Tarea Programada: El Algoritmo de Retropropagación

José Castro, Inteligencia Artificial
Computación San José - ITCR
1er Semestre

13 de abril de 2016

1. Introducción

El reconocimiento de patrones utilizando redes neurales es una tarea común en el campo de la inteligencia artificial. En esta tarea se introduce al estudiante en la programción de un algoritmo de redes neurales: el algoritmo de retropropagación. El propósito de la tarea es que el estudiante tenga experiencia de primera mano en el uso de estos algoritmos y su utilidad para el reconocimiento de patrones complejos.

2. Marco Teórico

El algoritmo de retropropagación es un agoritmo de aprendizaje supervisado para ejecutar en redes neurales, a continuación una explicación mas a fondo de estos conceptos.

2.1. Aprendizaje supervisado

En aprendizaje supervisado se cuenta con un conjunto de aprendizaje que esta conformado por *pares* de entrenamiento. Estos pares representan los patrones a identificar, donde a cada patrón se le asocia una categoría o patrón de salida. Es común que el patrón de salida sea mucho mas simple que el patrón de entrada, en cuyo caso la red lo que hace es un proceso de abstracción.

Por ejemplo, el conjunto de aprendizaje puede ser un conjunto de imágenes que han sido pre-clasificadas, de esta forma el patron de entrada es la imagen, y el patrón de salida es la clase o categoría a la cual pertenece la imágen.

Para el caso particular de redes neurales, tenemos la siguiente definición:

Definición 2.1 *El* conjunto de aprendizaje *en un problema de aprendizaje supervisado es un conjunto*

$$\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i) : i \in \{1, 2, \ldots, N\}\} \tag{1}$$

Donde los patrones de entrada $\mathbf{x}_i$ son de dimension N_{in}, y las clasificaciones de salida $\mathbf{y}_i$ son de dimension N_{out}. De esta manera, cada patrón de entrada $\mathbf{x}_i$ se puede representar como:

$$\mathbf{x}_i = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N_{\text{in}}} \end{bmatrix}$$

Y cada clasificación $\mathbf{y}_i$ se puede representar como:

$$\mathbf{y}_i = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N_{\text{out}}} \end{bmatrix}$$

Es comun que estos valores se encuentren normalizados tal que $x_i \in [0,1],\ \forall i$, y $y_i \in [0,1],\ \forall i$.

El objetivo de los algoritmos de aprendizaje supervisado es descubrir una manera de automáticamente asociar los patrones de entrada del conjunto de aprendizaje con sus respectivas salidas. Es deseable que los algoritmos de aprendizaje supervisado respondan bien a valores que no se encuentran en el conjunto de aprendizaje. Esta capacidad de responder correctamente a patrones que no se han visto por el algoritmo se llama *generalización*.

Puede suceder que los pares de entrenamiento asocien a cada vector con la etiqueta de una categoría, en cuyo caso la representación de $\mathbf{y}$ en el par de entrenamiento $(\mathbf{x}, \mathbf{y}) \in \mathcal{S}$ se simplifica y se suele denotar par por:

$$\mathcal{S} = \{(\mathbf{x}_i, \textit{clase}(\mathbf{x}_i)) : i \in \{1, 2, \ldots, N\}\} \tag{2}$$

2.2. La neurona artificial

Este algoritmo de redes neurales trabaja con una red compuesta de pequeños procesadores numéricos, estos procesadores numéricos son un modelo simplificado de la neurona natural. Cada neurona consta de un conjunto de entradas numéricas que afectan linealmente la activación de la neurona que llamaremos el net de la neurona. La activación net a su vez incide en la salida out de la neurona de manera no lineal gracias a una una función de achatamiento (squashing function).

Mas concretamente, si la red cuenta con un conjunto de entradas x_i con $i = 1, 2, \ldots n$, entonces esta también debe tener un conjunto de pesos w_i con $i = 1, 2, \ldots, n$. y la activación de la neurona esta dada por la suma:

$$net = \sum_{i=1}^{n} w_i x_i$$

El out de la neurona a su vez es la aplicación de una función continua no lineal g al valor del activación net de la neurona, esto es:

$$out = g(net) = g\left(\sum_{i=1}^{n} w_i x_i\right)$$

Casi que cualquier función no lineal puede ser utilizada, pero en nuestro caso utilizaremos la función lógistica, que es la mas utilizada para redes neurales cuyos valores de salida se encuentran en el intervalo $[0, 1]$.

