Documento de Trabajo

Aplicaciones de las Redes Neuronales en las Finanzas

Autor: **Prof Dr. Pablo García Estévez**.
Universidad Complutense de Madrid.
Facultad de Ciencias Económicas y Empresariales.

Madrid, abril de 2002

1 Redes Neuronales Supervisadas

1.1 NEURONAS ARTIFICIALES

Las Redes Neuronales Supervisadas (RNS) son un conjunto de algoritmos matemáticos que encuentran las relaciones no lineales entre conjuntos de datos. Suelen ser utilizadas como herramientas para la predicción de tendencias y como clasificadoras de conjuntos de datos.

Se denominan Neuronales porque están basadas en el funcionamiento de una neurona biológica cuando procesa información. Éstas tienen dos partes bien diferenciadas: la soma, con las dendritas y el axón. La neurona biológica toma los datos por las dendritas que están conectadas a otras neuronas o a centros de información nerviosa. Realmente las neuronas no están unidas unas con otras sino que tienen una pequeña distancia entre ellas; la conexión, denominada sinapsis se realiza mediante los llamados neurotransmisores. Cuando los estímulos procedentes de las conexiones sinápticas son lo suficientemente grandes la neurona se activa enviando a través del axón una corriente eléctrica destinada a liberar neurotransmisores hacia otras neuronas. Así que una neurona biológica puede estar estimulada o no estarlo y en función de esto pasará información alterada de algún modo o estará estática.

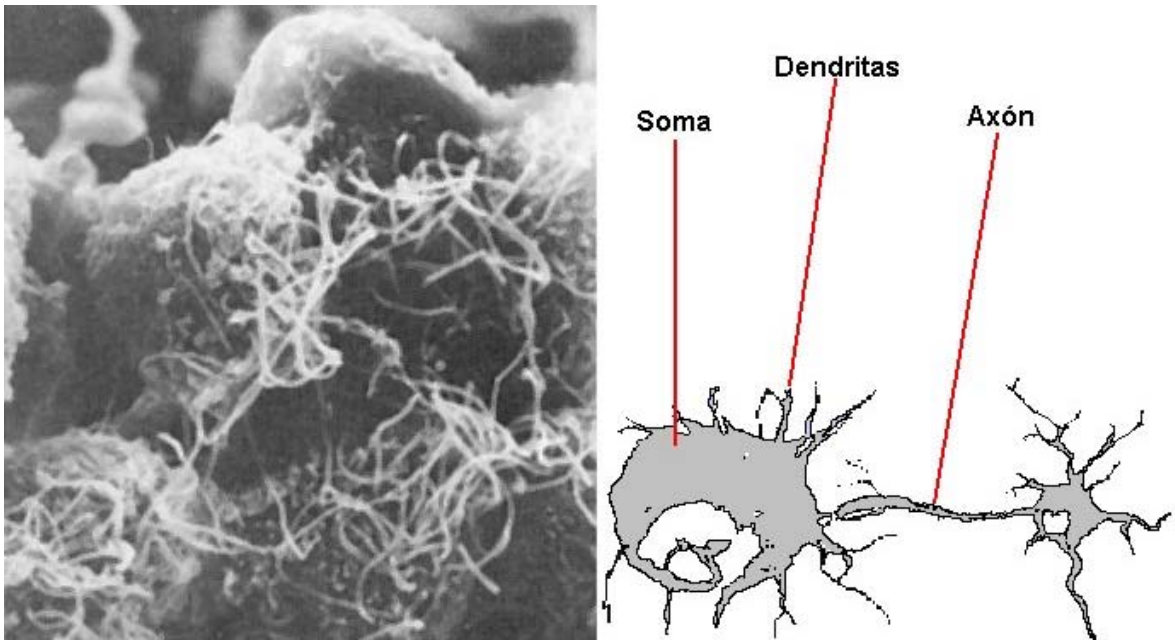


Fig. 1.1 En la parte izquierda se puede observar un cúmulo de neuronas en el cerebro humano. Micrografía ampliada en 15.000 aumentos. En la parte derecha se muestra un esquema de una neurona. (Fuente: Brain Research Institute. UCLA en *SAGAN* 1980)

Desde 1960 varios grupos de investigadores han intentado replicar este esquema de funcionamiento en problemas matemáticos y el resultado ha sido la neurona artificial que está representada en la figura 1.2. La neurona artificial recibe información de diversas fuentes, que están representadas por la letra X; así tenemos desde X_1 hasta X_N. La información no le llega a la neurona en estado puro, sino que es ponderada debido a que un tipo de información tendrá diferente importancia respecto a otro. Por eso a la información X_i se le multiplica por un peso designado por W_i. Lógicamente, si hay N puntos de información, habrá N pesos, uno por cada punto de información.

Al llegar a la neurona todos los datos de las informaciones ponderados por sus pesos se suman. Esto es lo que se denomina la *Regla de Propagación*. Podría adoptar otras fórmulas pero la más habitual es la simple suma de todos las entradas ponderadas que toma el nombre de Potencial sináptico.

$$H_i(t) = \sum_{j=1}^{N} X_j \times W_j$$

Siendo

$H_i(t)$ el potencial sináptico de la neurona i en el momento t.
X_j la entrada de datos procedentes de la fuente de información j.
W_j el peso sináptico asociado a la entrada X_j

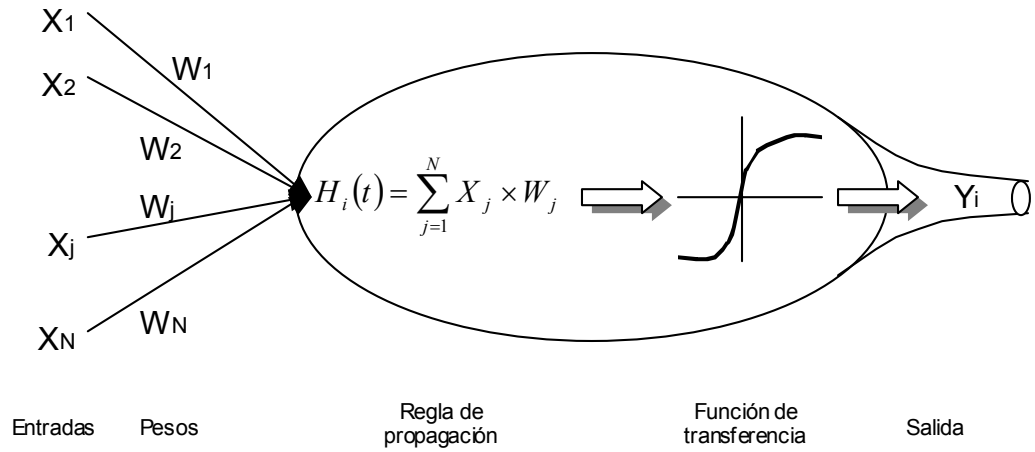


Fig. 1.2 Esquema de una neurona artificial

Cuando el resultado de la regla de propagación supera un cierto número, denominado umbral, entonces la neurona se activa y el número resultante de la regla de propagación se "introduce" en una función denominada *Función de transferencia*. Esquemáticamente se podría representar de la siguiente manera:

$$H_i(t) > \theta \text{ entonces } f[H_i(t)]$$

Se puede escoger diferentes funciones para la función de transferencia Las cinco funciones de transferencia típicas que determinan los distintos tipos de neuronas son:

1. Función escalón
2. Función lineal
3. Función Sigmoidea
4. Función Tangente Sigmoidea
5. Función Gaussiana

Hagamos un breve repaso por cada una de ellas con el objeto de familiarizarnos con ellas y saber que características tienen.

Función escalón.

La función escalón se utiliza cuando la neurona tiene salidas binarias: 0, 1. La neurona se activa cuando el valor del potencial postsináptico es mayor o igual a cierto valor umbral. Por ejemplo:

Sea $f(x) = 1$ cuando $H_i(t) \geq 0$
Y $f(x) = 0$ cuando $H_i(t) < 0$

En este caso el umbral es cero. Cuando la función de propagación supera el valor cero, la función tomará valor uno. En caso contrario la función tomará el valor cero. Esta función está representada en la figura 1.3.

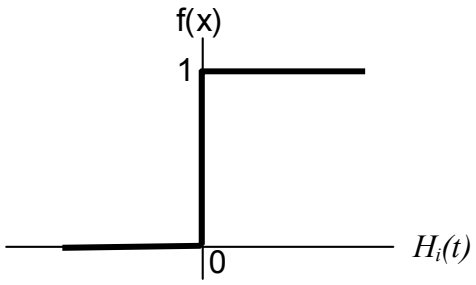


Fig. 1.3 Función escalón

Función lineal

La función lineal o identidad responde a la expresión $f(x) = H_i(t)$ Una variación de la función lineal sería la función *lineal a tramos* donde la salida de la neurona sería la función identidad siempre y cuando el valor del potencial postsináptico estuviese dentro de un rango de valores. Al estar fuera del rango la función se torna constante. Por ejemplo:

Sea $f(x) = 1$ cuando $H_i(t) > 1$
Y $f(x) = H_i(t)$ cuando $-1 \leq H_i(t) \leq 1$
Y $f(x) = -1$ cuando $H_i(t) < -1$

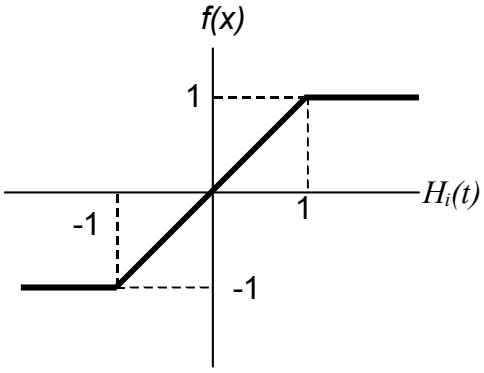


Fig. 1.4 Función lineal a tramos

En la figura 1.4 se aprecia como la función pasa a ser la función identidad a partir de que la función de propagación tome el valor –1 hasta el valor + 1. Fuera de este rango la función se torna constante con un valor de –1 desde - ∞ hasta –1 y con un valor de +1 desde +1 hasta + ∞.

Función Logarítmica Sigmoidea

La salida de esta función siempre será continua en el rango entre cero y uno. Con esta familia de funciones se pueden utilizar datos continuos o digitales proporcionando salidas exclusivamente continuas.

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

Siendo $x = H_i(t)$

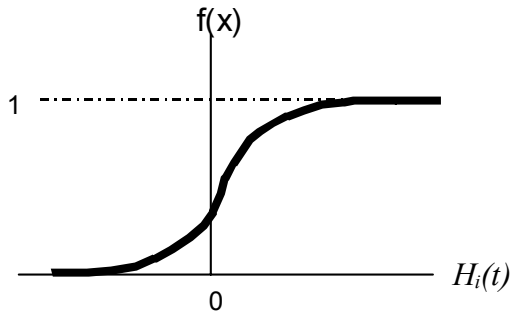


Fig. 1.5 Función logarítmica sigmoidea

En la función representada en la figura 1.5 se observa como la función adopta valores muy próximos a cero cuando X es pequeño, pero que según aumenta el valor en el eje de las abscisas la función pasa a ser creciente. Al principio la pendiente de la función crece hasta llegar a un punto de inflexión, momento en el cual la pendiente comienza a descender hasta llegar a cero, a partir del cual la función vuelve a dar un valor constante e igual a uno.

Función Tangente Sigmoidea

Esta es una de las funciones más utilizadas en las redes neuronales por su flexibilidad y el amplio rango de resultados que ofrece. Las ventajas de utilizar una tangente sigmoidea frente a una sigmoidea reside en que la segunda sólo ofrece resultados en el rango positivo entre cero y uno, en cambio la tangente sigmoidea da resultados entre –1 y 1, por lo que se amplía a los números negativos los posibles resultados. La función tiene una tipología como la siguiente:

Sea $$f(x) = \frac{2}{1+e^{-2x}} - 1$$

Siendo $x = H_i(t)$

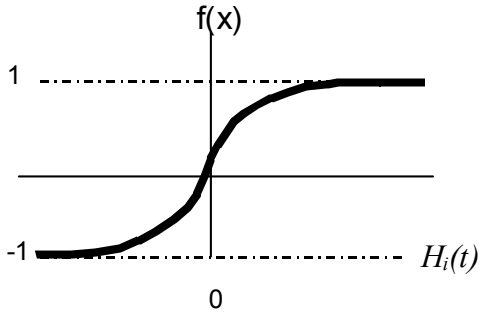


Fig. 1.6 Función Tangente Sigmoidea

El resultado que nos ofrecen cada una de estas funciones será el dato de salida de la neurona que se dirigirá bien a otra neurona, bien al resultado final.

1.2 REDES NEURONALES ARTIFICIALES.

Una neurona no tiene capacidad para realizar un proceso lógico, pero un conjunto de ellas sí. Por tanto, las neuronas se agrupan para realizar trabajos de cálculo lógico en redes. Las redes están compuestas de capas. Generalmente, una red neuronal típica tendrá tres capas: una de entrada de datos, otra oculta donde se procesan los datos y una tercera de salida de los resultados. Cada una de las capas contendrá un número determinado de neuronas en función del diseño que haya decidido el analista y del tipo de trabajo que vaya a realizar la red. Todas las neuronas que contiene una capa se conectan con todas las neuronas de la siguiente capa. De esta manera, cuando una neurona obtiene un resultado, lo envía a todas las neuronas de la capa siguiente. Evidentemente ese resultado será ponderado por cada neurona por el peso sináptico.

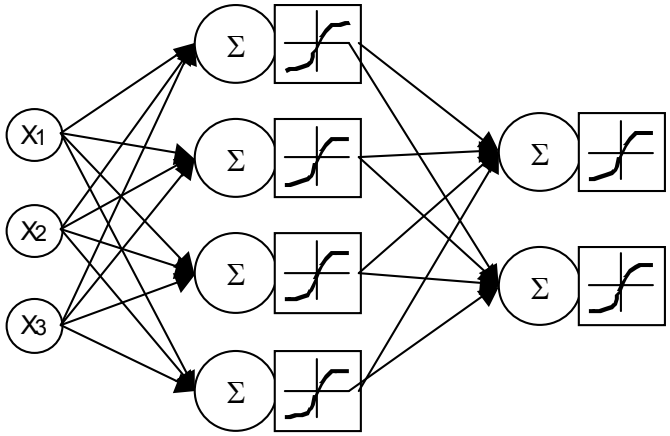


Fig. 1.7 Representación de una red neuronal de tres neuronas de entrada, cuatro ocultas y dos de salida [3-4-2].

En la figura 1.7 se muestra un ejemplo de red neuronal compuesto por tres capas: entrada, oculta y salida. La capa de entrada está compuesta, en este ejemplo, por tres neuronas de entrada. Éstas no hacen ningún proceso, sólo envían información a las neuronas de la capa oculta. Por tanto cada neurona de entrada tendrá tantas conexiones como neuronas tiene la capa oculta, y cada neurona de la capa oculta recibirá tanta información como neuronas tiene la capa de entrada. En nuestro ejemplo, cada neurona de entrada está conectada a las cuatro neuronas de la capa oculta, y cada neurona de la capa oculta recibe tres flujos de información procedente de cada una de las neurona de la capa de entrada. Este flujo de información estará ponderado por un peso sináptico, por tanto al haber tres flujos de información por neurona de la capa oculta y hay cuatro de éstas, habrá doce pesos sinápticos.

Para poder identificar a cada uno de los pesos sinápticos, se les añade un subíndice compuesto de dos números que identifican, por este orden, la neurona de destino (de la capa oculta) y la neurona de procedencia (de la capa de entrada).

$$W_{j,i}$$

Así, por ejemplo, $W_{3,2}$ se refiere al peso sináptico que conecta a la neurona número 2 de la capa de entrada con la neurona número 3 de la capa oculta.

En cada neurona de la capa oculta ocurre un proceso matemático consistente en la suma de las multiplicaciones de cada dato de información por sus pesos ponderados para determinar el potencial sináptico y que será el dato a introducir en la función de transferencia. El resultado de esta función en cada neurona de la capa oculta será su salida que se enviará a cada neurona de la

capa de salida. Por tanto cada neurona de la capa oculta tendrá tantas conexiones como neuronas de salida hay y cada neurona de salida recibirá tantos flujos de información como neuronas existan en la capa oculta. Los flujos de información son ponderados por los pesos que unen las neuronas de la capa oculta con las neuronas de la capa de salida. En las neuronas de la capa de salida se realiza el mismo proceso que hemos visto antes en las neuronas de la capa oculta. Al final, el resultado de la red será la salida de las neuronas de la capa de salida.

