VNIVERSITAT DE VALÈNCIA

MASTER DE INGENIERÍA BIOMÉDICA.

Métodos de ayuda al diagnóstico clínico.

Tema 5: Redes Neuronales

Emilio Soria, Antonio José Serrano y José David Martín Dpto Ingeniería Electrónica, ETSE
Sistemas de Ayuda a la Decisión Clínica, Curso 2009-2010

Objetivos del tema

Conocer las limitaciones de los modelos lineales en problemas de modelización/ clasificación.

Aprender los problemas que pueden surgir al aplicar estos métodos neuronales.

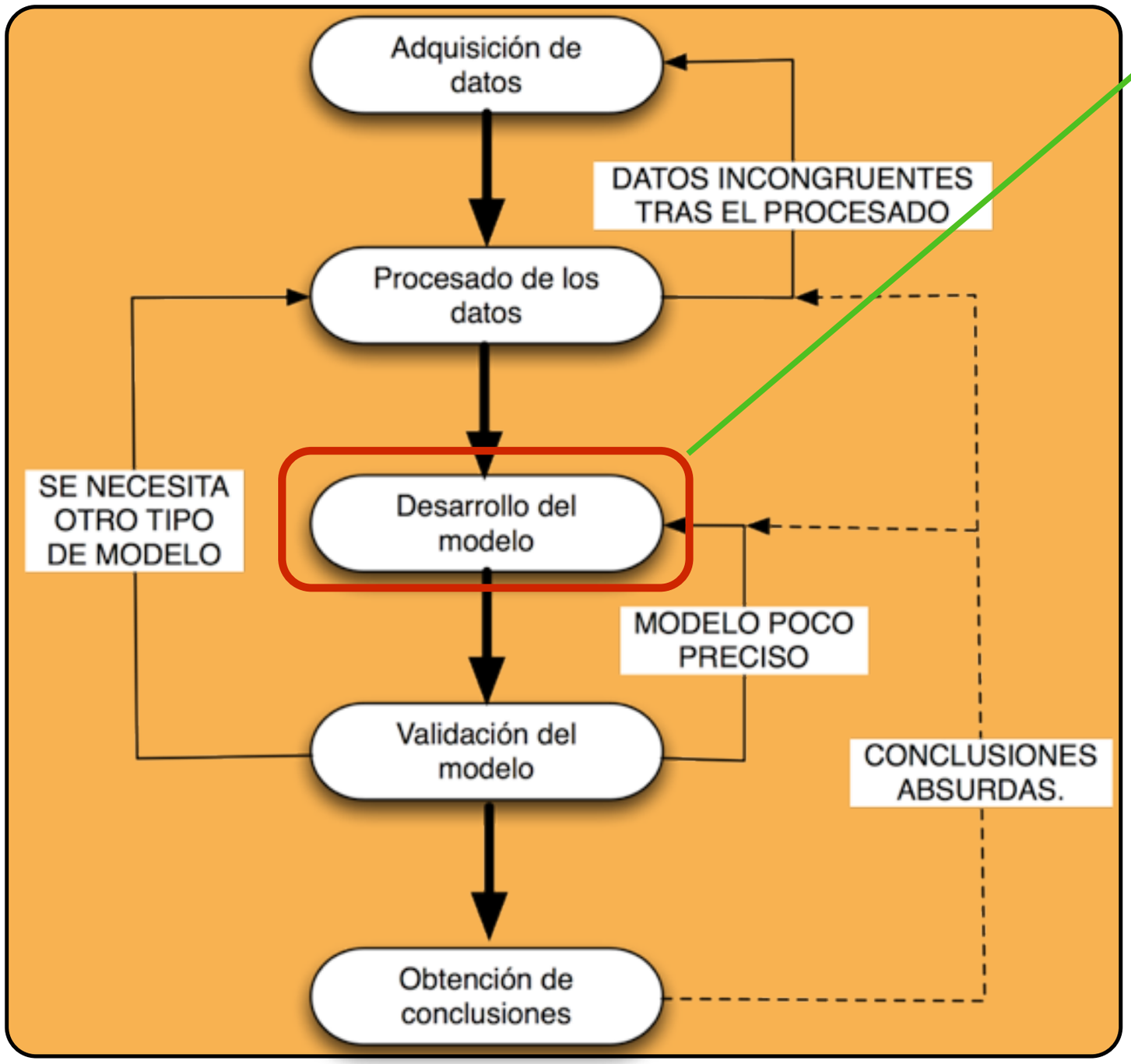
Aprender a aplicar la regla delta para cualquier estructura que se tenga.