La función logística es:

$$g(x) = \frac{1}{1 + e^{-z}}$$

A esta función también se le llama función sigmoide ya que al graficarla tiene forma de S. Esta función evalua a 0 en $-\infty$ y a 1 en ∞.

La función sigmoide tiene la propiedad deseable de que su derivada es:

$$g'(x) = g(x)(1 - g(x))$$

2.2.1. Arquitectura de la Red

En el algoritmo de retropropagación la red de neuronas se puede considerar como un grafo acíclico donde cada nodo es una neurona, y donde las señales se propagan desde las entradas hasta las salidas. La red de retropropagación se suele organizar en capas, numeradas 1, 2 hasta M, donde cada neurona de la capa i conecta con todas las neuronas de la capa $i + 1$.

La arquitectura típica de una red de retropropagación que se utiliza para aprender los pares de entrenamiento de $\mathcal{S}$ es la siguiente:

1. Un total de $M > 2$ capas, las capas se encuentran enumeradas, donde cada capa tiene un indice k que pertenece al rango $k \in \{1, 2, \ldots, M\}$.

2. Cada capa $k \in \{1, 2, \ldots M\}$ tiene un tamaño que denotamos por una constante N_k y en total tenemos el conjunto $\{N_1, N_2, \ldots, N_M\}$ de constantes. Cada capa k de la red tiene $N_k + 1$ neuronas indexadas por un índice $i \in \{1, 2, \ldots, N_M\}$ (la neurona 0 tiene una función especial que se explica mas adelante), excepto la última capa M que tiene N_M neuronas (no tiene neurona 0). Cada neurona de una capa $k < M$ tiene Las neuronas de la capa M tienen indice i que pertenece al rango $i \in \{0, 1, 2, \ldots, N_M\}$. La neurona i de la capa k la denotaremos por n_i^k.

3. Para que esta red pueda aprender el conjunto de aprendizaje $\mathcal{S}$ la dimensión de la primer capa debe coincidir con el tamaño de los patrones de entrada $\mathbf{x}$, y la dimensión de la última capa debe coincidir con la dimensión de las clasificaciones de salida $\mathbf{y}$. Es decir $N_1 = N_{in}$ y $N_M = N_{out}$ (ver la figura adjunta).

4. Cada neurona n_i^k para las capas $k = 1, 2, \ldots, M$ contiene los valores Δ, *net*, y *out*. Estos valores serán calculados en el proceso de propagar señales por la red. Cuando el contexto este claro, omitiremos los indices de los valores Δ, *net* y *out*, de lo contrario los denotaremos por Δ_i^k, net_i^k y out_i^k a los valores Δ, *net* y *out* de la neurona n_i^k.

5. La neurona con indice 0 para cada capa k (es decir n_0^k) no tiene entradas, y su salida siempre es 1, esto es $out_0^k = 1, \ \forall k$.

6. El vector columna conformado por los valores *net* de la capa k, se denotará por $\mathbf{net}_k$, esto es:

$$\boldsymbol{net}_k = \begin{bmatrix} net_1^k \\ net_2^k \\ \vdots \\ net_{N_k}^k \end{bmatrix}$$

Definición similar se da para los vectores $\boldsymbol{out}_k$ y Δ_k, excepto que el vector $\boldsymbol{out}_k$ contiene una entrada extra dada por out_0^k, esto es:

$$\boldsymbol{out}_k = \begin{bmatrix} 1 \\ net_1^k \\ net_2^k \\ \vdots \\ net_{N_k}^k \end{bmatrix}$$

7. Cada neurona n_i^k con $i > 0$ para las capas $k = 2, 3, \ldots, M$ tiene una función $g(\cdot)$ de *achatamiento*, en nuestro caso, utilizaremos para todas las neuronas la función:

$$g(x) = \frac{1}{1 + e^{-x}}$$

8. Todas las neuronas de la capa k estan conectadas con las neuronas de la capa $k+1$ (excepto con la neurona n_0^{k+1}, ya que esta neurona no tiene entradas) mediante conexiones que tienen un peso (número) asociado. La conexión de la neurona j de la capa k denotada por n_j^k, con una neurona i de la capa $k+1$ denotada por n_i^{k+1}, se representa por el peso $w_{i,j}^k$. La matriz de pesos que conecta a las neuronas de la capa k con la capa $k+1$ se le llama $\mathbf{W}_k$, esto es:

$$\mathbf{W}_k = \begin{bmatrix} w_{1,0}^k & w_{1,1}^k & \cdots & w_{1,N_k}^k \\ w_{2,0}^k & w_{2,1}^k & \cdots & w_{2,N_k}^k \\ \vdots & \vdots & \ddots & \vdots \\ w_{N_{k+1},0}^k & w_{N_{k+1},1}^k & \cdots & w_{N_{k+1},N_k}^k \end{bmatrix}$$

Note entonces que con esta notación

- $\boldsymbol{net}_{k+1} = \mathbf{W}_k \mathbf{out}_k$
- $\boldsymbol{out}_{k+1} = g(\boldsymbol{net}_{k+1})$

2.2.2. Propagación hacia adelante o funcionamiento normal de la red

Dada una red neural especificada como se indicó anteriormente, y dado un patrón de entrada $\mathbf{x}$, la red puede calcular una salida asociada a este patrón de la siguiente manera:

PROPAGACION-HACIA-ADELANTE($\mathbf{x} = (x_1, x_2, \ldots, x_{N_1})$)

1. Fije las salidas de las neuronas de la primer capa a los valores del patrón de entrada $\mathbf{x}$. Esto es: haga $out_i^1 = x_i, \forall i \in \{1, 2, \ldots, N_1\}$.

2. Para $k = 2, 3, \ldots, M$ haga:

   - Para cada neurona n_i^k, con $i = 1, 2, \ldots, N_k$ haga:
     a) $net_i^k \leftarrow \sum_{j=0}^{N_{k-1}} out_j^{k-1} w_{i,j}^{k-1}$
     b) $out_i^k \leftarrow g(net_i^k)$

Hecho esto, la salida de la red neural para el patrón de entrada $\mathbf{x}$ será el vector $\mathbf{out}$:

$$\mathbf{out} = \begin{bmatrix} out_1^M \\ out_2^M \\ \vdots \\ out_{N_M}^M \end{bmatrix}$$

2.2.3. Propagación hacia adelante en notación vectorial

El algoritmo anterior puede expresarse demanera mas succinta si utilizamos notación vectorial. Esto es:

PROPAGACION-HACIA-ADELANTE($\mathbf{x}$)

1. $\boldsymbol{out}_1 = \mathbf{x}$.

2. Para $k = 1, 2, \ldots, M-1$ haga:

   a) $\boldsymbol{net}_{k+1} \leftarrow \mathbf{W}_k \boldsymbol{out}_k$
   b) $\boldsymbol{out}_{k+1} \leftarrow [1 \ ; \ g(\boldsymbol{net}_k)]$ si $k < M-1$
   c) $\boldsymbol{out}_{k+1} \leftarrow g(\boldsymbol{net}_k)$ si $k = M-1$

2.3. El Algoritmo de Retropropagación

El algoritmo de retropropagación es un algoritmo de aprendizaje supervisado. El aprendizaje en la red sucede mediante la modificación de los pesos w_{ij} que se encuentran en las conexiones entre las neuronas.

2.4. Aprendizaje y la función de error

Para aprender un conjunto de aprendizaje $\mathcal{S}$ se utiliza una función de error que cuantifica la diferencia por mínimos cuadrados entre el resultado que calcula la red y el valor que deseamos que la red produzca. Si decimos que la salida de la red es el vector out_M

$$\mathbf{out} = \begin{bmatrix} out_1 \\ out_2 \\ \vdots \\ out_N \end{bmatrix}$$

Y el vector deseado esta denotado por

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Entonces nuestra función de error es una distancia de mínimos cuadrados dada por la fórmula:

$$E = \frac{1}{2}\sum_{i=1}^{N}(out_i - y_i)^2$$

Como la función de error es continua, se puede derivar con respecto a los pesos de la red y utilizar el método de Newton para encontrar el mínimo de esta función. El proceso de calcular esta derivada y aplicarla a los pesos de la red es la base de la propagación hacia atras que hace el algoirtmo de retropropagación.

