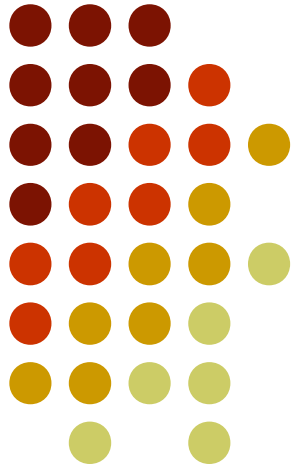
PGCA009 –Inteligência Computacional
Aula 5 Rede RBF

Prof. Angelo Loula

Mestrado em Computação Aplicada (UEFS)



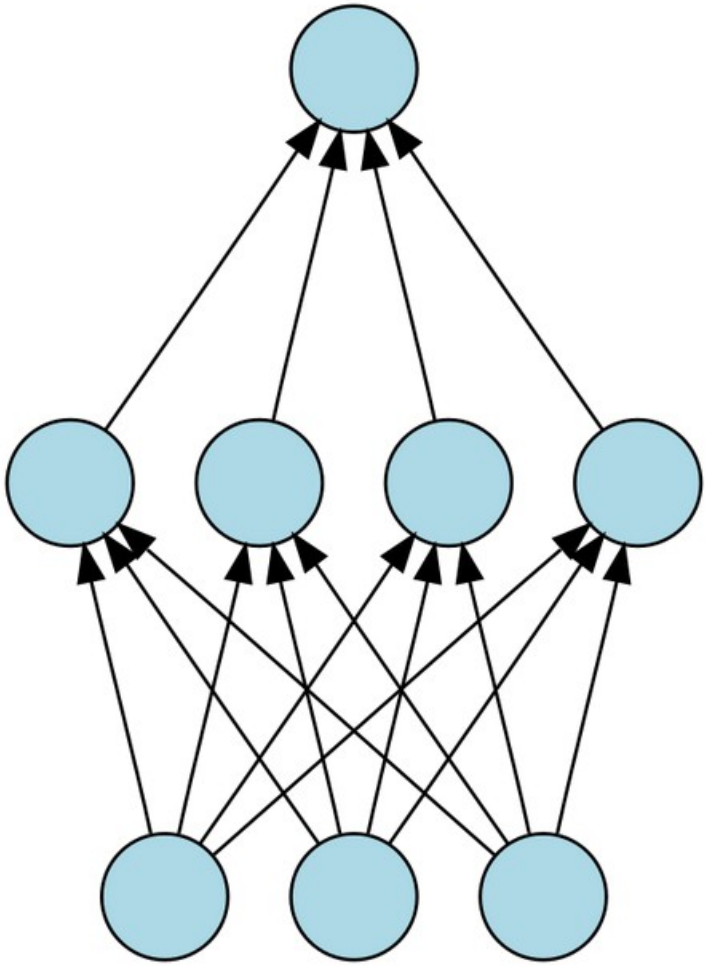
Redes Neurais RBF

- Radial Basis Function Neural Network
  - Utilizam uma função de base radial como função de transferência
    - Somente na camada intermediária
  - Realiza uma combinação linear destas funções para compor uma aproximação do mapeamento entrada-saída
    - Neurônios de saída são lineares,
    - mas as saídas são discretas, como em classificação, pode usar sigmóide ou degrau

Redes Neurais RBF

- Radial Basis Function Neural Network
  - Uma rede RBF tem capacidade de aproximação universal dada uma quantidade suficiente de neurônios na camada intermediária
  - vs MLP: treinamento mais rápido, interpretação clara, mais dados para treinar, aproximações locais, não apropriada para dados de alta dimensionalidade

