Práctica 2: Perceptrón multicapa para problemas de clasificación

Convocatoria de enero (curso académico 2023/2024)

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

18 de octubre de 2023

Resumen

Esta práctica sirve para familiarizar al alumno con la adaptación de un modelo computacional de red neuronal a problemas de clasificación, en concreto, con la adaptación del perceptrón multicapa programado en la práctica anterior. Por otro lado, también se implementará la versión *off-line* del algoritmo de entrenamiento. El alumno deberá programar estas modificaciones y comprobar el efecto de distintos parámetros sobre una serie de bases de datos reales, con el objetivo de obtener los mejores resultados posibles en clasificación. 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 **5 de noviembre de 2023**. En caso de que dos alumnos entreguen prácticas copiadas, no se puntuarán ninguna de las dos.

1. Introducción

El trabajo que se debe a realizar en la práctica consiste en adaptar el algoritmo de retropropagación realizado en la práctica anterior a problemas de clasificación. En concreto, se dotará un significado probabilístico a dicho algoritmo mediante dos elementos:

- Utilización de la función de activación *softmax* en la capa de salida.
- Utilización de la función de error basada en entropía cruzada.

Además, se implementará la versión *off-line* del algoritmo.

El alumno deberá desarrollar un programa capaz de realizar el entrenamiento con las modificaciones anteriormente mencionadas. Este programa se utilizará para entrenar modelos que clasifiquen 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 única condición es que no se puede modificar el número máximo de iteraciones del bucle externo (establecido a 1000 iteraciones para los problemas XOR y *ProPublica COMPAS*, y 500 iteraciones para la base de datos *noMNIST*).

La sección 2 describe una serie de pautas generales a la hora de implementar las modificaciones del algoritmo de retropropagación. La sección 3 explica los experimentos que se deben a realizar una vez implementadas dichas modificaciones. Finalmente, la sección 4 especifica los ficheros que se tienen que entregar para esta práctica.

2. Implementación del algoritmo de retropropagación

Se deben de seguir las indicaciones aportadas en las diapositivas de clase para añadir las siguientes características al algoritmo implementado en la práctica anterior:

1. *Incorporación de la función softmax*: Se incorporará la posibilidad de que las neuronas de capa de salida sean de tipo *softmax*, quedando su salida definida de la siguiente forma:

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

$$out_j^H = o_j = \frac{\exp(net_j^H)}{\sum_{l=1}^{n_H} \exp(net_l^H)}. \tag{2}$$

   Debes implementar el modelo optimizado en cuanto al número de pesos, de forma que, para la última salida (salida n_H), se considerará que $net_{n_H}^H = 0$. Esto nos permitirá reducir el número de pesos, de forma que podremos descartar los pesos $w_{n_H 0}^H, w_{n_H 1}^H, \ldots, w_{n_H n_{H-1}}^H$. Aunque se podría reservar una neurona menos en la capa de salida, se recomienda, por comodidad, establecer a `NULL` todos los vectores asociados a dicha neurona, utilizar $net_{n_H}^H = 0$ cuando nos encontremos `NULL` en la propagación hacia delante (solo para la última capa y cuando usemos *softmax*) e ignorar la neurona en el resto de métodos.

2. *Utilización de la función de error basada en entropía cruzada*: Se dará la posibilidad de utilizar la entropía cruzada como función de error, es decir:

$$L = -\frac{1}{N} \sum_{p=1}^{N} \left(\frac{1}{k} \sum_{o=1}^{k} d_{po} \ln(o_{po}) \right), \tag{3}$$

   donde N es el número de patrones de la base de datos considerada, k es el número de salidas, d_{po} es 1 si el patrón p pertenece a la clase o (y 0 en caso contrario) y o_{po} es el valor de probabilidad obtenido por el modelo para el patrón p y la clase o.

