Práctica 2: Perceptrón multicapa para problemas de clasificación

Convocatoria de febrero (curso académico 2020/2021)

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

13 de octubre de 2020

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 **25 de octubre de 2020**. 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 *divorce*, 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 cuatro 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.

- *Base de datos divorce*: *divorce* contiene 127 patrones de entrenamiento y 43 patrones de test. La base de datos contiene la respuesta a una serie de preguntas de un conjunto de encuestas en las que se pretende predecir si se va a producir un divorcio en la pareja. Las respuestas a las preguntas de la encuesta se proporcionan en escala de Likert con valores de 0 a 4. Todas las variables de entrada se consideran numéricas. A continuación se muestran dos ejemplos de preguntas asociadas a la encuesta:

  - *23. I know my spouse's favorite food.*
  - *24. I can tell you what kind of stress my spouse is facing in her/his life.*

  La base de datos tiene un total de 54 preguntas (por lo tanto, 54 variables de entrada) y dos categorías (0 no hay divorcio, 1 hay divorcio)[2].

- *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`.

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

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

[2] Para más información, consultar `https://archive.ics.uci.edu/ml/datasets/Divorce+Predictors+data+set`

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

Figura 1: Subconjunto de letras del conjunto de entrenamiento.

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

- 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:

- *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*. No emplees validación ($v = 0{,}0$) y desactiva el factor de decremento ($F = 1$).
  - Para el problema XOR utilizar la arquitectura que resultó mejor en la práctica anterior.
  - Para los problemas *divorce* 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*, $v = 0{,}0$ y $F = 1$):
  - 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* ($v = 0{,}0$ y $F = 1$).
- Finalmente, establece los mejores valores por los parámetros v y F, utilizando $v \in \{0{,}0; 0{,}15; 0{,}25\}$ y $F \in \{1, 2\}$.
- NOTA1: para la base de datos XOR, considerar siempre que $v = 0{,}0$ (no hay validación).

- NOTA2: observa si cuando activamos la validación ($v = 0{,}15$ o $v = 0{,}25$), el número de iteraciones medio disminuye con respecto a no considerar validación ($v = 0{,}0$), ya que esto implica un menor coste computacional y supone una ventaja.

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*: $CCR_{\text{entrenamiento}} = CCR_{\text{test}} = 50\,\%$.
- *Base de datos divorce*: $CCR_{\text{entrenamiento}} = 90{,}5512\,\%; CCR_{\text{test}} = 90{,}6977\,\%$.
- *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* (también se pueden incluir el CCR de validación).

- 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 $test$.
    - 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 `v`: Indica el ratio de patrones de entrenamiento que se van a utilizar como patrones de validación. Por defecto, utilizar $v = 0{,}0$.
    - Argumento `d`: Indica el valor del factor de decremento (F en las diapositivas) a utilizar por cada una de las capas. Por defecto, utilizar $F = 1$.
    - 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.

