# Práctica 2. Perceptrón multicapa para problemas de clasificación

## Introducción a los modelos computacionales

Ingeniería Informática (Rama Computación)

Escuela Politécnica Superior

Universidad de Córdoba

Curso 2016 – 2017

**Autor:** Rubén Medina Zamorano - 46068656R

**Email:** *i32mezar@uco.es*

# ÍNDICE

## 1. INTRODUCCIÓN

Se trata de la segunda práctica para la asignatura Introducción a los Modelos Computacionales. En ella, procederemos a elaborar una memoria que describa el problema a tratar, analice el pseudocódigo y el funcionamiento del algoritmo utilizado y por último, obtendremos unos resultados que estudiaremos y compararemos para llegar a unas conclusiones.

## 2. DESCRIPCIÓN DE LOS MODELOS DE REDES NEURONALES

### 2.1. Arquitectura

El modelo que se desarrollará serán redes neuronales basadas en un perceptrón multicapa. Esto quiere decir, que nuestro modelo tendrá unas entradas y mediante las neuronas internas, se obtendrá una probabilidad de pertenencia a la clase positiva o las probabilidades de varias clases. Las entradas serán suministradas mediante un fichero de texto, aunque también se podría implementar para introducir los datos manualmente. Y posteriormente, se procesan unas salidas que representan la probabilidad de pertenencia a la clase Ci, normalmente con N-1 salidas siendo N el número de clases totales y con $P(x=Cn) = 1- (P(C1) + P(C2) + \dots + P(Cn-1))$.

Aún así, cabe aclarar que el programa que se abordará estará configurado para parametrizar un problema de regresión con funciones sigmoides o bien realizar una clasificación con una función softmax que aporte la probabilidad.

El funcionamiento de entrenamiento consiste en propagar las entradas, mediante los pesos de los enlaces y comparar las salidas con el valor objetivo. Y retropropagar el error de la salida, para poder reajustar los pesos en los enlaces. Así, intentar minimizar la función del error. Y por último, comprobar el error para los test cuando se finaliza el entrenamiento.

Hasta aquí, el funcionamiento es similar a la anterior práctica con la regresión pero con clasificación, sin embargo, se ha modificado el modo de entrenamiento para online y offline. El modo online anterior iba actualizando los pesos para cada patrón, pero en offline los pesos se actualizan cuando se ha finalizado 1 iteración recorriendo todos los patrones. Por otro lado, se ha incorporado el cálculo del error por la entropía cruzada, esto produce una modificación con logaritmos más brusca en los pesos, a diferencia del MSE implementado anteriormente.

### 2.2. Organización de capas

La estructura de nuestro modelo se organizará generalmente con una capa de n entradas, una capa oculta con 5 neuronas por defecto y con un único valor de salida. Para el caso de clasificación, la organización sería similar, a diferencia que la salida determinaría la probabilidad de pertenencia a la clase positiva y la probabilidad de la clase negativa sería 1-P(C1). A sí mismo, se formularía para una mayor cantidad de clases.

El número de entradas se establecerán mediante el fichero de texto, al igual que las salidas obtenidas. Y el número de capas ocultas o el número de neuronas se establecerán mediante los parámetros de entrada definidos más adelante. Además, se podrá configurar la existencia de sesgo en estas capas ocultas.

Por otro lado, los pesos de las conexiones entre nodos se establecen en la neurona que tiene la entrada de ese enlace. Es decir, cada neurona de capa oculta o las neuronas de salida (capa h), tienen los vectores de pesos que conectan directamente con las salidas de la capa anterior (capa h-1).

Por último, la opción online aplica un entrenamiento online con actualización de pesos por cada patrón y la offline actualiza al final de cada iteración. La función softmax realiza la tarea de producir las probabilidades de salida o en su defecto, se aplicará la función sigmoide de regresión. Y se dará la opción de utilizar la fórmula MSE o entropía cruzada para el cálculo del error al retropropagar.