3. *Modo de funcionamiento*: Además de trabajar en modo *on-line* (práctica anterior), el algoritmo podrá trabajar en modo *off-line* o *batch*. Esto es, por cada patrón de entrenamiento (bucle interno), calcularemos el error y acumularemos el cambio, pero no ajustaremos los pesos de la red. Una vez procesados todos los patrones de entrenamiento (y acumulados todos los cambios), entonces ajustaremos los pesos, comprobaremos la condición de parada del bucle externo y volveremos a empezar por el primer patrón, si la condición no se cumple. Recuerda promediar las derivadas durante el ajuste de pesos para el modo *off-line*, tal y como se explica en la presentación de este práctica.

4. El resto de características del algoritmo (utilización de ficheros de *entrenamiento* y un fichero de *test*, *condición de parada*, *copias de los pesos* y *semillas para los números aleatorios*) se mantendrán similares a como se implementaron en la práctica anterior. Sin embargo, para esta práctica, se recomienda tomar como valores por defecto para la tasa de aprendizaje y el factor de momento los siguientes $\eta = 0{,}7$ y $\mu = 1$, ajustándolos si fuera necesario hasta obtener una convergencia.

Se recomienda implementar los puntos anteriores en orden, comprobando que todo funciona (con al menos dos bases de datos) antes de pasar al siguiente punto.

3. Experimentos

Probaremos distintas configuraciones de la red neuronal y ejecutaremos cada configuración con cinco semillas (1, 2, 3, 4 y 5). A partir de los resultados obtenidos, se obtendrá la media y

la desviación típica del error. Aunque el entrenamiento va a guiarse utilizando la función de entropía cruzada o el MSE, el programa deberá mostrar siempre el porcentaje de patrones bien clasificados, ya que en problemas de clasificación, es esta medida de rendimiento la que más nos interesa[1]. El porcentaje de patrones bien clasificados se puede expresar de la siguiente forma:

$$CCR = 100 \times \frac{1}{N} \sum_{p=1}^{N} \left(I(y_p = y_p^*) \right), \tag{4}$$

donde N es el número de patrones de la base de datos considerada, y_p es la clase deseada para el patrón p (es decir, el índice del valor máximo del vector $\mathbf{d}_p$, $y_p = \arg\max_o d_{po}$ o lo que es lo mismo, el índice de la posición en la que aparece un 1) e y_p^* es la clase obtenida para el patrón p (es decir, el índice del valor máximo del vector $\mathbf{o}_p$ o la neurona de salida que obtiene la máxima probabilidad para el patrón p, $y_p^* = \arg\max_o o_{po}$).

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

- *Problema XOR*: esta base de datos representa el problema de clasificación no lineal del XOR. Se utilizará el mismo fichero para entrenamiento y para $test$. Como puede verse, se ha adaptado dicho fichero a la codificación 1-de-k, encontrándonos en este caso con dos salidas en lugar de una.

- *ProPublica COMPAS*: Este conjunto de datos trata sobre el rendimiento del algoritmo COMPAS, un método estadístico para asignar puntajes de riesgo en el sistema de justicia penal de los Estados Unidos creado por Northpointe. Fue publicado por ProPublica en 2016 [2], afirmando que esta herramienta de riesgo estaba sesgada contra individuos afroamericanos (trataremos esto en la siguiente práctica). En este conjunto de datos, se analizó la puntuación de COMPAS para "riesgo de reincidencia", por lo que cada individuo tiene un resultado binario de "reincidencia", que es la tarea de predicción, indicando si fueron arrestados nuevamente dentro de dos años después del primer arresto (el cargo descrito en los datos). Hemos reducido el conjunto de datos original de 52 a 9 atributos de manera similar al conjunto de datos original: sex, age, age_cat, race, juv_fel_count, juv_misd_count, juv_other_count, priors_count, c_charge_degree. La variable de predicción es si el individuo será arrestado nuevamente en dos años o no.

