Práctica 3: Redes neuronales de funciones de base radial

Convocatoria de febrero (curso académico 2016/2017)

Asignatura: Introducción a los modelos computacionales
4º Grado Ingeniería Informática (Universidad de Córdoba)

11 de noviembre de 2016

Resumen

Esta práctica sirve para familiarizar al alumno con el concepto de red neuronal de funciones de base radial (RBF). De esta forma, desarrollaremos un código que entrene una red de este tipo, utilizando Python y la librería de aprendizaje automático `scikit-learn`[1]. Al mismo tiempo, la práctica servirá para familiarizarse con librerías externas, que tan a menudo son necesarias en entornos de aprendizaje automático. El alumno deberá programar el algoritmo y comprobar el efecto de distintos parámetros sobre un conjunto de bases de datos reales. La entrega se hará utilizando la tarea en Moodle habilitada al efecto. Se deberá subir en un único fichero comprimido todos los entregables indicados en este guión. El día tope para la entrega es el **1 de diciembre de 2016**. En caso de que dos alumnos entreguen prácticas copiadas, no se puntuarán ninguna de las dos.

1. Introducción

El trabajo a realizar en la práctica consiste en implementar una red neuronal de tipo RBF realizando un entrenamiento en tres etapas:

1. Aplicación de un algoritmo de *clustering* que servirá para establecer los centros de las funciones RBF (pesos de capa de entrada a capa oculta).
2. Ajuste de los radios de las RBF, mediante una heurística simple (media de las distancias hacia el resto de centros).
3. Aprendizaje de los pesos de capa oculta a capa de salida.
   - Para problemas de regresión, utilización de la pseudo-inversa de Moore Penrose.
   - Para problemas de clasificación, utilización de un modelo lineal de regresión logística.

El alumno deberá desarrollar un *script* de Python capaz de realizar el entrenamiento de una red RBF con las características anteriormente mencionadas. Este *script* se utilizará para entrenar modelos que predigan de la forma más correcta posible un conjunto de bases de datos disponible en Moodle y se realizará un análisis de los resultados obtenidos. **Este análisis influirá en gran medida en la calificación de la práctica**.

En el enunciado de esta práctica, se proporcionan valores orientativos para todos los parámetros del algoritmo. Sin embargo, se valorará positivamente si el alumno encuentra otros valores para estos parámetros que le ayuden a mejorar los resultados obtenidos.

La sección 2 describe una serie de pautas generales a la hora de implementar el algoritmo de entrenamiento de redes neuronales de tipo RBF. La sección 3 explica los experimentos a realizar

[1] `http://scikit-learn.org/`

una vez implementado el algoritmo. Finalmente, la sección 4 especifica los ficheros a entregar para esta práctica.

2. Implementación del algoritmo de entrenamiento de redes RBF

2.1. Arquitectura de los modelos a considerar

Los modelos de redes neuronales RBF que vamos a considerar tienen la siguiente arquitectura:

- Una capa de entrada con tantas neuronas como variables tenga la base de datos considerada.
- Una capa oculta con un número de neuronas a especificar por el usuario del *script* a desarrollar. Es importante recalcar que, en las dos prácticas anteriores, el número de capas ocultas era variable. En esta práctica **siempre tendremos una sola capa oculta**. Todas las neuronas de la capa oculta serán de tipo RBF (en contraposición a las neuronas de tipo sigmoide, utilizadas en prácticas anteriores).
- Una capa de salida con tantas neuronas como variables de salida tenga la base de datos considerada:
  - Si la base de datos es de regresión, todas las neuronas de la capa de salida serán de tipo lineal (igual que las neuronas de tipo sigmoide, pero sin aplicar la transformación $1/(1+\exp(-x))$).
  - Si la base de datos es de clasificación, todas las neuronas de la capa de salida serán de tipo *softmax*.

2.2. Ajuste de los pesos

Se deben de seguir las indicaciones aportadas en las diapositivas de clase para que el entrenamiento se realice de la siguiente forma

1. Aplicación de un algoritmo de *clustering* que servirá para establecer los centros de las funciones RBF (pesos de capa de entrada a capa oculta). Para problemas de clasificación, la inicialización de los centroides se realizará seleccionando aleatoriamente n_1/k patrones de cada una de las clases, donde n_1 es el número de *clusters* y k el número de clases[2]. Para problemas de regresión, seleccionaremos aleatoriamente n_1 patrones. Después de inicializar los centroides, para realizar el *clustering*, utilizaremos la clase `sklearn.cluster.KMeans`, con una sola inicialización de los centroides (`n_init`) y un máximo de 500 iteraciones (`max_iter`).

