

Introducción a los modelos computacionales

Práctica 3. Redes neuronales de funciones de base radial

Pedro Antonio Gutiérrez
pagutierrez@uco.es

Asignatura “Introducción a los modelos computacionales”
4º Curso Grado en Ingeniería Informática
Especialidad Computación
Escuela Politécnica Superior
(Universidad de Córdoba)

3 de noviembre de 2020



- 1 Contenidos
- 2 Introducción
- 3 Arquitectura de una red RBF
- 4 Entrenamiento de una red RBF
 - Fase 1: clustering
 - Fase 2: ajuste de los radios
 - Fase 3: pesos de la capa de salida



Objetivos de la práctica

- Familiarizar al alumno con el concepto de red neuronal de funciones de base radial (RBF).
- Implementar una red de este tipo.
- Familiarizar al alumno con el uso de `scikit-learn` como entorno para la creación de modelos de aprendizaje automático.



Redes neuronales de funciones de base radial

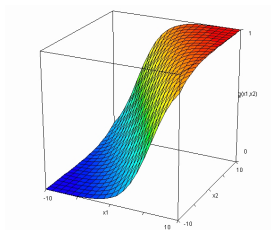
- Redes Neuronales de Funciones de Base Radial (**RBFNN**): se basan en una aproximación local.
 - Las neuronas de capa oculta son funciones de base radial (RBF): **funciones locales**.
 - Al contrario que en las redes tipo perceptrón multicapa, donde las neuronas de capa oculta son tipo sigmoide: **funciones de proyección**.
- **Funciones de proyección**: valor alto, distinto de cero, sobre una región amplia del espacio de entrada.
- **Funciones locales**: valor alto, distinto de cero, sólo sobre una región localizada del espacio de entrada.



Funciones globales Vs locales

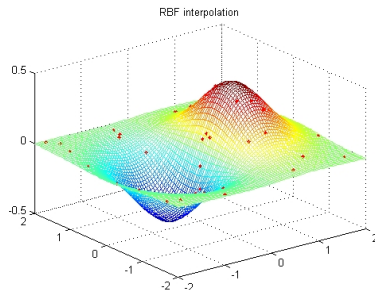
Unidades sigmoide

- Modelo **aditivo** de proyección.

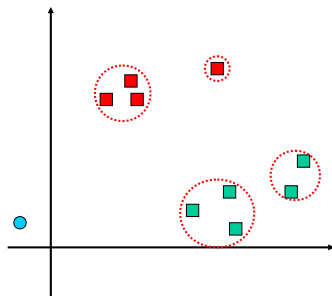
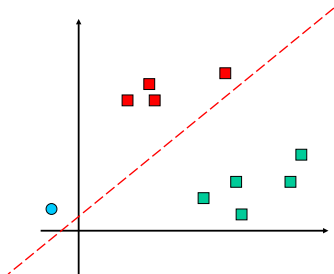


Funciones de Base Radial

- Modelo **local**.



Funciones globales Vs locales



Funciones RBF

- Una función se dice que es RBF si su salida depende de la distancia que hay entre el vector de entrada a la función y un vector almacenado en ella (**centro**).
- Cada RBF guarda un **centro** como referencia y, cada vez que se le presenta un patrón nuevo, se calcula la distancia a dicho centro.
 - Si la distancia es pequeña, la RBF **se activa** (su salida es 1).
 - Si la distancia es grande, la RBF **no se activa** (su salida es 0).
- ¿Qué consideramos como grande o pequeño? \Rightarrow Para eso incorporamos un elemento adicional: el **radio**.
 - Si el radio es pequeño, la activación solo se producirá cuando el patrón esté muy cerca del centro.
 - Si el radio es grande, la activación se producirá a más distancia.



Funciones RBF

- Hay muchas funciones que cumplen estas propiedades: RBF de Cauchy, inversa multicuadrática...
- Pero la más común es la función Gaussiana:

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

donde \mathbf{c} es el centro de la RBF, σ es su ancho y \mathbf{x} es el patrón que estamos evaluando.