Conocer el Teorema de Cybenko y sus implicaciones en cuanto al uso de redes multicapa en la resolución de problemas.

Saber cómo analizar los resultados obtenidos con una red neuronal al tiempo que se extrae conocimiento de dicho modelo neuronal.

Conocer las funciones de base radial (RBF).

Dónde estamos



Se ha implementado un modelo lineal en los parámetros y hemos comprobado que no ha funcionado; esta comprobación la podemos tener a varios niveles:

Los errores cometidos por el modelo son muy grandes
El modelo desarrollado no da buenos resultados en datos no usados para construir el modelo
Las conclusiones que se obtienen del modelo son absurdas.
No se cumplen las hipótesis de partida del modelo lineal (errores i.i.d, normales, de varianza cte)
No cumple algunos de los tests estadísticos planteados para ese modelo

Ahora tenemos dos opciones volvemos atrás para obtener más datos o bien usamos modelos no lineales; los primeros que describiremos serán las ***redes neuronales***.

Redes neuronales, algunas definiciones

Haykin; "Una red neuronal es un procesador distribuido y con una estructura paralela que tiene una tendencia natural a almacenar conocimiento experimental, haciéndolo apto para su uso. Se parece al cerebro en dos cosas:

El conocimiento es adquirido por la red a través de un proceso de aprendizaje

Ese conocimiento se almacena en los pesos sinápticos o conexiones entre neuronas"

Fausset; "Una red neuronal artificial es un sistema de procesado de la información que tiene ciertas características de funcionamiento en común con redes neuronales biológicas".

Hassoun; "Las redes neuronales son modelos computacionales compuestos de unidades de proceso adaptativas: las neuronas".

En todas las definiciones aparece el concepto de neurona como elemento individual de proceso. Son definiciones muy generales porque existen gran cantidad de modelos neuronales.

Algunas cuestiones a tener en cuenta.

Son modelos matemáticos que se ajustan a los datos que se tienen sin necesidad de hacer ninguna suposición a priori (los métodos bayesianos deben suponer alguna distribución en los datos; la de normalidad es la más usada).

Muchos de ellos suponen una generalización a métodos estadísticos usados desde hace mucho tiempo. Por ejemplo el perceptrón multicapa es una generalización de la regresión logística.

Se pueden establecer relaciones no lineales entre conjuntos de datos **sin necesidad de conocer el tipo de relación de antemano.** Por ejemplo, si se realiza un análisis multivariante y queremos introducir un efecto no lineal de las variables de entrada entonces debemos conocer la expresión matemática de ese efecto (logarítmico, exponencial, etc).

Como cualquier modelo matemático se pueden analizar y extraer conclusiones cualitativas de ellos. De hecho, por su gran flexibilidad es absolutamente necesario realizar esto (en la bibliografía existente se observa una ausencia de este análisis final).

Son de uso habitual en otras áreas de conocimiento por ser modelos no lineales, no paramétricos y con gran robustez al ruido en los datos; ¿por qué no usarlos en problemas clínicos?.

Un pequeño test sobre el problema a resolver

No se tiene (o no se plantea) ninguna suposición a priori sobre la cuestión a resolver. Si se tiene un modelo estructural del problema y es correcto este modelo no tendrá rival. ¿TENEMOS DICHO MODELO?.