2. Ajuste de los radios de las RBF, mediante una heurística simple (la mitad de la media de las distancias hacia el resto de centros). Es decir, el radio de la neurona j será[3]:

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

3. Aprendizaje de los pesos de capa oculta a capa de salida.

[2]Para esta labor, puedes consultar la clase `sklearn.cross_validation.StratifiedShuffleSplit`, que realiza una o varias particiones de una base de datos de forma "estratificada", es decir, manteniendo la proporción de patrones de cada clase en la base de datos original
`http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedShuffleSplit.html`

[3]Considera el uso conjunto de las funciones `pdist` y `squareform` para obtener la matriz de distancias)

- Para problemas de regresión, utilización de la pseudo-inversa de Moore Penrose. Es decir:

$$\boldsymbol{\beta}^{\mathrm{T}}_{((n_1+1)\times k)} = (\mathbf{R}^{+})_{((n_1+1)\times N)}\,\mathbf{Y}_{(N\times k)} = \quad (2)$$

$$= \left(\mathbf{R}^{\mathrm{T}}_{((n_1+1)\times N)} \times \mathbf{R}_{(N\times(n_1+1))}\right)^{-1}\mathbf{R}^{\mathrm{T}}_{((n_1+1)\times N)}\mathbf{Y}_{(N\times k)} \quad (3)$$

  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 $\mathbf{Y}$ es una matriz con todas las salidas deseadas. Para realizar estas operaciones, utilizaremos las funciones matriciales de `numpy`, que es una de las dependencias de `scikit-learn`.

- Para problemas de clasificación, utilización de un modelo lineal de regresión logística. Haremos uso de la clase `sklearn.linear_model.LogisticRegression`, aportando un valor para el parámetro C que aplica regularización. Es necesario indicar que en esta librería lo que especificamos es el valor de coste C (importancia del error de aproximación frente al error de regularización), de forma que $\eta = \frac{1}{C}$. Utilizaremos la expresión de regularización de tipo L2[4] y el algoritmo de optimización Liblinear.

3. Experimentos a realizar

Probaremos distintas configuraciones de la red neuronal y ejecutaremos cada configuración con cinco semillas (10, 20, 30, 40 y 50). A partir de los resultados obtenidos, se obtendrá la media y la desviación típica del error. Para problemas de regresión mostraremos el error de tipo MSE. Para problemas de clasificación, mostraremos el error de tipo MSE[5]y, además, el *script* deberá mostrar el porcentaje de patrones bien clasificados o CCR.

Para valorar cómo funciona el algoritmo implementado en esta práctica, emplearemos tres bases de datos de regresión:

- *Función seno*: esta base de datos está compuesta por 120 patrones de entrenamiento y 41 patrones de *test*. Ha sido obtenido añadiendo cierto ruido aleatorio a la función seno (ver Figura 1).

- *Base de datos forest*: esta base de datos está compuesta por 387 patrones de entrenamiento y 130 patrones de *test*. Contiene, como entradas o variables independientes, una serie de datos meteorológicos y otras variables descriptoras sobre incendios en bosques al norte de Portugal, y, como salida o variable dependiente, el área quemada[6].

- *Base de datos CPU*: esta base de datos está compuesta por 109 patrones de entrenamiento y 100 patrones de *test*. Contiene como entradas o variables independientes las características de un conjunto de procesadores, y como salida o variable dependiente el rendimiento relativo del procesador[7].

Y dos bases de datos de clasificación:

- *Base de datos iris*: *iris* contiene 112 patrones de entrenamiento y 38 patrones de test. Las clases son tres especies distintas de la flor *iris*, de manera que para cada flor se extraen cuatro medidas o variables de entrada (longitud y ancho de los pétalos y los sépalos, en cm). Las tres especies a distinguir son *iris setosa*, *iris virginica* e *iris versicolor*.