2.4.1. Propagación hacia atrás

Supongamos ahora que se tiene un patrón de entrada $\mathbf{x}$ y su respectivo vector de clasificación deseado $\mathbf{y}$. Notese que este vector $\mathbf{y}$ no necesariamente corresponde al resultado que emite la red neural cuando esta hace propagación hacia adelante, el cual denotamos por $\mathbf{out}$.

La propagación hacia atrás modifica los pesos $w_{j,i}^k$ de la red neural y depende de un parámetro $\eta > 0$, el patrón de entrada $\mathbf{x}$ y la clasificación de salida $\mathbf{y}$. Este proceso se aplica después de hacer propagación hacia adelante y corresponde al siguiente procedimiento:

PROPAGACION-HACIA-ATRAS$(\mathbf{x}, \mathbf{y}, \eta)$

1. Para cada neurona n_i^M de la última capa M con $i = 1, 2, \ldots N_M$ calcule

$$\Delta_i^M \leftarrow \left[out_i^M \left(1 - out_i^M\right)\right] \times \left(y_i - out_i^M\right)$$

2. Para cada capa $k = (M-1)$ hasta 1 haga:

   - Para cada neurona n_j^k, con $j = 0, 1, 2, \ldots, N_k$ haga:

     a) Obtenga el valor de Δ_j^k con la siguiente ecuación:

$$\Delta_j^k \leftarrow \boldsymbol{out}_j^k \left(1 - \boldsymbol{out}_j^k\right) \sum_{i=1}^{N_{k+1}} \Delta_i^{k+1} w_{i,j}^k$$

     NOTA: Cuando $j = 0$, el valor de $\Delta_0^k = 0,\ \forall k$ es constante. Además, este Δ no se utiliza en ningún cálculo y se puede obviar.

     b) Modifique los pesos $w_{i,j}^k$ para cada $i = 1, 2, \ldots, N_{k+1}$ de la siguiente manera:

$$\begin{aligned} \Delta w_{i,j}^k &\leftarrow \eta \Delta_i^{k+1} \boldsymbol{out}_j^k \\ w_{i,j}^k &\leftarrow w_{i,j}^k + \Delta w_{i,j}^k \end{aligned}$$

2.4.2. Propagación hacia atrás notación vectorial

PROPAGACION-HACIA-ATRAS$(\mathbf{x}, \mathbf{y}, \eta)$

1.

$$\Delta_M \leftarrow [\boldsymbol{out}_M (1 - \boldsymbol{out}_M)] \times (\mathbf{y} - \boldsymbol{out}_M)$$

2. Para cada capa $k = (M-1)$ hasta 1 haga:

   a)

$$\Delta_k \leftarrow \boldsymbol{out}_k (1 - \boldsymbol{out}_k) \mathbf{W}_k' \Delta_{k+1}$$

   b) Modifique los pesos $\mathbf{W}_k$ para cada $k = 1, 2, \ldots, M-1$ de la siguiente manera:

$$\begin{aligned} \Delta \mathbf{W}_k &\leftarrow \eta \Delta_{k+1} \boldsymbol{out}_k' \\ \mathbf{W}_k &\leftarrow \mathbf{W}_k + \Delta \mathbf{W}_k \end{aligned}$$

En el algoritmo anterior tanto $\mathbf{W}_k'$ como $\boldsymbol{out}_k'$ son las transpuestas respectivas de $\mathbf{W}_k$ y $\boldsymbol{out}_k$.

2.4.3. Aprendizaje en el Algoritmo de Retropropagación

Teniendo claro los procesos de propagación hacia adelante y propagación hacia atrás, podemos proceder al aprendizaje de un conjunto de entrenamiento $\mathcal{S}$. Dicho procedimiento de aprendizae es:

RETROPROPAGACION$(\mathcal{S}, \eta)$

1. Inicialize los pesos de la red neural $w_{i,j}^k$ en valores aleatorios, usualmente dentro del rango $w_{i,j}^k \in [-3, 3]$.

2. Desordene el conjunto de aprendizaje $\mathcal{S}$.

3. Para cada par de entrenamiento $(\mathbf{x}, \mathbf{y}) \in \mathcal{S}$ haga:

   a) propage hacia adelante con el valor $\mathbf{x}$.

   b) propage hacia atrás utilizando los valores de $\mathbf{y}$ y η.

4. revise el criterio de finalización, si no se cumple vuelva al punto número 2, de lo contrario termine.

