Introduction to computational models

Lab assignment 3. Radial basis function neural networks

Pedro Antonio Gutiérrez
pagutierrez@uco.es

Module "Introduction to computational models"
4th year of "Grado en Ingeniería Informática"
Especialidad Computación
Escuela Politécnica Superior
(Universidad de Córdoba)

16th November 2021

Objectives of the lab assignment

- To familiarise the student with the concept of the neuronal network of radially based functions (RBF).
- To implement such a network.
- To familiarise the learner with the use of `scikit-learn` as an environment for the creation of automatic learning models.

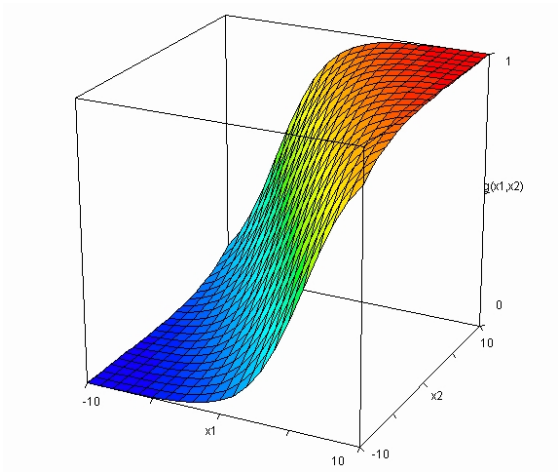
Radial basis function neural network

- Radial Basis Function Neural Networks (RBFNNs): they are based on a local approach.
  - The hidden layer neurons are radial basis functions (RBFs): local functions.
  - Opposite to MLP networks, where the hidden layer neurons are of the sigmoid type: projection functions.
- Projection functions: high value, not zero, over a wide region of the input space.
- Local functions: high value, not zero, only on a localised region of the input space.

Local Vs global functions

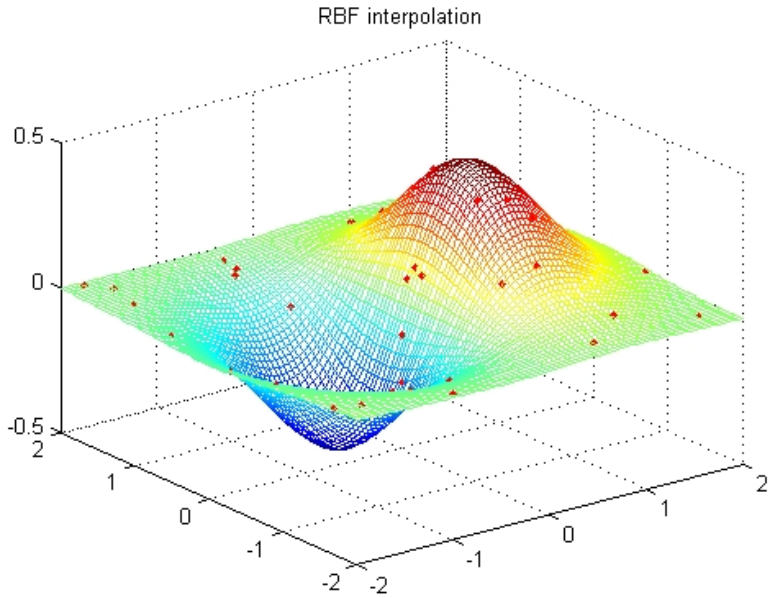
Sigmoidal units

- Additive projection model.