Redes Neurais RBF

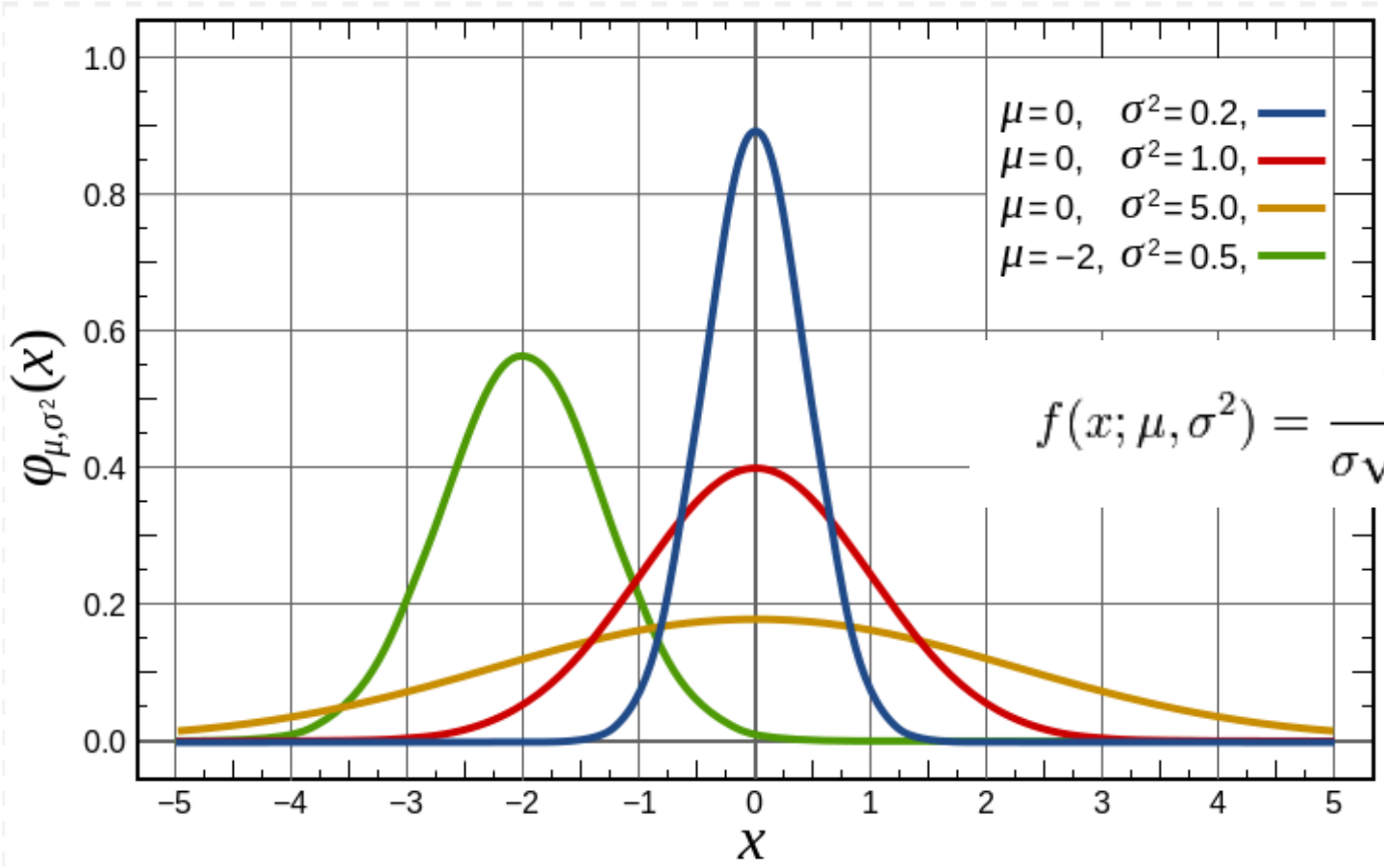


Redes Neurais RBF

- Função de Base Radial
  - Função com valores monotonicamente decrescentes (ou crescentes) a partir de um ponto central
  - Ponto central e taxa de decréscimo são parâmetros da função

Redes Neurais RBF

Exemplo: Gaussiana



$$f(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Redes Neurais RBF

- Ativação da camada intermediária pela entrada
  - Ao invés de multiplicação de entradas por pesos
  - Usamos a distância (norma) da entrada em relação ao centro da RBF

$$\|X - C_i\|$$

Redes Neurais RBF

- Saída da rede para uma função de transferência gaussiana

$$y_j = \sum_{i=1}^{m} w_{ij} h_i(X) \qquad h_i(x) = e^{-\frac{\|X - C_i\|^2}{\sigma_i}}$$

