* the rule should increase the chances of winning by m^{th} neuron when this update rule is used.

$$\Rightarrow \quad w_m^t x \leq (w_m + \Delta w_m)^t x$$

$$\Rightarrow \quad \Delta w_m^t x \geq 0 \iff \alpha (x - w_m)^t x \geq 0$$

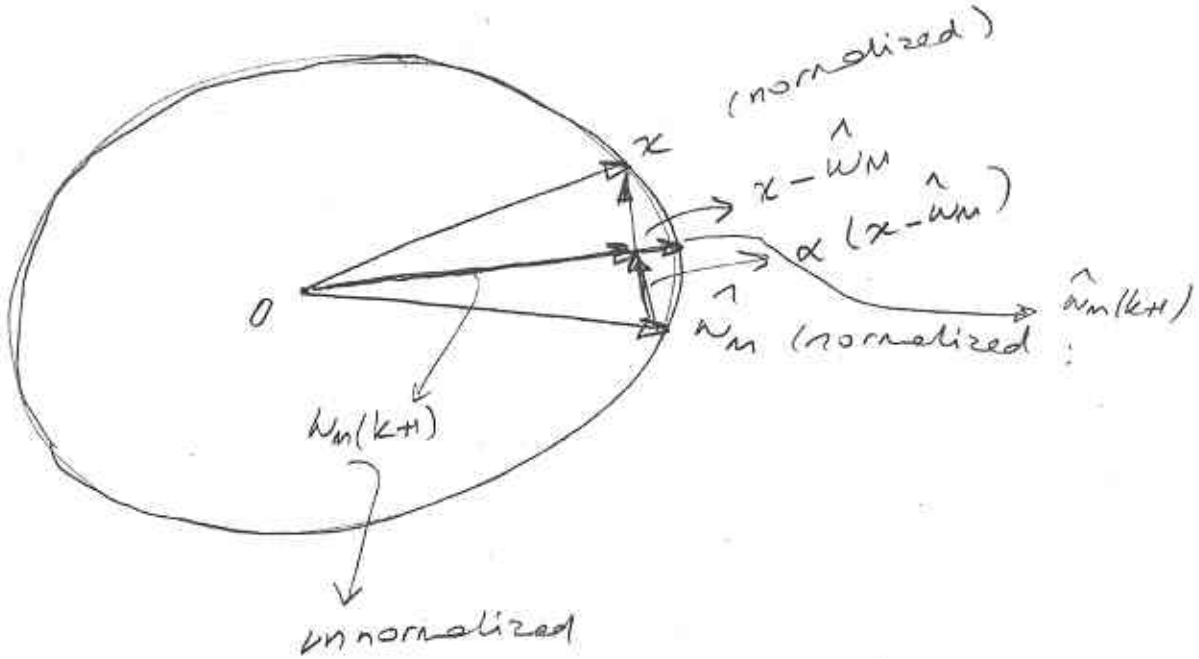
$$\iff x^T x - w_m^t x = \|x\|^2 - w_m^t x \geq 0$$