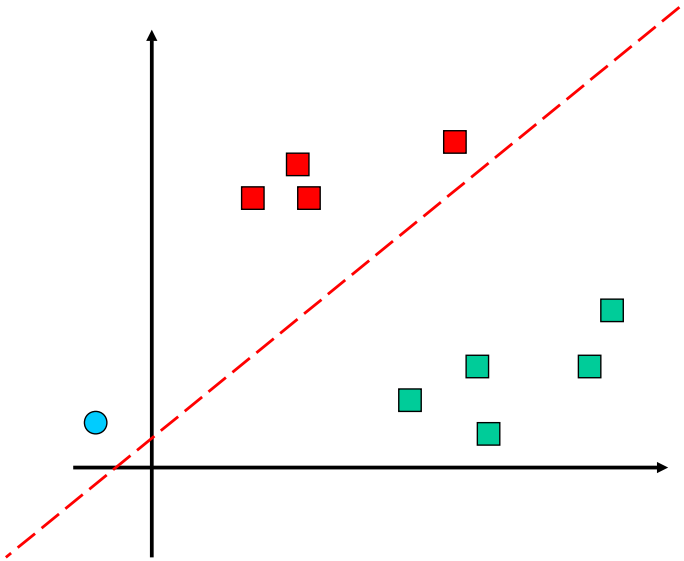


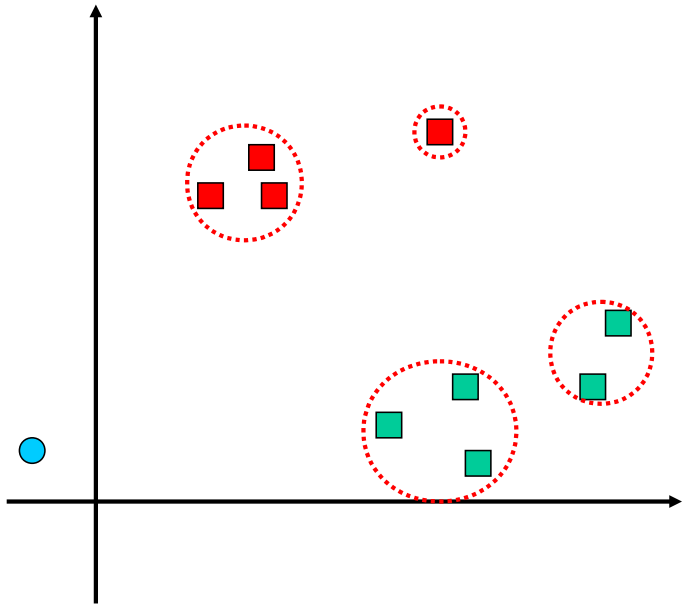
Radial basis function

- Local model.



Local Vs global functions





RBFs

- A function is said to be RBF if its output depends on the distance between the input vector and a vector stored in it (centre).
- Each RBF stores a centre as a reference, and every time a new pattern is presented to it, the distance to this centre is calculated.
  - If the distance is small, the RBF is activated (its output is 1).
  - If the distance is high, the RBF is not activated (its output is 0).
- What do we consider as large or small? $\Rightarrow$ For this, we incorporate an additional element: the radius.
  - If the radius is small, activation will only occur when the pattern is very close to the centre.
  - If the radius is large, the activation will take place at a greater distance.

RBFs

- There are many functions that fulfil these properties: Cauchy RBF, inverse multi-quadratic...
- But the most common is the Gaussian function:

$$\varphi(\mathbf{x}, \mathbf{c}, \sigma) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}\|^2}{2(\sigma)^2}\right)$$

  where $\mathbf{c}$ if the centre of the RBF, σ is the radius and $\mathbf{x}$ is the pattern we are evaluating.
- $\|\mathbf{x} - \mathbf{c}\|$ is the norm of the difference vector between the centre and the pattern, or what is the same as the Euclidean distance:

$$\|\mathbf{x} - \mathbf{c}\| = \sqrt{\sum_{i=1}^{n}(x_i - c_i)^2}$$

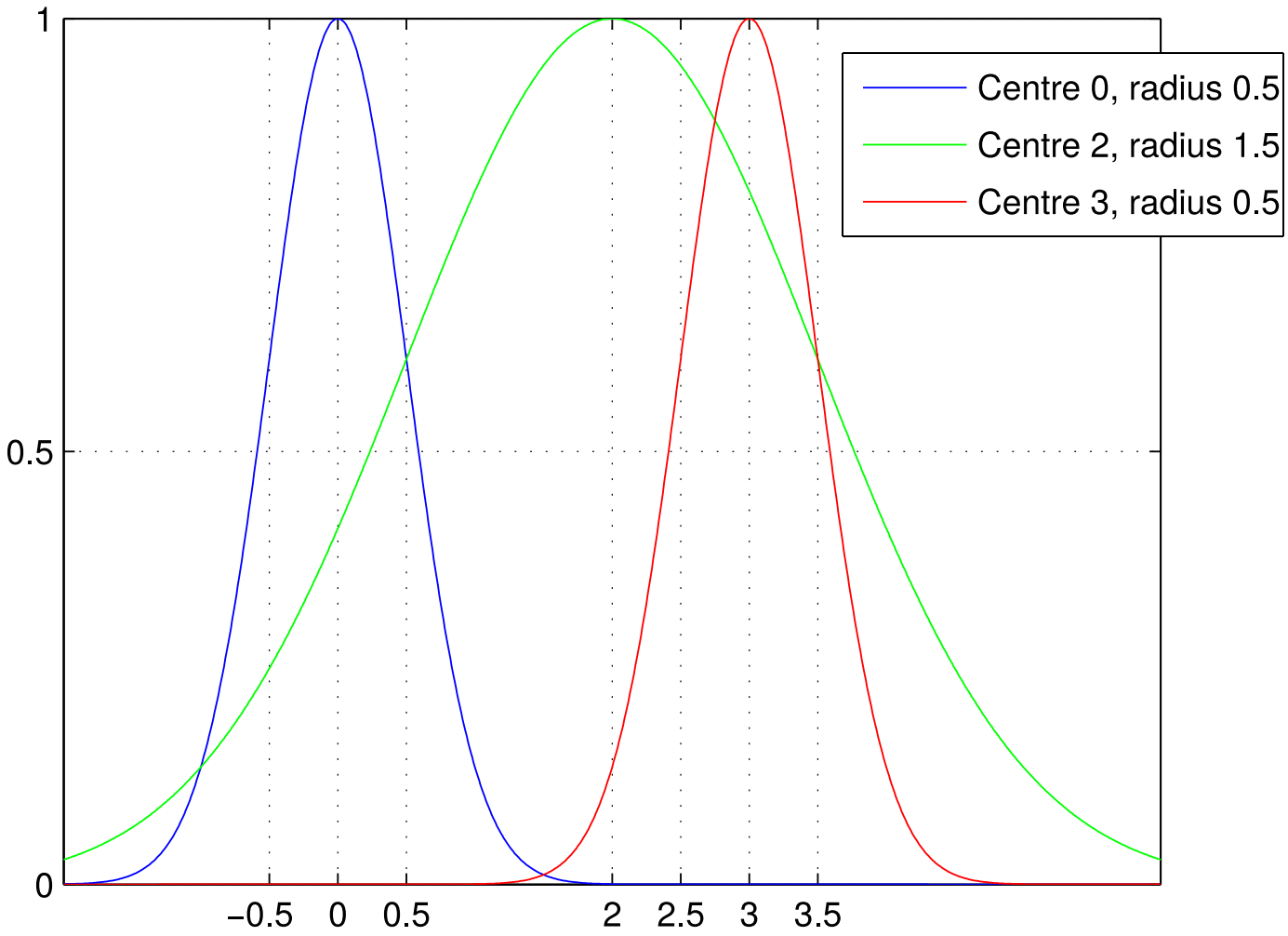
RBFs

- By linking the two expressions and simplifying:

$$\varphi(\mathbf{x}, \mathbf{c}, \sigma) = \exp\left(-\frac{\left(\sqrt{\sum_{i=1}^{n}(x_i - c_i)^2}\right)^2}{2(\sigma)^2}\right)$$

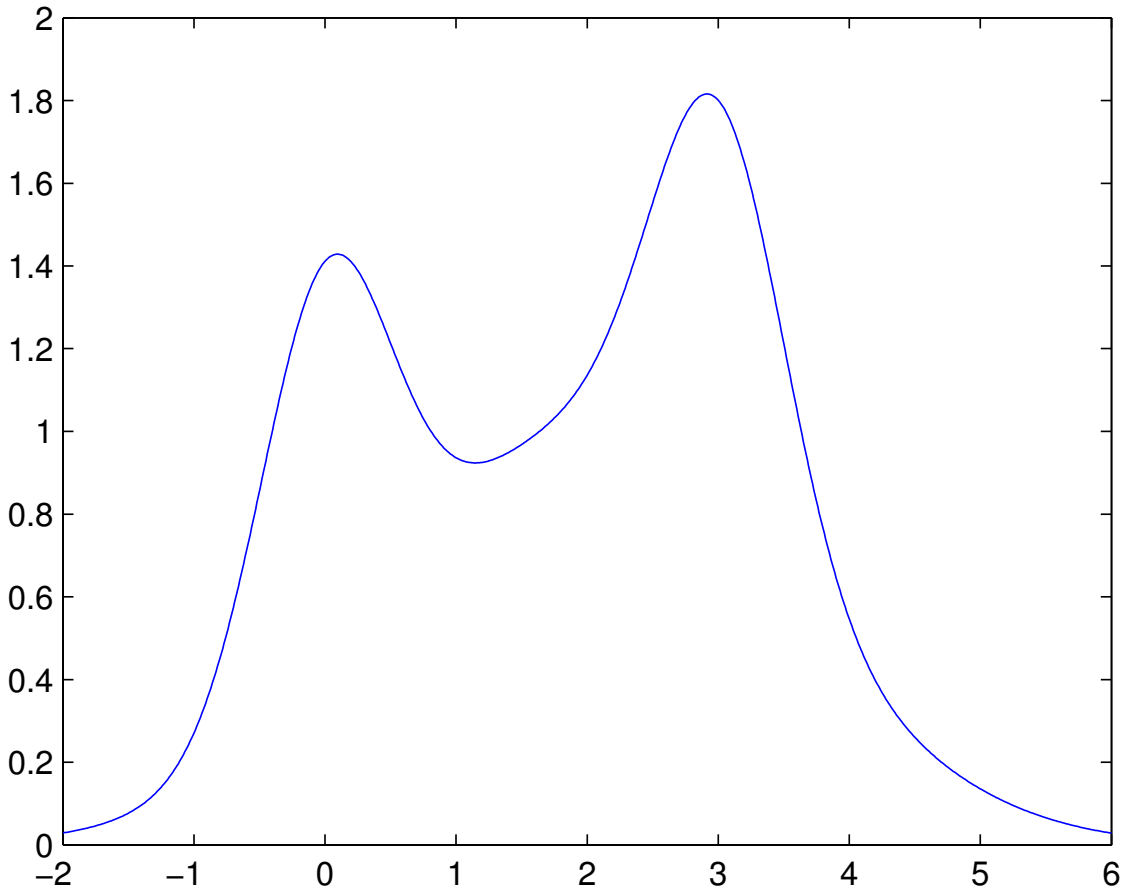
$$\varphi(\mathbf{x}, \mathbf{c}, \sigma) = \exp\left(-\frac{\sum_{i=1}^{n}(x_i - c_i)^2}{2(\sigma)^2}\right)$$