Este proceso se puede representar de manera matricial, que facilita la comprensión del sistema; siguiendo el ejemplo de la figura 1.7, la matriz de entradas estará representada por X, mientras que la matriz de los pesos sinápticos viene representada por W y la matriz H representa los potenciales sinápticos de cada neurona de la capa oculta.

$$\begin{pmatrix} X_1 & X_2 & X_3 \end{pmatrix} \times \begin{pmatrix} W_{1,1} & W_{2,1} & W_{3,1} & W_{4,1} \\ W_{1,2} & W_{2,2} & W_{3,2} & W_{4,2} \\ W_{1,3} & W_{2,3} & W_{3,3} & W_{4,3} \end{pmatrix} = \begin{pmatrix} H_1 & H_2 & H_3 & H_4 \end{pmatrix}$$

Cuando alimentamos las funciones de transferencia con los valores de la matriz de los potenciales sinápticos obtenemos la matriz Y de salidas de las neuronas de la capa oculta. Estás serán ponderadas por la matriz W' de pesos sinápticos que unen la capa oculta con la capa de salida. El resultado será la matriz H' de potencial sináptico de la capa de salida que servirá para alimentar las funciones de transferencia de esta capa y que su resultado será la matriz Z de resultado de la red.

$$f\begin{pmatrix} H_1 & H_2 & H_3 & H_4 \end{pmatrix} = \begin{pmatrix} Y_1 & Y_2 & Y_3 & Y_4 \end{pmatrix}$$

$$\begin{pmatrix} Y_1 & Y_2 & Y_3 & Y_4 \end{pmatrix} \times \begin{pmatrix} W'_{1,1} & W'_{2,1} \\ W'_{1,2} & W'_{2,2} \\ W'_{1,3} & W'_{2,3} \\ W'_{1,4} & W'_{2,4} \end{pmatrix} = \begin{pmatrix} H'_1 & H'_2 \end{pmatrix}$$

$$f\begin{pmatrix} H'_1 & H'_2 \end{pmatrix} = \begin{pmatrix} Z_1 & Z_2 \end{pmatrix}$$

1.3 LAS REDES SUPERVISADAS: EL PERCEPTRÓN MULTICAPA Ó MLP

Para que una red neuronal descubra las conexiones no lineales entre dos conjuntos de datos hay que entrenarla. Para esto se le presentan a la red los datos de entrada y los resultados queridos por el analista. La red, utilizando de manera reiterada un algoritmo, denominado de aprendizaje, irá modificando los pesos, (que en el inicio de la red tienen un valor aleatorio) una y otra vez hasta encontrar el conjunto de ellos que consigue que con los datos suministrados obtener los resultados queridos.

Una vez entrenada, se le presentan nuevos datos y se hace un test para comprobar la bondad del conjunto de pesos. Si no es satisfactorio se vuelve a ajustar los pesos. Cuando la red es testada y ofrece un rendimiento óptimo, ya está lista para trabajar. Se le puede introducir datos nuevos y la red ofrecerá los resultados del problema en el que se está trabajando. Este tipo de redes neuronales se les denomina supervisadas, debido a que al introducir los datos queridos, la red, en la fase de entrenamiento, puede calcular el error que comete y modificar los pesos sinápticos con el objetivo de disminuir este error.

La red neuronal supervisada más utilizada es la denominada Perceptrón Multicapa o MLP (*Multi - Layer Perceptron*). Esta es una red de varias capas, usualmente tres (entrada, oculta y salida) que utiliza como función de transferencia en la capa oculta funciones sigmoideas. Las funciones de la capa de salida pueden ser lineales o sigmoideas, dependiendo del tipo de salida que

se quiera. Pero la característica más importante de las MLP es que utiliza como función de aprendizaje la Retropropagación hacía atrás o Regla BP. En la realidad, existen diferentes variantes de la Regla BP que se utilizan en función del problema que se quiera resolver.

La Regla BP puede trabajar de dos maneras: Aprendizaje por lotes o aprendizaje en serie. El aprendizaje por lotes acumula las variaciones de los pesos y al final de cada ciclo, actualiza a la vez todos los pesos. El aprendizaje en serie va actualizando los pesos cada vez que se presenta un dato. Una desventaja del aprendizaje en serie es que se debe respetar el orden de presentación de las entradas, mientras que en el aprendizaje por lotes el orden no tiene importancia. La ventaja del aprendizaje en serie es su velocidad mientras que la ventaja del aprendizaje por lotes es que se puede aplicar a la mayoría de los problemas.

Existen diferentes programas informáticos que permiten manejar redes neuronales de manera sencilla. Uno de los más potentes, y el que aquí vamos a utilizar, el la Toolbox de Redes Neuronales de MatLab. Una de las decisiones que debe realizar el analista a la hora de diseñar una red neuronal es elegir el algoritmo de aprendizaje. El Toolbox de MatLab ofrece diferentes algoritmos que tienen unas características específicas. Es muy difícil elegir el algoritmo adecuado para cada problema. Un algoritmo que es rápido y consigue soluciones satisfactorias para un problema no es eficaz para otro, o se vuelve muy lento. Hay veces que es preferible un algoritmo que trabaje de manera lenta, debido al resultado que se quiere obtener, y otras veces se necesita rapidez y la precisión de los datos no es tan importante. En la siguiente tabla se muestra una lista de los algoritmos que ofrece MatLab ordenador de más lento al más rápido. Generalmente se aconseja elegir el más rápido e ir eligiendo, si no es muy preciso, los siguientes, que aunque menos rápidos pueden darnos mejores precisiones. De todos modos el algoritmo Levenberg – Marquardt es el mejor algoritmo para problemas que no sean muy grandes.

Tabla 1.1 Diferentes algoritmos de aprendizaje del Toolbox de Redes Neuronales del programa MatLab, ordenados de lentos a rápidos.

Algoritmo	***Sentencia MatLab***
Variable (*Variable learning rate*)	traingdx
Rprop	trainrp
Gradiente conjugado escalar (*Scaled Conjugated. Gradient*)	trainscg
Fletcher Powell CG	traincgf
Polak Ribiére CG	traincgp
Powell Beale CG	traincgb
Secante de un paso (*One Step Secant*)	trainoss
Quasi Newton BFGS	trainbfg
Levenberg Marquardt	trainlm

1.4 CÁLCULO DEL ALGORITMO DE LA NORMAL MEDIANTE UNA MLP

Para ilustrar el proceso de construcción de una red neuronal MLP vamos a desarrollar una red neuronal que nos calcule un algoritmo matemático que nos permita de manera sencilla calcular el resultado de una distribución normal de media cero y desviación típica uno. Además, vamos a comparar el resultado con la aproximación de Hasting, utilizada por algunos autores para realizar cálculos sin utilizar las tablas estadísticas.

La ecuación que explica la distribución normal estándar es la siguiente:

$$\Phi(x) = \int_{-\infty}^{t} \frac{1}{\sqrt{2 \times \pi}} \times e^{-\frac{x^2}{2}} dx$$

Ecuación 1

Vamos a trabajar con el programa MATLAB y su Toolbox de Redes Neuronales. Para entrenar la red vamos a construir dos conjuntos de datos: uno que tenga una serie de números entre –10 y 10 con una variación de 0,25. Es decir

[-10; -9,75; -9,50;....;+9,75;+10]

Calculando el resultado de una distribución normal estándar para cada uno de estos datos, obtenemos el conjunto de los resultados deseados. Para hacer este calculo, podemos en una hoja de cálculo Excel escribir la siguiente función:

= DISTR.NORM.ESTAND(*Celda*)

Grabamos cada uno de estos dos conjuntos en dos ficheros independientes con extensión DAT. Por ejemplo, el conjunto de datos lo llamaremos:

Pdesnorm.DAT

Y el conjunto de resultados deseados lo llamaremos:

Tdesnorm.DAT

Estos ficheros, los podrá encontrar el lector en el disco adjunto. Para cargar los datos al MatLab desde el disco duro se debe teclear el siguiente código, teniendo en cuenta que los datos están en C:

```
Dlmread c:/Pdesnorm.dat;
p = ans;
Dlmread c:/Tdesnom.dat;
t = ans;
```

De esta manera, todos los datos están en la matriz llamada p y todos los resultados deseados están en la matriz llamada t.

Para realizar este trabajo voy a utilizar una red que tenga tres capas: una de entrada, una oculta y una de salida. En la capa de entrada sólo pondremos una neurona ya que sólo tenemos un dato, que es el que corresponde al conjunto de la matriz p. La neurona oculta tendrá tres neuronas con función de transferencia tangente sigmoidea. Y por último la capa de salida tendrá una sola neurona con función de transferencia logaritmo sigmoidea, que será la respuesta de la red y buscará coincidir con el resultado de la distribución normal estándar. He decidido que la última función sea logaritmo sigmoidea debido a que los resultados de esta función están en el rango entre cero y uno, igual que el resultado de una distribución normal estándar. Reflejamos en la siguiente figura la red utilizada:

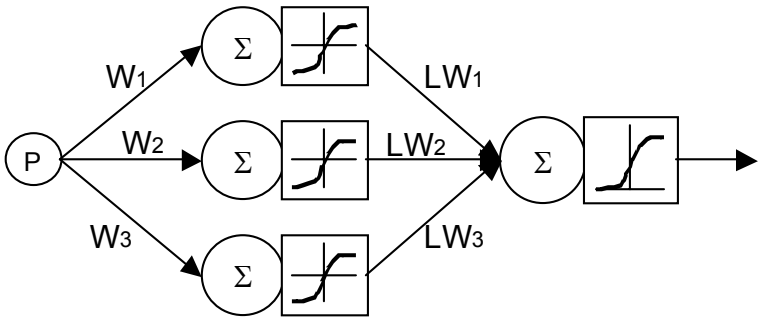
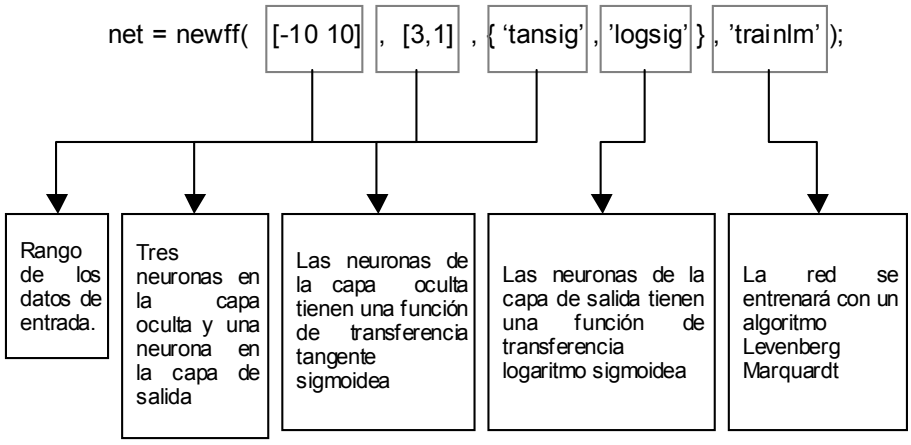


Fig. 1.8 Red Neuronal MLP [1 3 1]

Para crear esta red en el MatLab se debe escribir la siguientes sentencia:

```
net = newff([-10 10],[3,1],{'tansig','logsig'},'trainlm');
```

Con esta sentencia estamos creando una variable llamada net que contendrá nuestra red neuronal. Dentro de la sentencia hemos escrito todas las características de nuestra red.



Hay que hacer notar que si la red tuviera más neuronas en la capa de entrada tendríamos que escribir el rango de los datos de cada una de las neuronas que la compusiesen.

Ya tenemos creada la red neuronal y tenemos cargados los datos en el MatLab. Ahora sólo nos queda entrenarla. Pero primero vamos a escribir tres sentencias para configurar este entrenamiento:

```
net.trainParam.show = 50;
net.trainParam.epochs = 3000;
net.trainParam.goal = 1e-10;
```

- Con la primera sentencia establecemos que nos muestre los resultados del entrenamiento cada 50 ciclos.
- Con la segunda sentencia establecemos 3.000 ciclos de entrenamiento.
- Con la tercera sentencia marcamos un objetivo. El error de los resultados obtenidos por la red y los resultados deseados deberá tender a 1×10^{-10}.

Ahora estamos preparados para escribir la sentencia para iniciar el entrenamiento de la red:

```
net = train(net, p, t);
```

Estamos entrenando la variable net, con las matrices de datos de entrada p y con la matriz de resultados deseados t. La red comienza a realizar ciclos y a modificar pesos sinápticos y nos

mostrará cada 50 ciclos el resultado. El entrenamiento parará bien cuando se llegue al final de los ciclos, bien cuando se consiga que el error entre los datos obtenidos por la red llegue al objetivo conseguido.

En la siguiente figura se muestra el gráfico del error que nos muestra el programa cada 50 ciclos. Como se observa a partir de los 500 ciclos se estabiliza el error muy cerca del objetivo, aunque no se ha conseguido. El entrenamiento se terminó porque se llegó al final de los ciclos establecidos. No hubiese servido de nada el haber puesto muchos más ciclos ya que el error está muy estabilizado.

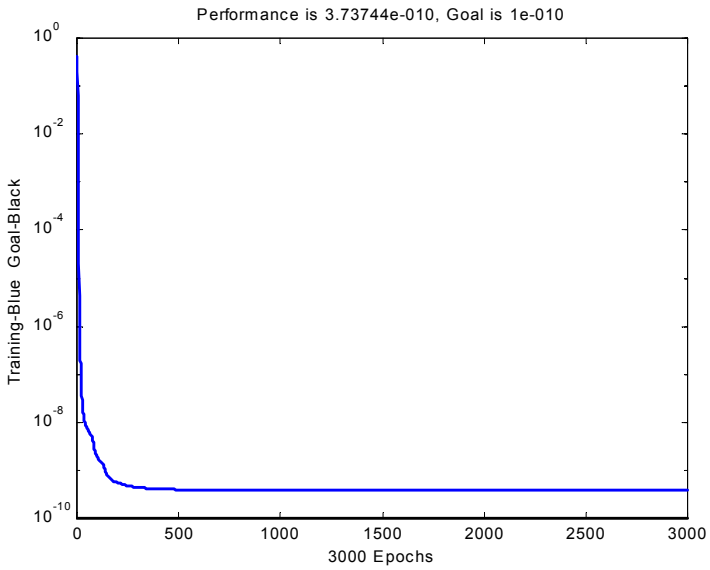


Fig. 1.9 Gráfico del error en el entrenamiento de la red

Para saber si los resultados obtenidos son los deseados vamos a calcular el coeficiente de correlación entre los resultados de la red y los deseados. Primero hagamos que la red calcule los resultados con los datos de entrada.

```
Res = sim(net, p);
```

En la variable Res recogemos los resultados de la red. Ahora calculamos el coeficiente de correlación entre la variable Res y la variable de resultados deseados t.

```
corrcoef(Res, t)
```

Y el programa nos devuelve la matriz de correlaciones entre las dos matrices:

```
ans =
     1     1
     1     1
```

La correlación es perfecta. Los resultados son óptimos. En el disco adjunto se muestran los resultados y el lector verá que los datos obtenidos por la red coinciden completamente con los datos

deseados. Es decir, hemos conseguido que la red calcule la distribución normal estándar. Para deducir el algoritmo, primero obtenemos los pesos sinápticos de la red.

```
W = net.IW{1,1};
LW = net.LW{2,1};
B = net.b{1};
LB=net.b{2};
```

Donde W es la matriz de los pesos que unen la capa de entrada y la capa oculta.
LW es la matriz de los pesos que unen la capa oculta y la capa de salida
B es la matriz de los umbrales de la capa oculta
LB es la matriz de los umbrales de la capa de salida

Y obtenemos los pesos y umbrales, que se muestran en la siguiente tabla:

Tabla 1.2 Pesos y umbrales de la red

W	**B**
-0.3297 0.9368 0.2999	1.3914 -6.7991 1.4651
LW	**LB**
-10.5728 156.8703 14.3815	153.2871

Con estos pesos podemos deducir el algoritmo que calcula, de manera alternativa la distribución normal estándar. La ecuación resultante es la siguiente:

$$t = \frac{-21{,}1456}{1+e^{D_1}} + \frac{313{,}7406}{1+e^{D_2}} + \frac{28{,}763}{1+e^{D_3}} - 7{,}3919$$

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

Siendo:

$D_1 = 0{,}6594 \cdot x - 2{,}7828$
$D_2 = -1{,}8736 \cdot x + 13{,}5982$
$D_3 = -0{,}5998 \cdot x - 2{,}9302$

Comparemos este algoritmo con la aproximación de Hasting que adopta un forma con la siguiente:

$$t = \frac{1}{1+P\cdot x}$$

$$\Phi(x) = 1 - \left(\frac{1}{\sqrt{2\times\pi}} \times e^{-\frac{x^2}{2}}\right) \times \left(C_1 t + C_2 t^2 + C_3 t^3 + C_4 t^4 + C_5 t^5\right)$$

Siendo

P = 0,2316419
C_1 = 0,31938153
C_2 = – 0,356563782
C_3 = 1,78147937
C_4 = – 1,821255978
C_5 = 1,330274429

Para realizar esta comparación compararemos la respuesta de la red y del algoritmo de Hasting con los datos reales de la distribución normal estándar y calcularemos el coeficiente R^2 de los resultados con los datos reales. Éste cálculo se puede observar en la siguiente tabla

Tabla 1.3 Prestaciones de la red neuronal en comparación con la aproximación de Hasting para el cálculo de la distribución normal estándar.