  1. sex: sexo binario.
  2. age: edad numérica.
  3. age_cat: categoría de edad (edad $<$ 25, edad $\geq$ 25 y edad $<$ 45, edad $\geq$ 45).
  4. race: atributo binario (0 significa 'blanca' y 1 significa 'negra').
  5. juv_fel_count: una variable continua que contiene el número de delitos graves juveniles.
  6. juv_misd_count: una variable continua que contiene el número de delitos menores juveniles.
  7. juv_other_count: una variable continua que contiene el número de condenas juveniles previas que no se consideran delitos graves ni delitos menores.
  8. priors_count: una variable continua que contiene el número de delitos previos cometidos.
  9. c_charge_degree: grado del delito, que puede ser M (delito menor), F (delito grave) u O (no conlleva tiempo en la cárcel). Esta variable ya está preprocesada.

[1] Lástima que no sea derivable y no la podamos usar para ajustar pesos

[2] `https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm`

- *Base de datos noMNIST*: originalmente, esta base de datos estaba compuesta por $200{,}000$ patrones de entrenamiento y $10{,}000$ patrones de *test*, con un total de 10 clases. No obstante, para la práctica que nos ocupa, se ha reducido considerablemente el tamaño de la base de datos con objeto de realizar las pruebas en menor tiempo. Por lo tanto, la base de datos que se utilizará está compuesta por 900 patrones de entrenamiento y 300 patrones de *test*. Está formada por un conjunto de letras (de la a a la f) escritas con diferentes tipografías o simbologías. Están ajustadas a una rejilla cuadrada de 28×28 píxeles. Las imágenes están en escala de grises en el intervalo $[-1{,}0; +1{,}0]$[3]. Cada uno de los píxeles es una variable de entrada (con un total de $28 \times 28 = 784$ variables de entrada) y las clases se corresponden con la letra escrita (a, b, c, d, e y f, con un total de 6 clases). La figura 1 representa un subconjunto de los 180 patrones del conjunto de entrenamiento, mientras que la figura 2 representa un subconjunto de 180 letras del conjunto de *test*. Además, todas las letras, ordenadas dentro de cada conjunto, está colgadas en la plataforma Moodle en los ficheros `train_img_nomnist.tar.gz` y `test_img_nomnist.tar.gz`.

Figura 1: Subconjunto de letras del conjunto de entrenamiento.

Figura 2: subconjunto de letras del conjunto de *test*.

Todas las bases de datos están normalizadas (**Las salidas nunca se normalizan en clasificación.**).

Se deberá construir **una tabla para cada base de datos**, en la que se compare la media y desviación típica de cuatro medidas:

- Error de entrenamiento y de *test*. Se utilizará la función que el usuario haya elegido para ajustar los pesos (MSE o entropía cruzada).
- CCR de entrenamiento y de *test*.

Se deberá utilizar siempre factor de momento. Se recomienda utilizar los valores de $\eta = 0{,}7$ y $\mu = 1$, ajustándolos si fuera necesario hasta obtener una convergencia. Se deben probar, al menos, las siguientes configuraciones:

[3]Para más información, consultar `http://yaroslavvb.blogspot.com.es/2011/09/notmnist-dataset.html`

- *Arquitectura de la red*: Para esta primera prueba, utiliza la función de error entropía cruzada y la función de activación *softmax* en la capa de salida, con el algoritmo configurado como *off-line*.
    - Para el problema XOR utilizar la arquitectura que resultó mejor en la práctica anterior.
    - Para los problemas *compas* y *noMNIST*, se deberán probar 8 arquitecturas (una o dos capas ocultas con 4, 8, 16 o 64 neuronas).
- Una vez decidida la mejor arquitectura, probar las siguientes combinaciones (con algoritmo *off-line*):
    - Función de error MSE y función de activación *sigmoide* en la capa de salida.
    - Función de error MSE y función de activación *softmax* en la capa de salida.
    - Función de error entropía cruzada y función de activación *softmax* en la capa de salida.
    - No probar la combinación de entropía cruzada y función de activación *sigmoide*, ya que llevará a malos resultados (explicar por qué).
- Una vez decidida la mejor combinación de las anteriores, comparar los resultados con el algoritmo *on-line* frente a los obtenidos con el algoritmo *off-line*.