## 3. DESCRIPCIÓN DEL PSEUDOCÓDIGO DEL ALGORITMO DE RETROPROPAGACIÓN

En esta apartado, procederemos a analizar el pseudocódigo y el funcionamiento de las funciones para el algoritmo de retropropagación:

Esta función copia el vector de entradas en la primera capa:

```
void PerceptronMulticapa::alimentarEntradas(double* input) {
   Para i desde  0 hasta pCapas[0].nNumNeuronas
   {
         pCapas[0].pNeuronas[i].x <- input[i];
         pCapas[0].pNeuronas[i].dX <- input[i];
   }
}
```

Esta función propaga las entradas, comprobando el sesgo y los pesos de cada enlace junto con el valor de los nodos en la capa anterior. En la última capa, la función se aplica según el valor de softmax, o sigmoide por defecto.

```
void PerceptronMulticapa::propagarEntradas(bool softmax) {
      double sumSoftmax<-0.0;
      int salidas <- pCapas[nNumCapas-1].nNumNeuronas;
      double exponentes[salidas];
      Para i desde 0 hasta nCapas//La capa 0 es la entrada
            Para j desde 0 hasta nNeuronas
                  int nEntradas <- pCapas[i-1].nNumNeuronas;
                  double exponente <- 0.0;
                  Si hay sesgo
                        nEntradas++;
                        exponente+= pCapas[i].pNeuronas[j].w[0];
```

```
                  Para k desde 0 hasta nEntradas
                        double weight;
                        weight <- pCapas[i].pNeuronas[j].w[k];
                        double x <- pCapas[i-1].pNeuronas[k].x;
                        exponente += (weight*x);
                  Si es softmax y i== (nCapas-1)
                              sumSoftmax+= exp(exponente);
                              exponentes[j] <- exp(exponente);
                  sino
                  pCapas[i].pNeuronas[j].x<-1/(1+exp(-exponente));

Si es softmax
      Para i desde 0 hasta nSalidas
      pCapas[nNumCapas-1].pNeuronas[i].x<-exponentes[i]/sumSoftmax;

}
```

---

Realiza la retropropagación del error desde la última capa hasta la inicial, tomando como referencia el vector objetivo de salida. Está considerando los valores para dX, pero aún no esta aplicando el cambio en los pesos. Considera si la función de salida aplicada es softmax o sigmoide, y después diferencia entre un error MSE o entropía cruzada para retropropagar.

---

```
void  PerceptronMulticapa::retropropagarError(double*  objetivo,  int
funcionError, bool softmax) {
      //Solo capa de salida
      Si es softmax
            Para j desde 0 hasta nSalidas
                  double sum<-0.0;
                  double outj <- pCapas[nNumCapas-1].pNeuronas[j].x;
                  Para i desde 0 hasta nSalidas{
                     double outi <-pCapas[nNumCapas-1].pNeuronas[i].x;
                  Si i==j
                              Si MSE
                        sum += (objetivo[i] - outi )*outj*(1-outi);
                              sino//Entropia Cruzada
                        sum += (objetivo[i] / outi )*outj*(1-outi);
                  sino
                              Si MSE
                        sum += (objetivo[i] - outi )*outj*(0-outi);
                              sino//Entropia cruzada
                        sum += (objetivo[i] / outi )*outj*(0-outi);

                              pCapas[nNumCapas-1].pNeuronas[j].dX <- -sum;
                  }
      sino //Sigmoide
            Para i desde 0 hasta nSalidas
                  double out = pCapas[nNumCapas-1].pNeuronas[i].x;
                  Si MSE
pCapas[nNumCapas-1].pNeuronas[i].dX<- - (objetivo[i]-out)*out*(1-out);
                  sino //entropía cruzada
pCapas[nNumCapas-1].pNeuronas[i].dX<- - (objetivo[i]/out)*out*(1-out);
```

---

```
      //Capas ocultas
      Para i desde nCapas-2 hasta 0//La capa 0 es la entrada
            Para j desde 0 hasta nNeuronas
                  int nEntradas <- pCapas[i+1].nNumNeuronas;
                  double sum <- 0.0;
                  Para k desde 0 hasta nEntradas
      sum+=pCapas[i+1].pNeuronas[k].w[j]*pCapas[i+1].pNeuronas[k].dX;

      double out =  pCapas[i].pNeuronas[j].x;
      pCapas[i].pNeuronas[j].dX = sum*out*(1-out);
}
```