RBFs: effect of the centre and the radius

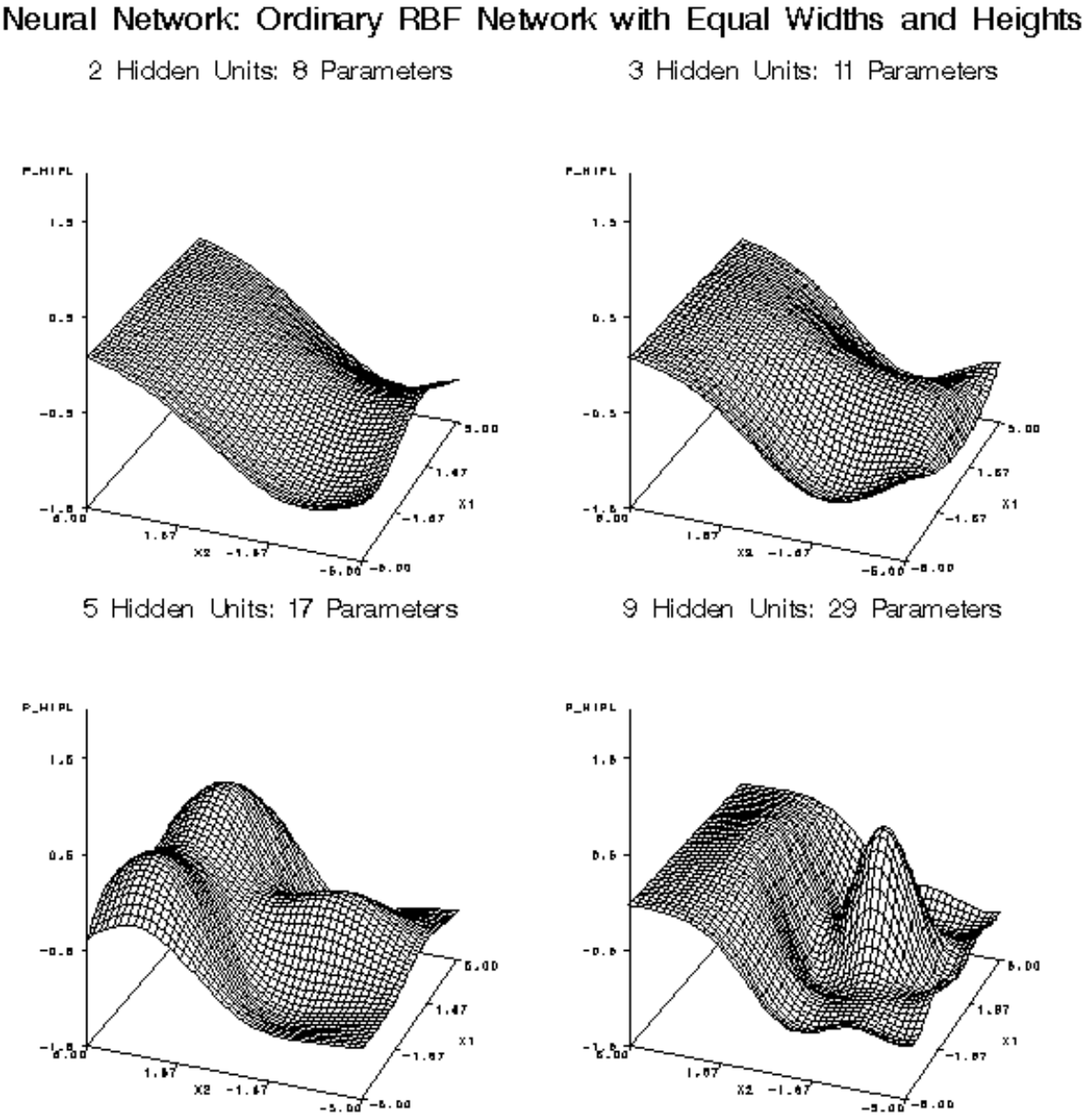


RBF neural network

Sum of the previous two RBFs:



Example in 2D:



Architecture of an RBFNN

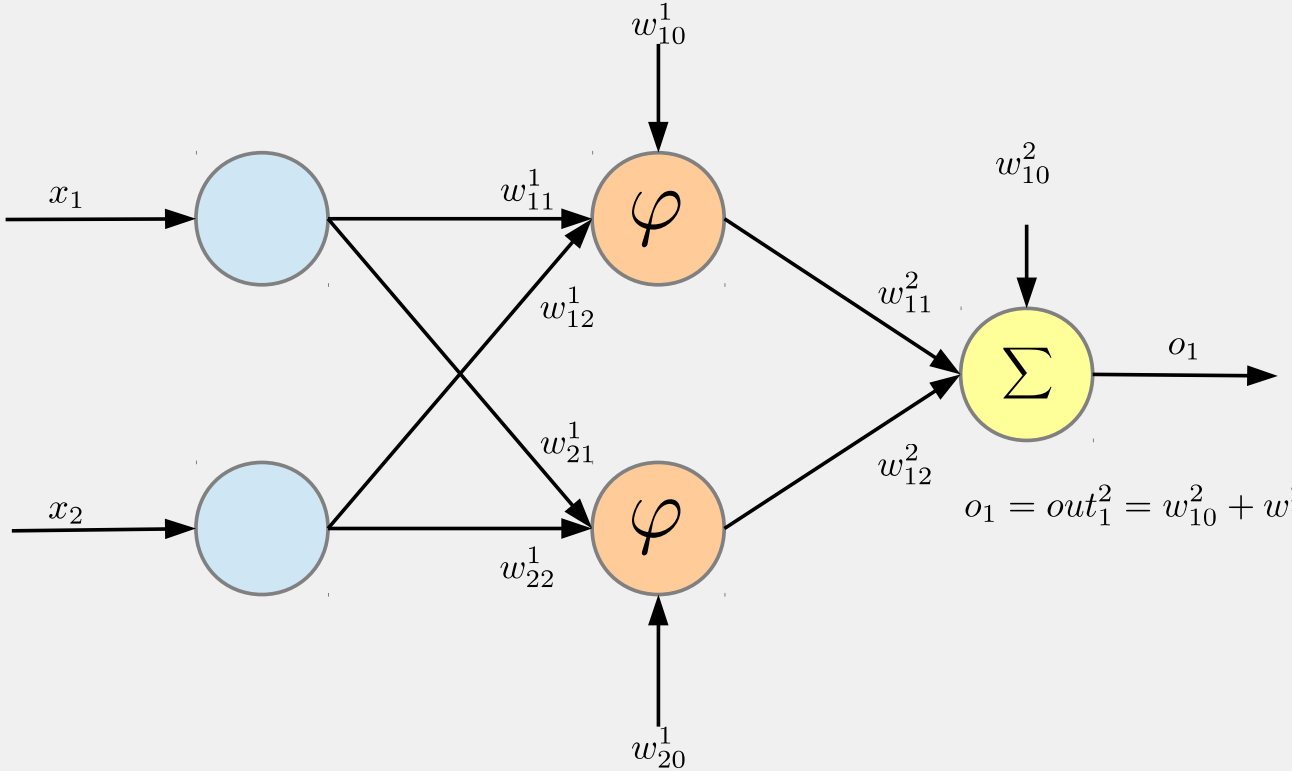
- Three layers:
  - Input layer.
  - Hidden layer (RBF functions).
  - Output layer:
    - Regression: linear function (weighted sum of the RBF outputs).
    - Classification: *softmax* function.

Architecture of an RBFNN

- Following the notation we used for the MLP:
  - RBF neurons:
    - $net_j^h = \sum_{i=1}^{n_{h-1}} \left(w_{ji}^h - out_i^{h-1} \right)^2$
    - $out_j^h = \exp\left(-\frac{net_j^h}{2\left(w_{j0}^h\right)^2} \right)$
  - Linear type neurons:
    - $out_j^H = net_j^H = w_{j0}^H + \sum_{i=1}^{n_{H-1}} w_{ji}^H out_i^{H-1}$
  - *softmax* neurons:
    - $net_j^H = w_{j0}^H + \sum_{i=1}^{n_{H-1}} w_{ji}^H out_i^{H-1}$
    - $out_j^H = \frac{\exp(net_j^H)}{\sum_{i=1}^{n_H} \exp(net_i^H)}$

Architecture of an RBFNN (regression)

$$out_1^1 = \varphi(\mathbf{x}, \mathbf{c}_1, \sigma_1) = \varphi(\{x_1, x_2\}, \{w_{11}^1, w_{12}^1\}, w_{10}^1)$$



$$o_1 = out_1^2 = w_{10}^2 + w_{11}^2 \cdot out_1^1 + w_{12}^2 \cdot out_2^1$$

$$out_2^1 = \varphi(\mathbf{x}, \mathbf{c}_2, \sigma_2) = \varphi(\{x_1, x_2\}, \{w_{21}^1, w_{22}^1\}, w_{20}^1)$$

Training of an RBFNN

- How do we adjust the parameters?
  - RBF functions are derivable $\rightarrow$ Apply the error backpropagation algorithm (fully supervised training).
    - Derivatives would have to be calculated with respect to radius and centres.
    - They are complex and have a higher computational cost than for the MLP.
  - Hybrid training: unsupervised part (*clustering*) and supervised part (logistic regression or matrix inversion).
    - The local properties of the RBF networks are better exploited.
    - The computational cost is generally lower than using the gradient descent algorithm.