	Hasting	Red Neuronal
Para un rango –10 a 10	0,014	1
Para un rango –3 a 6	0,9616	1
Para un rango –1 a 1	1	1

Como se puede observar el algoritmo derivado de la red neuronal realiza una estimación perfecta de la distribución normal estándar mientras que el algoritmo de Hasting es un buen estimador sólo para un rango entre –1 y 1 ya que fuera de este rango, dicho algoritmo tiene imperfecciones. En la siguiente

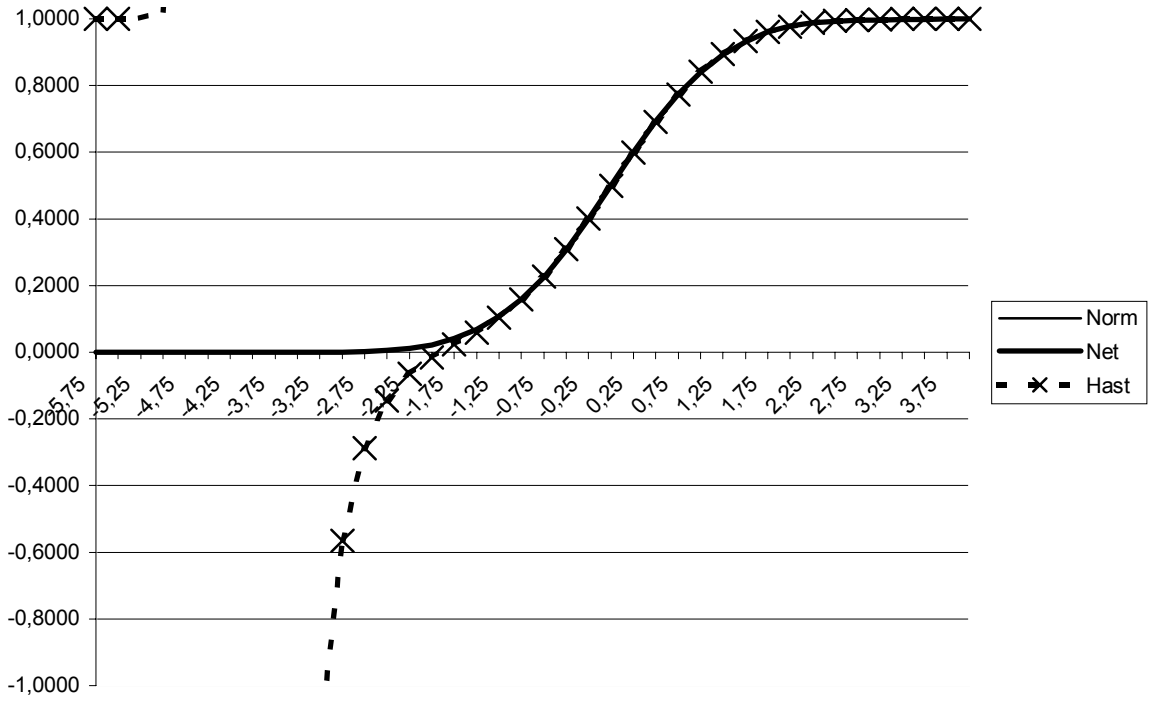


Fig. 1.10 Comparación de los resultados de la estimación de una distribución normal estándar mediante red neuronal y la aproximación de Hasting.

Como se puede observar en la figura 1.10 a partir del valor -1, aproximadamente, la aproximación de Hasting diverge de los resultados reales. En los valores más pequeños esta aproximación da un valor constante de 1 cuando la distribución normal estándar devuelve como valor el cero. Este problema es importante ya que desvirtúa los cálculos. Por ejemplo, no se podría utilizar la aproximación de Hasting para el cálculo del precio de una opción mediante la ecuación Black Scholes ya que estaríamos expuestos a obtener valores erróneos. Sin embargo, si que podemos utilizar con toda confianza el algoritmo obtenido por las redes neuronales porque replica exactamente la función que queremos calcular.

1.5 CONSTRUCCIÓN DE UN ALGORITMO PARA CALCULAR EL PRECIO DE LAS OPCIONES.

1.5.1 Objetivo del trabajo.

Ahora, en este ejemplo, pretendemos conseguir un algoritmo mediante una red neuronal supervisada que calcule el precio de una opción de compra de la compañía Telefónica utilizando datos históricos del mercado. El algoritmo conseguido se trasladará a una hoja de cálculo Microsoft Excel para facilitar la utilización del mismo.

Los datos utilizados proceden de MEFF y de los ficheros históricos de la Bolsa de Madrid. Para construir el algoritmo se han empleado seis variables mas los valores de las primas de las opciones de compra sobre Telefónica que se utilizarán como objetivos que el modelo deberá alcanzar. Las variables utilizadas son:

1. Precio de cierre de Telefónica.
2. Precio de ejercicio
3. Volumen
4. Open Interest
5. Volatilidad
6. Tiempo hasta el vencimiento.

7. Primas de las Call de Telefónica

Cada día de negocio hay varias referencias de contratos de opciones de compra sobre Telefónica en función de los precios de ejercicio y el vencimiento del contrato. Por ello, el número de datos es muy elevado. Para mostrar este ejemplo, se han utilizado sólo los datos del mercado de las opciones call de Telefónica desde 3 de enero de 2000 hasta 30 de junio de 2000, eligiendo, sólo, las referencias diarias qua han tenido movimiento. De éstas, se ha utilizado el último precio cruzado en el día, que no coincide siempre con el valor teórico calculado con la ecuación Black Scholes ya que depende de la oferta y demanda que en ese momento tenga esa referencia. El número referencias utilizadas es de 2.384 contratos. Teniendo en cuenta que el modelo utilizará seis variables mas un dato que hará la función de objetivo a conseguir, el número de datos utilizados ascenderá a 16.688:

$$2.384\ contratos \times 7\ variables = 16.688\ datos.$$

1.5.2 Mecánica del modelo.

Para no crear un modelo sesgado por el orden cronológico procedemos a ordenar de manera aleatoria la muestra para deshacer cualquier tendencia temporal que pudiera existir. Dividimos el conjunto de datos en seis subconjuntos:

1. Datos para el Entrenamiento: 1.192 contratos × 6 variables = 7.152 datos
2. Objetivo del Entrenamiento: 1.192 precios.
3. Datos para la Validación: 596 contratos × 6 variable = 3.576 datos.
4. Objetivo de la Validación: 596 precios.
5. Datos del Test: 596 contratos × 6 variables = 3.576 datos.
6. Objetivo del Test = 596 precios.

Con los datos del entrenamiento realizaremos las iteraciones necesarias en la red para descubrir la relación entre las seis variables (1) y los precios de las opciones (2). Los datos para la validación (3) y sus objetivos (4) servirán para parar el proceso iterativo cuando la red empiece a distanciarse de un generalización, ya que si no se parase la red realizaría mal las predicciones. Y por último, los datos del Test (5) servirán para alimentar el modelo final y se comparará la respuesta obtenida con los datos reales (6)

Se normalizan las seis variables utilizadas por la red a un rango entre –1 y 1 mediante los máximos y mínimos valores de cada variable.

Creamos una red neuronal de seis neuronas de entrada, cuatro ocultas y una de salida. Las funciones de transferencia de las neuronas ocultas son del tipo tangente sigmoideas mientras que la función de transferencia de la neurona de la capa de salida es del tipo entidad. El algoritmo de aprendizaje es el Levenberg – Marquardt con parada temprana.

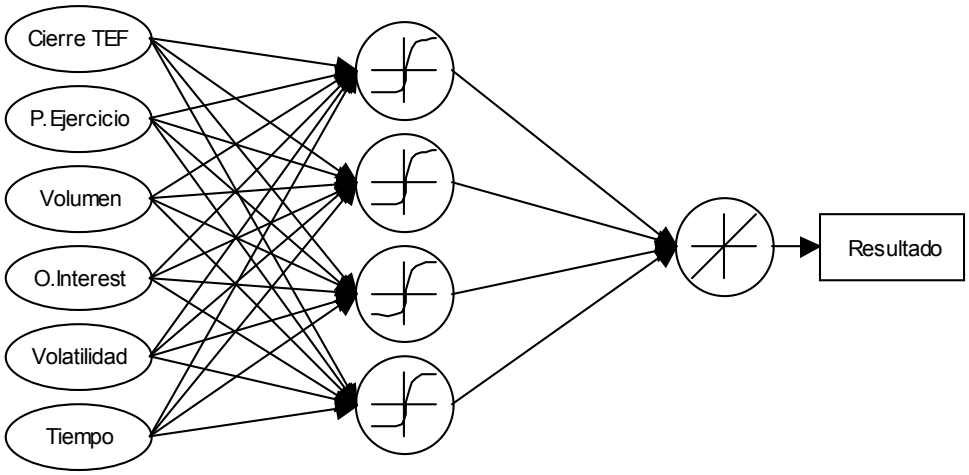


Fig. 1.11 Esquema de la red neuronal utilizada en este ejemplo

Todos estos pasos se realizarían en MatLab escribiendo las siguientes que se detallan más adelante.

Siendo:

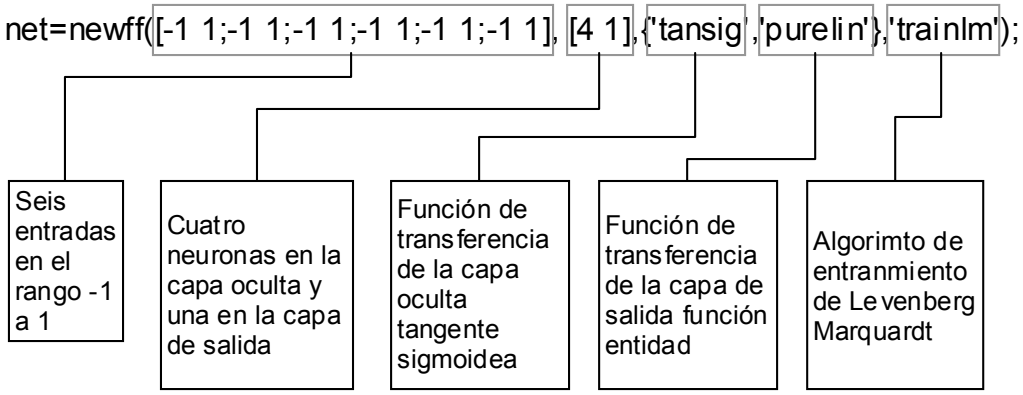
todosp	Conjunto de todos los datos para calcular mínimos y máximos
maxtodosp	Máximo del conjunto de todos los datos
mintodosp	Mínimo del conjunto de todos los datos
todost	Conjunto de todos los objetivos para calcular mínimos y máximos
maxtodost	Máximo del conjunto de todos los objetivos
mintodost	Mínimo del conjunto de todos los objetivos
trp	Conjunto de datos sin normalizar para entrenar.
trpn	Conjunto de datos destinado para entrenar normalizado en un rango –1 a 1.
vp	Conjunto de datos sin normalizar para validar.
vpn	Conjunto de datos destinado para validar normalizado en un rango –1 a 1
vt	Conjunto de objetivos sin normalizar para validar .
tsp	Conjunto de datos sin normalizar para test.
tspn	Conjunto de datos destinado para test normalizado en un rango –1 a 1
tst	Conjunto de objetivos sin normalizar para el test.

todosp=todosp';	Utilizamos la matriz traspuesta de los datos para operar
[todospn,mintodosp,maxtodosp]=premnmx(todosp);	Establecemos el conjunto de mínimos y de máximos para la normalización de las entradas en el rango –1 +1
trp=trp';	Utilizamos la matriz traspuesta del conjunto destinado al entrenamiento
trpn=tramnmx(trp,mintodosp,maxtodosp);	Normalizamos el conjunto de datos destinado al entrenamiento en el rango –1 +1

trt=trt';	Utilizamos la matriz traspuesta del conjunto de objetivos para el entrenamiento.
vp=vp';	Utilizamos la matriz traspuesta del conjunto de datos para la validación.
vpn=tramnmx(vp,mintodosp,maxtodosp);	Normalizamos el conjunto de datos para la validación en el rango –1 +1
vt=vt';	Utilizamos la matriz traspuesta del conjunto de objetivos para la validación.
v.P=vpn;	Establecemos el conjunto vpn como conjunto de datos para la validación
v.T=vt;	Establecemos el conjunto vt como conjunto de objetivos para la validación

Ahora tenemos preparados todos los datos que utilizaremos en la red neuronal, por lo que se procede a la construcción de la misma. Recordamos que ésta es una red de seis entradas, cada una de ellas con un rango entre –1 y 1, con cuatro neuronas en la capa oculta y una neurona de salida, es decir una red del tipo [6 4 1] La función de transferencia de la capa oculta es tangente sigmoidea (*tansig*) mientras que la función de transferencia de la capa de salida es la función entidad (*purelin*). El algoritmo de aprendizaje será del tipo Levenberg Marquardt (*trainlm*). El código sería el siguiente:

```
net=newff([-1 1;-1 1;-1 1;-1 1;-1 1;-1 1],[4 1],{'tansig','purelin'},'trainlm');
```



Ahora solo nos queda reiniciar los pesos sinápticos de la red y entrenarla.

net=init(net);	Inicializamos los pesos.
[net,tr]=train(net,trpn,trt,[],[],v);	Ordenamos al MatLab que entrene la red y realice la validación.

Al escribir estas órdenes la red comienza mediante iteraciones a buscar la combinación de pesos que optimiza el modelo. Además, realiza al mismo tiempo una simulación con los datos de la validación para comprobar la bondad del modelo. El entrenamiento se parará cuando el error de validación se dispare alejándose del error de entrenamiento. Si no fuese así, la red tendría un sobre aprendizaje. Es decir, la red se adaptaría a los datos de entrenamiento pero sería incapaz de hacer una predicción fuera de los datos de entrenamiento; dejaría de generalizar para especializarse en la muestra dada. En este caso no nos interesa que nos descubra el algoritmo que explica los datos de entrenamiento sino que nos descubra un algoritmo que sea capaz de predecir en cualquier momento.

Efectivamente la red se paró después de la decimonovena iteración:

```
TRAINLM, Epoch 0/100, MSE 10.4857/0, Gradient 5399.07/1e-010
TRAINLM, Epoch 19/100, MSE 0.167355/0, Gradient 270.434/1e-010
TRAINLM, Validation stop.
```

A continuación se alimenta la red con los datos reservados para realizar el test. El objeto de esto es comparar la respuesta de la red con los datos del test con los últimos precios cruzados que realmente hubo en el mercado para cada una de esas referencias. Para comprobar el grado de predicción del modelo utilizamos el coeficiente de correlación. Este coeficiente entre los resultados obtenidos en el Test y los resultados reales es del 98%, un valor muy aceptable teniendo en cuenta el tamaño pequeño de la muestra utilizado.