- $\|\mathbf{x} - \mathbf{c}\|$ es la norma del vector diferencia entre el centro y patrón, o lo que es lo mismo la distancia euclídea:

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



Funciones RBF

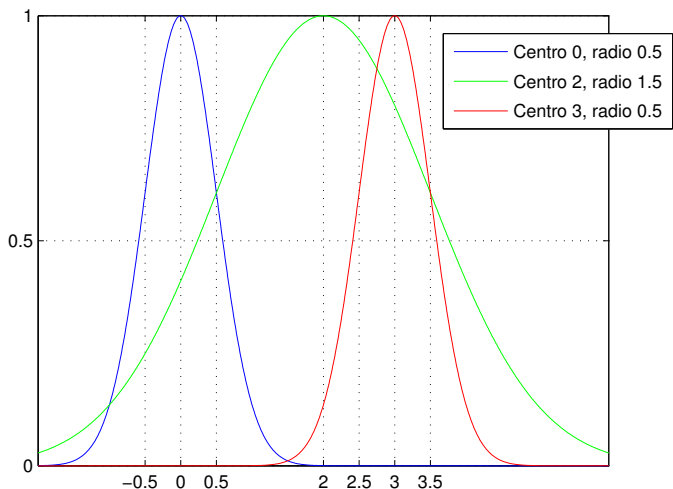
- Uniendo ambas expresiones y simplificando:

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

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

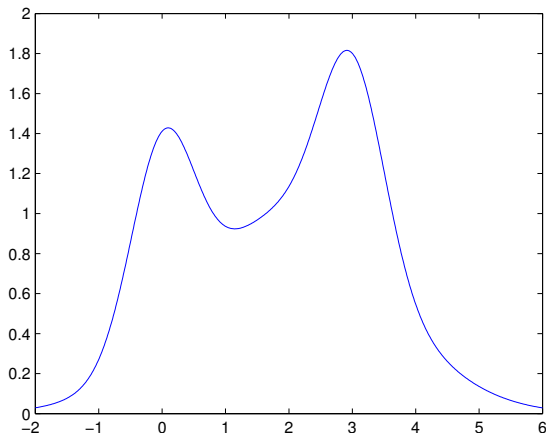


Funciones RBF: efecto del centro y del radio



Red neuronal RBF

Suma de las RBF anteriores:



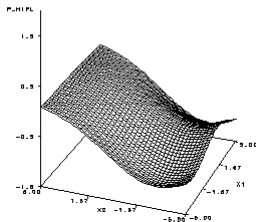
Red neuronal RBF

Ejemplo 2D:

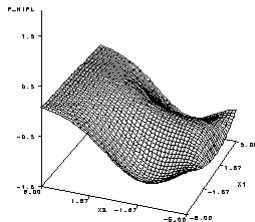
Neural Network: Ordinary RBF Network with Equal Widths and Heights

2 Hidden Units: 8 Parameters

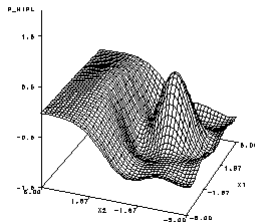
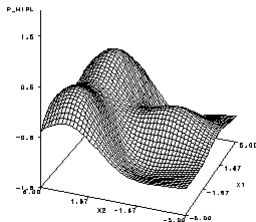
3 Hidden Units: 11 Parameters



5 Hidden Units: 17 Parameters



9 Hidden Units: 29 Parameters



Arquitectura de una RBFNN

- Tres capas:
 - Capa de entrada.
 - Capa oculta (funciones **RBF**).
 - Capa de salida:
 - **Regresión**: función lineal (suma ponderada de la salidas de las RBFs).
 - **Clasificación**: función tipo *softmax*.



Arquitectura de una RBFNN

- Siguiendo la notación que utilizamos para el MLP:

- Neuronas de tipo RBF:

- $net_j^h = \sum_{i=1}^{n_{h-1}} (w_{ji}^h - out_i^{h-1})^2$

- $out_j^h = \exp\left(-\frac{net_j^h}{2 (w_{j0}^h)^2}\right)$

- Neuronas de tipo lineal (en nuestro caso, $h = H$):