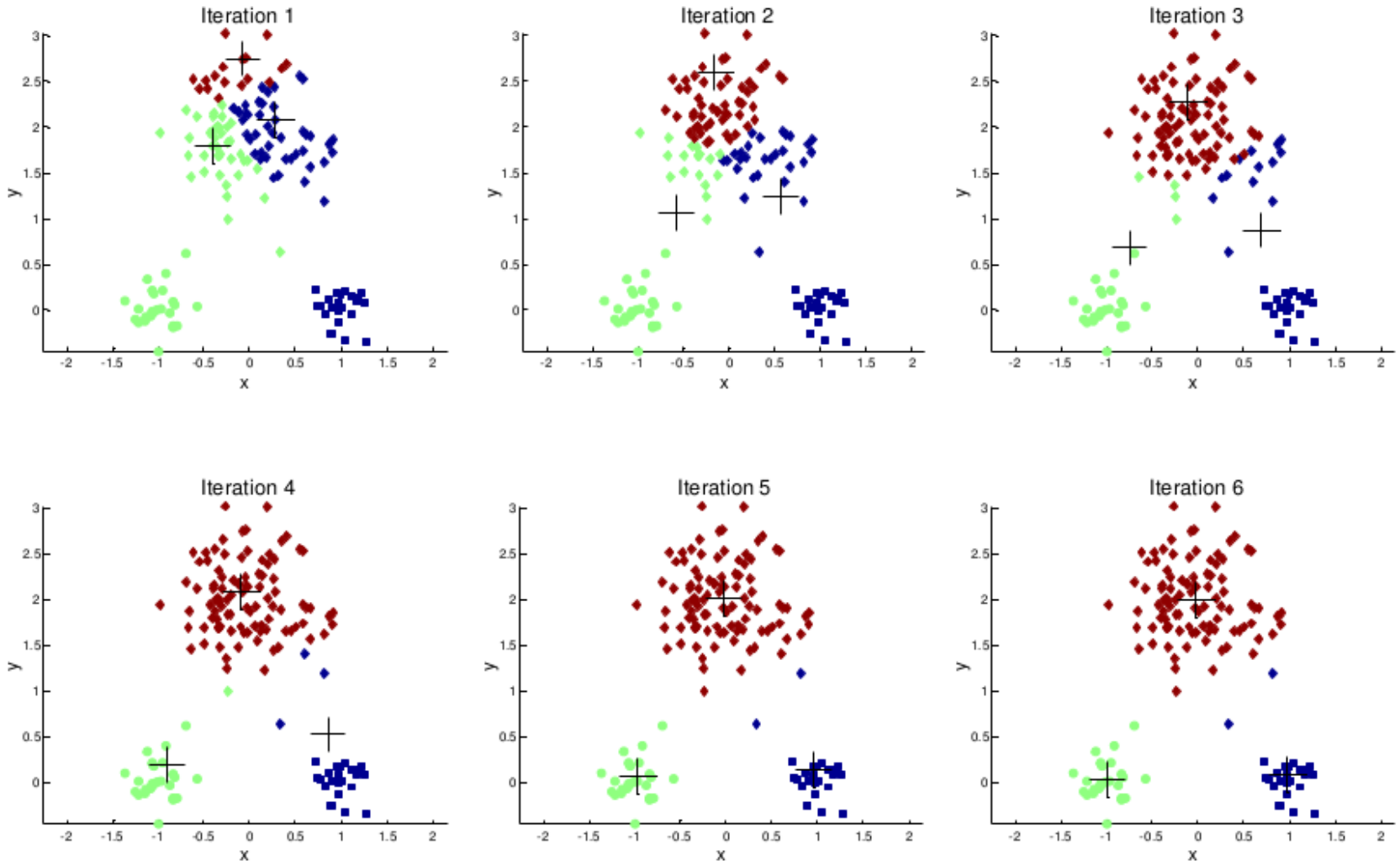
Training of an RBFNN

- We have to obtain three things:
  1. Coordinates of the RBF centres: $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{n_1}$.
     - Weights from input layer to hidden layer:
       $\mathbf{c}_1 = \{w_{11}^1, w_{12}^1, \ldots, w_{1n}^1\}$
       $\mathbf{c}_2 = \{w_{21}^1, w_{22}^1, \ldots, w_{2n}^1\}$
       $\ldots$
       $\mathbf{c}_{n_1} = \{w_{n_1 1}^1, w_{n_1 2}^1, \ldots, w_{n_1 n}^1\}$.
  2. Width of the RBF: $\sigma_1, \sigma_2, \ldots, \sigma_{n_1}$.
     - We will use the bias position:
       $\sigma_1 = w_{10}^1, \sigma_2 = w_{20}^1, \ldots, \sigma_{n_1} = w_{n_1 0}^1$
  3. Weights from hidden layer to output layer (with bias):
     $w_{10}^2, w_{11}^2, w_{12}^2, \ldots, w_{1n_1}^2$
     $w_{20}^2, w_{21}^2, w_{22}^2, \ldots, w_{2n_1}^2$
     $\ldots$
     $w_{k0}^2, w_{k1}^2, w_{k2}^2, \ldots, w_{kn_1}^2$

Training of an RBFNN: phase 1 (clustering)

- The adjustment of the centres in the network can be done by a *clustering* procedure.
- The idea is to detect the group of patterns (or *clusters*) in the input space and locate one RBF in each *cluster*.
- We are going to use the most popular clustering method, the K-means.
- The coordinates of the centroids of each *cluster* will be the centres of the RBFs.