`tspn=tramnmx(tsp,mintodosp,maxtodosp);`	Normalizamos los datos que vamos a utilizar para el test en el rango –1 a 1
`rdo=sim(net,tspn);`	En el conjunto rdo calculamos la respuesta que da la red con los datos tspn
`[m,b,r]=postreg(rdo,tst)`	Comparamos las respuestas en el conjunto rdo con el conjunto de objetivos del test tst

La red nos ofrecerá la siguiente respuesta:

```
m =
   0.9429
b =
   0.1448
r =
   0.9803
```

Los parámetros m y b son, respectivamente el origen y la pendiente de la recta que mejor se ajusta a los resultados obtenidos. El coeficiente r nos muestra el coeficiente de correlación serial, que en este caso alcanza a 0,9803.

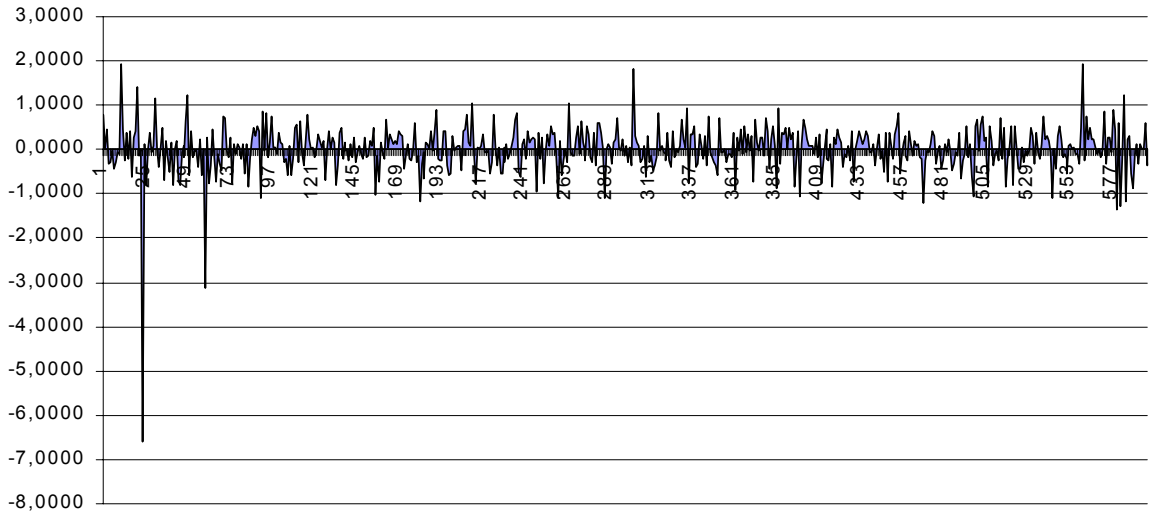


Fig. 1.12 Representación de los errores de la simulación.

En la figura anterior se muestran la representación de los errores de la estimación de las primas; éstos se calculan restando el valor real con el valor estimado por la red. Como se puede observar la mayoría de los errores están dentro de la banda ±1 euro. La media de los errores es de 0,0105 euros. La desviación típica asciende a 0,5306 euros. El máximo error cometido de predicción de precio es de 6,58 euros.

1.5.3 Aplicación en Excel

Para poder manejar de manera sencilla el modelo se obtienen los pesos sinápticos de la red y se traspasan a una hoja de cálculo tipo Excel con el objeto de calcular el precio de la opción. Para ello escribimos las siguientes sentencias para obtener las matrices con los pesos sinápticos de la red.

W=net.IW{1,1}	En la matriz W se sitúan los pesos sinápticos que unen la capa de entrada con la capa oculta de la red.
umbral=net.b{1}	En la matriz umbral situamos el valor de los umbrales de cada una de las neuronas de la capa oculta.
LW=net.LW{2,1}	En la matriz LW situamos a los pesos sinápticos que unen la capa oculta con la capa de salida
Lumbral=net.b{2}	En la matriz Lumbral establecemos el valor del único umbral que tiene la neurona situada en la capa de salida.

Llevamos estas matrices de pesos a una hoja de cálculo del tipo Microsoft Excel. Para ello se puede hacer uso del Toolbox llamado Excel Link que permite el traspaso de datos entre el MatLab y la Excel mediante un botón. Una vez realizado esto, ordenamos los pesos y los umbrales de la siguiente manera:

Tabla 1.4 Modelo neuronal en una hoja de cálculo Excel

	A	B	C	D	E	F	G	H	I
1	**Variables**		**Mín**	**Máx**		**Pesos**	**Oculta**		
2	**Cierre TEF**	25,113	19,3657	31,3311	-0,039346783	0,538165499	-2,44316488	0,156266482	2,663643599
3	**P. Ejercicio**	22,87	12,09	46	-0,364199351	-1,224177039	-1,654930615	-3,478016741	-9,134046489
4	**Volumen**	53	1	32987	-0,996847147	-0,045013662	-4,290328994	-2,593317133	1,118302342
5	**O. Interest**	21163	0	92205	-0,540957649	0,027180322	-2,756704538	6,410952035	-0,009625371
6	**Volatilidad**	48,5	0,02	63,6	0,525007864	0,160196812	2,003717387	-3,003431648	-0,60623487
7	**Tiempo**	0,19178	0	0,980	-0,608938547	0,360546698	-1,270512929	8,70043523	-1,886655999
8						0,319391543	8,292549643	-6,497225521	2,942817906
9				**Umbrales**	**Oculta**	-0,877020415	5,32142879	-1,099281432	-4,305271033
10						-0,557628872	13,61397843	-7,596506953	-1,362453127
11						-0,506216035	1	-0,999999496	-0,876960812
12				**Pesos**	**Salida**	9,714313789	1,602725107	-0,395070728	1,074670143
13						-3,862189377			
14				**Umbral**	**Salida**	7,787316956			
15					**Respuesta**	**3,925**			

El analista escribirá los datos en la columna B. Estos datos se normalizan a un rango –1 +1 con los mínimos y máximos que están en las columnas C y D. La matriz obtenida (E2:E7) se multiplica por la matriz de pesos sinápticos (F2:I7). El producto resultante (F8:I7) se le suma la matriz de los umbrales (F9:I9) y su resultado (F10:I10) se modifica mediante una función tangente sigmoidea:

$$\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

El resultado se muestra en la matriz (F11:I11). Esta es multiplicada por la matriz de pesos de salida (F12:I12) y el resultado de este producto (F13) se le suma el umbral de la capa de salida (F14) mostrando la hoja de cálculo la respuesta en F15.

1.5.4 Conclusión.

Este ejemplo sirve para comprobar el potencial de las redes neuronales supervisadas. La red ha sido capaz de calcular los precios de 596 contratos diferentes puestos al azar con una precisión del 98%. Para realizar una generalización superior se debería utilizar una cantidad de datos mayor, con el consiguiente problemática del proceso de datos.

La red neuronal consigue encontrar, si existe, la relación no lineal que existe entre conjuntos de datos, por lo que los analistas pueden utilizarla para encontrar relaciones no estudiadas y realizar predicciones alternativas a las del mercado. En este caso el cálculo del precio de la opción viene dado por las relaciones que la red encuentra en las variables típicas utilizadas por el algoritmo Black Scholes mas el volumen de contratos y el Open interest. Una ventaja de este modelo sobre Black Scholes es la no utilización de la distribución Normal estándar. Y es ventaja no sólo por los requerimientos cuantitativos a la hora del cálculo, sino por que algunos analistas pueden llegar a cuestionar que la evolución de los precios de las acciones se distribuyan mediante esa ley estadística.

1.6 APÉNDICE: LAS REGLAS DE APRENDIZAJE[1].

1.6.1 La regla LMS

También denominada Regla de Widrow–Hoff o LMS. En un caso particular, que veremos más adelante, se denomina regla Delta. Esta regla es muy importante, ya que es la base de la mayoría de los algoritmos de aprendizaje de un amplio conjunto de redes neuronales. Básicamente, la metodología para deducir una regla de aprendizaje de una red neuronal se resumen dos puntos:

1. Definición de la función de error, también denominada función de coste: mide la bondad del modelo. Cuanto más pequeña sea, más eficiente es el modelo. Lógicamente depende de los pesos sinápticos, que son las incógnitas a resolver en el entrenamiento.
2. Optimización de la función de error. Se busca un conjunto de pesos sinápticos que minimice la función de error. Esta búsqueda se realizará mediante un proceso iterativo denominado *Descenso por el Gradiente*.

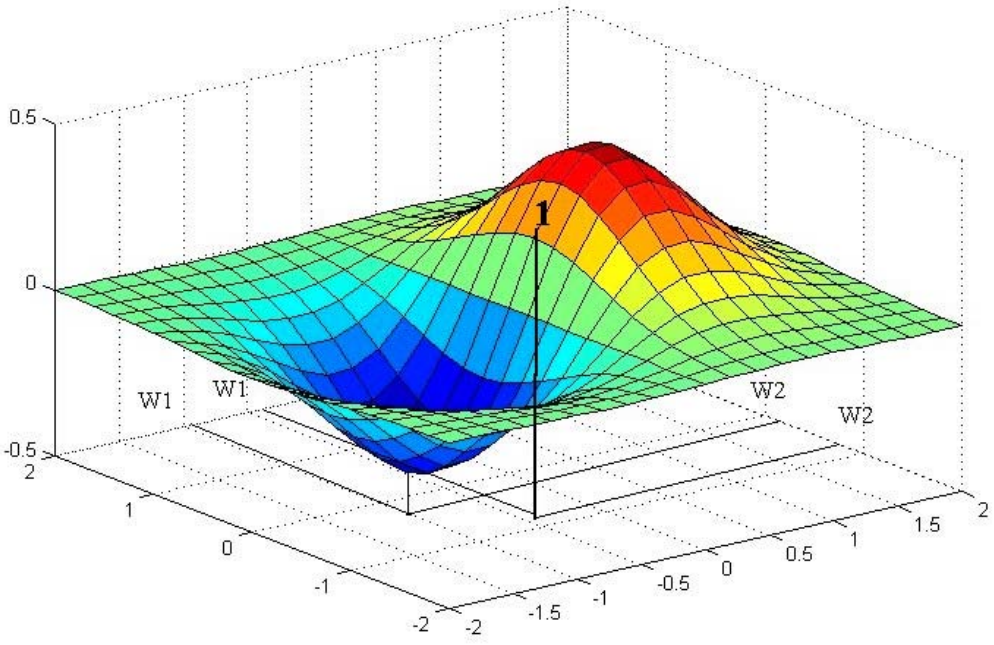


Fig. 1.13

Podemos realizar una aproximación gráfica del descenso por el gradiente observando la figura 8: El punto uno está representado por un conjunto de pesos (en nuestro caso y para simplificar, dos pesos: W_1 y W_2). Se calcula el sentido de la máxima variación de la función de error y se toma el camino opuesto. Esta máxima variación viene dada por el Gradiente de la función de error en el punto uno. Lógicamente, al tomar la dirección contraria del gradiente apuntaremos hacía un mínimo, que seguramente será local. El proceso se itera hasta que se alcance ese mínimo.

Dicho con otras palabras: la variación de los pesos depende de su gradiente, pero además, a éste se le multiplicará por un número infinitesimal para que la variación sea pequeña, ya que de

---

[1] *El desarrollo de esta parte está basada en Martín del Brío, B y Sanz de Molina, A.* Redes Neuronales y Sistemas Borrosos*. Ed. RA-MA. Madrid 1997. Pag. 59 – 62 y 66 – 67*

otro modo, si ésta fuese grande tenemos el riesgo de que en una variación nos pasemos del mínimo y volvamos a estar en una colina. Por tanto la el conjunto de pesos en el momento t + 1 será igual al valor de esos pesos en el momento t menos el gradiente por un número infinitesimal que se llama ratio de aprendizaje y nos definirá el tamaño de cada iteración.

$$W(t+1) = W(t) - \alpha \nabla E(W)$$

$\nabla E(W)$ es el gradiente de la función de coste o función de error.

α es el coeficiente de aprendizaje

Hay que llegar a encontrar cierto equilibrio en la definición del coeficiente de aprendizaje ya que si es muy pequeño, el proceso se hace eterno, mientras que si es muy grande no encontraremos jamás el mínimo porque nos dedicamos a oscilar de una pared a otra.

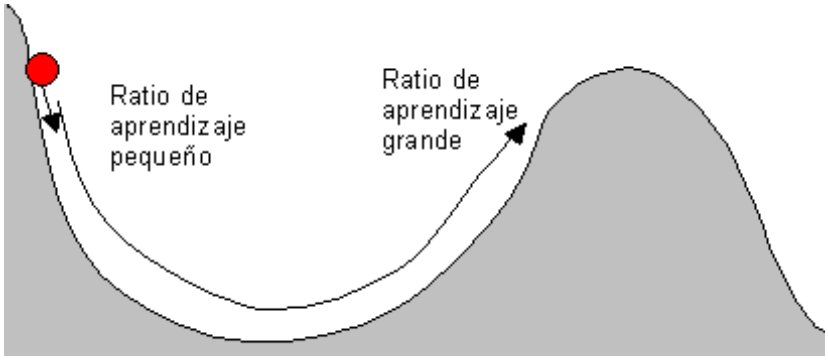


Fig. 1.14 Comparación entre un aprendizaje pequeño y un aprendizaje de gran tamaño

Si tenemos una red neuronal compuesta exclusivamente por una capa de neuronas cuyas funciones de transferencia son la entidad[2], la salida de la neurona i sería la suma de los pesos por las entradas menos el umbral de disparo:

$$y_i = \sum_{j=1}^{N} W_{ij} \cdot x_j - \theta_i$$

El error vendrá dado por la diferencia entre el resultado de la red y el objetivo que buscamos. En una muestra finita el error viene dado por:

$$E[W_{ij}] = \frac{1}{2} \cdot \sum_{\mu=1}^{p} \sum_{i=1}^{n} \left(t_i^{\mu} - y_i^{\mu} \right)^2$$

Se deberá buscar el conjunto de pesos W^*_{ij} que minimice este error. Para resolver este problema de minimización se recurre al descenso por el gradiente. Habíamos comentado más arriba que la variación de los pesos sinápticos es igual al gradiente del error por el ratio de aprendizaje. El gradiente del error no es más que la cuantificación de cuanto varía el error cuando variamos los pesos, es decir, la derivada del error respecto a los pesos sinápticos:

$$\Delta W_{ij} = -\alpha \cdot \nabla E(W_{ij}) = -\alpha \cdot \frac{\delta E(W_{ij})}{\delta W_{ij}}$$

[2] *Estas neuronas con función de transferencia la entidad se denominan Adalinas.*

$$\frac{\delta E\lfloor W_{ij}\rfloor}{\delta W_{ij}} = -\frac{1}{2}\cdot 2\sum_{\mu=1}^{p}\left(t_i^{\mu} - y_i^{\mu}\right)^2 \cdot \frac{dy_i^{\mu}}{dW_{ij}} = -\sum_{\mu=1}^{p}\left(t_i^{\mu} - y_i^{\mu}\right)\cdot x_i^{\mu}$$

Por lo tanto la variación de los pesos quedará como:

$$\Delta W_{ij} = -\alpha \cdot \sum_{\mu=1}^{p}\left(t_i^{\mu} - y_i^{\mu}\right)\cdot x_i^{\mu}$$

Siendo α el ritmo de aprendizaje

En definitiva el nuevo peso se calculará como:

$$\boxed{W_{ij}(t+1) = W_{ij}(t) - \alpha\cdot(t_i - y_i)\cdot x_j}$$

Y ésta es la regla LMS o de Widrow – Hoff.. Si la deducimos en neuronas que tengan como función de transferencia una función sigmoidea, la regla LMS se denomina regla Delta. Por lo tanto, al calcular el gradiente como la derivada de la función de error respecto a los pesos sinápticos, hay que tener en cuenta que la respuesta de la neurona está influida por la función sigmoidea utilizada.