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

- 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        Validation error: 0
7  Iteration 2      Training error: 0.428928        Validation error: 0
8  Iteration 3      Training error: 0.359314        Validation error: 0
9  Iteration 4      Training error: 0.404694        Validation error: 0
10 Iteration 5      Training error: 0.354386        Validation error: 0
11 Iteration 6      Training error: 0.38935         Validation error: 0
12 Iteration 7      Training error: 0.351464        Validation error: 0
13 Iteration 8      Training error: 0.376896        Validation error: 0
14 Iteration 9      Training error: 0.348207        Validation error: 0
15
16 ....
17
18 Iteration 997    Training error: 0.000854865     Validation error: 0
19 Iteration 998    Training error: 0.000853851     Validation error: 0
20 Iteration 999    Training error: 0.000852839     Validation error: 0
21 Iteration 1000   Training error: 0.000851829     Validation error: 0
22 NETWORK WEIGHTS
23 ===============
24 Layer 1
25 ------
26 1.890014 -1.701798 1.897538
27 0.330608 -0.183608 -0.265321
28 -0.494545 0.501866 -0.647277
29 -0.127540 0.794900 0.055426
30 -1.714361 1.917311 1.665996
31 1.877923 1.585010 1.831484
32 -2.962570 2.817746 -2.966236
33 -2.119120 -2.356071 2.169989
34 -2.635774 -2.427666 -2.633887
35 -1.783463 2.066134 1.723217
36 -0.212003 1.037069 0.177721
37 -1.790252 2.016734 1.726581
38 2.405366 2.624863 -2.454070
39 2.159760 1.917194 2.151348
40 -2.390844 2.232019 -2.398387
41 0.712774 0.364864 0.987507
42 Layer 2
43 ------
44 -2.034694 0.645946 0.169628 -0.859297 -2.580101 2.436878 4.976043 3.776574
       -4.019719 -2.962101 -0.721652 -2.916534 -4.314177 3.248817 3.412149 1.108400
       0.247830
45 Desired output Vs Obtained output (test)
46 ========================================
47 1 -- 0.998009 0 -- 0.00199144
48 0 -- 0.00141374 1 -- 0.998586
49 1 -- 0.998492 0 -- 0.00150804
50 0 -- 0.00189549 1 -- 0.998105
51 We end!! => Final test CCR: 100
52 **********
53 SEED 2
54 **********
55 Iteration 1      Training error: 0.373712        Validation error: 0
56 Iteration 2      Training error: 0.446452        Validation error: 0
```

```
Iteration 3      Training error: 0.382052        Validation error: 0
Iteration 4      Training error: 0.403883        Validation error: 0
Iteration 5      Training error: 0.383328        Validation error: 0

....

Iteration 997    Training error: 0.00091654      Validation error: 0
Iteration 998    Training error: 0.000915438     Validation error: 0
Iteration 999    Training error: 0.000914339     Validation error: 0
Iteration 1000   Training error: 0.000913242     Validation error: 0
NETWORK WEIGHTS
===============
Layer 1
------
-2.587098 2.578320 -2.619753
-0.759566 -0.149755 -0.002021
2.047701 -2.066665 -2.047086
1.337941 1.483398 -1.350814
1.836233 -1.846479 1.891185
-2.385468 -2.426681 2.388022
0.701236 -0.490246 -0.277441
-1.694265 1.704642 -1.750630
-0.416997 0.865281 0.703919
-2.532052 -2.493683 -2.503934
-2.877679 2.886804 2.877883
0.194349 0.803944 -0.534277
-2.335808 2.323417 -2.366562
2.216600 -2.226776 -2.218230
-0.963446 0.411973 -0.568539
1.548762 1.482188 1.418917
Layer 2
------
4.304145 0.188449 2.996299 -1.668890 -2.590935 3.737036 0.263283 2.188783 -0.513814
     -4.092572 -5.431650 -0.199261 3.583364 3.350478 0.350937 1.630442 -0.783647
Desired output Vs Obtained output (test)
=========================================
1 -- 0.99812 0 -- 0.0018796
0 -- 0.0018402 1 -- 0.99816
1 -- 0.998286 0 -- 0.00171411
0 -- 0.00186534 1 -- 0.998135
We end!! => Final test CCR: 100
**********
SEED 3
**********

....

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

....

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

....

Iteration 995    Training error: 0.000996214     Validation error: 0
Iteration 996    Training error: 0.000995021     Validation error: 0
Iteration 997    Training error: 0.00099383      Validation error: 0
Iteration 998    Training error: 0.000992642     Validation error: 0
Iteration 999    Training error: 0.000991457     Validation error: 0
Iteration 1000   Training error: 0.000990274     Validation error: 0
NETWORK WEIGHTS
===============
```