- $out_j^H = net_j^H = w_{j0}^H + \sum_{i=1}^{n_{H-1}} w_{ji}^H out_i^{H-1}$

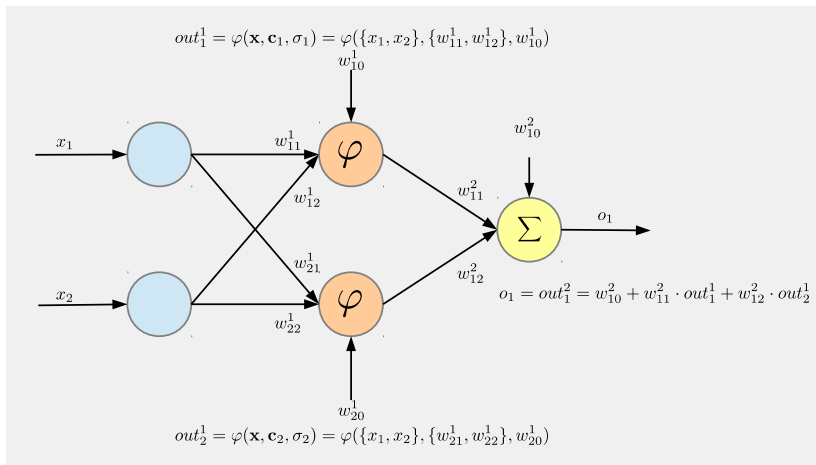
- Neuronas de tipo *softmax* (en nuestro caso, $h = H$):

- $net_j^H = w_{j0}^H + \sum_{i=1}^{n_{H-1}} w_{ji}^H out_i^{H-1}$

- $out_j^H = \frac{\exp(net_j^H)}{\sum_{i=1}^{n_H} \exp(net_i^H)}$



Arquitectura de una RBFNN (Regresión)



Entrenamiento de una red tipo RBF

- ¿Cómo ajustamos los parámetros?
 - Las funciones RBF son derivables → Aplicar el algoritmo de retropropagación (entrenamiento **totalmente supervisado**).
 - Habría que calcular las derivadas con respecto al radio y a los centros.
 - Son complejas y tiene un coste computacional algo mayor que para el perceptrón multicapa.
 - Entrenamiento **híbrido**: parte **no supervisada** (*clustering*) y parte **supervisada** (regresión logística o inversión de una matriz).
 - Las propiedades locales de las redes RBF se aprovechan mejor.
 - El coste computacional es, en general, menor que con un algoritmo de retropropagación.



Entrenamiento de una red tipo RBF

- Tenemos que obtener tres cosas:

- 1 Coordenadas de los centros de las RBF: $\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_{n_1}$.

- Pesos de capa de entrada a capa oculta:

$$\mathbf{c}_1 = \{w_{11}^1, w_{12}^1, \dots, w_{1n}^1\}$$

$$\mathbf{c}_2 = \{w_{21}^1, w_{22}^1, \dots, w_{2n}^1\}$$

...

$$\mathbf{c}_{n_1} = \{w_{n_1 1}^1, w_{n_1 2}^1, \dots, w_{n_1 n}^1\}.$$

- 2 Ancho de las RBF: $\sigma_1, \sigma_2, \dots, \sigma_{n_1}$.

- Usaremos el hueco del sesgo:

$$\sigma_1 = w_{10}^1, \sigma_2 = w_{20}^1, \dots, \sigma_{n_1} = w_{n_1 0}^1$$

- 3 Pesos de capa oculta a capa de salida (con sesgo):

$$w_{10}^2, w_{11}^2, w_{12}^2, \dots, w_{1n_1}^2$$

$$w_{20}^2, w_{21}^2, w_{22}^2, \dots, w_{2n_1}^2$$

...

$$w_{k0}^2, w_{k1}^2, w_{k2}^2, \dots, w_{kn_1}^2$$

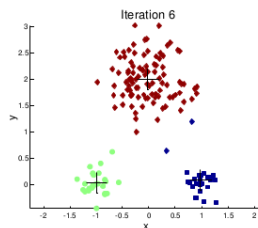
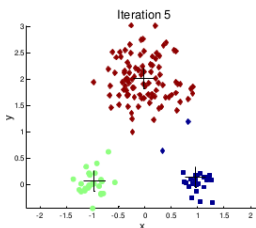
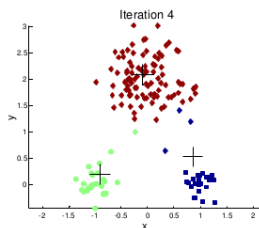
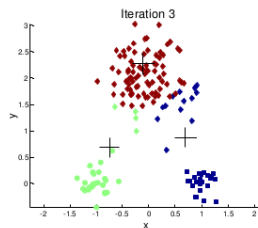
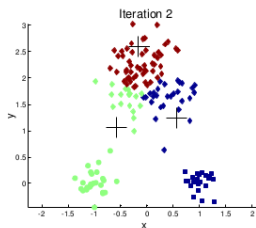
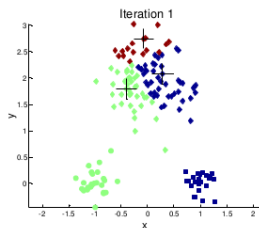


Entrenamiento de una red tipo RBF: fase 1 (clustering)

- El ajuste de los centros de la red puede hacerse mediante un procedimiento de *clustering*.
- La idea es detectar los grupos de patrones (o *clusters*) en el espacio de entrada y poner una RBF en cada *cluster*.
- Para este proceso vamos a utilizar el algoritmo de clustering más popular, el *K-medias*.
- Las coordenadas de los centroides de cada *cluster* serán las coordenadas de los centros de las RBF.