m: quantidade de neurônios na camada intermediária

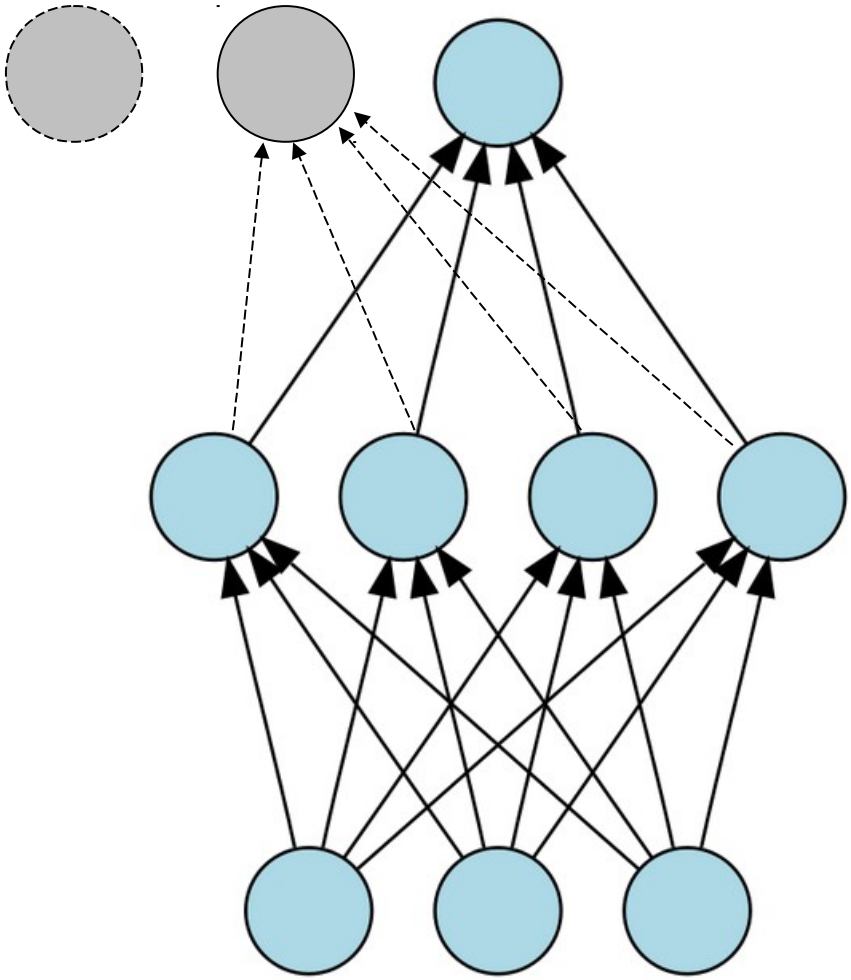
w_{ij}: peso entre o neurônio intermediário i e o neurônio de saída j