1.6.2 La regla BP

Las redes neuronales MLP utilizan como algoritmo de aprendizaje la regla BP, siglas de *Retropropagación hacía atrás* (Back Propagation). Básicamente, la BP es la aplicación de la regla Delta en una MLP. Si nos fijamos en la figura 10, está representada una MLP del tipo [n m r] con objetivos desde t_1 hasta t_p.

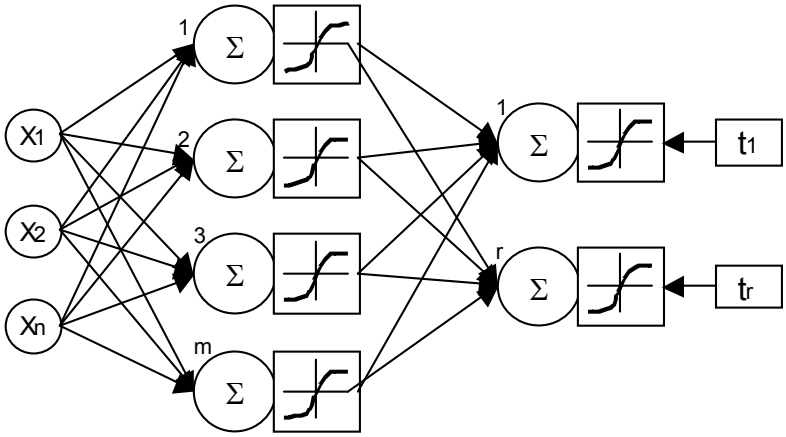


Fig. 1.15 Representación de una MLP

La respuesta que nos ofrece la neurona k, en el momento μ, es función de la multiplicación de los pesos, que unen la capa oculta con la capa de salida, por el resultado de cada neurona de la capa oculta. Estos últimos son función de la multiplicación de los pesos, entre la capa de entrada y la capa oculta, por las entradas de la red. Todo esto se puede representar de la siguiente manera:

$$Z_k^{\mu} = f\left\{\sum_{j=1}^{m} W_{kj}^{'}\cdot\left[f\left(\sum_{i=1}^{n} W_{ji}\cdot X_i - \theta_j\right)\right] - \theta_k^{'}\right\}$$

Las funciones de transferencia son sigmoideas y derivables. La manera de comprobar la bondad del modelo es mediante el error cuadrático medio. En una muestra finita este error viene representado por la función E.

$$E = \frac{1}{2} \cdot \sum_{\mu=1}^{p} \sum_{k=1}^{r} \left(t_k^{\mu} - Z_k^{\mu} \right)^2$$

Si sustituimos en la respuesta de la red por la función que la define la ecuación quedará como sigue:

$$E = \frac{1}{2} \cdot \sum_{\mu=1}^{p} \sum_{k=1}^{r} \left[t_k^{\mu} - f\left(W_{kj}^{'} \cdot y_j^{\mu} - \theta_k^{'} \right) \right]^2$$

Lógicamente cuanto más pequeña sea esta función, más eficiente será nuestro modelo, por lo que hay que proceder a minimizarla. Se hará mediante el *descenso por el gradiente*, tanto en la capa de salida como en la capa oculta. Hay que recordar que se van a derivar funciones sigmoideas.

La variación de los pesos que unen la capa oculta con la capa de salida es igual al coeficiente de aprendizaje por el gradiente del error:

$$\delta W_{kj}^{'} = -\varepsilon \cdot \frac{\delta E}{\delta W_{kj}^{'}}$$

El gradiente se calcula como la variación del error cuando varían los pesos sinápticos que unen la capa oculta con la capa de salida: derivando la función de error respecto a los pesos sinápticos. Hay que tener en cuenta que ahora la función de salida es afectada por una función de la familia de las sigmoideas y que al derivar la función de error hay que derivar la función utilizada. La variación de los pesos la podemos escribir de la siguiente manera:

$$\delta W_{kj}^{'} = \varepsilon \sum_{\mu=1}^{p} \left[t_k^{\mu} - f\left(v_k^{'\mu} \right) \right] \cdot \frac{\delta f\left(v_k^{'\mu} \right)}{\delta v_k^{'\mu}} \cdot y_j^{\mu} {}' = \varepsilon \sum_{\mu=1}^{p} \Delta_k^{'\mu} \cdot y_j^{\mu}$$

Siendo $v_k^{'\mu}$ la derivada de la respuesta de la red.

$\Delta_k^{'\mu}$ se denomina Señal de error de la capa de salida.

Con la Señal de error de la capa de salida, $\Delta_k^{'\mu}$, calculamos la Señal de error de la capa oculta: Δ_j^{μ}

$$\Delta_j^{\mu} = \left(\sum_{k=1}^{r} \Delta_k^{'\mu} \cdot W_{kj}^{'} \right) \cdot \frac{\delta f\left(v_j^{\mu} \right)}{\delta v_j^{\mu}}$$

Ahora podemos calcular la variación de los pesos que unen la capa oculta con la capa de entrada..

$$\delta W_{ji} = \varepsilon \cdot \sum_{\mu=1}^{p} \Delta_j^{\mu} \cdot x_i^{\mu}$$

2 Redes Neuronales Autoorganizadas

2.1 INTRODUCCIÓN

El cerebro de los mamíferos se puede dividir en diversas áreas encargadas de diferentes funciones del cuerpo. Varías de estas áreas, como el cortex, están organizadas en función de diferentes órganos sensoriales. En la figura 2-1 se han señalado, en la representación de un cerebro humano, las zonas más importantes responsables de diferentes funciones.

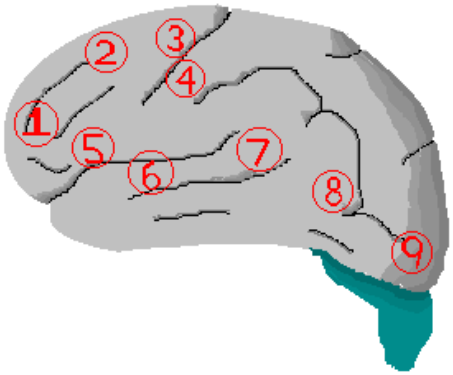


Fig. 2.1 Representación de las áreas del cerebro. Los diferentes números corresponden: 1. Pensamiento. 2. Planificación de acciones. 3. Funciones motoras. 4. Mapa somatosensorial. 5. Habla. 6. Oído primario. 7. Reconocimiento de palabras. 8. Movimientos de los ojos. 9. Visión. (Fuente: Kohonen 1989)

Las áreas sensoriales son escasas, sólo comprenden el 10 % del área cortical; y entre éstas están las áreas asociativas, que son menos conocidas, en las que convergen diferentes informaciones sensoriales que se ordenan siguiendo unos patrones establecidos. Así, por ejemplo, en la figura 2-2 se representa las diferentes funciones del área denominada *mapa Somatosensorial* (localizado con el número 4 en la figura 2 –1) mostrando el orden de la funcionalidad de diversos órganos. En este orden subyace una estructura en función de los órganos; por ejemplo, los dedos están todos próximos entre si, al igual que la lengua, la boca y la faringe.

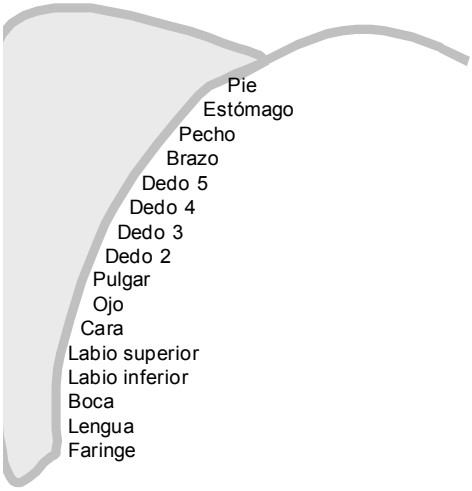


Fig. 2.2 Representación del Mapa Somatosensorial del Cerebro. (Fuente: Kohonen 1989)

En algunas partes del cerebro de los vertebrados, las neuronas, están interconectadas siguiendo una determinada estructura, donde las conexiones de las neuronas forman una estructura que es el reflejo del entorno sensitivo. Esta organización adopta la forma de un mapa. En el cerebro podemos encontrarnos mapas topológicos de los órganos sensoriales de nuestro cuerpo. Los mapas que forman estas interconexiones están dispuestos en dos dimensiones.

Estos mapas se organizan de manera autónoma, sin una referencia en la que corregir errores. El cerebro tiene la plasticidad suficiente para clasificar la información nueva sin tener una referencia anterior donde apoyarse. El cerebro se organiza de manera automática, o dicho de otra manera, se autoorganiza. El cerebro procesa la nueva información clasificándola en función de las características de la nueva información de manera topológica tal que al final la podemos localizar en determinadas zonas. De esta manera, toda información que reciba el cerebro con unos patrones de entrada similares será situada en las mismas zonas.

Esta característica aplicada en el campo de las redes neuronales artificiales, da el nombre a uno de los conjuntos de redes más fascinantes: las redes autoorganizadas. Estas redes pueden aprender a detectar regularidades y correlaciones en los datos de entrada y adaptar su respuesta futura de acuerdo con los datos de entrada. Las neuronas de una red competitiva aprenden a reconocer grupos de vectores de datos de entrada similares. Los mapas autoorganizados (SOFM)[3] aprenden a reconocer grupos de vectores de entrada similares y provoca el agrupamiento de las neuronas que responden a los mismos vectores de entrada. La información que llega a la red está organizada en un cierto orden que la red detecta y refleja en la capa de salida. En este libro analizaremos dos de las redes autoorganizadas más usadas: las Redes Competitivas y los Mapas de Kohonen.

2.2 LAS REDES COMPETITIVAS.

Una red competitiva tiene dos capas: una de entrada y la otra, la capa de salida. La única función de las neuronas de entrada es el envío de información a la capa de salida. Las neuronas de la capa de entrada están conectadas mediante pesos sinápticos con todas las neuronas de la capa de salida. De esta manera las informaciones que aportan cada neurona de la capa de entrada es enviada a todas, y a cada una de las neuronas de la capa de salida. Cada neurona, de la capa de salida, recibe el mismo conjunto de entradas desde la capa de entrada.

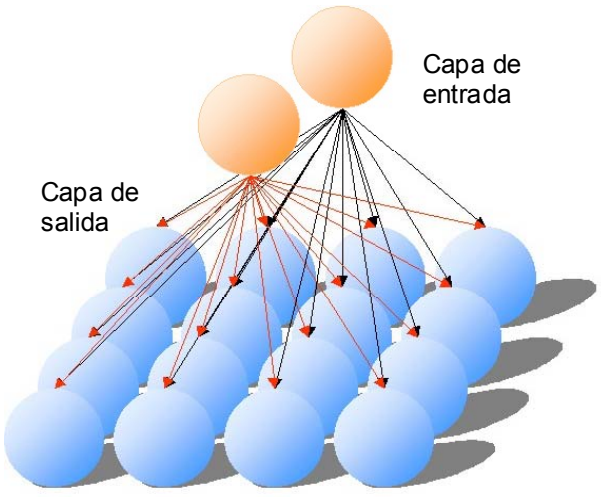


Fig. 2.3 Representación de un mapa autoorganizado.

[3] *SOFM son las siglas inglesas de Self – Organized Feature Maps. Estas siglas se las debemos al profesor de la Universidad de Helsinki Teuvo Kohonen*

El objetivo en una red competitiva es buscar la neurona de la capa de salida que tenga un conjunto de pesos sinápticos más parecidos a los valores de las neuronas de la capa de entrada. Con otras palabras, lo que busca es el vector de pesos sinápticos correspondientes a una sola neurona que sea más parecido al patrón de entrada de la red en un momento determinado. Para ello, cada neurona calcula la diferencia entre el valor del patrón de la entrada y el conjunto de los pesos sinápticos de cada neurona de salida. En función de este cálculo se determinará la neurona vencedora entre todas las neuronas; ésta será la que tiene menor diferencia entre sus pesos y el conjunto de entradas.

Supongamos una red competitiva como la mostrada en la figura 2.4. Esta red tiene dos capas: la capa de entrada contiene dos neuronas que mandan información a la segunda capa, donde las neuronas están ordenadas en una disposición de 4 x 4 neuronas. El número de neuronas de la capa de salida es decidido por el creador de la red en función del problema a resolver, así como la disposición de las neuronas. En este ejemplo se ha escogido una disposición cuadrada de 4×4, pero se podría haber escogido otra rectangular de, por ejemplo, 10×5. Las neuronas de la capa de entrada están conectadas con las neuronas de la capa de salida mediante pesos sinápticos.

Utilizaremos la siguiente nomenclatura:

$$W_{i,j,k}$$

Donde $W_{i,j,k}$ es el peso sináptico que parte de la neurona (k) de la capa de entrada y se conecta con la neurona en la posición (i, j) de salida. En nuestro ejemplo la neurona de la capa de salida en la posición 3,2 tiene dos pesos sinápticos:

$$W_{3,2,1}$$

$$W_{3,2,2,}$$

Siendo el primero, $W_{3,2,1}$ el peso procedente de la neurona 1 de la capa de entrada y el segundo, $W_{3,2,2,}$ el de la neurona 2 de la capa de salida.

Se podría utilizar otra nomenclatura que sería el numerar las neuronas de la capa de salida. En nuestro ejemplo, habría 16 neuronas. Comenzando a contar por filas de arriba hacia abajo y de izquierda a derecha, la neurona en la posición 3,2 sería la neurona número 10, y los pesos sinápticos, procedentes de la neurona 1 y 2 respectivamente, se podrían escribir del siguiente modo:

$$W_{10,1}$$

$$W_{10,2}$$

La entrada de datos en el momento t la representamos mediante el vector X

$$X(t) = \{x_1\ x_2\}$$

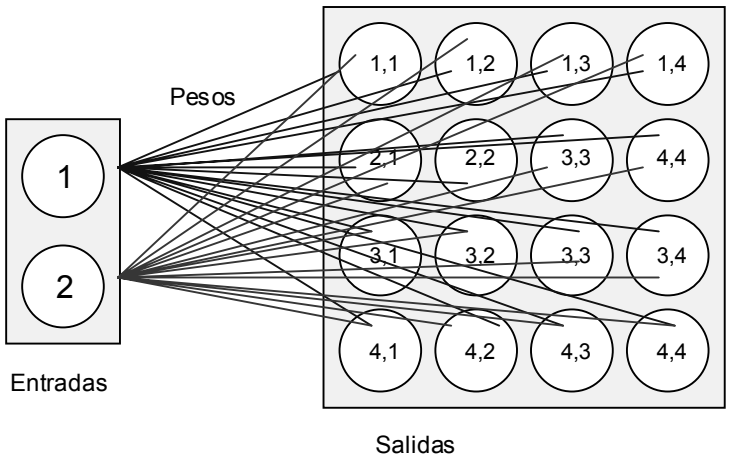


Fig. 2.4 Representación de una red competitiva.

Las distancias netas entre las neuronas de la capa de salida y el vector de patrones de entrada se calcula con la siguiente ecuación:

$$d_{i,j,(t)} = \sqrt{\sum_{h=1}^{k} \left(W_{i,j,h} - X_k\right)^2}$$

Siendo X_k la entrada de la neurona de entrada k y $d_{i,j,(t)}$ la distancia Euclídea de la neurona en la posición (i, j) en el momento t, respecto al patrón de entrada del momento t, para una red con i × j neuronas en la capa de salida y k neuronas en la capa de entrada.

La distancia neta de la neurona (3, 2) de la capa de salida la calculamos aplicando la ecuación de la siguiente manera:

$$d_{3,2,(t)} = \sqrt{\left(W_{3,2,1} - X_1\right)^2 + \left(W_{3,2,2} - X_2\right)^2}$$

Entonces se declara neurona ganadora aquella cuya distancia es la menor de todas. Es decir, la que muestra la menor distancia Euclídea.

$$g_{i,j} = \text{Min}\{\forall d_{i,j}\}$$

Establecida la neurona ganadora, todas las neuronas de la red mostrarán una salida igual a cero menos la neurona ganadora que muestra una salida igual a uno. La red nos muestra qué neurona responde ante ese determinado patrón de entrada y patrones similares deberán hacer responder a la misma neurona.