[4] `https://msdn.microsoft.com/en-us/magazine/dn904675.aspx`

[5] En clasificación, el MSE debe obtenerse como se hizo en la práctica 2, es decir, convirtiendo las etiquetas de clase a valores binarios y comparándolos con las probabilidades predichas, que pueden obtenerse con el método `predict_proba`

[6] Para más información, consultar `https://archive.ics.uci.edu/ml/datasets/Forest+Fires`

[7] Para más información, consultar `http://archive.ics.uci.edu/ml/datasets/Computer+Hardware`

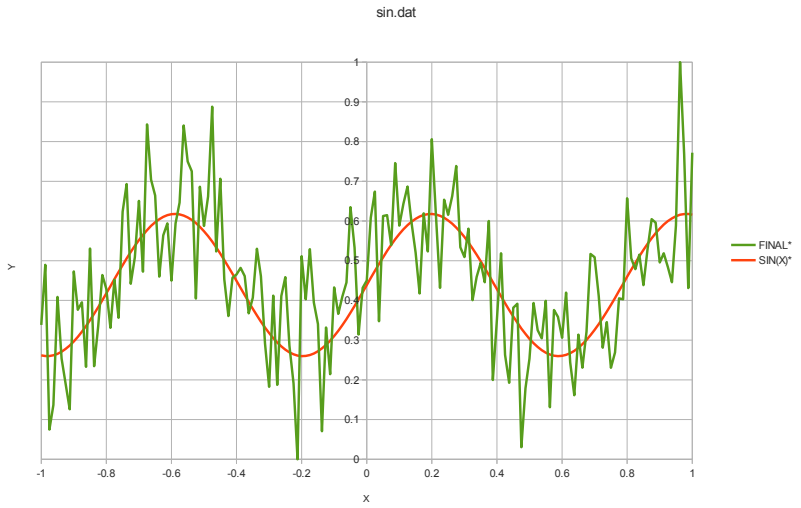

Figura 1: Representación de los datos incluidos para el problema de estimación de la función seno.

- *Base de datos digits*: esta base de datos está compuesta por 1274 patrones de entrenamiento y 319 patrones de *test*. Está formada por un conjunto de dígitos (del 0 al 9) escritos a mano por 80 personas distintas, y ajustados a una rejilla cuadrada de 16×16 píxeles. Aunque las imágenes originales estaban en escala de grises, éstas fueron binarizadas, con un valor de umbral fijo[8]. Cada uno de los píxeles forman parte de las variables de entrada (con un total de $16 \times 16 = 256$ variables de entrada) y las clases se corresponden con el dígito escrito (0, 1, $\ldots$, 9, con un total de 10 clases). La figura 2 presenta los 1274 patrones del conjunto de entrenamiento, mientras que los dígitos del conjunto de test están en la figura 3. Además, todos los dígitos está colgados en la plataforma Moodle en los ficheros `train_digits.tar.gz` y `test_digits.tar.gz`.

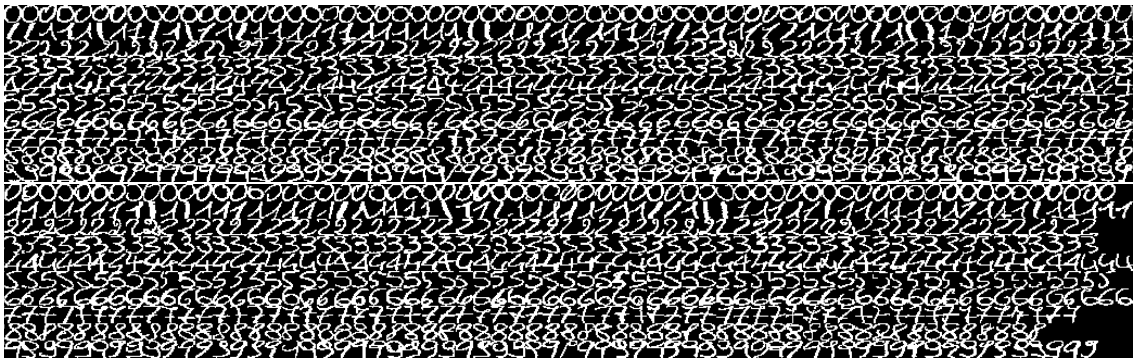

Figura 2: Dígitos del conjunto de entrenamiento.