El criterio de finalización puede ser cualquiera de los siguientes:

- ya se cumplieron una cantidad predeterminada de iteraciones del algoritmo.
- El error calculado entre el valor obtenido $\mathbf{out}$ y el valor deseado $\mathbf{y}$ es menor a una constante ε para cada elemento $(\mathbf{x}, \mathbf{y}) \in \mathcal{S}$.
- El error calculado entre el valor obtenido $\mathbf{out}$ y el valor deseado $\mathbf{y}$ esta incrementando para un conjunto de valores $(\mathbf{x}', \mathbf{y}')$ que se utiliza de prueba y no estan en el conjunto de aprendizaje $\mathcal{S}$.

3. Tarea

En esta tarea usted debe implementar el algoritmo de retropropagación en Python, su base de datos de entrada será un conjunto de 10 digitos: 1,2,3,4,5,6,7,8,9 y 0. Los digitos vendrán en un vector binario de 64 entradas que representan los pixeles de una matriz 8x8. la base de datos tiene al inicio 3 números, el primero es la cantidad de pares de entrenamiento que tiene el archivo, el segundo es la dimensión de la entrada, y el tercero la dimensión de la salida. La base de datos se lista a continuación

```
10  64  10
0  0  0  1  1  0  0  0
0  0  1  1  1  0  0  0
0  0  0  1  1  0  0  0
0  0  0  1  1  0  0  0
0  0  0  1  1  0  0  0
0  0  0  1  1  0  0  0
0  0  0  1  1  0  0  0
0  0  1  1  1  1  0  0

1  0  0  0  0  0  0  0  0  0

0  0  1  1  1  1  0  0
0  1  1  1  1  1  1  0
0  1  1  0  0  0  1  1
0  0  0  0  0  1  1  0
0  0  0  0  1  1  0  0
0  0  0  1  1  0  0  0
0  0  1  1  1  1  1  1
0  1  1  1  1  1  1  1

0  1  0  0  0  0  0  0  0  0

0  1  1  1  1  1  1  1
0  1  1  1  1  1  1  1
0  0  0  0  0  1  1  0
0  0  0  1  1  0  0  0
0  0  0  1  1  1  1  0
0  0  0  0  0  0  1  1
0  1  1  0  0  1  1  1
0  0  1  1  1  1  1  0

0  0  1  0  0  0  0  0  0  0

0  1  1  0  1  1  0  0
0  1  1  0  1  1  0  0
0  1  1  0  1  1  0  0
0  1  1  1  1  1  1  1
0  0  0  0  1  1  0  0
0  0  0  0  1  1  0  0
0  0  0  0  1  1  0  0
0  0  0  0  1  1  0  0

0  0  0  1  0  0  0  0  0  0
```

```
0 1 1 1 1 1 1 1
0 1 1 0 0 0 0 0
0 1 1 0 0 0 0 0
0 1 1 1 1 1 0 0
0 0 0 0 0 1 1 0
0 0 0 0 0 0 1 1
0 1 1 1 1 1 1 1
0 1 1 1 1 1 1 0

0 0 0 0 1 0 0 0 0 0

0 0 0 0 1 1 0 0
0 0 0 1 1 0 0 0
0 0 1 1 0 0 0 0
0 1 1 0 0 0 0 0
0 1 1 1 1 1 1 0
0 1 1 0 0 0 1 1
0 1 1 0 0 0 1 1
0 0 1 1 1 1 1 0

0 0 0 0 0 1 0 0 0 0

0 1 1 1 1 1 1 1
0 0 0 0 0 0 1 1
0 0 0 0 0 1 1 0
0 0 0 0 0 1 1 0
0 0 0 0 1 1 0 0
0 0 0 0 1 1 0 0
0 0 0 1 1 0 0 0
0 0 0 1 1 0 0 0

0 0 0 0 0 0 1 0 0 0

0 0 1 1 1 1 1 0
0 1 1 0 0 0 1 1
0 1 1 0 0 0 1 1
0 0 1 1 1 1 1 0
0 1 1 0 0 0 1 1
0 1 1 0 0 0 1 1
0 1 1 0 0 0 1 1
0 0 1 1 1 1 1 0

0 0 0 0 0 0 0 1 0 0
```

```
0 0 1 1 1 1 1 0
0 1 1 0 0 0 1 1
0 1 1 0 0 0 1 1
0 0 1 1 1 1 1 1
0 0 0 0 0 1 1 0
0 0 0 0 1 1 0 0
0 0 0 0 1 1 0 0
0 0 0 1 1 0 0 0

0 0 0 0 0 0 0 0 1 0

0 0 1 1 1 1 1 0
0 1 1 0 0 0 1 1
0 1 1 0 0 0 1 1
0 1 1 0 0 0 1 1
0 1 1 0 0 0 1 1
0 1 1 0 0 0 1 1
0 1 1 0 0 0 1 1
0 0 1 1 1 1 1 0

0 0 0 0 0 0 0 0 0 1
```