Training of an RBFNN: phase 1 (clustering)



Training of an RBFNN: phase 1 (clustering)

- K-means: partitional clustering method.
  - The number of *clusters* must be specified, in our case, it will be the number of hidden neurons of our RBFNN (n_1).
  - Each *cluster* has a centroid (geometric mean of the *cluster*).
  - The points are assigned to the *cluster* with the closest centroid (using any distance metric).
  - Iteratively, we update the centroids as a function of the assignments of the points to the *clusters*, until the centroids no longer change.
  - The results depend on the initialization of the centroids:
    - For classification, we select randomly, and in a stratified manner, n_1 patterns.
    - For regression, we randomly select n_1 patterns.

Training of an RBFNN: phase 2 (adjustment of the radii)

- More complex procedures (density estimation) can be used to fine tune the radii of the RBFs.
- However, we will adjust the radii in a very simple way, we will take half (because it is a radius and not a diameter) of the average distance to the rest of centroids.
- That is, the radius of the neuron j will be:

$$\sigma_j = w_{j0}^1 = \frac{1}{2 \cdot (n_1 - 1)} \sum_{i \neq j} \|c_j - c_i\| = \quad (1)$$

$$= \frac{1}{2 \cdot (n_1 - 1)} \sum_{i \neq j} \sqrt{\sum_{d=1}^{n} (c_{jd} - c_{id})^2} \quad (2)$$

Training of an RBFNN: phase 3 (output layer weights, case 1 classification)

- We will adjust the weights of the output layer in two ways, depending on whether we are facing a classification or regression problem.
  - For classification, the weights will be adjusted using logistic regression.
  - For regression, the weights will be adjusted using the pseudo-inverse.

Training of an RBFNN: phase 3 (output layer weights, case 1 classification)

- In both cases, we will need the RBF output matrix, which we will call **R**:

$$\mathbf{R} = \begin{pmatrix} out_1^1(\mathbf{x}_1) & out_2^1(\mathbf{x}_1) & \dots & out_{n_1}^1(\mathbf{x}_1) & 1 \\ out_1^1(\mathbf{x}_2) & out_2^1(\mathbf{x}_2) & \dots & out_{n_1}^1(\mathbf{x}_2) & 1 \\ \dots & \dots & \dots & \dots & \\ out_1^1(\mathbf{x}_N) & out_2^1(\mathbf{x}_N) & \dots & out_{n_1}^1(\mathbf{x}_N) & 1 \end{pmatrix} \quad (3)$$

where $out_j^1(\mathbf{x}_i)$ is the output of the j-th RBF neuron when it is feed using the training pattern $\mathbf{x}_i$. To simulate the bias we have included a constant column equal to 1.

Training of an RBFNN: phase 3 (output layer weights, case 1 classification)

- Once the matrix has been constructed, in the case of classification, we will apply logistic regression.
  - We will use a function to which we will pass the **R** matrix as if it were the input matrix of my database.
  - Logistic regression is a linear classification model, which approximates the probability of belonging to a class in the following way (*softmax* function):

$$P(\mathbf{x} \in C_j) = o_j = \frac{exp(\beta_{j0} + \sum_{i=1}^{n} \beta_{ji} x_i)}{\sum_{l=1}^{k} exp(\beta_{l0} + \sum_{i=1}^{n} \beta_{li} x_i)} \quad (4)$$

  - The objective of the logistic regression is to obtain the values of β_{ji} that maximize the cross entropy:

$$L = \frac{1}{N} \sum_{p=1}^{N} \left(\frac{1}{k} \sum_{o=1}^{k} d_j \ln(o_j) \right) \quad (5)$$

Training of an RBFNN: phase 3 (output layer weights, case 1 classification)

- ...we will apply regresión logística.
  - Logistic regression can include regularization, which is a mechanism for making the maximum number of parameters β_{ji} tend to zero (or almost zero).
    - L2 regularization:

$$L = \left(\frac{1}{N} \sum_{p=1}^{N} \left(\frac{1}{k} \sum_{o=1}^{k} d_j \ln(o_j) \right) \right) - \eta \left(\sum_{j=1}^{k} \sum_{i=0}^{n} \beta_{ji}^2 \right) \quad (6)$$

    - L1 regularization:

$$L = \left(\frac{1}{N} \sum_{p=1}^{N} \left(\frac{1}{k} \sum_{o=1}^{k} d_j \ln(o_j) \right) \right) - \eta \left(\sum_{j=1}^{k} \sum_{i=0}^{n} |\beta_{ji}| \right) \quad (7)$$

  - The η parameter must be set by the user and establishes the importance given to regularization.

Training of an RBFNN: phase 3 (output layer weights, case 1 classification)

- ...we will apply regresión logística.
  - Regularisation provides simpler models that are less prone to over-fitting.
  - Difference between L2 and L1:
    - The L2 regularisation tends to provide smaller weights (although not necessarily equal to zero).
    - L1 regularization tends to prune more variables, making many weights equal to zero (although non-zero weights are not necessarily small in absolute value).