Esta función acumula los cambios producidos en el valor deltaW para aplicarlo posteriormente al peso de los enlaces

```
   // Acumular los cambios producidos por un patrón en deltaW
   void PerceptronMulticapa::acumularCambio() {
      Para i desde 1 hasta nNumCapas
            Para j desde 0 hasta pCapas[i].nNumNeuronas
                  int nEntradas <- pCapas[i-1].nNumNeuronas;
                  Para k desde 0 hasta nEntradas
                        Si existe bSesgo && k==0) //Si tiene sesgo, el
   primer peso se considera diferente, porque no tiene entrada x.
                              deltaW[k]+= pCapas[i].pNeuronas[j].dX;
                        sino
                              deltaW[k]+=pNeuronas[j].dX   *   pCapas[i-
   1].pNeuronas[k].x;
   }
```

Esta función ajusta los pesos, aplicando los cambios obtenidos en las funciones anteriores de retropropagar el error y acumular el cambio

```
void PerceptronMulticapa::ajustarPesos() {
   Para i desde 1 hasta nNumCapas
      Para j desde 0 hasta pCapas[i].nNumNeuronas
            int nEntradas <- pCapas[i-1].nNumNeuronas;
            Si existe bSesgo
                  nEntradas++;
            Para k desde 0 hasta nEntradas
            Si existe bSesgo y k==0
                  w[k]   +=   (-dEta)*   deltaW[k]   -   dMu   *   (dEta*
   ultimoDeltaW[k]);
            sino
                  w[k]     +=     (-dEta)*deltaW[k]     -     dMu     *
   (dEta*ultimoDeltaW[k]);
   }
```

Esta función simula el funcionamiento de una red neuronal para un patrón de entrada: en el que inicializa los valores, propaga las entradas, comprueba el error, retropropaga el error. Si es online, realiza la actualización y ajuste de pesos por cada patrón.

---

```
void    PerceptronMulticapa::simularRed(double*    entrada,    double*
objetivo) {

Si es online
      Para i desde 1 hasta nNumCapas
            Para j desde 0 hasta pCapas[i].nNumNeuronas
                  int nEntradas <- pCapas[i-1].nNumNeuronas;
                  Para k desde 0 hasta nEntradas
                        ultimoDeltaW[k] <- deltaW[k];
                        pCapas[i].pNeuronas[j].deltaW[k] <- 0.0;
alimentarEntradas(entrada);
propagarEntradas();
retropropagarError(objetivo);
acumularCambio();
   Si es online
      ajustarPesos();
}
```

---

Entrena los n patrones de los ficheros de entrada, llamando n veces a simular red, y si es offline realiza el ajuste de pesos cuando termina todos los patrones.

---

```
void     PerceptronMulticapa::entrenar(Datos*     pDatosTrain,     int
funcionError, bool softmax) {
Si es offline
      Para i desde 1 hasta nNumCapas
            Para j desde 0 hasta pCapas[i].nNumNeuronas
                  int nEntradas <- pCapas[i-1].nNumNeuronas;
                  Para k desde 0 hasta nEntradas
                        ultimoDeltaW[k]<- deltaW[k];
                        pCapas[i].pNeuronas[j].deltaW[k]<- 0.0;
Para i desde 0 hasta el nPatrones
      simularRed(pDatosTrain->entradas[i],    pDatosTrain->salidas[i],
funcionError, softmax);

Si es offline
      ajustarPesos();
}
```

## 4. EXPERIMENTOS Y ANÁLISIS DE RESULTADOS

### 4.1. Bases de datos utilizadas

Las bases de datos que utilizaremos serán las suministradas en la carpeta /dat, y se dividen en 4 conjuntos, todos con ficheros de train y test:

- **Problema XOR**: esta base de datos representa el problema de clasificación no lineal del XOR. Aproximaríamos una clasificación mediante una regresión, cuyas entradas son -1 y 1 y las salidas son 1 o 0.
- **Base de datos iris**: tenemos 112 patrones de entrenamiento y 38 para test. Esta base de datos posee 4 variables independientes de entrada, que especifican longitud y ancho de sépalo y pétalo, con 3 clases de salida según corresponda a la clasificación de la flor iris: setosa, virginica o versicolor. Por ejemplo, las salidas serían 0 0 1 perteneciendo a la clase versicolor.
- **Base de datos digits**: está formada por 1275 patrones entrenamiento y 319 patrones de test. La base de datos está formada por un conjunto de dígitos del 0 al 9 escritos en una rejilla de 16x16 píxeles, estos píxeles son las entradas independientes que dan lugar a 10 salidas para clasificar el número correspondiente.