Se deberá extraer la media y desviación típica de dos medidas (en regresión) o cuatro medidas (en clasificación):

- Regresión: media y desviación típica del MSE de entrenamiento y de *test*.

[8]Para más información, consultar `http://archive.ics.uci.edu/ml/datasets/Semeion+Handwritten+Digit`

Figura 3: Dígitos del conjunto de test.

- Clasificación: media y desviación típica del CCR de entrenamiento y de *test* y media y desviación típica del MSE de entrenamiento y de *test*.

Se deben probar, al menos, las siguientes configuraciones:

- *Arquitectura de la red*:
  - Para todas las bases de datos, considerar un número de neuronas en capa oculta (n_1) igual al $5\,\%$, $10\,\%$, $25\,\%$ y $50\,\%$ del número de patrones de la base de datos. En esta fase, para problemas de clasificación, utilizar regularización L1 y un valor para el parámetro $\eta = 10^{-5}$.
- Para los problemas de clasificación, una vez decidida la mejor arquitectura, probar los siguientes valores para η: $\eta = 0$, $\eta = 0{,}1$, $\eta = 0{,}01$, $\eta = 0{,}001$, ..., $\eta = 10^{-10}$, junto con los dos tipos de regularización (L2 y L1). ¿Qué sucede?. Calcula la diferencia en número de coeficientes en iris y digits cuando modificas el tipo de regularización (L2 Vs L1).
- Finalmente, en alguno de los problemas de clasificación, probar a lanzar el *script* considerando el problema como si fuera un problema de regresión (es decir, incluyendo un `False` en el parámetro `clasificacion` y calculando el CCR redondeando las predicciones hasta el entero más cercano). ¿Qué sucede en este caso?.

Como valor orientativo, se muestra a continuación el error de entrenamiento y de generalización obtenido por una regresión lineal utilizando Weka en las tres bases de datos:

- *Función seno*: $MSE_{\text{entrenamiento}} = 0{,}02968729$; $MSE_{\text{test}} = 0{,}03636649$.
- *Base de datos CPU*: $MSE_{\text{entrenamiento}} = 0{,}00145161$; $MSE_{\text{test}} = 0{,}00438244$.

y el CCR de entrenamiento y de generalización obtenido por una regresión logística lineal utilizando Weka en las dos bases de datos de clasificación:

- *Base de datos iris*: $CCR_{\text{entrenamiento}} = 83{,}0357\,\%$; $CCR_{\text{test}} = 92{,}1053\,\%$.
- *Base de datos digits*: $CCR_{\text{entrenamiento}} = 94{,}3485\,\%$; $CCR_{\text{test}} = 73{,}0408\,\%$.
- *Base de datos Forest*: $MSE_{\text{entrenamiento}} = 0{,}00154449$; $MSE_{\text{test}} = 0{,}00891136$.

El alumno debería ser capaz de superar estos valores con algunas de las configuraciones y semillas.

3.1. Formato de los ficheros

Los ficheros que contienen las bases de datos tendrán formato CSV, de forma que los valores vendrán separados por comas. En este caso, no tendremos cabeceras. Para realizar la lectura de los ficheros, utilizaremos la función `read_csv` de la librería `pandas`.

4. Entregables

Los ficheros a entregar serán los siguientes:

- Memoria de la práctica en un fichero `pdf` que describa el *script* generado, incluya las tablas de resultados y analice estos resultados.
- *Script* de Python correspondiente a la práctica.

4.1. Memoria de la práctica

La memoria de la práctica deberá incluir, al menos, el siguiente contenido:

- Portada con el número de práctica, título de la práctica, asignatura, titulación, escuela, universidad, curso académico, nombre, DNI y correo electrónico del alumno.
- Índice del contenido de la memoria con numeración de las páginas.
- Descripción de los pasos a realizar para llevar a cabo el entrenamiento de las redes RBF (**máximo 1 carilla**).
- Experimentos y análisis de resultados:
  - Breve descripción de las bases de datos utilizadas.
  - Breve descripción de los valores de los parámetros considerados.
  - Resultados obtenidos, según el formato especificado en la sección anterior.
  - Análisis de resultados. El análisis deberá estar orientado a justificar los resultados obtenidos, en lugar de realizar un análisis meramente descriptivo de las tablas. Tener en cuenta que esta parte es decisiva en la nota de la práctica. Se valorará la inclusión de los siguientes elementos de comparación:
    - Matriz de confusión en test del mejor modelo de red neuronal obtenido para la base de datos *digits*. Analizar los errores cometidos, incluyendo las imágenes de aquellos dígitos en los que el modelo de red se equivoca, para comprobar si son confusos. Comparación de esta matriz con la matriz obtenida para el perceptrón multicapa en la práctica anterior.
    - Tiempo computacional necesario para entrenar la base de datos `digits` y comparativa con el tiempo necesario para la práctica anterior.
    - Incluir la ecuación del mejor modelo de red neuronal obtenido para la base de datos iris. Deberás incluir la fórmula correspondiente a cómo se estima la probabilidad de pertenencia a cada una de las clases.
- Referencias bibliográficas u otro tipo de material distinto del proporcionado en la asignatura que se haya consultado para realizar la práctica (en caso de haberlo hecho).

Aunque lo importante es el contenido, se valorará también la presentación, incluyendo formato, estilo y estructuración del documento. La presencia de demasiadas faltas ortográficas puede disminuir la nota obtenida.

4.2. Código fuente

Junto con la memoria, se deberá incluir el *script* de Python preparado para funcionar en las máquinas de la UCO (en concreto, probar por `ssh` en `ts.uco.es`). Un ejemplo de ejecución de dicho *script* puede verse en la siguiente salida [9]:

```
i02gupep@NEWTS:~/imc/practica3$ cat rbf.py | grep 'csv' | grep -v 'df'
            entrenar_rbf(fichero_train='./csv/train_iris.csv',
                         fichero_test='./csv/test_iris.csv', num_rbf=10, clasificacion=
                             True, eta=10e-3, l2=True)

i02gupep@NEWTS:~/imc/tmp$ ./rbf.py
-----------
Semilla: 10
-----------
MSE de entrenamiento: 0.013421
MSE de test: 0.014756
CCR de entrenamiento: 96.43%
CCR de test: 97.37%
-----------
Semilla: 20
-----------
MSE de entrenamiento: 0.013316
MSE de test: 0.008545
CCR de entrenamiento: 97.32%
CCR de test: 97.37%
-----------
Semilla: 30
-----------
MSE de entrenamiento: 0.013238
MSE de test: 0.009993
CCR de entrenamiento: 96.43%
CCR de test: 97.37%
-----------
Semilla: 40
-----------
MSE de entrenamiento: 0.012379
MSE de test: 0.016311
CCR de entrenamiento: 96.43%
CCR de test: 94.74%
-----------
Semilla: 50
-----------
MSE de entrenamiento: 0.012143
MSE de test: 0.009002
CCR de entrenamiento: 96.43%
CCR de test: 100.00%
**********
Resumen de resultados
**********
MSE de entrenamiento: 0.012899 +- 0.000530
MSE de test: 0.011721 +- 0.003186
CCR de entrenamiento: 96.61% +- 0.36%
CCR de test: 97.37% +- 1.66%


i02gupep@NEWTS:~/imc/practica3$ cat rbf.py | grep 'csv' | grep -v 'df'
            entrenar_rbf(fichero_train='./csv/train_cpu.csv',
                         fichero_test='./csv/test_cpu.csv', num_rbf=20, clasificacion=
                             False, eta=10e-3, l2=True)


i02gupep@NEWTS:~/imc/practica3$ ./rbf.py
-----------
```

[9] El primer `cat` y `grep` solo sirve para ver como está configurado el algoritmo en cada ejecución. El cambio de parámetros debe hacerse sobre el mismo fichero.

```
Semilla: 10
-----------
MSE de entrenamiento: 0.000467
MSE de test: 0.002108
-----------
Semilla: 20
-----------
MSE de entrenamiento: 0.001810
MSE de test: 0.004512
-----------
Semilla: 30
-----------
MSE de entrenamiento: 0.000599
MSE de test: 0.002363
-----------
Semilla: 40
-----------
MSE de entrenamiento: 0.003718
MSE de test: 0.004678
-----------
Semilla: 50
-----------
MSE de entrenamiento: 0.003692
MSE de test: 0.004393
**********
Resumen de resultados
**********
MSE de entrenamiento: 0.002057 +- 0.001425
MSE de test: 0.003611 +- 0.001129
```