$$w_m^t x = \|w_m\| \cdot \|x\| \cdot \cos(\underbrace{w_m, x}_{\theta})$$

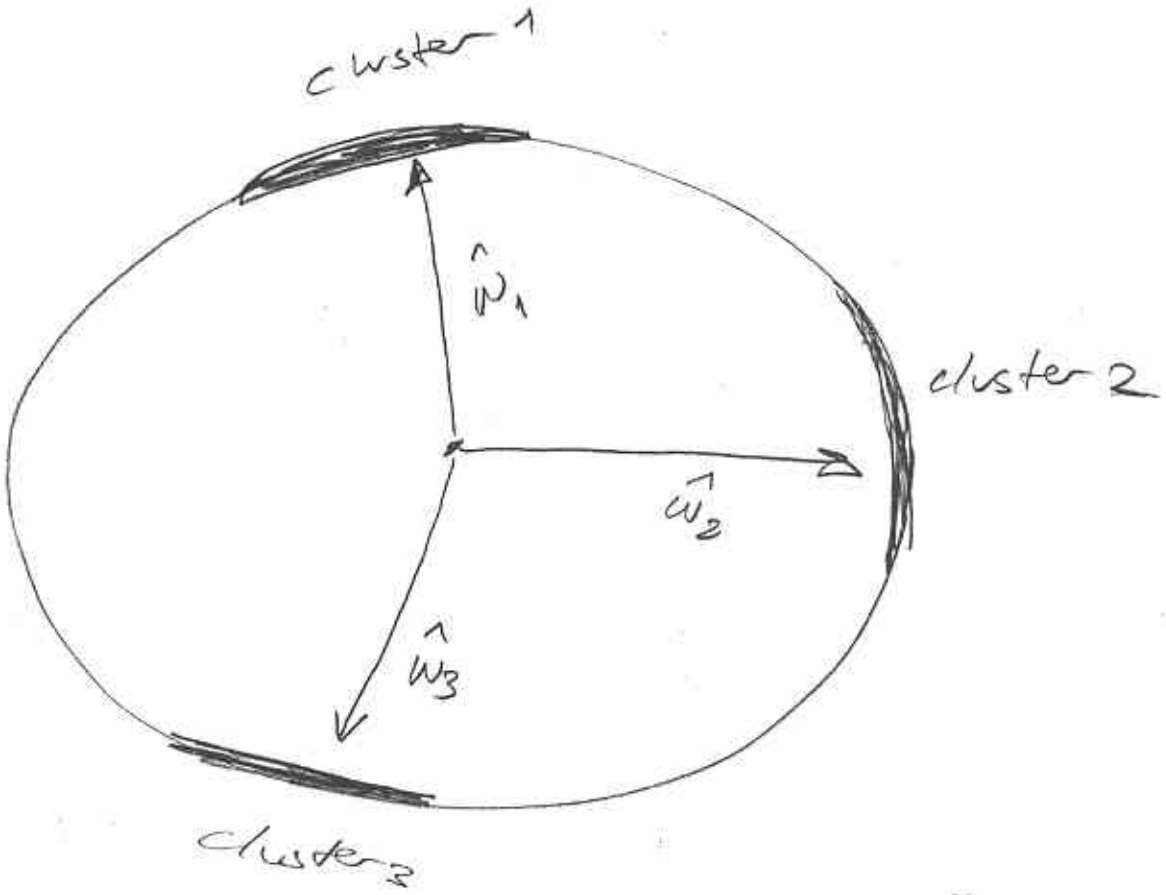
$$\|x\|^2 - w_m^t x = \|x\| \left(\|x\| - \|w_m\| \cos(w_m, x) \right) \overset{?}{\geq} 0$$

if $\|x\|$ and $\|w_m\|$ are normalized this is always the true!

GEOMETRIC IDEA:



IDEA :



- success may depend on how the clusters are separated
- initial weights

TRICK: If we know the cluster patterns, initial weights may be chosen as the cluster centers

i.e. $C\hat{l}^0 = \{x_{i1}, \dots, x_{ip}\}$

choose $w_i = \frac{1}{m}\{x_{i1} + \dots + x_{ip}\}$ $\quad i = 1, 2, \dots, p$

ALGORITHM:

Given the training set:

$\{x_1, x_2, \ldots, x_N\}$ $y \in \mathbb{R}^n$ (in our case $N=100$ (patterns), $n=100$ (10x10 grid))

And p distinct classes ($p=10$ in our case)

- Choose the initial weights randomly and uniformly distributed on the unit hypercube in $\mathbb{R}^n$.

(suggestion: $W = \begin{pmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_p^T \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ \vdots & & & \\ w_{p1} & w_{p2} & \cdots & w_{pn} \end{pmatrix}$

choose $w_{ij} \in \{-1, 1\}$ uniformly distributed and randomly.

$\Rightarrow$ $\hat{w}_i = \frac{w_i}{\|w_i\|}$ $\|w_i\| = \sqrt{w_{i1}^2 + \cdots + w_{in}^2}$

- Normalize the pattern set.

$y_i = \frac{x_i}{\|x_i\|}$ (since $x_{ij} = \pm 1 \rightarrow \|x_i\| = \sqrt{1 + \cdots + 1} = \sqrt{100} = 10$)

For each pattern y_i in the pattern set

(1) $\longrightarrow$ ($i = 1, 2, \ldots, N$)

- find $\arg\max_{1 \le k \le p} \hat{w}_k^T y_i$ (i.e. $w_{y_i} \rightarrow$ find max. entry)

Let m be the max index (winning neuron)

- update only w_m

$w_m(t+1) = w_m(t) + \alpha (y_i - w_m(t))$

- Do not update the remaining neuron weights.