### 4.2. Parámetros de entrada

A la hora de ejecutar nuestro programa, tiene una serie de parámetros de entrada para modificar el aprendizaje o las estructura de nuestra red. Además, estos parámetros cuentan con un valor por defecto:

**- t**: Indica el nombre del fichero que contiene los datos de entrenamiento a utilizar. Sin este argumento, el programa no puede funcionar.

**- 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.

**- i**: Indica el numero de iteraciones del bucle externo a realizar. Si no se especifica, utilizar 1000 iteraciones.

**- l**: Indica el numero de capas ocultas del modelo de red neuronal. Si no se especifica, utilizar 1 capa oculta.

**- h:** Indica el numero de neuronas a introducir en cada una de las capas ocultas. Si no se especifica, utilizar 5 neuronas.

**- e:** Indica el valor del parámetro eta ($\eta$). Por defecto, utilizar $\eta = 0{,}1$.

**- m**: Indica el valor del parámetro mu ($\mu$). Por defecto, utilizar $\mu = 0{,}9$.

**- b**: Indica si se va a utilizar sesgo en las neuronas. Por defecto, no debemos utilizar sesgo.

**- f arg**: Indica la función de error que se utilizará para retropropagar, bien 0 es MSE o bien 1 es entropía cruzada.

**- s**: Indica que se utilizará una función softmax de clasificación en la salida, por defecto se utilizará la función sigmoide.

**- o**: Indica que el algoritmo se ejecutará de forma online, si no se especifica, por defecto será offline.

### 4.3. Resultados obtenidos

Para las bases de datos dadas y los diferentes parámetros realizaremos una serie de experimentos modificando esos parámetros, y ajustando una mejor estructura para minimizar el error en test y obtener un buen modelo que clasifique maximizando la precisión.

#### Errores en IRIS

Los resultados siguientes han sido obtenidos con sesgo y factor de momento por defecto 0.9. Además, para esta primera etapa vamos a considerar entropía cruzada y activación softmax en una configuración offline.

| Arquitectura de la red | Estadísticos | | | |
|---|---|---|---|---|
| | Error Train | Error Test | CCR Train | CCR Test |
| | (media, DV) | (media, DV) | (media, DV) | (media, DV) |
| **{n:5:k}** | 0.1734, 0.1552 | 0.1745, 0.1562 | 96.785, 0.7985 | 94.736, 0 |
| **{n:10:k}** | 0.1822, 0.1635 | 0.1777, 0.1603 | 96.071, 3.001 | 96.842, 2.882 |
| **{n:50:k}** | 0.1309, 0.1171 | 0.1089, 0.0974 | 96.607, 0.3992 | 96.315, 2.353 |
| **{n:75:k}** | 0.1243, 0.1111 | 0.1049, 0.0939 | 96.607, 0.399 | 95.789, 2.353 |
| **{n:5:5:k}** | 0.3234, 0.2899 | 0.3002, 0.2692 | 96.785, 1.851 | 95.263, 2.882 |
| **{n:10:10:k}** | 0.2037, 0.1841 | 0.1728, 0.1563 | 94.821, 5.144 | 95.263, 2.882 |
| **{n:50:50:k}** | 0.1036, 0.0926 | 0.1104, 0.0989 | 97.321, 0 | 95.263, 1.176 |
| **{n:75:75:k}** | 0.0986, 0.0882 | 0.1058, 0.0946 | 97.5, 0.3992 | 94.7368, 0 |