h_i: função de transferência do neurônio intermediário i

X: vetor de entrada

C_i: vetor do centro da função h_i

Redes Neurais RBF



Output y

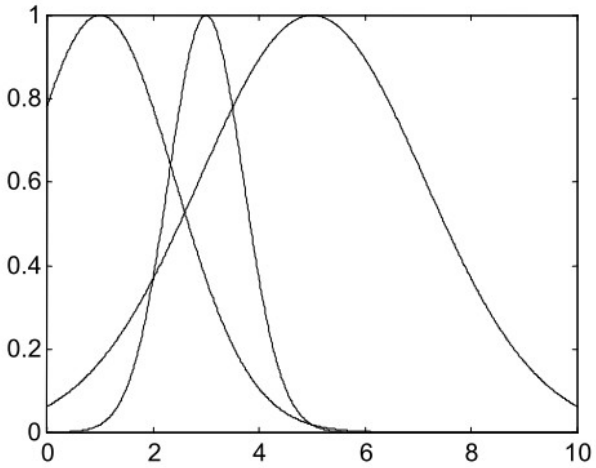
Linear weights w_{ij}

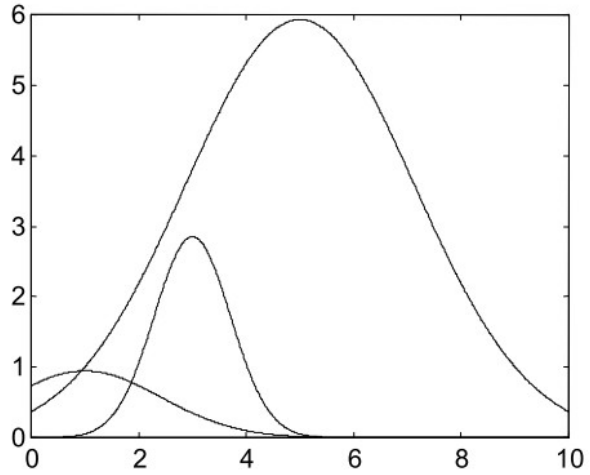
Radial basis functions h_i

Weights C_i

Input x X

Redes Neurais RBF



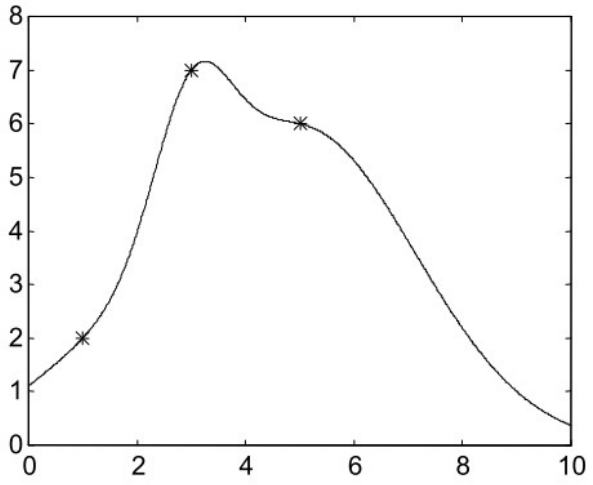


Caso 1: $m = N$

Pontos amostrados: (1,2); (3,7); (5,6)

$$\mathbf{c} = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}; \quad \mathbf{r} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}; \quad \mathbf{w} = \begin{bmatrix} 0.945 \\ 2.850 \\ 5.930 \end{bmatrix}$$

Obs: As funções de base radial têm centros n valores de x e dispersões arbitrárias.



baseado em slides do prof Fernando von Zuben (Unicamp)

Redes Neurais RBF

- Treinamento da rede
  - Determinar centros C_i
  - Determinar larguras σ_i
    - Por aprendizado não supervisionado
  - Determinar pesos w_{ij}
    - Por aprendizado supervisionado

Redes Neurais RBF

- Treinamento da rede
  - Determinar centros C_i
    - Seleção aleatória
      - Muitos centros, centros próximos
    - Distribuição regular no espaço de entrada
      - Muitos centros
    - Clustering (Agrupamento)
      - ex: K-means, SOM
    - Outras: ex computação evolutiva

Redes Neurais RBF

- Treinamento da rede
  - Determinar larguras σ_i
    - Distância média entre centros
    - Distância do centro aos dados próximos
    - Distância do centro aos centros próximos

Redes Neurais RBF

- Treinamento da rede
  - Determinar pesos w_{ij}
    - Admitindo C_i e σ_i fixos
    - Ajuste linear→ método dos quadrados mínimos
    - Pode usar regra Delta

Redes Neurais RBF

- Método dos quadrados mínimos

$$\mathbf{w} = \left(\mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{s}$$

saídas esperadas

$$\mathbf{s} = \begin{bmatrix} s_1 \\ \vdots \\ s_N \end{bmatrix}$$

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \cdots & \mathbf{h}_m \end{bmatrix} = \begin{bmatrix} h_1(\mathbf{x}_1) & h_2(\mathbf{x}_1) & \cdots & h_m(\mathbf{x}_1) \\ h_1(\mathbf{x}_2) & h_2(\mathbf{x}_2) & \cdots & h_m(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ h_1(\mathbf{x}_N) & h_2(\mathbf{x}_N) & \cdots & h_m(\mathbf{x}_N) \end{bmatrix}$$

Example

- An often quoted example which shows how the RBF network can handle a non-linearly separable function is the exclusive-or problem.
- One solution has 2 inputs, 2 hidden units and 1 output.
- The centres for the two hidden units are set at $c_1 = 0,0$ and $c_2 = 1,1$, and the value of radius σ is chosen such that $2\sigma^2 = 1$.