Una vez declarada la neurona ganadora sus pesos son ajustados mediante una regla de aprendizaje que tiene como objetivo acercar más los pesos de la neurona ganadora al patrón de entrada que la ha hecho ganar. De este modo, la neurona cuyos pesos estén más cerca del patrón de entrada es actualizada para estar, todavía, más cerca teniendo como resultado que la neurona ganadora tiene más posibilidades de ganar la competición en la siguiente presentación de datos de entrada para un vector de entrada similar; y menos posibilidades de ganar la competición si el vector de presentado es diferente. En definitiva, la neurona se ha especializado en ese patrón de entrada.

Cuantos más y más vectores de entrada son presentados, cada neurona de la capa de salida se aproximará al grupo de entrada parecidas, ajustando sus pesos hacía esos valores de los datos de las entradas. La ecuación que aproxima los pesos de la neurona vencedora hacía el vector de entrada es la siguiente:

$$W_{jik}(t+1) = W_{jik}(t) + \alpha \cdot \left[X_k(t) - W_{jik}(t)\right]$$

Siendo α el ratio de aprendizaje, $X_k(t)$ el patrón de las entradas en el momento t y $W_{jik}(t)$ el peso sináptico que conecta la entrada k con la neurona ji en el momento t.

Leamos la ecuación anterior: El peso que conecta la entrada k con la neurona j,i, en el próximo periodo será igual al valor del mismo peso en el periodo actual mas un factor de incremento. Este factor de incremento es el producto del coeficiente de aprendizaje por la diferencia entre el valor de la entrada y el valor del peso en el momento t. Dicho de otro modo: el factor de incremento es igual al producto del coeficiente de aprendizaje por el error entre el patrón de entrada y el conjunto de pesos.

Veamos el funcionamiento de las redes competitivas mediante un ejemplo. Vamos a clasificar 20 valores de los cuales sabemos su rendimiento, su riesgo medido por la desviación típica y la liquidez del mismo, medido por la probabilidad de transformar el producto en efectivo en las siguientes 24 horas. Todos estos datos se resumen en la siguiente tabla:

Tabla 5

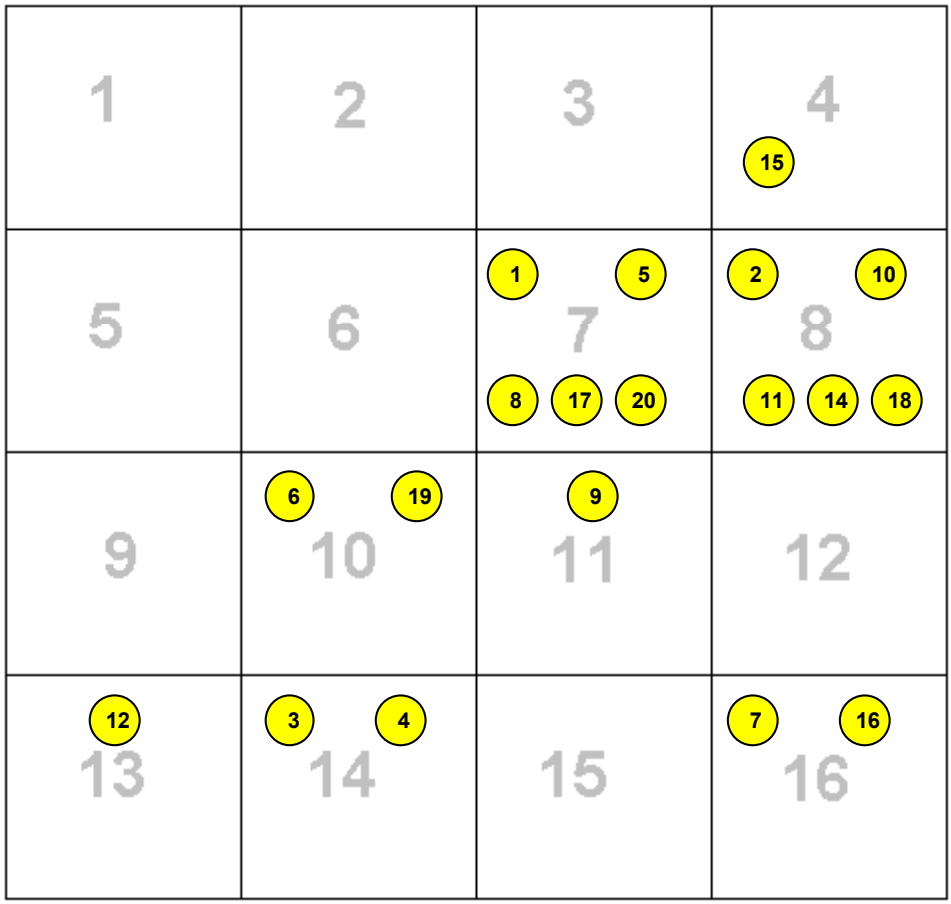
Valor	Rendimiento	Riesgo	Liquidez	Valor	Rendimiento	Riesgo	Liquidez
1	14,68%	53,10%	43,15%	**11**	21,86%	19,13%	86,87%
2	24,39%	23,14%	72,57%	**12**	27,79%	39,94%	7,34%
3	14,26%	34,34%	96,25%	**13**	25,24%	54,83%	25,12%
4	19,80%	44,92%	99,18%	**14**	39,12%	9,83%	48,98%
5	31,01%	22,79%	39,41%	**15**	34,83%	58,75%	32,15%
6	7,96%	5,75%	17,84%	**16**	15,59%	40,96%	79,92%
7	17,13%	41,69%	82,73%	**17**	34,13%	46,69%	53,06%
8	3,36%	59,23%	42,89%	**18**	14,75%	2,49%	86,14%
9	18,25%	58,16%	6,19%	**19**	0,25%	19,11%	1,50%
10	5,33%	14,89%	69,54%	**20**	19,88%	51,02%	31,65%

Utilizaremos una red competitiva de 4 × 4 neuronas en la capa de salida y tres neuronas en la capa de entrada. Para realizar este ejemplo desarrollamos un programa en QBASIC. El código es el siguiente:

Código	Comentario
10 REM red competitiva de 4 x 4, 3 de entrada	Título del programa
20 CLS : ref = 1000: g = 0	Borramos todo y damos a la variable ref el valor de 1.000. La variable g tiene el valor cero
30 DIM x(20): DIM y(20): DIM w(48): DIM z(20)	Damos a la variable X, la variable Y, y la variable Z la dimensión de 20 datos. W tendrá 48
40 RANDOMIZE TIMER	Activamos el generador de números aleatorios
50 FOR b = 1 TO 48	Establecemos 48 ciclos
60 w(b) = RND	El peso W(b) tiene valor aleatorio.
70 NEXT b	Siguiente ciclo
80 PRINT "Introducción de los datos de entrada"	
85 PRINT : PRINT	
90 FOR a = 1 TO 20	Establecemos 20 ciclos
100 PRINT "x "; a;	
110 INPUT x(a)	Preguntamos el valor de entrada X
120 PRINT "y "; a;	
130 INPUT y(a)	Preguntamos el valor de entrada Y
132 PRINT "z "; a;	
134 INPUT z(a)	Preguntamos el valor de entrada Z
140 NEXT a	Siguiente ciclo
150 CLS	Borramos la pantalla
160 INPUT "Nº de ciclos", C	Preguntamos el número de ciclos de la red
170 FOR d = 1 TO C	Establecemos el número de ciclos determinado
175 CLS	Borramos pantalla
180 PRINT d	Mostramos el número de ciclo
190 FOR e = 1 TO 20	Establecemos 20 ciclos
200 FOR f = 1 TO 48 STEP 3	Establecemos 20 ciclos con salto de 3 en 3
210 dist = SQR(((w(f) - x(e)) ^ 2) + ((w(f + 1) - y(e)) ^ 2) + ((w(f + 2) - z(e)) ^ 2))	Calculamos la distancia entre los tres grupos de entrada con los tres pesos de cada neurona
220 IF dist < ref THEN ref = dist : g=f	Si la distancia es menor al valor de ref entonces ref toma el valor de dist como nuevo valor.
230 IF dist < ref THEN g = e	Si distancia es menor entonces g valdrá el número de la
240 NEXT f	
250 w(g) = w(g) + (.1 * (x(e) - w(g)))	
260 w(g + 1) = w(g + 1) + (.1 * (y(e) - w(g + 1)))	
265 w(g + 2) = w(g + 2) + (.1 + (z(e) - w(g + 2)))	
270 NEXT e	
280 NEXT d	

290 FOR h = 1 TO 48	
300 PRINT h, w(h)	
305 LPRINT h, w(h)	
310 NEXT h	
320 END	

En un primer momento establecemos de manera aleatoria los pesos que unen la capa de entrada con la capa de salida. Se van presentando los datos a la red y esta, utilizando la distancia Euclídea selecciona la neurona ganadora y aproxima un poco más los pesos. Al final después de 1.000 ciclos la red ha clasificado a todos los valores de la siguiente manera.



La red ha especializado las neuronas en diferentes valores; así la neurona 1 responde ante el valor 1, 5, 20, 17 y el valor 20.

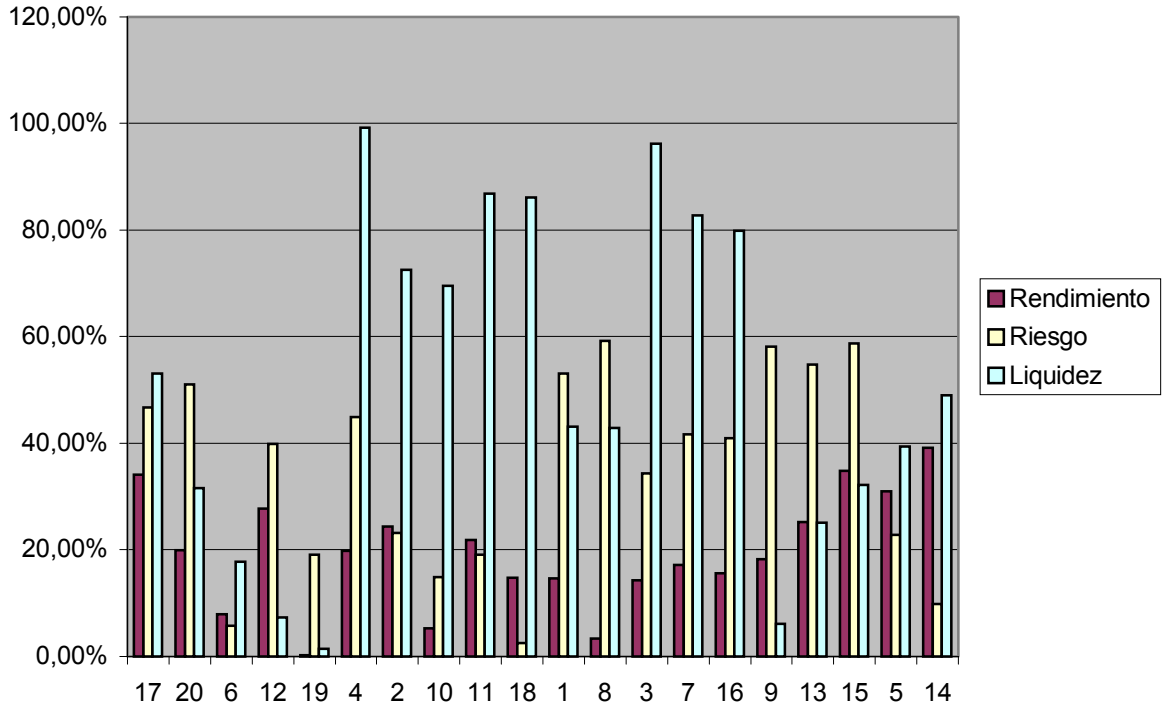


Fig. 2.5 Los parámetros de los valores en función de su clasificación en la red competitiva

En la figura anterior se muestra en un diagrama de barras los diferentes valores, agrupándolos por su situación en las neuronas de la red competitiva. Como se puede observar hay cierto parecido entre los niveles de los parámetros de los valores situados en las mismas neuronas. Por ejemplo, los valores 2, 10, 11, 14 y 18 se han situado en la neurona número 8 y todos estos tienen una gran liquidez, y un riesgo y rendimiento por debajo de 30%.

Con esta red, se ha podido clasificar valores atendiendo a tres parámetros; pero además se puede realizar el mismo trabajo con más parámetros, obteniendo la clasificación de los valores en diferentes neuronas.

2.3 LOS MAPAS AUTOORGANIZADOS

En las redes competitivas, cada neurona de la capa de salida, calcula la similitud entre el vector de entrada y su propio vector de pesos sinápticos. La neurona con el vector de pesos más parecido al de entrada, se declara neurona "ganadora". La neurona ganadora modifica su vector de pesos para que se parezcan un poco más al vector de entrada. Así, si se vuelve a repetir ese vector de entrada, la misma neurona responderá y se acercará un poco más a la imagen de ese vector de entrada. De esta manera, cada neurona representará a cada vector de entrada.

Teuvo Kohonen introduce en las redes competitivas la función de vecindad creando los Mapas de rasgos autoorganizados o SOFM (*Self-Ornanizing Feature Maps)*. Como en las redes neuronales autoorganizadas, en los SOFM se declara una neurona ganadora. Esta es la que tiene una distancia menor entre sus pesos sinápticos y el vector de datos de entrada. Pero ahora, tiene, además, una función de vecindad. Esta función define el entorno alrededor de la neurona ganadora, y en la fase de aprendizaje se actualizan tanto los pesos de las neurona ganadora como los pesos de las neuronas pertenecientes a la vecindad.

El efecto de la introducción de la función de vecindad es que las neuronas próximas a la ganadora sintonizan con los patrones de entrada que ha hecho ganar a la vencedora. Fuera de la vecindad no se actualizan los pesos de las neuronas.

La vecindad está en función de la distancia entre la neurona ganadora y sus vecinas. La distancia es una zona bidimensional que existe alrededor de cada neurona. Esta zona puede ser circular cuadrada o hexagonal. En la figura 7 se presenta una zona de vecindad creada mediante una función de Gauss. La altura representa la influencia de la neurona ganadora en las neuronas de esa región. Cuanta más altura, mayor será la influencia de esa neurona en las neuronas que constituyen su vecindad. Como se puede observar esta influencia decrece con la distancia a las neuronas.

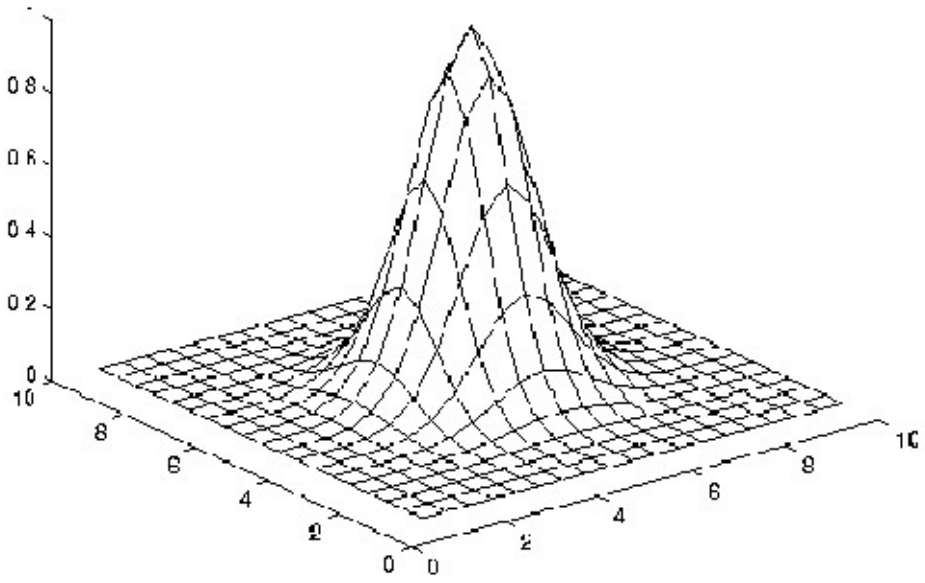


Fig. 2.6 Forma gaussiana de la función de vecindad.