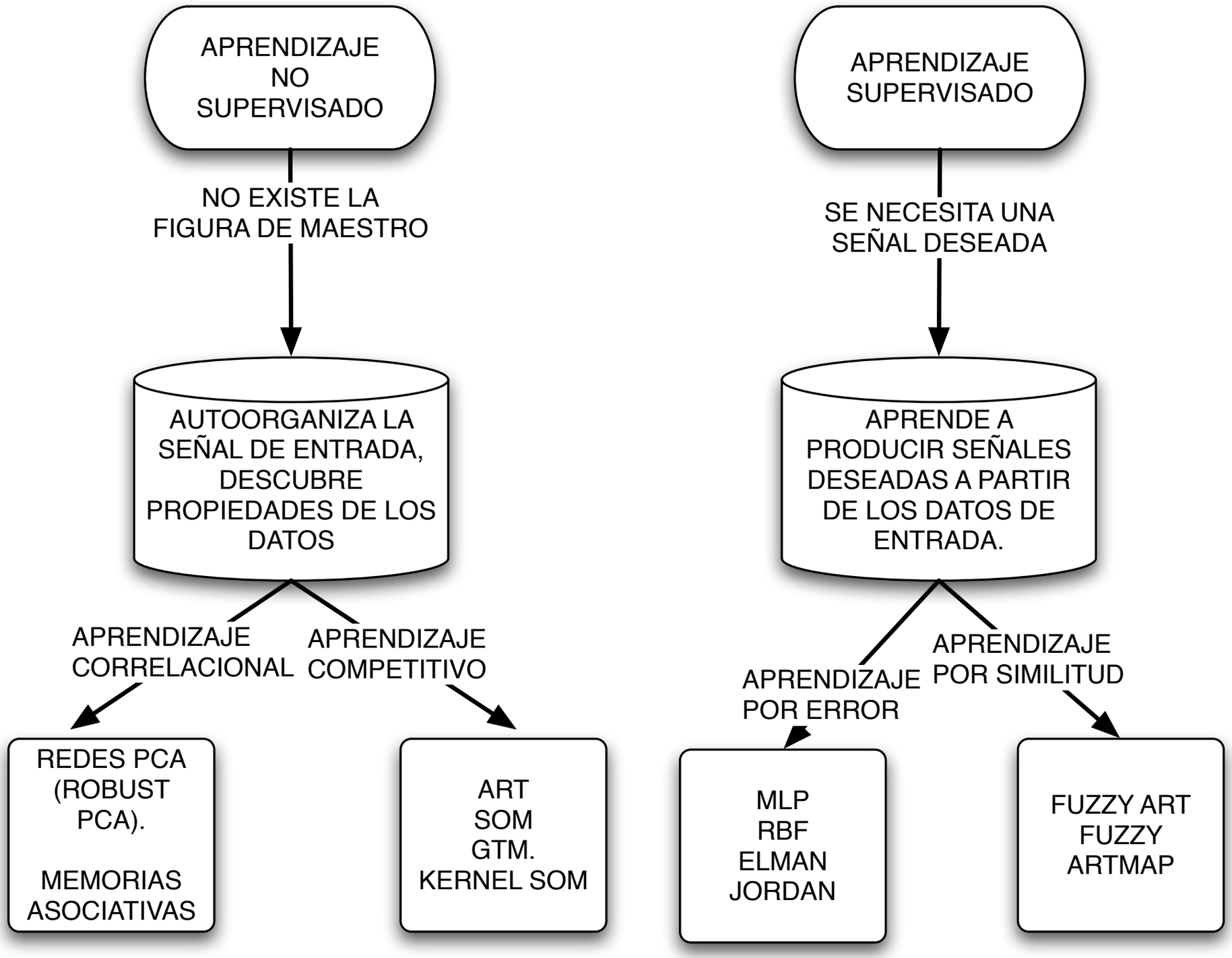
Los datos presentan imprecisiones bien por errores de los sensores, por error del experimentador, además algunos datos están incompletos... ¿NUESTROS DATOS SON PERFECTOS?.

El problema presenta a priori una alta complejidad ya que suponemos que aparecen efectos de memoria, saturación, relaciones no lineales.... ¿QUEREMOS UN MODELO SIMPLE PERO FALSO?.

Nuestro problema conlleva un gran número de variables a priori porque no conocemos si existe o no la relación que queremos establecer ¿VAMOS A PLANTEAR UN MODELO CON POCAS VARIABLES DE ENTRADA?.

RESULTADO DEL TEST:
Si ha contestado a la mayoría de preguntas NO su problema puede tener como solución un modelo neuronal artificial.

Una posible clasificación.



Existen muchos más modelos de los que aparecen en la figura aplicándose cada uno de ellos según las características del problema a resolver. A modo de ejemplo, en un problema de modelización con una fuerte componente local, sería más apropiado el uso de las RBF (Funciones de Base Radial) que el perceptron multicapa.

Algunos puntos importantes en redes neuronales

NO SON CAJAS NEGRAS: si se han entrenado correctamente se puede extraer nuevo conocimiento sobre el problema a resolver: obtención de nuevas conclusiones y posibilidad de obtener modelos neuronales más refinados.