Training of an RBFNN: phase 3 (output layer weights, case 2 regression)

- Once the matrix has been constructed, in the case of regression, we will apply the pseudo-inverse.
  - From the point of view of linear algebra, the output of the network can be written as:

$$\mathbf{R}_{(N\times(n_1+1))} \times \boldsymbol{\beta}^{\mathrm{T}}_{((n_1+1)\times k)} = \hat{\mathbf{Y}}_{(N\times k)} \tag{8}$$

where $\mathbf{R}$ is the matrix containing the outputs of the RBF neurons, $\boldsymbol{\beta}$ is a matrix containing a vector of parameters for each output to be predicted and $\hat{\mathbf{Y}}$ s a matrix with all the estimated outputs.

$$\begin{pmatrix} out^1_{11} & \dots & out^1_{n_1 1} & 1 \\ out^1_{12} & \dots & out^1_{n_1 2} & 1 \\ \dots & \dots & \dots & \dots \\ out^1_{1N} & \dots & out^1_{n_1 N} & 1 \end{pmatrix} \begin{pmatrix} \beta_{11} & \beta_{21} & \dots & \beta_{k1} \\ \dots & \dots & \dots & \dots \\ \beta_{1n_1} & \beta_{2n_1} & \dots & \beta_{kn_1} \\ \beta_{10} & \beta_{20} & \dots & \beta_{k0} \end{pmatrix} \tag{9}$$

Training of an RBFNN: phase 3 (output layer weights, case 2 regression)

- ...we will apply the pseudo-inverse.
  - If we want to obtain the best possible values for the parameters, we use the following equation:

$$\mathbf{R}_{(N\times(n_1+1))} \times \boldsymbol{\beta}^{\mathrm{T}}_{((n_1+1)\times k)} = \mathbf{Y}_{(N\times k)} \tag{10}$$

where $\mathbf{Y}$ is the matrix with desired outputs:

$$\mathbf{Y} = \begin{pmatrix} d_{11} & \dots & d_{1k} \\ d_{21} & \dots & d_{2k} \\ \dots & \dots & \dots \\ d_{N1} & \dots & d_{Nk} \end{pmatrix} \tag{11}$$

Training of an RBFNN: phase 3 (output layer weights, case 2 regression)

- ...we will apply the pseudo-inverse.
  - If $\mathbf{R}$ is square ($N = (n_1 + 1)$), then $\mathbf{R}$ has an inverse and we can clear it directly:

$$\boldsymbol{\beta}^{\mathrm{T}}_{((n_1+1)\times k)} = \left(\mathbf{R}_{(N\times N)}\right)^{-1} \mathbf{Y}_{(N\times k)} \tag{12}$$

  - If $(n_1 + 1) > N$, then there are many solutions, and some kind of feature selection algorithm must be used to reduce the value of n_1.
  - If $(n_1 + 1) < N$ (most common case), there is one single solution, but, given that $\mathbf{R}$ is not square, we have to use the Moore Penrose pseudo-inverse.

Training of an RBFNN: phase 3 (output layer weights, case 2 regression)

- ...we will apply the pseudo-inverse.
  - Moore Penrose pseudo-inverse:

$$\boldsymbol{\beta}^{\mathrm{T}}_{((n_1+1)\times k)} = \left(\mathbf{R}^{+}\right)_{((n_1+1)\times N)} \mathbf{Y}_{(N\times k)} \quad (13)$$

$$\left(\mathbf{R}^{+}\right)_{((n_1+1)\times N)} = \left(\mathbf{R}^{\mathrm{T}}_{((n_1+1)\times N)} \times \mathbf{R}_{(N\times(n_1+1))}\right)^{-1} \mathbf{R}^{\mathrm{T}}_{((n_1+1)\times N)} \quad (14)$$

  - We will use a matrix library to do these operations and obtain $\boldsymbol{\beta}^{\mathrm{T}}_{((n_1+1)\times k)}$.

RBF training algorithm *off-line*

Start

1. initialCentroids $\leftarrow$ randomly select n_1 patterns (regression) or n_1 patterns in a stratified way (classification).
2. centroids $\leftarrow$ K-means($\mathbf{X}$,n_1,initialCentroids) *// $\mathbf{X}$ is the input matrix with the inputs of all patterns*
3. $\sigma_j \leftarrow$ (average of the distance from j to the rest of centroids)/2.
4. Construct the $\mathbf{R}_{(N\times(n_1+1))}$ matrix, where $\mathbf{R}_{ij} = out_j^1(\mathbf{x}_i)$ for $j \neq (n_1+1)$, and $\mathbf{R}_{ij} = 1$ for $j = (n_1+1)$.
5. **If** classification
   1. outputWeights $\leftarrow$ applyLogisticRegression($\mathbf{R}$,*eta*)
6. **Si** regresión
   1. outputWeights $\leftarrow$ calculatePseudoInverse($\mathbf{R}$)

End

Introduction to computational models

Lab assignment 3. Radial basis function neural networks

Pedro Antonio Gutiérrez
pagutierrez@uco.es

Module "Introduction to computational models"
4th year of "Grado en Ingeniería Informática"
Especialidad Computación
Escuela Politécnica Superior
(Universidad de Córdoba)

16th November 2021