Ojo: Dependiendo de la función de error, puede ser necesario adaptar los valores de la tasa de aprendizaje (η) y del factor de momento (μ).

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

- *Problema XOR*: $CCR_{\text{entrenamiento}} = CCR_{\text{test}} = 50\,\%$.
- *ProPublica dataset*: $CCR_{\text{entrenamiento}} = 67{,}7348\,\%; CCR_{\text{test}} = 66{,}8514\,\%$.
- *Base de datos noMNIST*: $CCR_{\text{entrenamiento}} = 80{,}4444\,\%; CCR_{\text{test}} = 82{,}6667\,\%$.

El alumno debería ser capaz de superar estos porcentajes con alguna de las configuraciones y semillas.

3.1. Formato de los ficheros

Los ficheros que contienen las bases de datos tendrán el mismo formato que en la práctica anterior. Tan solo observar que en este caso todos los ficheros tienen múltiples salidas (una salida por clase).

4. Entregables

Los ficheros a entregar serán los siguientes:

- Memoria de la práctica en un fichero `pdf` que describa el programa generado, incluya las tablas de resultados y analice estos resultados.
- Fichero ejecutable de la práctica y código fuente.

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 las adaptaciones aplicadas a los modelos de red utilizados para tener en cuenta problemas de clasificación (**máximo 1 carilla**).
- Descripción en pseudocódigo de aquellas funciones que haya sido necesario modificar respecto a las implementaciones de la práctica anterior (**máximo 3 carillas**).
- 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 *noMNIST*.
    - De nuevo para *noMNIST*, analizar los errores cometidos, **incluyendo las imágenes de algunas de las letras en las que el modelo de red se equivoca**, para comprobar visualmente si son confusos.
    - Gráficas de convergencia: reflejan, en el eje x, el número de iteración del algoritmo y, en el eje y, el valor del CCR de entrenamiento y/o el valor del CCR de *test*.
- 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. Ejecutable y código fuente

Junto con la memoria, se deberá incluir el fichero ejecutable preparado para funcionar en las máquinas de la UCO (en concreto, probar por `ssh` en `ts.uco.es`). Además se incluirá todo el código fuente necesario. El fichero ejecutable deberá tener las siguientes características:

- Su nombre será `la2`.
- El programa que se debe desarrollar recibe doce argumentos por la línea de comandos (que pueden aparecer en cualquier orden)[4]. Los nueve primeros no han cambiado respecto a la práctica anterior. Los tres últimos incorporan las modificaciones realizadas en este práctica:
  - Argumento `t`: Indica el nombre del fichero que contiene los datos de entrenamiento a utilizar. Sin este argumento, el programa no puede funcionar.
  - Argumento `T`: Indica el nombre del fichero que contiene los datos de $test$ a utilizar. Si no se especifica este argumento, utilizar los datos de entrenamiento como $test$.

[4]Para procesar la secuencia de entrada, se recomienda utilizar la función `getopt()` de la librería `libc`

- Argumento `i`: Indica el número de iteraciones del bucle externo a realizar. Si no se especifica, utilizar 1000 iteraciones.
- Argumento `l`: Indica el número de capas ocultas del modelo de red neuronal. Si no se especifica, utilizar 1 capa oculta.
- Argumento `h`: Indica el número de neuronas a introducir en cada una de las capas ocultas. Si no se especifica, utilizar 4 neuronas.
- Argumento `e`: Indica el valor del parámetro eta (η). Por defecto, utilizar $\eta = 0{,}7$.
- Argumento `m`: Indica el valor del parámetro mu (μ). Por defecto, utilizar $\mu = 1$.
- Argumento `o`: Booleano que indica si se va a utilizar la versión *on-line*. Si no se especifica, utilizaremos la versión *off-line*.
- Argumento `f`: Indica la función de error que se va a utilizar durante el aprendizaje (0 para el error MSE y 1 para la entropía cruzada). Por defecto, utilizar el error MSE.
- Argumento `s`: Booleano que indica si vamos utilizar la función *softmax* en la capa de salida. Si no se especifica, utilizaremos la función sigmoide.
- Argumento `n`: Booleano que indica que los datos (entrenamiento y test) van a normalizarse tras su lectura. Se normalizará en el intervalo $[-1, 1]$ las entradas (las salidas no se normalizan en clasificación).