A partir de la anterior tabla, podemos deducir que la estructura **{n:10:k}** tiene la mayor precisión de clasificación en test con 96.8. Aunque hay otras estructuras que minimizan el error, pero realmente es más importante el número de patrones clasificados correctamente que el valor de la salida, sea semejante a la salida real, ya que estamos hablando de probabilidades de clasificación en lugar de regresión. Teniendo esta estructura probaremos varias configuraciones y posteriormente la versión online.

| Arquitectura de la red | Estadísticos | | | |
|---|---|---|---|---|
| | Error Train | Error Test | CCR Train | CCR Test |
| | (media, DV) | (media, DV) | (media, DV) | (media, DV) |
| **MSE y sigmoide** | 0.6102, 0.5475 | 0.5771, 0.5179 | 83.035, 13.932 | 82.631, 13.105 |
| **MSE y softmax** | 0.1914, 0.1715 | 0.1544, 0.1383 | 96.785, 1.018 | 97.894, 2.882 |
| **Entropía y softmax** | 0.1822, 0.1635 | 0.1777, 0.1603 | 96.071, 3.001 | 96.842, 2.882 |

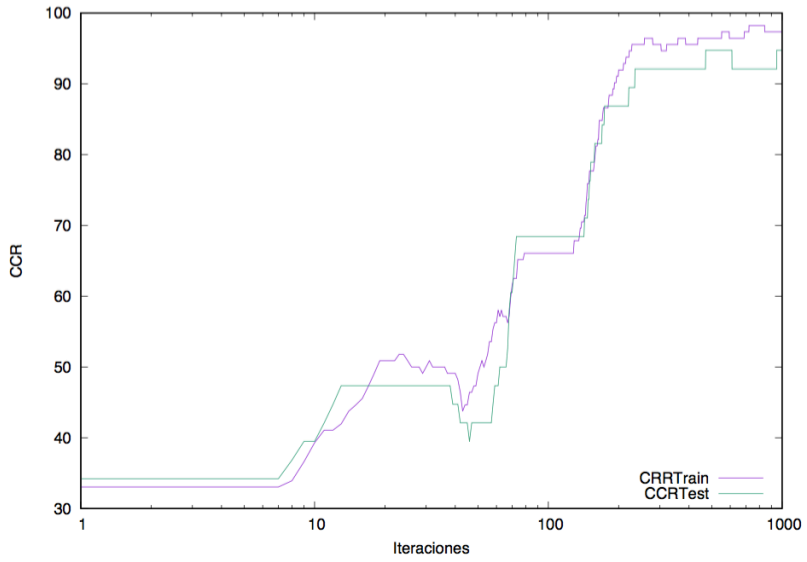
La **entropía cruzada en sigmoide**: se tiene que utilizar con la función softmax ya que la salida de softmax es un valor entre 0 y 1, porque todas ellas deben sumar probabilidades para llegar a 1. Y por tanto en la entropía se aplica el logaritmo a un

valor entre 0 y 1. En el caso de sigmoide, puede devolver un valor superior a 1 o inferior de 0 porque se trata de un problema de regresión y la entropía no funciona correctamente.

Obviamente, con softmax obtenemos una mejor precisión de clasificación ya que es la función que aporta la probabilidad de pertenencia. Y luego, dentro de softmax, parece ser que MSE proporciona mejores resultados, pero hay escasa diferencia con entropía. Ahora probaremos online, y con distintos parámetros.

| Modo | Mu | Eta | MSE Train/Test | CCR Train/Test |
|---|---|---|---|---|
| **Online** | 0.9 | 0.1 | 0.8463/0.8255 | 73.392/72.105 |
| **Offline** | " | 0.1 | 0.1914/0.1544 | 96.785/97.894 |
| " | " | 0.5 | 0.0667/0.0761 | 97.678/97.368 |
| " | " | 0.9 | 0.0529/0.0604 | 98.214/96.842 |
| " | 0.5 | 0.1 | 0.4822/0.4435 | 88.214/87.894 |
| " | 0.1 | 0.1 | 0.583 / 0.5543 | 83.75 / 82.105 |

Aumentando eta, se obtienen mejores resultados para train pero se empeora el test, y nuestro objetivo es mejorar el test. Y los mejores valores son para mu = 0.9 y eta=0.1. Vamos a representar la función offline gráficamente.