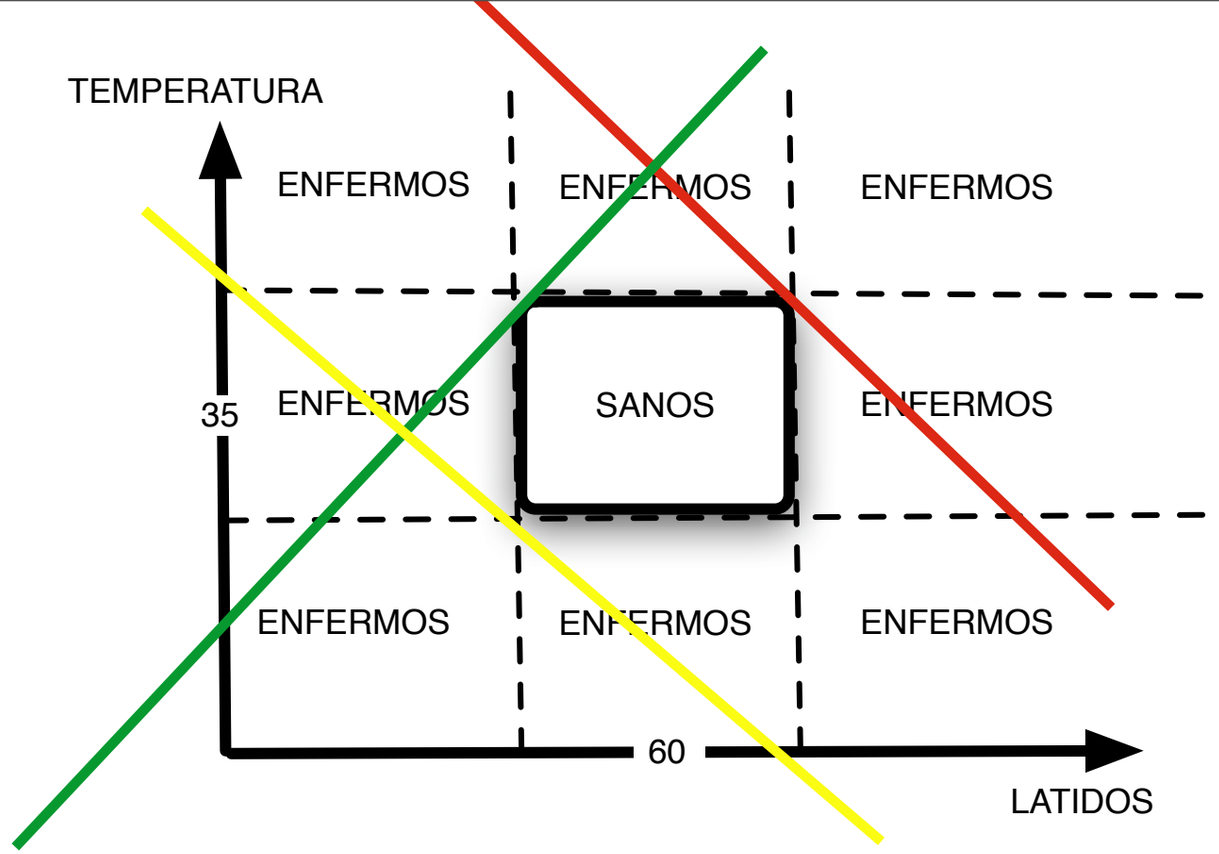
LOS ESTADÍSTICOS LAS REHUYEN: libros modernos de análisis multivariante las incluyen en sus últimos capítulos, importantes estadísticos tienen libros sobre redes neuronales tratadas como modelos estadísticos (B. Ripley, 1993) y paquetes de estadística (SPSS) las incluyen como librerías.

CUIDADO CON LOS PROGRAMAS (I): existe un gran cantidad de trabajos donde se han aplicado por no expertos dando los resultados numéricos como reflejo de la bondad de la red; SE NECESITA UNA VALIDACIÓN CUALITATIVA DEL MODELO; ¿es bueno un ajuste a un polinomio de 7º grado usando 8 datos?.

CUIDADO CON LOS PROGRAMAS (II): el otro peligro de las redes viene del otro extremo; en el anterior caso se sobreajustaban a los datos pero puede pasar que, debido a una mala elección de los parámetros las redes no aprendan el problema a resolver.....HAY QUE TENER EXPERIENCIA PARA LLEGAR A RESULTADOS CORRECTOS.

Problemas de los modelos lineales.