- Normalize $w_m(t+1)$

$\hat{w}_m(t+1) \leftarrow \frac{w_m(t+1)}{\|w_m(t+1)\|}$

- Go to (1) if all patterns are exhausted

— How to reduce α ?

$0.1 < \alpha < 0.7$ initially

as epochs increase $\alpha \to 0.01$ (or even smaller)

This could be done in various ways.

- In discrete epoch steps. i.e.

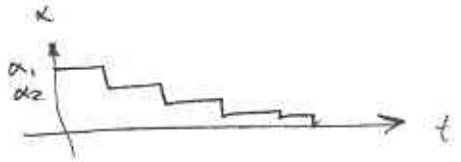
let $t_1, t_2, \ldots, t_M$ be discrete epoch numbers

t : current epoch number.

$0 \le t \le t_1 \rightarrow \alpha = \alpha_1$

$t_1 < t \le t_2 \rightarrow \alpha = \alpha_2$

:



Or :

$$\alpha = \alpha_0 + \alpha_1 e^{-t/\tau}$$

t : # of epochs

$\alpha_0 = 0.01$

$0.1 \le \alpha_1 = \le 0.7$

— IF CONVERGED

— Calibration :

Now it is not known which output corresponds to which class. To determine this, we could do calibration as follows:

- Select candidates from the training set representing class i.e. let $y_1 \in C_1 \Rightarrow$ apply $A \Rightarrow$ find the max. output
  $\Rightarrow$ This output represents class C_1.

Do this for any other class as well.

TRICK: For initial weights, instead of using w_i randomly, we could choose w_i as the arithmetic average of patterns in class C_i :-

KOHONEN'S SELF ORGANIZING FEATURE MAPS:

In its basic form, we have p output neurons representing p classes. We could also increase the number of neurons, and allow the output neurons to compete among themselves.

procedure is the same as before:

Given the training patterns and p output neurons

$\{y_1, y_2, \ldots, y_N\}$

p: # of classes
P: # of output neurons

$P > p$

In epoch t
- Find the max. output:

$$\arg\max_{1 \le k \le P} w_k^T y$$

Let m be the winning neuron

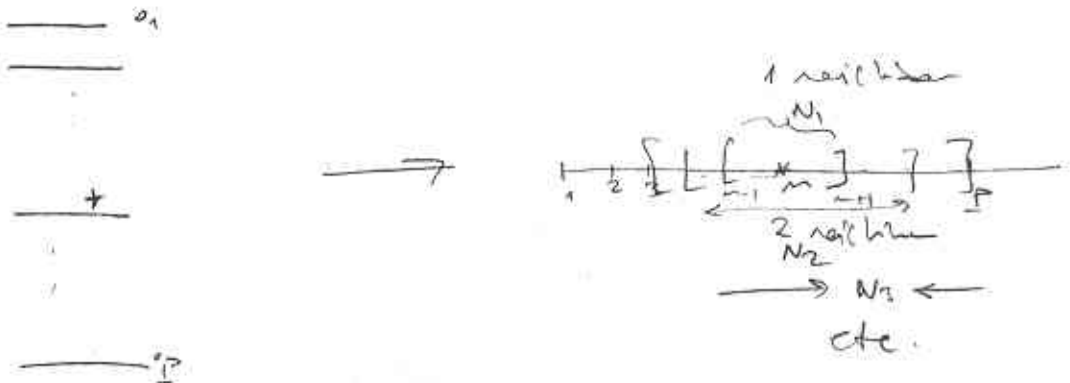
Define a neighborhood of winning neuron $N_m(t)$
(t: epoch number)

$$w_j(n+1) = w_j(n) + \alpha(t)\,(y - w_j(n)) \qquad j \in N_m(t)$$

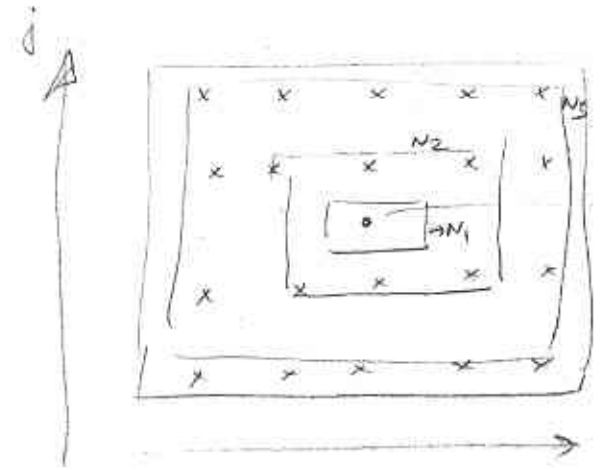
i.e. all neurons in the neighborhood of the winning neuron are updated.