Exemplo obtido em: http://www.computing.surrey.ac.uk/courses/csm10/

Example

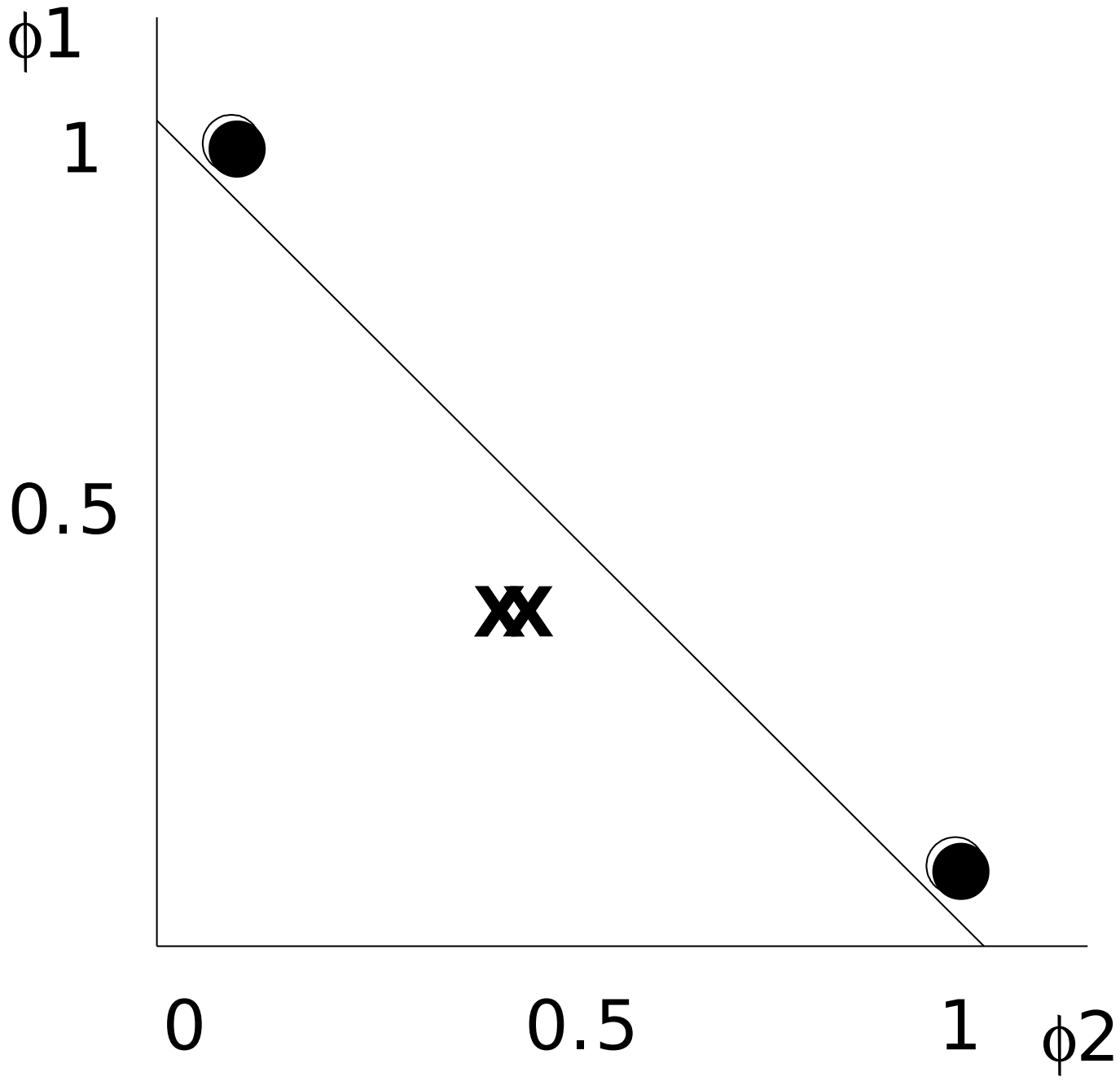
- The inputs are x, the distances from the centres squared are r, and the outputs from the hidden units are φ.
- When all four examples of input patterns are shown to the network, the outputs of the two hidden units are shown in the following table.

distância ao centro *ativação do neurônio*

x_1	x_2	r_1	r_2	φ_1	φ_2
0	0	0	2	1	0.1
0	1	1	1	0.4	0.4
1	0	1	1	0.4	0.4
1	1	2	0	0.1	1

Example

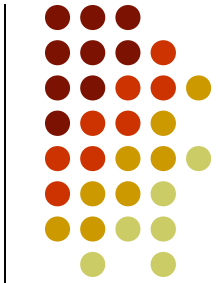
- Next Fig. shows the position of the four input patterns using the output of the two hidden units as the axes on the graph - it can be seen that the patterns are now linearly separable.
- This is an ideal solution - the centres were chosen carefully to show this result.
- Methods generally adopted for learning in an RBF network would find it impossible to arrive at those centre values - later learning methods that are usually adopted will be described.