- Opcionalmente, se puede añadir otro argumento para guardar la configuración del modelo entrenado (será necesario para hacer predicciones para la competición de Kaggle):
  - Argumento `w`: Indica el nombre del fichero en el que se almacenarán la configuración y el valor de los pesos del modelo entrenado.
- Un ejemplo de ejecución se puede ver en la siguiente salida:

```
1
2  i02gupep@NEWTS:~/imc/workspace/la2/Debug$ ./la2 -t ../train_xor.dat -T ../test_xor.
       dat -i 1000 -l 1 -h 16 -e 0.7 -m 1 -f 1 -s
3  **********
4  SEED 1
5  **********
6  Iteration 1        Training error: 0.366974
7  Iteration 2        Training error: 0.394686
8  Iteration 3        Training error: 0.357086
9  Iteration 4        Training error: 0.366077
10 Iteration 5        Training error: 0.349135
11 Iteration 6        Training error: 0.351968
12 Iteration 7        Training error: 0.343165
13 Iteration 8        Training error: 0.343468
14 Iteration 9        Training error: 0.338281
15 Iteration 10       Training error: 0.337465
16
17 ....
18
19 Iteration 996      Training error: 0.00102801
20 Iteration 997      Training error: 0.00102678
21 Iteration 998      Training error: 0.00102554
22 Iteration 999      Training error: 0.00102431
23 Iteration 1000     Training error: 0.00102309
24 NETWORK WEIGHTS
25 ===============
26 Layer 1
27 ------
28 1.932040 -1.737134 1.926088
29 0.326163 -0.146441 -0.182055
30 -0.490515 0.503036 -0.586897
31 -0.163416 0.775937 0.142567
32 -1.648976 1.859996 1.620257
33 1.865697 1.598651 1.845814
```

```
-2.915683 2.762668 -2.906463
-2.120769 -2.339651 2.162988
-2.523937 -2.326467 -2.534853
-1.837088 2.108107 1.788306
-0.238769 1.013190 0.253558
-1.826408 2.054005 1.775930
2.317019 2.528668 -2.357329
2.145112 1.922300 2.156072
-2.327206 2.158176 -2.320141
0.708153 0.410257 1.074749
Layer 2
------
-2.145596 0.658908 0.172077 -0.855432 -2.428730 2.420198 4.861758 3.750326
    -3.734557 -3.039142 -0.680678 -2.974963 -4.075474 3.229442 3.288709 1.130524
    0.243718
Desired output Vs Obtained output (test)
=========================================
1 -- 0.997629 0 -- 0.00237091
0 -- 0.00168976 1 -- 0.99831
1 -- 0.99817 0 -- 0.00183032
0 -- 0.00228516 1 -- 0.997715
We end!! => Final test CCR: 100
**********
SEED 2
**********
Iteration 1        Training error: 0.373712
Iteration 2        Training error: 0.402586
Iteration 3        Training error: 0.368335
Iteration 4        Training error: 0.369898
Iteration 5        Training error: 0.361395
Iteration 6        Training error: 0.358421
Iteration 7        Training error: 0.356269
Iteration 8        Training error: 0.353413
Iteration 9        Training error: 0.352642
Iteration 10       Training error: 0.350569

....

Iteration 997      Training error: 0.00111981
Iteration 998      Training error: 0.00111843
Iteration 999      Training error: 0.00111706
Iteration 1000     Training error: 0.0011157
NETWORK WEIGHTS
===============
Layer 1
------
-2.561168 2.538790 -2.589251
-0.770332 -0.212399 0.075913
1.886265 -1.926663 -1.888277
1.233641 1.415421 -1.243881
1.888271 -1.880845 1.937680
-2.398075 -2.444980 2.397720
0.675080 -0.449499 -0.187134
-1.626142 1.626569 -1.680457
-0.432686 0.870079 0.746575
-2.467137 -2.423554 -2.444506
-2.905327 2.927639 2.908822
0.178809 0.807696 -0.499326
-2.263792 2.236110 -2.289030
2.141228 -2.166324 -2.146262
-0.961022 0.368665 -0.519352
1.671652 1.605829 1.578243
Layer 2
------
4.218926 0.256066 2.682362 -1.515218 -2.665360 3.794517 0.199035 2.051653 -0.533876
     -3.897894 -5.539269 -0.179128 3.387229 3.200096 0.315796 1.884826 -0.745766
Desired output Vs Obtained output (test)
```

```
========================================
1 -- 0.997695 0 -- 0.00230545