— How to choose the neighborhoods of neurons?

That depends on the position of output neurons.



If the atoms are selected as 2-dimensional



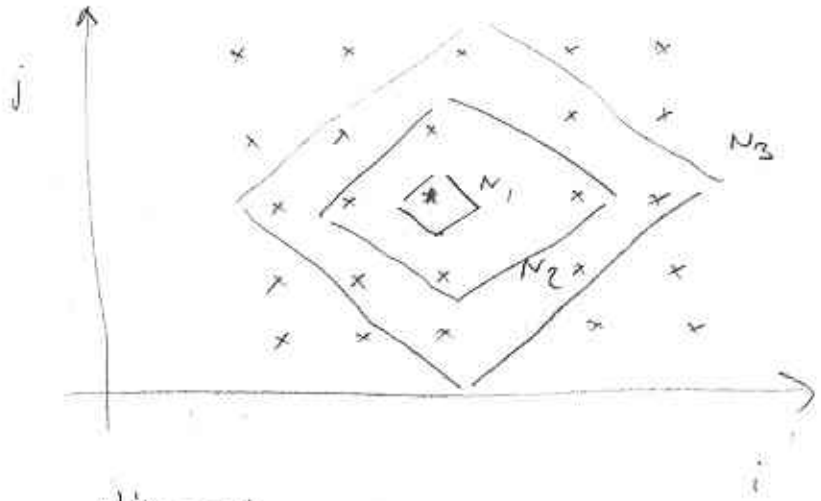
rectangle

⇒ winning neuron.
(m_1, m_2) location

$N_1 \rightarrow$ only (m_1, m_2)

$N_2 \rightarrow \{(i,j) \mid i = m_1-1, m_1, m_1+1 \;;\; j = m_2-1, m_2, m_2+1\}$

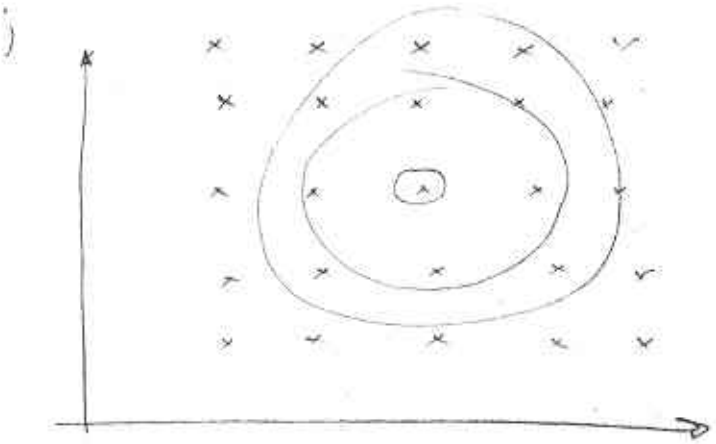
$N_3 \rightarrow \{(i,j) \mid i = m_1-2, \ldots, m_1+2 \;;\; j = m_2-1, \ldots, m_2+1\}$



diamond

winning neuron (m_1, m_2)

$N_2 = \{(i,j) \mid |i+j| \leq 1$



circular neighborhood.

— As a result, as epoch number t increases, we decrease both the neighborhood size and α.

— Use predetermined epoch number $t_1, t_2, t_3 \ldots$ and decrease the neighborhood size as well as α as $t \to \infty$ or $t_1, t_2, \ldots$

i.e.

$1 \leq t \leq t_1$	$N_m = N_3$	$\alpha = \alpha_3$
$t_1 < t \leq t_2$	$N_m = N_2$	$\alpha = \alpha_2$
⋮		

OR do it automatically:

$$w_j(n+1) = w_j(n) + \alpha(t) \cdot e^{-\frac{\|r_j - r_m\|}{\sigma^2(t)}} \left(x - w_j(n) \right) \quad j \in N_m(t)$$

r_j: position of j^{th} neuron wrt winning neuron

$$\alpha(t) = \alpha_0 \, e^{-t/\tau_2}$$

$$\sigma(t) = \sigma_0 \, e^{t/\tau_1}$$

— After convergence

Do calibration.