ϕ1
1
0.5
0
0.5
1
ϕ2

Training hidden layer

- The hidden layer in a RBF network has units which have weights that correspond to the vector representation of the centre of a cluster.
- These weights are found either using a traditional clustering algorithm such as the *k*-means algorithm, or adaptively using essentially the Kohonen algorithm.

Training hidden layer



- In either case, the training is unsupervised but the number of clusters that you expect, *k*, is set in advance. The algorithms then find the best fit to these clusters.
- The *k* -means algorithm will be briefly outlined.
- Initially *k* points in the pattern space are randomly set.

Training hidden layer

- Then for each item of data in the training set, the distances are found from all of the *k* centres.
- The closest centre is chosen for each item of data - this is the initial classification, so all items of data will be assigned a class from 1 to *k*.
- Then, for all data which has been found to be class 1, the average or mean values are found for each of co-ordinates.

Training hidden layer

- These become the new values for the centre corresponding to class 1.
- Repeated for all data found to be in class 2, then class 3 and so on until class *k* is dealt with - we now have *k* new centres.
- Process of measuring the distance between the centres and each item of data and re-classifying the data is repeated until there is no further change – i.e. the sum of the distances monitored and training halts when the total distance no longer falls.

Adaptive k-means

- The alternative is to use an adaptive *k* -means algorithm which similar to Kohonen learning.
- Input patterns are presented to all of the cluster centres one at a time, and the cluster centres adjusted after each one. The cluster centre that is nearest to the input data wins, and is shifted slightly towards the new data.
- This has the advantage that you don't have to store all of the training data so can be done on-line.

Finding radius of Gaussians

- Having found the cluster centres using one or other of these methods, the next step is determining the radius of the Gaussian curves.
- This is usually done using the *P*-nearest neighbour algorithm.
- A number *P* is chosen, and for each centre, the *P* nearest centres are found.

Finding radius of Gaussians

- The root-mean squared distance between the current cluster centre and its *P* nearest neighbours is calculated, and this is the value chosen for σ.
- So, if the current cluster centre is c_j, the value is:

$$\sigma_j = \sqrt{\frac{1}{P}\sum_{i=1}^{P}(c_k - c_i)^2}$$

Finding radius of Gaussians

- A typical value for P is 2, in which case σ is set to be the average distance from the two nearest neighbouring cluster centres.

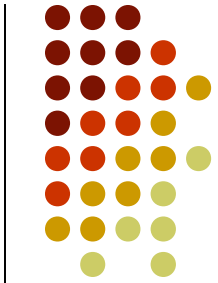
XOR example

- Using this method, XOR function can be implemented using a minimum of 4 hidden units.
- If more than four units are used, the additional units duplicate the centres and therefore do not contribute any further discrimination to the network.
- So, assuming four neurons in the hidden layer, each unit is centred on one of the four input patterns, namely 00, 01, 10 and 11.

- The *P*-nearest neighbour algorithm with *P* set to 2 is used to find the size of the radii.
- In each of the neurons, the distances to the other three neurons is 1, 1 and 1.414, so the two nearest cluster centres are at a distance of 1.
- Using the mean squared distance as the radii gives each neuron a radius of 1.
- Using these values for the centres and radius, if each of the four input patterns is presented to the network, the output of the hidden layer would be:

input	neuron 1	neuron 2	neuron 3	neuron 4
00	0.6	0.4	1.0	0.6
01	0.4	0.6	0.6	1.0
10	1.0	0.6	0.6	0.4
11	0.6	1.0	0.4	0.6

Training output layer



- Having trained the hidden layer with some unsupervised learning, the final step is to train the output layer using a standard gradient descent technique such as the Least Mean Squares algorithm.
- In the example of the exclusive-or function given above a suitable set of weights would be +1, –1, –1 and +1. With these weights the value of *net* and the output is:

input	neuron 1	neuron 2	neuron 3	neuron 4	*net*	output
00	0.6	0.4	1.0	0.6	–0.2	0
01	0.4	0.6	0.6	1.0	0.2	1
10	1.0	0.6	0.6	0.4	0.2	1
11	0.6	1.0	0.4	0.6	–0.2	0