Entrenamiento de una red tipo RBF: fase 1 (clustering)



Entrenamiento de una red tipo RBF: fase 1 (clustering)

- K -medias: algoritmo de agrupamiento por particiones.
 - Hay que decirle el número de *clusters*, en nuestro caso será el número de neuronas de la RBF.
 - Cada *cluster* tiene asociado un centroide (centro geométrico del *cluster*).
 - Los puntos se asignan al *cluster* cuyo centroide esté más cerca (utilizando cualquier métrica de distancia).
 - Iterativamente, se van actualizando los centroides en función de las asignaciones de puntos a *clusters*, hasta que los centroides dejen de cambiar.
 - Los resultados dependen de la **inicialización de los centroides**:
 - En **clasificación**, escogemos aleatoriamente, y de **forma estratificada**, n_1 patrones.
 - En **regresión**, escogemos aleatoriamente n_1 patrones.



Entrenamiento de una red tipo RBF: fase 2 (ajuste de los radios)

- Se pueden utilizar procedimientos más complejos (estimación de densidades) para ajustar bien los radios de las RBF.
- Sin embargo, nosotros ajustaremos los radios de una forma muy simple, tomaremos la mitad (por ser un radio y no un diámetro) de la distancia media al resto de centroides.
- Es decir, el radio de la neurona j será:

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

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



Entrenamiento de una red tipo RBF: fase 3 (pesos de la capa de salida, caso 1 clasificación)

- Los pesos de la capa de salida los ajustaremos de dos formas, dependiendo de si estamos ante un problema de clasificación o de regresión.
 - Si el problema es de **clasificación**, los pesos se ajustarán utilizando **regresión logística**.
 - Si el problema es de **regresión**, los pesos se ajustarán utilizando la **pseudo-inversa**.



Entrenamiento de una red tipo RBF: fase 3 (pesos de la capa de salida, caso 1 clasificación)

- En ambos casos, necesitaremos la matriz de salidas de las RBF, que llamaremos **R**:

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

donde $out_j^1(\mathbf{x}_i)$ es la salida de la j -ésima neurona RBF cuando alimentamos las entradas con el patrón de entrenamiento \mathbf{x}_i . Para simular el sesgo hemos incluido una columna constante e igual a 1.



Entrenamiento de una red tipo RBF: fase 3 (pesos de la capa de salida, caso 1 clasificación)

- Una vez construida la matriz, para el caso de clasificación, aplicaremos **regresión logística**.
 - Utilizaremos una función a la que le pasaremos la matriz **R** como si fuese la matriz de entradas de mi base de datos.
 - La regresión logística es un modelo lineal de clasificación, que aproxima la probabilidad de pertenencia a una clase de la siguiente forma (función *softmax*):

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

- El objetivo de la regresión logística es obtener los valores de β_{ji} que maximizan la entropía cruzada:

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



Entrenamiento de una red tipo RBF: fase 3 (pesos de la capa de salida, caso 1 clasificación)

- ...aplicaremos **regresión logística**.
 - La regresión logística puede incluir regularización, que es un mecanismo para lograr que el máximo número de parámetros β_{ji} tiendan a valores muy pequeños (o casi cero).
 - Regularización L2:

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

- Regularización L1:

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

- El parámetro η lo debe fijar el usuario y establece la importancia que se le da a la regularización.



Entrenamiento de una red tipo RBF: fase 3 (pesos de la capa de salida, caso 1 clasificación)

- ...aplicaremos **regresión logística**.
 - La regularización proporciona modelos más simples y que tienden menos a sobre-entrenar.
 - Diferencia entre L2 y L1:
 - La regularización L2 tiende a proporcionar pesos más pequeños (aunque no necesariamente iguales a cero).
 - La regularización L1 tiende a podar más variables, haciendo que muchos pesos sean iguales a cero (aunque los que no son cero no tienen porque ser pequeños en valor absoluto).