Un ejemplo simple.....supongamos que vivimos en un mundo donde el diagnóstico de las enfermedades viene dado por la temperatura corporal y el número de latidos/minuto. En un mundo así se tendría lo siguiente



Recopilamos datos y queremos sacar un modelo que, dadas las variables temperatura y latidos nos proporcione como salida la probabilidad que el paciente esté enfermo: USAMOS EL CLÁSICO: UNA REGRESIÓN LOGÍSTICA!!.....PERO.....

$$y = \frac{1}{1 + e^{-[w_0 + w_1 \cdot Latidos + w_2 \cdot Temperatura]}} \begin{cases} > (k)\ umbral & enfermo \\ < (k)\ umbral & sano \end{cases}$$

Superficie de separación y=k.

$$Temperatura = -\left(\frac{w_1}{w_2}\right) \cdot Latidos + \left[\ln\left(\frac{k}{1-k}\right) - w_0\right] \cdot \frac{1}{w_2}$$

$$Temperatura = A \cdot Latidos + B$$

A nivel geométrico la separación entre enfermos y sanos es una ¡¡ LÍNEA RECTA!!..NO PODEMOS RESOLVER ESTE PROBLEMA CON LA R.L.

Problemas de los modelos lineales (II).

Una solución a nuestro problema puede venir desde dos puntos de vista; uno geométrico y otro estadístico.

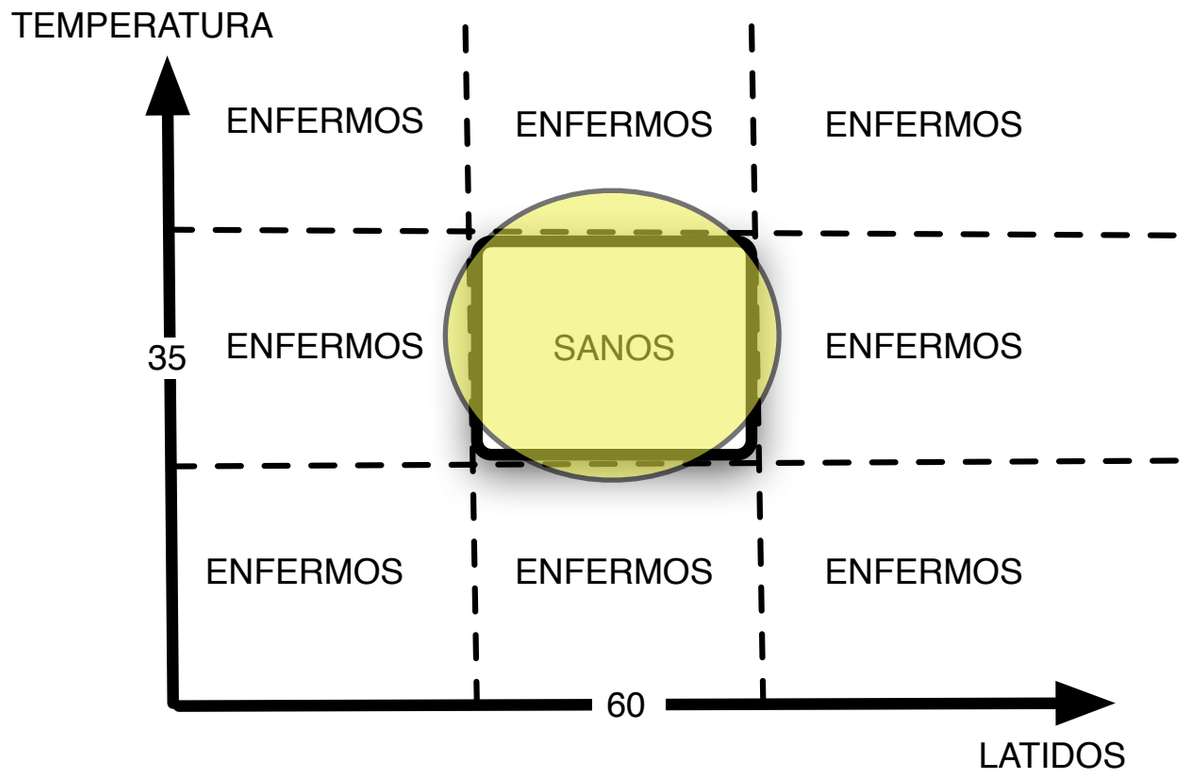
Transformamos las variables de entrada usando alguna transformación geométrica; en nuestro ejemplo una posible transformación sería

$$[z_1\ z_2\ z_3\ z_4\ z_5] = [T\ \ L\ \ T \cdot L\ \ T^2\ \ L^2]$$

Atención al aumento del número de variables; si hubiésemos tenido tres variables esta transformación me hubiese conducido a 10 variables nuevas; cuatro variables a 19....