```
Layer 1
------
1.924121 -1.907170 1.880177
-2.877435 -2.894421 -2.923764
-0.465113 -0.098912 0.013589
2.301898 -2.325585 -2.286912
2.269386 -2.242283 2.208006
-0.260069 -0.097105 -0.120448
0.117679 -0.533281 -0.353352
-0.936595 -0.984372 -1.101798
-1.456342 1.467005 1.490475
-1.701050 -1.676890 1.751372
-0.054140 0.301263 -0.964476
2.406749 2.388844 -2.422281
-0.296602 -1.008889 0.356140
-2.691590 2.674935 -2.642923
0.907242 1.057448 1.125275
2.963765 -2.980305 -2.951356
Layer 2
------
-2.556302 -5.030181 0.300620 3.561749 -3.354764 1.023757 0.480647 -1.176288
    -1.501465 2.462988 0.125039 -3.878418 0.627243 4.759606 1.263119 5.504183
    0.443612
Desired output Vs Obtained output (test)
========================================
1 -- 0.997981 0 -- 0.00201925
0 -- 0.00193022 1 -- 0.99807
1 -- 0.99795 0 -- 0.00204989
0 -- 0.00191499 1 -- 0.998085
We end!! => Final test CCR: 100
WE HAVE FINISHED WITH ALL THE SEEDS
FINAL REPORT
*************
Train error (Mean +- SD): 0.000950977 +- 8.47617e-05
Test error (Mean +- SD): 0.000950977 +- 8.47617e-05
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 -e 0.7 -m 1 -f 1 -s

....

FINAL REPORT
*************
Train error (Mean +- SD): 0.069922 +- 0.00569867
Test error (Mean +- SD): 0.123385 +- 0.0128565
Train CCR (Mean +- SD): 89 +- 0.906084
Test CCR (Mean +- SD): 80.6667 +- 1.5456
```

```
i02gupep@NEWTS:~/imc/la2/Debug$ ./la2 -t ../train_nomnist.dat -T ../test_nomnist.
    dat -i 500 -l 1 -h 8 -e 0.7 -m 1 -f 1 -s -v 0.25 -d 2

....

FINAL REPORT
*************
Train error (Mean +- SD): 0.0698994 +- 0.00589789
Test error (Mean +- SD): 0.11215 +- 0.00631624
Train CCR (Mean +- SD): 87.9111 +- 1.02426
Test CCR (Mean +- SD): 80.4667 +- 1.84992
```

```
186 ____________________________________________________________________________________
187
188
189 i02gupep@NEWTS:~/imc/la2/Debug$ ./la2 -t ../train_nomnist.dat -T ../test_nomnist.
        dat -i 500 -l 1 -h 4 -e 0.7 -m 1 -f 0 -s
190
191 ....
192
193 FINAL REPORT
194 *************
195 Train error (Mean +- SD): 0.0607302 +- 0.00429784
196 Test error (Mean +- SD): 0.0688165 +- 0.00518643
197 Train CCR (Mean +- SD): 77.1778 +- 4.43652
198 Test CCR (Mean +- SD): 72.0667 +- 4.29082
199
200 ____________________________________________________________________________________
201
202
203 i02gupep@NEWTS:~/imc/la2/Debug$ ./la2 -t ../train_nomnist.dat -T ../test_nomnist.
        dat -i 500 -l 1 -h 8 -e 0.7 -m 1 -f 1 -s -v 0.25 -d 2 -o
204 ....
205
206 FINAL REPORT
207 *************
208 Train error (Mean +- SD): 0.205463 +- 0.0543647
209 Test error (Mean +- SD): 0.225942 +- 0.057252
210 Train CCR (Mean +- SD): 68.6815 +- 4.17005
211 Test CCR (Mean +- SD): 66.6667 +- 5.07171
212
213 ____________________________________________________________________________________
214
215 i02gupep@NEWTS:~/imc/workspace/la2/Debug$ ./la2 -t ../train_divorce.dat -T ../
        test_divorce.dat -i 1000 -l 1 -h 8 -e 0.7 -m 1 -f 0 -s
216 ...
217 FINAL REPORT
218 *************
219 Train error (Mean +- SD): 0.000646809 +- 0.000112738
220 Test error (Mean +- SD): 0.0212406 +- 0.000947102
221 Train CCR (Mean +- SD): 100 +- 0
222 Test CCR (Mean +- SD): 97.6744 +- 0
```

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.

```
1 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
2
```

```
**********
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
```