Su tarea es implementar en python el algoritmo de retropropagación y hacerlo converger con esta base de datos. Su red debe tener solo 3 capas, una capa de entrada de 65 neuronas (las 64 entradas mas la neurona de sesgo que siempre dispara 1), una capa oculta (de tamaño definido a su gusto), y una capa de salida de 10 neuronas, una para cada dígito.

Para esto debe implementar las siguientes funciones:

- `newWeights(n,m)` : genera una nueva matriz de pesos de tamaño nxm que conecta una capa con la siguiente
- `newNetwork(archivo,input,oculta,output)` : genera una nueva red neural con pesos aleatorios que tiene `input` número de neuronas en la primer capa, `oculta` número de neuronas en la capa oculta y `output` número de neuronas en la capa de salida. Esta red se guarda en un archivo de texto cuyo nombre esta en el parámetro `archivo`.
- `readNetwork(archivo)` : carga una red neural que ha sido cuardada en un archivo, el resultado es un par de matrices correspondientes a los pesos entre la capa de input y la capa oculta, y entre la capa oculta y la salida (los pesos incluyen a la neurona de sesgo)
- `saveNetwork(archivo,W1,W2)` guarda una red neural en un archivo de texto, la red se asume que tiene tres capas y por lo tanto esta definida por las dos matrices de pesos entre las capas 1 y 2 y la 2 y 3.
- `forward(X,W1,W2)` : calcula la propagación hacia adelante y deja el resultado en tres las variables `net1, out1, net2, out2`

- `backward(out1,out2,W1,W2)` calcula los valores `delta1` Δ_1 y `delta2` Δ_2 necesarios para la retropropagación
- `update(delta1,delta2,out1,out2,W1,W2)` : actualiza los pesos de la red y retorna los nuevos valores de `W1` y `W2`

Además su programa debe tener dos funciones: una función de entrenamiento y una función de ejecución/prueba. La función de entrenamiento debe estar declarada de la siguiente manera:

```
def train(input, red, output, eta, error, max_iter)
    ...
```

El resultado de su programa debe ser la red entrenada en un archivo, los parámetros significan lo siguiente:

- `input`: nombre del archivo de entrada que contiene los pares de entrenamiento.
- `red`: nombre del archivo que contiene los pesos de la red neural.
- `output`: nombre del archivo de salida donde guarda los pesos de la red entrenada.
- `eta`: valor η que corresponde al ritmo de aprendizaje.
- `error`: error tolerado, cuando su error este por debajo de este monto puede dar por terminado el entrenamiento. El error lo debe calcular con la formula de mínimos cuadrados data anteriormente.
- `max_iter`: número máximo de iteraciones al set de entrenamiento, cuando llega a esta cantidad aborta el entrenamiento de su red.

El formato para el archivo archivo de la red neural consta de dos matrices de pesos, para ello debe dar la dimensión de cada una de las capas, por ejemplo su archivo de salida puede ser:

```
64 30 10

<Pesos entre capa de entrada y capa oculta>
una matriz de tama\˜no 30x65
...
<Pesos entre capa oculta y capa de salida>
una matriz de tama˜no 10x31
```

Note que la neurona de sesgo no se incluye entre la dimensión de las capas, decimos que la primer capa tiene 64 entradas pero la matriz tiene dimensión 30x65, ya que debe contemplar la neurona de sesgo.

La función de prueba se declara de la siguiente manera:

```
def test(red,casos):
```

Donde

- `red`: es el archivo que contiene los pesos de la red.
- `casos`: archivo con igual formato al de entrenamiento

El procedimento de test debe generar un reporte en pantalla de cuantos de los casos en el archivo de prueba fueron clasificados correctamente y cuantos no.