Podemos ver que al tener muchos patrones, se entrena rápidamente en un tan solo 100 iteraciones nos acercamos a un 100% de precisión y ahí converge nuestro algoritmo. Y ahora representamos la matriz de confusión en test de los valores obtenidos frente a los valores esperados.

| | | **Salida esperada** | | |
|---|---|---|---|---|
| | | Setosa | Virginica | Versicolor |
| **Salida obtenida** | Setosa | 13 | 0 | 0 |
| | Virginica | 0 | 10 | 0 |
| | Versicolor | 0 | 2 | 13 |

Tenemos finalmente una precisión del 97% en el que solo 2 patrones han sido clasificados incorrectamente de 38 valores de test.

**Errores en DIGITS**

Extraemos los siguientes valores a partir de una configuración con sesgo, 300 iteraciones, error de entropía cruzada y función softmax.

| **Arquitectura de la red** | **Estadísticos** | | | |
|---|---|---|---|---|
| | Error Train | Error Test | CCR Train | CCR Test |
| | (media, DV) | (media, DV) | (media, DV) | (media, DV) |
| **{n:5:k}** | 1.078, 0.9644 | 1.0819, 0.9679 | 24.725, 6.238 | 24.514, 6.498 |
| **{n:10:k}** | 2.9849, 2.7254 | 2.808, 2.5586 | 44.7567, 8.969 | 45.078, 9.614 |
| **{n:50:k}** | 0.2864, 0.2562 | 0.3708, 0.3316 | 84.05, 1.05 | 75.799, 1.833 |
| **{n:75:k}** | 0.23, 0.2057 | 0.3188, 0.2852 | 87.142, 1.133 | 80.125, 2.434 |
| **{n:5:5:k}** | 1.1411, 1.0207 | 1.1411, 1.0207 | 13.956, 6.416 | 14.67, 7.599 |
| **{n:10:10:k}** | 1.0356, 0.9264 | 1.0376, 0.9282 | 28.995, 6.624 | 27.899, 5.155 |
| **{n:50:50:k}** | 0.3513, 0.3143 | 0.4047, 0.362 | 80.627, 0.4131 | 75.548, 1.551 |
| **{n:75:75:k}** | 0.2971, 0.2658 | 0.371, 0.3319 | 83.249, 3.857 | 77.304, 5.067 |

Podemos ver que con dos capas ocultas se obtienen peores resultados que con una por tanto escogeremos una sola capa. Y además, con 50 o 75 neuronas se obtienen mejores resultados pero el tiempo de cómputo y la complejidad de la red es tan elevada que no se hace interpretable, por lo que escogeremos la estructura {n:10:k} que podrá mejorar modificando los parámetros.

| **Arquitectura de la red** | **Estadísticos** | | | |
|---|---|---|---|---|
| | Error Train | Error Test | CCR Train | CCR Test |
| | (media, DV) | (media, DV) | (media, DV) | (media, DV) |
| **MSE y sigmoide** | | | | |
| **MSE y softmax** | | | | |
| **Entropía y softmax** | 2.9849, 2.7254 | 2.808, 2.5586 | 44.7567, 8.969 | 45.078, 9.614 |

| **Modo** | **Mu** | **Eta** | **MSE Train/Test** | **CCR Train/Test** |
|---|---|---|---|---|
| **Online** | 0.9 | 0.1 | - | - |
| **Offline** | " | 0.1 | 2.9849/2.808 | 44.7567/45.078 |
| " | " | 0.5 | 0.2105/0.2532 | 91.475/84.702 |
| " | " | 0.9 | 0.082/1.9377 | 96.813/81.379 |
| " | 0.5 | 1 | | |
| " | 0.1 | 1 | 0.1469/0.6584 | 93.438/ 82.946 |

La versión online no produce buenos resultados, y modificando los parámetros siempre da salidas incorrectas. Sin embargo, para el modo offline, podemos ver que al aumentar eta a 0.9 se aumenta la precisión y se minimiza el error.

**Errores en XOR**

En la práctica anterior escogimos una estructura {n:10:k}, así que partiremos de esta para probar los distinto parámetros, tipos de errores y modos de ejecución.