0 -- 0.0022113 1 -- 0.997789
1 -- 0.997915 0 -- 0.00208488
0 -- 0.00231397 1 -- 0.997686
We end!! => Final test CCR: 100
**********
SEED 3
**********

....

**********
SEED 4
**********

....

**********
SEED 5
**********

....

Iteration 996      Training error: 0.00120944
Iteration 997      Training error: 0.00120797
Iteration 998      Training error: 0.0012065
Iteration 999      Training error: 0.00120504
Iteration 1000     Training error: 0.00120358
NETWORK WEIGHTS
===============
Layer 1
------
1.900581 -1.882362 1.857397
-2.842675 -2.859711 -2.889926
-0.465074 -0.098927 0.013740
2.268086 -2.294572 -2.254044
2.243438 -2.213363 2.180051
-0.256642 -0.105276 -0.120501
0.114388 -0.534062 -0.351653
-0.911573 -0.958489 -1.080537
-1.433847 1.444481 1.471491
-1.661529 -1.637269 1.715131
-0.052632 0.301068 -0.963939
2.370791 2.352868 -2.385332
-0.288566 -1.009871 0.350465
-2.658619 2.639775 -2.608652
0.887022 1.042201 1.112214
2.932930 -2.951392 -2.921607
Layer 2
------
-2.509621 -4.920168 0.292810 3.478740 -3.292094 1.022310 0.478927 -1.150676
    -1.462800 2.385706 0.120623 -3.790206 0.620506 4.659517 1.236610 5.406729
    0.433783
Desired output Vs Obtained output (test)
========================================
1 -- 0.997546 0 -- 0.00245364
0 -- 0.00234317 1 -- 0.997657
1 -- 0.99751 0 -- 0.00249031
0 -- 0.0023299 1 -- 0.99767
We end!! => Final test CCR: 100
WE HAVE FINISHED WITH ALL THE SEEDS
FINAL REPORT
*************
Train error (Mean +- SD): 0.00114768 +- 0.000104197
Test error (Mean +- SD): 0.00114768 +- 0.000104197
Train CCR (Mean +- SD): 100 +- 0
```

```
Test CCR (Mean +- SD): 100 +- 0

_____________________________________________________________________________

i02gupep@NEWTS:~/imc/la2/Debug$ ./la2 -t ../train_nomnist.dat -T ../test_nomnist.
    dat -i 500 -l 1 -h 4 -f 1 -s

....
FINAL REPORT
*************
Train error (Mean +- SD): 0.0751722 +- 0.00948445
Test error (Mean +- SD): 0.124133 +- 0.0127144
Train CCR (Mean +- SD): 86.7778 +- 4.27236
Test CCR (Mean +- SD): 78.4 +- 5.96937

_____________________________________________________________________________

i02gupep@NEWTS:~/imc/la2/Debug$ ./la2 -t ../train_nomnist.dat -T ../test_nomnist.
    dat -i 500 -l 1 -h 4 -e 1 -m 2 -f 1 -s

....

FINAL REPORT
*************
Train error (Mean +- SD): 0.048567 +- 0.00473395
Test error (Mean +- SD): 0.113218 +- 0.0124186
Train CCR (Mean +- SD): 92.1333 +- 0.535182
Test CCR (Mean +- SD): 83.6667 +- 1.31233



_____________________________________________________________________________


i02gupep@NEWTS:~/imc/la2/Debug$ ./la2 -t ../train_nomnist.dat -T ../test_nomnist.
    dat -i 500 -l 1 -h 4 -e 1 -m 2 -f 0 -s

....

FINAL REPORT
*************
Train error (Mean +- SD): 0.0390912 +- 0.00594294
Test error (Mean +- SD): 0.0544008 +- 0.00626936
Train CCR (Mean +- SD): 85.0222 +- 3.20243
Test CCR (Mean +- SD): 77.5333 +- 4.09336

_____________________________________________________________________________


i02gupep@NEWTS:~/imc/la2/Debug$ /la2 -t ../train_nomnist.dat -T ../test_nomnist.dat
     -i 500 -l 1 -h 8 -e 0.1 -m 2 -f 1 -s -o
....

FINAL REPORT
*************
Train error (Mean +- SD): 0.0425087 +- 0.0151428
Test error (Mean +- SD): 0.151445 +- 0.0202287
Train CCR (Mean +- SD): 91.8222 +- 3.62144
Test CCR (Mean +- SD): 84 +- 4.26224

_____________________________________________________________________________

i02gupep@NEWTS:~/imc/workspace/la2/Debug$ ./la2 -t ../train_compas.dat -T ../
    test_compas.dat
...
*************
Train error (Mean +- SD): 0.228885 +- 0.000211591
Test error (Mean +- SD): 0.230951 +- 0.000302507
```

```
Train CCR (Mean +- SD): 68.4249 +- 0.18029
Test CCR (Mean +- SD): 67.6718 +- 0.47264
```