Entrenamiento de una red tipo RBF: fase 3 (pesos de la capa de salida, caso 2 regresión)

- Una vez construida la matriz, para el caso de regresión, aplicaremos la **pseudo-inversa**.
 - Desde el punto de vista del álgebra lineal, la salida de la red se puede escribir como:

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

dónde \mathbf{R} es la matriz que contiene las salidas de las neuronas RBF, $\boldsymbol{\beta}$ es una matriz conteniendo un vector de parámetros por cada salida a predecir e $\hat{\mathbf{Y}}$ es una matriz con todas las salidas estimadas.

$$\begin{pmatrix} out_{11}^1 & \dots & out_{n_1 1}^1 & 1 \\ out_{12}^1 & \dots & out_{n_1 2}^1 & 1 \\ \dots & \dots & \dots & \dots \\ out_{1N}^1 & \dots & out_{n_1 N}^1 & 1 \end{pmatrix} \begin{pmatrix} \beta_{11} & \beta_{21} & \dots & \beta_{k1} \\ \dots & \dots & \dots & \dots \\ \beta_{1n_1} & \beta_{2n_1} & \dots & \beta_{kn_1} \\ \beta_{10} & \beta_{20} & \dots & \beta_{k0} \end{pmatrix} \quad (9)$$



Entrenamiento de una red tipo RBF: fase 3 (pesos de la capa de salida, caso 2 regresión)

- ...aplicaremos la **pseudo-inversa**.
 - Si queremos obtener los mejores valores posibles para los parámetros, planteamos la siguiente ecuación:

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

dónde \mathbf{Y} es la matriz de salidas deseadas, es decir:

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



Entrenamiento de una red tipo RBF: fase 3 (pesos de la capa de salida, caso 2 regresión)

- ...aplicaremos la **pseudo-inversa**.
 - Si \mathbf{R} fuese cuadrada ($N = (n_1 + 1)$), entonces \mathbf{R} tiene inversa y podemos despejar directamente:

$$\beta_{((n_1+1) \times k)}^T = (\mathbf{R}_{(N \times N)})^{-1} \mathbf{Y}_{(N \times k)} \quad (12)$$

- Si $(n_1 + 1) > N$, entonces existen muchas soluciones y hay que usar algún tipo de algoritmo de reducción de características para bajar el valor de n_1 .
- Si $(n_1 + 1) < N$ (el caso más común), existe una solución única, pero como \mathbf{R} no es cuadrada, tenemos que usar la pseudo-inversa de Moore Penrose.



Entrenamiento de una red tipo RBF: fase 3 (pesos de la capa de salida, caso 2 regresión)

- ...aplicaremos la **pseudo-inversa**.
 - Pseudo-inversa de Moore Penrose:

$$\beta_{((n_1+1) \times k)}^T = (\mathbf{R}^+)_{((n_1+1) \times N)} \mathbf{Y}_{(N \times k)} \quad (13)$$

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

- Utilizaremos una librería matricial para hacer estas operaciones y obtener $\beta_{((n_1+1) \times k)}^T$.



Algoritmo de entrenamiento RBF *off-line*

Inicio

- ➊ centroidesIniciales \leftarrow aleatoriamente n_1 patrones (**regresión**) o n_1 patrones de forma estratificada (**clasificación**).
- ➋ centroides \leftarrow K-medias($\mathbf{X}, n_1, \text{centroidesIniciales}$) // \mathbf{X} es el conjunto de entradas para todos los patrones
- ➌ $\sigma_j \leftarrow$ (media de las distancias al resto de centroides)/2.
- ➍ Construir la matriz $\mathbf{R}_{(N \times (n_1 + 1))}$, donde $\mathbf{R}_{ij} = \text{out}_j^1(\mathbf{x}_i)$ para $j \neq (n_1 + 1)$, y $\mathbf{R}_{ij} = 1$ para $j = (n_1 + 1)$.
- ➎ **Si clasificación**
 - ➊ pesosSalida \leftarrow aplicarRegresionLogistica(\mathbf{R}, eta)
- ➏ **Si regresión**
 - ➊ pesosSalida \leftarrow calcularPseudoinversa(\mathbf{R})

Fin



Introducción a los modelos computacionales

Práctica 3. Redes neuronales de funciones de base radial

Pedro Antonio Gutiérrez
pagutierrez@uco.es

Asignatura “Introducción a los modelos computacionales”
4º Curso Grado en Ingeniería Informática
Especialidad Computación
Escuela Politécnica Superior
(Universidad de Córdoba)

3 de noviembre de 2020