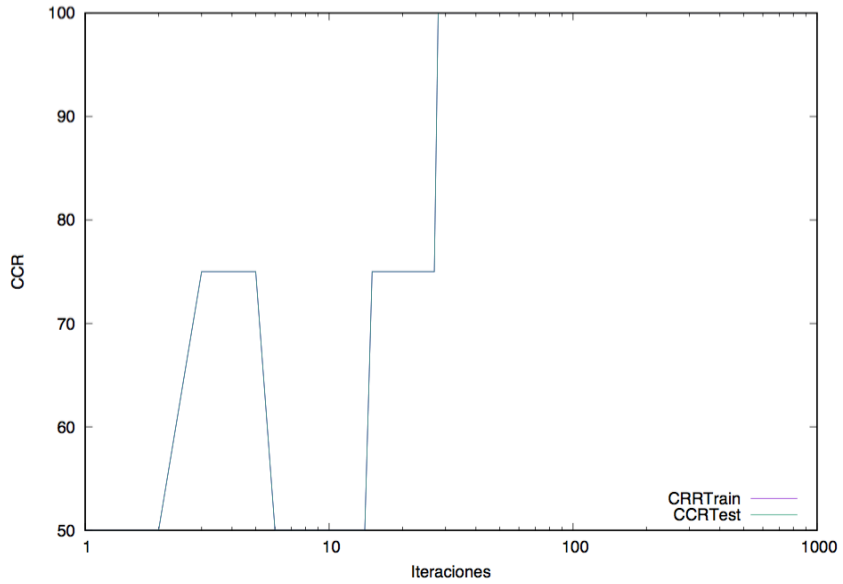
| **Arquitectura de la red** | **Estadísticos** | | | |
|---|---|---|---|---|
| | Error Train | Error Test | CCR Train | CCR Test |
| | (media, DV) | (media, DV) | (media, DV) | (media, DV) |
| **MSE y sigmoide** | 1.0809/ 0.967 | 1.0809/ 0.967 | 75/17.6777 | 75/17.6777 |
| **MSE y softmax** | 0.546/0.4911 | 0.546/0.4911 | 95/11.1803 | 95/11.1803 |
| **Entropía y softmax** | 0.3907/0.3508 | 0.3907/0.3508 | 100/0 | 100/0 |

Al tratarse de un problema de clasificación, escogeremos la función softmax y aplicaremos esta junto a la entropía cruzada.

| **Modo** | **Mu** | **Eta** | **MSE Train/Test** | **CCR Train/Test** |
|---|---|---|---|---|
| **Online** | 0.9 | 0.1 | 0.0892 | 100 |
| " | " | 0.5 | - | 55 |
| " | 0.5 | 0.1 | 0.6157 | 85 |
| " | 0.1 | " | 0.3497 | 90 |
| **Offline** | 0.9 | 0.1 | 0.39 | 100 |
| " | " | 0.5 | 0.174 | 95 |
| " | 0.5 | 0.1 | 0.5169 | 100 |
| " | 0.1 | 0.1 | 0.8489 | 95 |

Tratando de mejorar el modo offline, hemos obtenido peores resultados en error y precisión, por tanto los mejores valores son por defecto. Y además, hemos obtenido que online minimiza el error para una misma precisión. Por tanto, vamos a representar gráficamente para online, y la matriz de confusión final.

| | | **Salida esperada** | |
|---|---|---|---|
| | | 0 | 1 |
| **Salida obtenida** | 0 | 2 | 0 |
| | 1 | 0 | 2 |




Se puede observar en la gráfica que se obtiene rápidamente el 100 en CCR ya que al tener pocos patrones, todos iguales y usar el mismo fichero para entrenar y test, pues converge rápidamente y no hay lugar a confusión.

## 5. REFERENCIAS BIBLIOGRÁFICAS

- **Presentación del algoritmo de retropropagación del error:**

**http://moodle.uco.es/m1617/pluginfile.php/168808/mod_resource/content/1/Tema 4.- Clasificación No lineal y SVM 06-07 revisado.pdf**

- **Presentación del pseudocódigo del programa:**

**http://moodle.uco.es/m1617/mod/resource/view.php?id=66915**