De esta manera, en el modelo de SOFM se logra que las neuronas próximas sintonice con patrones similares, quedando de esta manera reflejada sobre el mapa una cierta imagen del orden topológico presente en el espacio de entradas. El orden de la información del conjunto de los patrones de entrada queda reflejado en el mapa de la capa de salida de una red del tipo SOFM

Lo que hacen los SOFM es una tarea de clasificación, ya que la neurona de salida activada ante una entrada, representa la clase a la que pertenece dicha información de entrada. Además, como ante otra entrada parecida se activa la misma neurona de salida, u otra cercana a la anterior, debido a la semejanza entre las clases, se garantiza que las neuronas topológicamente próximas sean sensibles a entradas físicamente similares. Por esta causa las redes de mapas autoorganizados son especialmente útiles para establecer relaciones desconocidas entre conjunto de datos. Un conjunto de datos que no se le conozca un orden predeterminado, se puede clasificar, es decir, ordenar, mediante una red SOFM.

Durante el proceso de aprendizaje se van presentando diferentes patrones de entrada. Cada patrón se identificará con una neurona ganadora y su vecindad. Al tiempo que aumenta el número de patrones de entrada presentados, disminuye el tamaño de la vecindad.

En el momento en el que se diseña la red, se debe decidir el tipo de vecindad que tendrá esta: La vecindad puede ser rectangular o hexagonal. Este tipo de vecindades se presenta en la figura 8. La

vecindad mostrada en la izquierda de la figura 8 es una vecindad hexagonal, mientras que la vecindad de la derecha es rectangular.

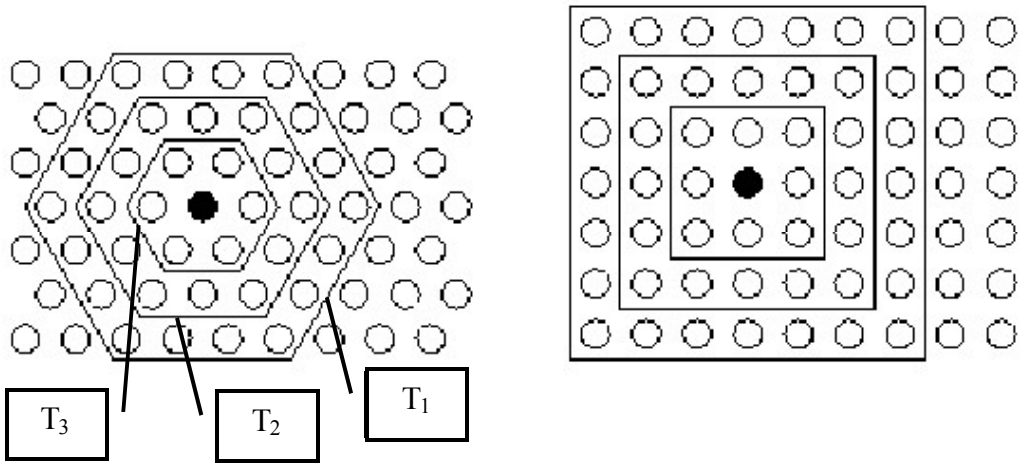


Fig. 2.7 Tipos de vecindad: la figura de la izquierda es una vecindad hexagonal, mientras que la figura de la derecha es una vecindad rectangular. Los círculos representan neuronas, siendo el círculo negro la neurona ganadora. T1, T2 y T3 representa el momento del tiempo donde se mide la vecindad; T1 es la iteración en el momento 1, mientras que T2 es la del momento 2.

Una vecindad de radio 2, incluye dos neuronas, además de la neurona ganadora; por otro lado una vecindad de radio 1 sólo afectará a las neuronas adyacentes a la neurona ganadora. En la figura 8 se puede observar cómo la vecindad se reduce con cada iteración. En este ejemplo, en el momento T_1 la vecindad es de radio 3, mientras que en el momento T_2 la vecindad es de radio 2, más pequeña.

Podemos diferenciar el proceso de aprendizaje en dos fases:

1. La ordenación: donde se identifican las neuronas ganadoras y su vecindad.
2. El ajuste fino: donde se especializan las neuronas ganadoras. En esta fase el radio de la vecindad es igual a 1. Es decir , la vecindad se reduce a la neurona ganadora.

Mecánica de los mapas autoorganizados

a. Los pesos, W_{ijk} al principio pueden ser aleatorios de pequeño valor absoluto o con un valor determinado.

b. Se presenta un patrón de entrada. X(t) Este, de todos los patrones que hay para el entrenamiento, es escogido al azar. Cada neurona calcula su similitud entre los pesos sinápticos y el vector de entrada mediante la Distancia Euclídea[4].

$$d = \sqrt{\sum_{k=1}^{N} \left(W_{ijk} - X_k \right)^2}$$

[4] *Hay otros criterios de medida, como la distancia Manhatann o el Producto escalar. De todos modos, la más habitual es el distancia Euclídea*

c. Determinación de la neurona ganadora: g*. Esta es la que muestra la menor distancia al patrón de entrada.

d. Actualización de los pesos de la neurona ganadora g* y sus vecinas mediante la siguiente ecuación:

$$W_{ijk}(t+1) = W_{ijk}(t) + \alpha(t) \cdot h(|i - g^*|, t) \cdot (X_k(t) - W_{ijk}(t))$$

Donde $\alpha(t)$ es un término de aprendizaje que toma valores comprendidos entre 0 y 1. Cuando se alcanza un número de iteraciones superior a 500, entonces $\alpha(t)$ tiende a valer 0. Para el cálculo de $\alpha(t)$ se suele utilizar una de las dos siguientes ecuaciones

$$\alpha(t) = \alpha_0 + (\alpha_f - \alpha_0) \cdot \frac{t}{t_\alpha}$$

O también:

$$\alpha(t) = \alpha_0 \cdot \left(\frac{\alpha_f}{\alpha_0}\right)^{\frac{1}{t_\alpha}}$$

En donde α_0 es el ritmo inicial, α_f es el ritmo final, que suele tomar valores de 0,01, t es la iteración actual y t_α es el número máximo de iteraciones que se desean realizar[5].

La función $h(|i - g^*|, t)$ es la función de vecindad. El tamaño de la vecindad se reduce en cada iteración.

En la fase de ajuste fino, α vale 0,01 y el radio de vecindad es igual a 1.

El número de iteraciones es proporcional al número de neuronas e independientes del número de entradas. Un número de 50 a 100 iteraciones suele ser suficiente.

En algunos casos y dependiendo de la regla de aprendizaje y la función de distancia escogida, puede haber conflicto entre ellas. En la mayoría de los casos se utiliza la distancia Euclídea y la regla de actualización mostrada.

La función de vecindad depende de:

1. La distancia
2. El radio de vecindad.

La distancia se mide como:

$$|i - g| = \sqrt{(i - g_1)^2 + (j - g_2)^2}$$

La función de vecindad decrece con la distancia a la vencedora. Cuanto más alejada, más pequeña será. Depende del radio de vecindad R(t) que representa el tamaño de la vecindad actual.

$$h(|1 - g^*|, t) = f[R(t)]$$

---

[5] *Existen otras ecuaciones que nos calculan el ritmo de aprendizaje para la iteración t: Un ejemplo de estas son: $\alpha(t) = 1 / t$ y $\alpha(t) = [1 - (t / \alpha_2)] \cdot \alpha_1$ donde α_1 toma un valor de 0,1 ó 0,2 y α_2 es un valor que intenta aproximar el número total de iteraciones. En la práctica se suele tomar valores cercanos a 10.000 iteraciones.*

Para el cálculo de la vecindad se utilizan funciones del tipo Escalón o del tipo Sombrero Mejicano. Un ejemplo de esta función se puede observar en la figura 9. La función representada responde a la ecuación:

$$y = 2 \cdot e^{-1 \cdot x^2} - 0{,}5 \cdot e^{-0{,}05 \cdot x^2}$$

El ratio de vecindad R(t) disminuye con el tiempo. Una ecuación, usualmente empleado, que disminuye el ratio de vecindad en función del tiempo es la siguiente:

$$R(t) = R_0 + (R_f - R_0) \cdot \frac{t}{t_R}$$

R_f es el radio final que toma valor igual a 1. Además, t_R es el número de iteraciones hasta alcanzar R_f.

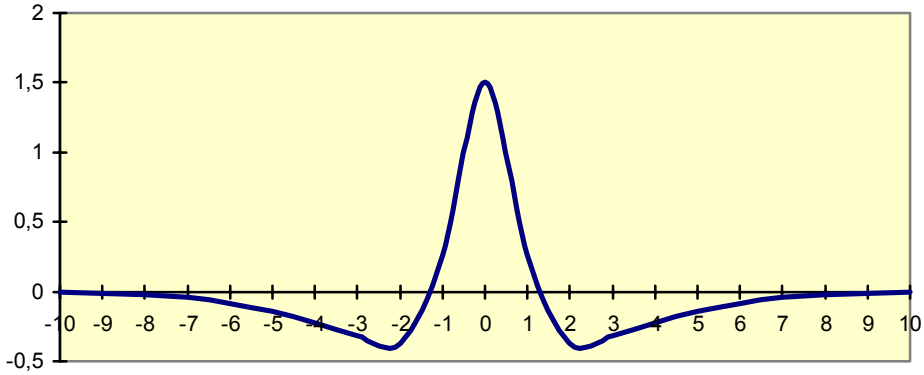


Fig. 2.8 Gráfica de una función del tipo Sombrero Mejicano utilizada para la determinación de la vecindad entre neuronas.

Cuanto mayor es el número de patrones iguales presentados, más neuronas se especializan en ese patrón. El número de neuronas que se especializan en reconocer un patrón de entrada depende de la probabilidad de aparición de este patrón. Así, el mapa resultante aproxima la forma de la función de densidad de probabilidad del espacio sensorial. La cantidad de neuronas concentradas en una región muestran la mayor probabilidad de aparición de ese tipo de patrones.

Después de declararse la neurona ganadora (*Best-Matching Unit*), BMU, los vectores de los pesos del SOM son actualizados. Los vectores de los pesos del BMU y sus vecinas topológicas se mueven hacía el vector de entrada, haciendo la distancia más reducida.

Esta adaptación provoca una estrechez entre la neurona ganadora y sus vecinas topológicas con el vector de entrada. Esto se ilustra en la figura 10 donde el vector de entrada está marcado con una x. La neurona ganadora está señalada mediante las siglas BMU. Se observa como la neurona ganadora como las vecinas se acercan al vector de entrada. El desplazamiento disminuye con la distancia a la BMU.

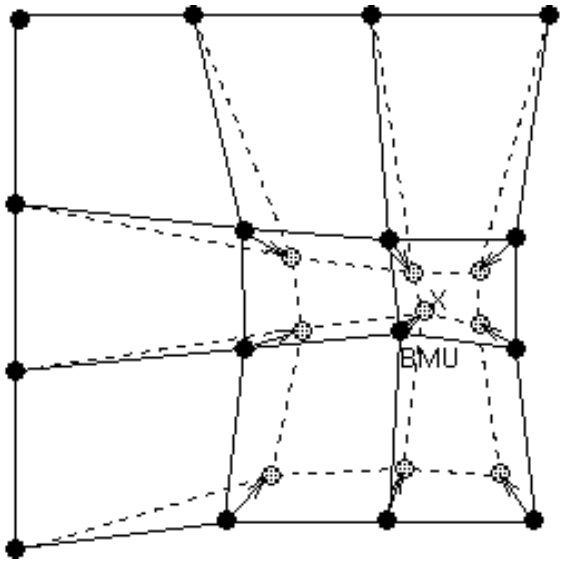


Fig. 2.9 La actualización de la neurona vencedora (BMU) y sus vecinas hacia el vector de entradas, marcado con una x. Las líneas continuas y punteadas representan la situación antes y después de la actualización, respectivamente

Para ilustrar todo lo anterior vamos a realizar un análisis de las opciones del primer semestre de 2000 sobre el futuro sobre el IBEX-35 mediante SOFM con tres variables: el open interest, el tiempo, el años, que queda hasta el vencimiento y la relación entre el valor del precio de ejercicio y el IBEX-35.

Para realizar este ejemplo utilizaremos los datos proporcionados por MEFF sobre el contrato Call sobre el futuro del IBEX-35 desde el 3 de enero de 2000 hasta el 30 junio de 2000. Sólo utilizaremos los datos de los contratos que tuvieron volumen de negociación; esto reduce a 4.779 referencias. Establecemos utilizar 4.559 referencias, elegidas de manera aleatoria para realizar tanto el entrenamiento como el ajuste fino, reservando las 180 referencias restantes para realizar un test de clasificación. Utilizaremos el Toolbox de MatLab proporcionado por T. Kohonen en la Universidad tecnológica de Helsinki.

Primero cargamos los datos de las 4.559 referencias. Teniendo en cuenta que son tres variables por cada referencia el programa trabajará con 13.677 datos.

```
» sD=som_read_data('opibex.data');
data read ok
```

Ahora, normalizaremos el conjunto de datos dentro del rango [0 1] pero realizando la transformación lineal en el rango medio y no lineal en los extremos. Acto seguido efectuamos el entrenamiento

```
» sD=som_normalize(sD,'logistic');
» sM=som_make(sD);
Determining map size...
 map size [23, 15]
Initialization...
Training using batch algorithm...
Rough training phase...

Training:   3/   3 s
Finetuning phase...
```

```
Training:   3/  12 s
Training:   6/  12 s
Training:   9/  12 s
Training:  12/  12 s
Final quantization error: 0.049
Final topographic error:  0.037
```

Mediante la orden som_autolabel vinculamos las etiquetas de los datos a los resultados de la red. De esta manera la red nos puede mostrar donde se sitúa cada referencia. Con la orden som_show mostramos los resultados del entrenamiento

```
» sM=som_autolabel(sM,sD,'vote');
» som_show(sM,'umat','all','comp',[1:3],'empty','Labels','norm','d');
```