4.3. [OPCIONAL] Obtención de predicciones para Kaggle

El mismo ejecutable de la práctica permitirá obtener las predicciones de salidas para un determinado conjunto de datos. Esta salida debe guardarse en un archivo `.csv` que deberéis subir a Kaggle para participar en la competición (ver el archivo `sampleSubmission.csv` en la plataforma Kaggle). Este modo de predicción, utiliza unos parámetros diferentes a los citados anteriormente:

- Argumento `p`: Flag que indica que el programa se ejecutará en modo de predicción.
- Argumento `T`: Indica el nombre del fichero que contiene los datos de *test* que se utilizarán (`kaggle.dat`).
- Argumento `w`: Indica el nombre del fichero que contiene la configuración y los pesos del modelo entrenado que se utilizará para predecir las salidas.

A continuación, se muestra un ejemplo de ejecución del modo de entrenamiento haciendo uso del parámetro `w` para guardar la configuración del modelo.

```
i02gupep@NEWTS:~/imc/workspace/la2/Debug$ ./la2 -t ../train.dat -T ../test.dat -i 1000 -
    l 1 -h 4 -e 0.7 -m 1 -f 1 -s -w weights.txt

**********
SEED 1
**********

...

**********
SEED 2
**********

...

**********
SEED 3
**********

...

**********
SEED 4
**********

...

**********
SEED 5
**********

...

FINAL REPORT
*************
Train error (Mean +- SD): 0.061885 +- 0.00883649
Test error (Mean +- SD): 0.0627513 +- 0.00875778
Train CCR (Mean +- SD): 83.04 +- 1.93203
Test CCR (Mean +- SD): 81.97 +- 1.8529
```

A continuación, se muestra un ejemplo de salida del modo de predicción:

```
i02gupep@NEWTS:~/imc/practica1/Debug$ ./la2 -T ../kaggle.dat -p -w weights.txt
Id,Category
0,9
1,5
2,2
3,4
4,8
5,3

...

1994,3
1995,0
1996,1
1997,1
1998,4
1999,6
```