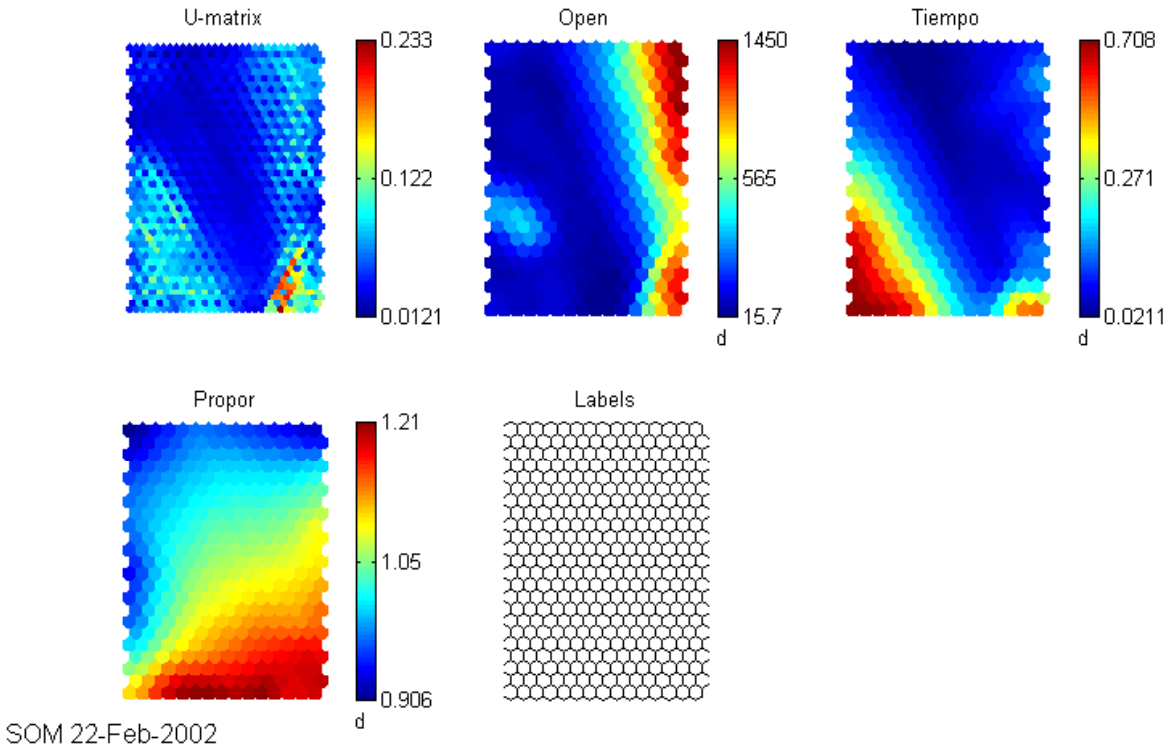


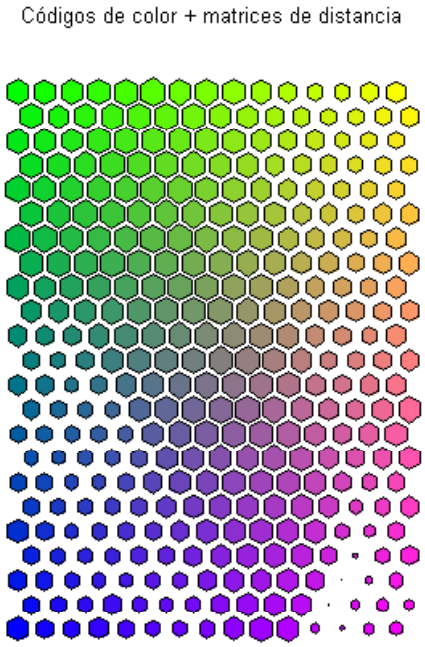
Fig. 2.10 Resultados de la Red

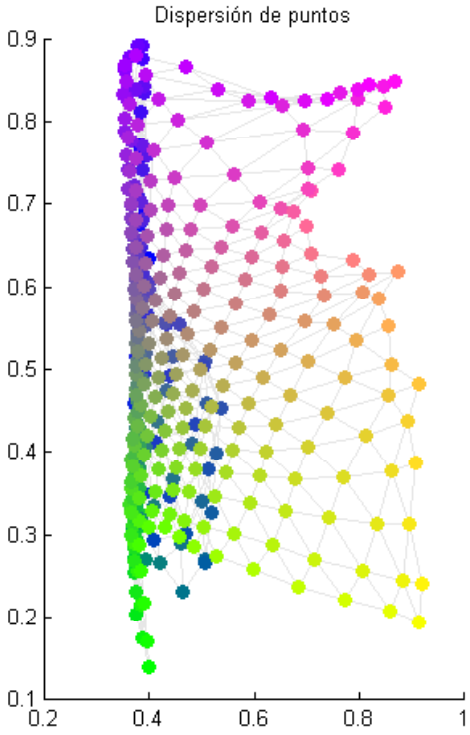
En la figura 11 se muestra el resultado del entrenamiento. La primera figura es el mapa de las neuronas. El color muestra la distancia que existe entre las neuronas. Las siguientes figuras nos muestra las funciones de densidad de las tres variables utilizadas. Se puede constatar que no existe una alta correlación entre las tres variables, por lo que las tres son significativas.

En las siguientes instrucciones se ordenará al programa que muestre las distancias entre las neuronas mediante el tamaño de estas y que a su lado nos muestre la maya en tres dimensiones de las neuronas.

```
» U=som_umat(sM);
» Um=U(1:2:size(U,1),1:2:size(U,2));
» C=som_colorcode(sM);
» subplot(1,2,1)
» som_cplane(sM,C,1-Um(:)/max(Um(:)));
» title('Códigos de color + matrices de distancia')
```

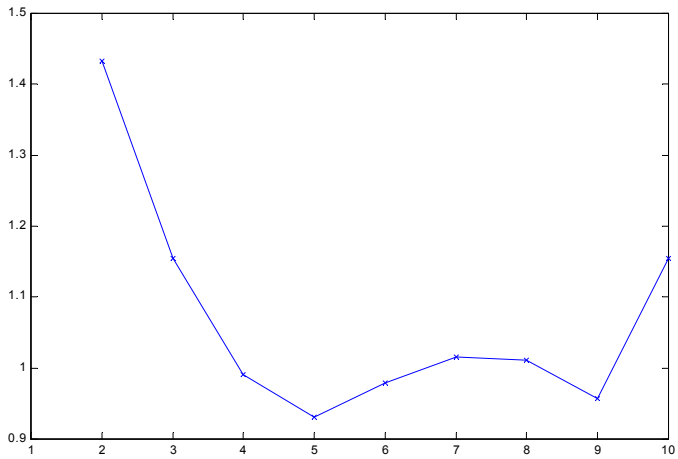
```
» subplot(1,2,2)
» som_grid(sM,'Coord',sM.codebook(:,[1 3]),'MarkerColor',C);
» title('Dispersión de puntos');
```





Ahora le pediremos al programa que busque el número de conjuntos que existe dentro del mapa mediante el algoritmo de D-B.

```
» [c,p,err,ind]=kmeans_clusters(sM,8);
» plot(1:length(ind),ind,'x-')
```



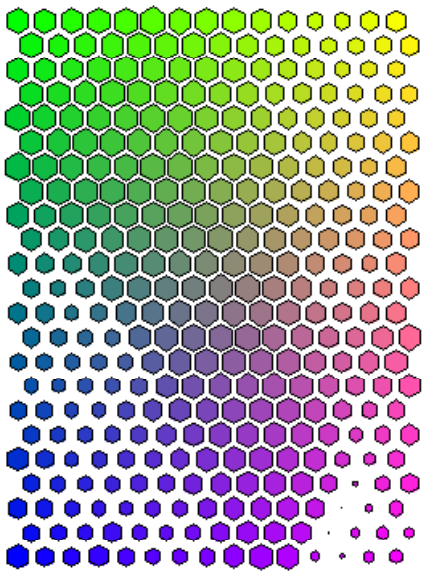
El gráfico nos dice que existen 5 posibles conjuntos, lo comprobamos numéricamente

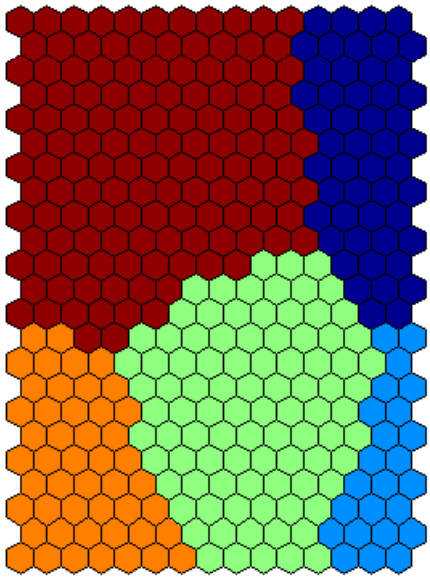
```
» [dummy,i]=min(ind)
dummy =
    0.9314
```

```
i =
     5
```

Ahora pediremos al mapa que nos detalle esos cinco conjuntos, al lado del mapa original para poder comparar

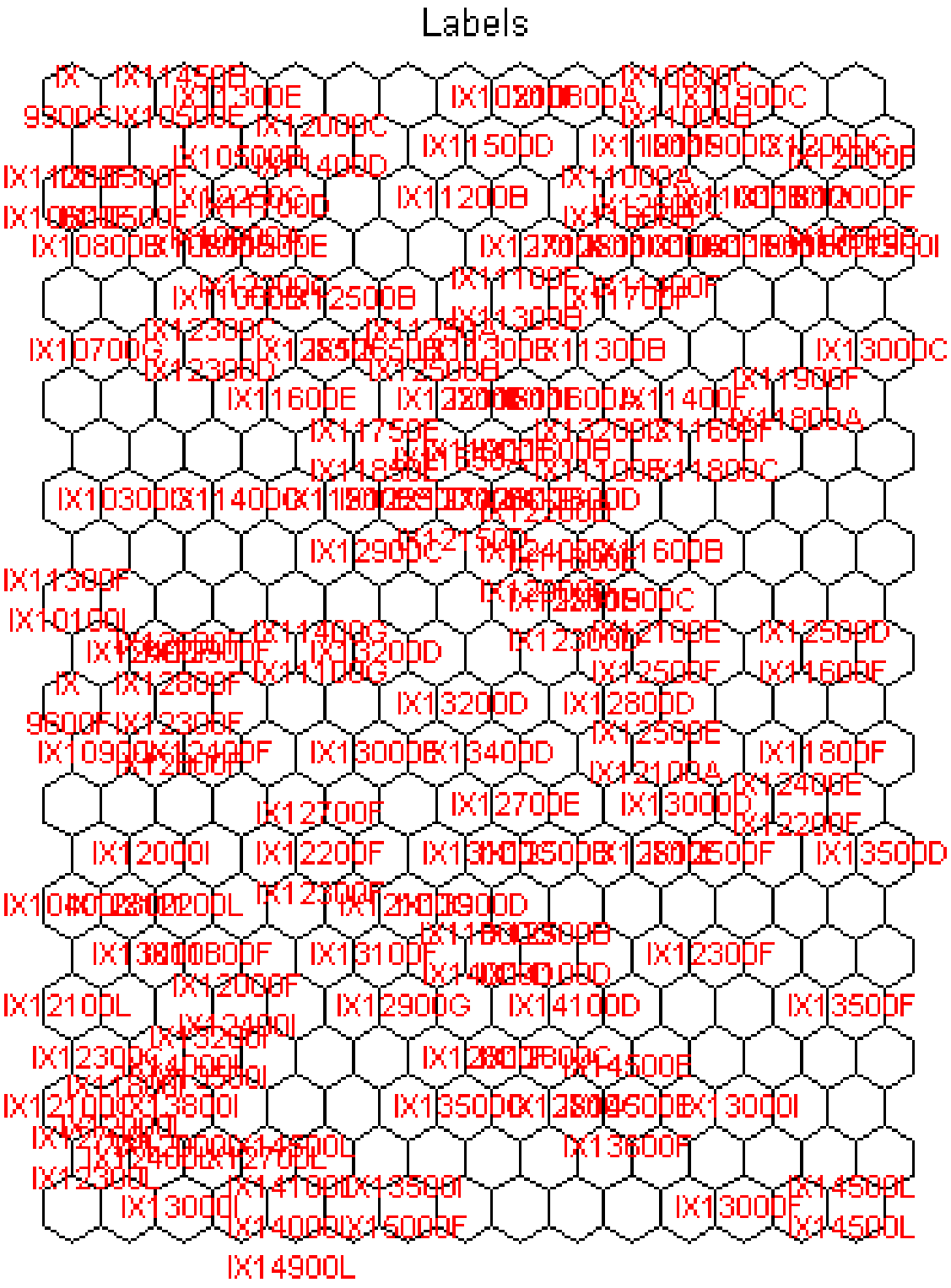
```
» cl=p{i};
» subplot(1,2,1)
» som_cplane(sM,C,1-Um(:)/max(Um(:)));
» subplot(1,2,2)
» som_cplane(sM,cl)
```





Por ultimo, vamos a cargar los datos del Test y a normalizarlo. Quitaremos las etiquetas anteriores y asignaremos las etiquetas del conjunto de Test y las mostraremos en un mapa

```
» sT=som_read_data('opibexb.data');
data read ok
» sT=som_normalize(sT,'logistic');
» sM=som_label(sM,'clear','all');
» sM=som_autolabel(sM,sT,'add');
» som_show(sM,'empty','Labels')
» som_show_add('label',sM,'textsize',8,'textcolor','r')
```



El último paso sería comprobar las características de los productos analizados y ver el patrón de comportamiento.

3 Bibliografía

ALHONIEMI, E HIMBERG, J PARVIAINEN, J y VESANTO, J.: *SOM Toolbox 2.0, a software library for Matlab 5 implementing the Self-Organizing Map algorithm* Laboratory of Computer and Information Science. Helsinki. 2000.

AMARI, S-I.: *Topographic organization of nerve field*. Bulletin of Mathematical Biology 42 1980: Pág. 339 - 364

APOSTOL, T.: *Cálculus.* Tomo I y II Reverté. Barcelona 1986

AZOFF, M.: *Neural network tome series forecasting of financial markets.* John Wiley & Sons. Sussex. 1997

DAVIES, D.L., BOULDIN, D.W.: *A Cluster Separation Measure*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-1, no. 2, 1979, pp. 224-227

DEBOECK, G.: *Self-Organizing Maps facilitate knowledge discovery in finance*. Financial Engineering news. Nº 8 . Seattle. 1998.

DEMUTH, H y BEALE, M.: *Neural Network Toolbox User's Guide* The MathWorks, Inc. Natick, MA. 1998.

DÍEZ DE CASTRO, L. y MASCAREÑAS, J.: *Ingeniería Financiera*. Segunda Edición. Ed. McGraw – Hill. Madrid 1995.

FREEMAN, J y SKAPURA, D.: *Redes neuronales. Algoritmos, aplicaciones y técnicas de programación.* Ed. Addison - Wesley. Wilmington, Delaware. USA 1993.

GURNEY, K. y WRIGHT, M.: *A self-organising neural network model of image velocity encoding*. Biological Cybernetics, 68: 173- 181

HILERA, J. y MARTÍNEZ, V.: *Redes Neuronales Artificiales*. Ed. Ra – Ma. Madrid 1995.

KOHONEN, T.: *Self – Organization and Associative Memory*. Springer – Verlag. Berlin 1989.

KOHONEN, T.: *Self – Organized formation of topologically correct feature maps.* Biological Cybernetics 43: 59 – 69 1982

KOHONEN. T.: *Introduction to SOM Toolbox*. EET DST Helsinki 1997

KRÖSE, B y SMAGT, P.: *An introduction to Neural Networks*. University of Amsterdam. Amsterdam 1996.

MARTÍN DEL BRÍO, B. SANZ MOLINA, A.: *Redes neuronales y sistemas borrosos*. . Ed. Ra – Ma. Madrid 1997.

MINSKY, M. PAPERT, S.: *Perceptrons*. MA: MIT Press. Cambridge 1969 Introducción y Pág. 1 – 20.

MORENO, A.: *Trabajando con Matlab y la Control System Toolbox*. Ed. Ra-Ma. Madrid 1999.

O'ROURKE, B.: *Neurnal nets forecast futures prices*. Financial Engineering News. Nº 1. Seattle 1998

PENG, S. DATTATREYA, R.: *The Structured Note Market.* Ed. Probus. Chicago. 1995.

REFENES, A.: (Editor) *Neural Networks in the Capital Markets*. Ed. John Wiley & Sons. Londres 1995.

RÍOS, J. PAZOS, A. BRISABOA, N. CARIDAD, S.: *Estructura Dinámica y Aplicaciones de las Redes de Neuronas Artificiales*. Ed. Centro de Estudio Ramón Areces. Madrid. 1991.

ROSENBLATT, F:. *Principles of Neurodynamics*. Spartan Books. New York 1989

ROSENBLATT, F:. *The Perceptrón*: *A probabilistic model for information storage and organization in the brain.* Psychological Review, 1958 Nº 65.

RUIZ, R. JIMÉNEZ, J.: *Las redes neuronales en su aplicación a las finanzas*. Banca Finanzas. Nº 54. Abril 2000. Pág 19 – 26

WILDROW, B. HOFF, M.: *Adaptive switching circuits*. 1960 IRE Wescon. Grabación de la convección. Nueva York. 1960.

WILLSHAW D.J. MALSBURG, C.: *How patterned connections can be set up by self-organization*. Proceedings of the Royal Society of London, B 194: Pág. 431- 445

Tabla de contenidos

1 REDES NEURONALES SUPERVISADAS 2

1.1 NEURONAS ARTIFICIALES 2
1.2 REDES NEURONALES ARTIFICIALES. 6
1.3 LAS REDES SUPERVISADAS: EL PERCEPTRÓN MULTICAPA Ó MLP 7
1.4 CÁLCULO DEL ALGORITMO DE LA NORMAL MEDIANTE UNA MLP 8
1.5 CONSTRUCCIÓN DE UN ALGORITMO PARA CALCULAR EL PRECIO DE LAS OPCIONES. 14
1.5.1 Objetivo del trabajo. 14
1.5.2 Mecánica del modelo. 14
1.5.3 Aplicación en Excel 18
1.5.4 Conclusión. 19
1.6 APÉNDICE: LAS REGLAS DE APRENDIZAJE. 20
1.6.1 La regla LMS 20
1.6.2 La regla BP 22

2 REDES NEURONALES AUTOORGANIZADAS 24

2.1 INTRODUCCIÓN 24
2.2 LAS REDES COMPETITIVAS. 25
2.3 LOS MAPAS AUTOORGANIZADOS 30

3 BIBLIOGRAFÍA 40