MALDICIÓN DE LA DIMENSIONALIDAD

Usamos un modelo estadístico más avanzado.....atacamos el problema con otro GLM (Generalized Linear Model).



Ahora la superficie se separación es más compleja.....¡se pueden tener elipses!

En las dos aproximaciones hay una cuestión a tener en cuenta: LA ELECCIÓN DE LA TRANSFORMACIÓN GEOMÉTRICA Y DEL MODELO.....¿CÓMO LA HACEMOS?.... en un problema multidimensional no "vemos" el problema como en este caso.

Problemas de los modelos lineales (III)

Antes hemos visto un problema de clasificación.....veamos ahora uno de modelización.....

La herramienta más utilizada es la regresión múltiple.....tenemos ventajas....sencillez, gran cantidad de programas lo realizan, intervalos de confianza para los parámetros obtenidos y las predicciones.....PERO.....

$$y = w_0 + w_1 \cdot x_1 + ...+ w_n \cdot x_n$$

a) No explica efectos de "memoria"; histéresis, en general efectos no lineales.

b) Se tiene siempre la misma relación salida-entrada; si aumento/disminuyo la entrada aumenta/disminuye la salida (absolutamente falso en problemas médicos).

Fácil solución.....vamos a transformar las entradas !!!!

$$y = w_0 + w_1 \cdot \varphi_1(x_1) + ...+ w_n \cdot \varphi_n(x_n)$$

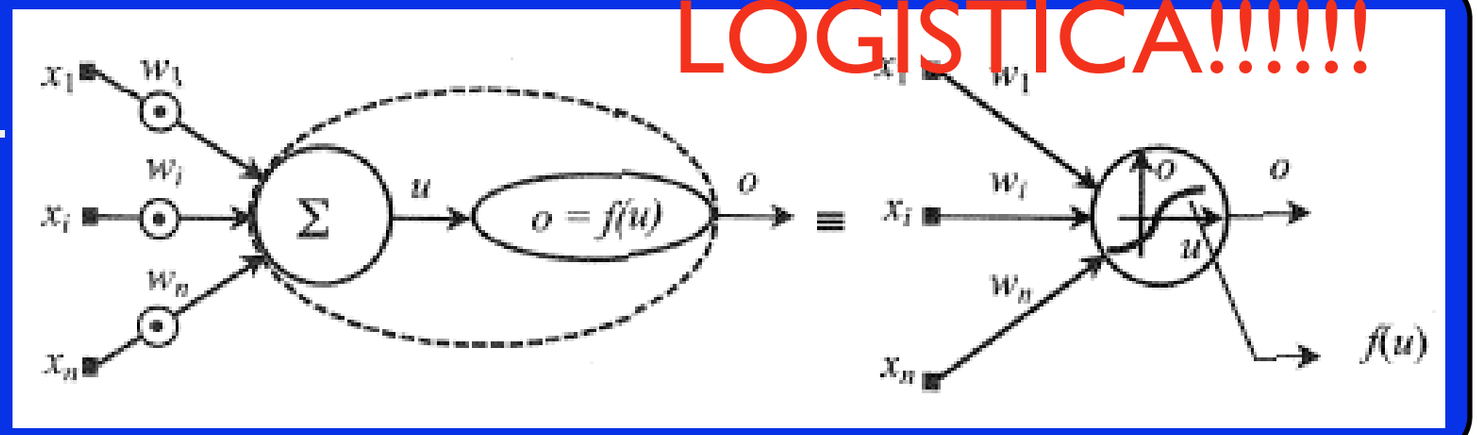
PERO.....¿QUÉ TRANSFORMACIÓN ESCOGEMOS PARA CADA UNA DE ELLAS?; ¿SABEMOS CON TOTAL CERTEZA ESA TRANSFORMACIÓN? SI LO SABEN HÁGANLA Y NO SE PREOCUPEN DE LAS REDES NEURONALES.... SI NO LA SABEN EL PERCEPTRON MULTICAPA PUEDE AYUDAR.

Neurona y Perceptron multicapa

ES UNA REGRESIÓN LOGÍSTICA!!!!!!

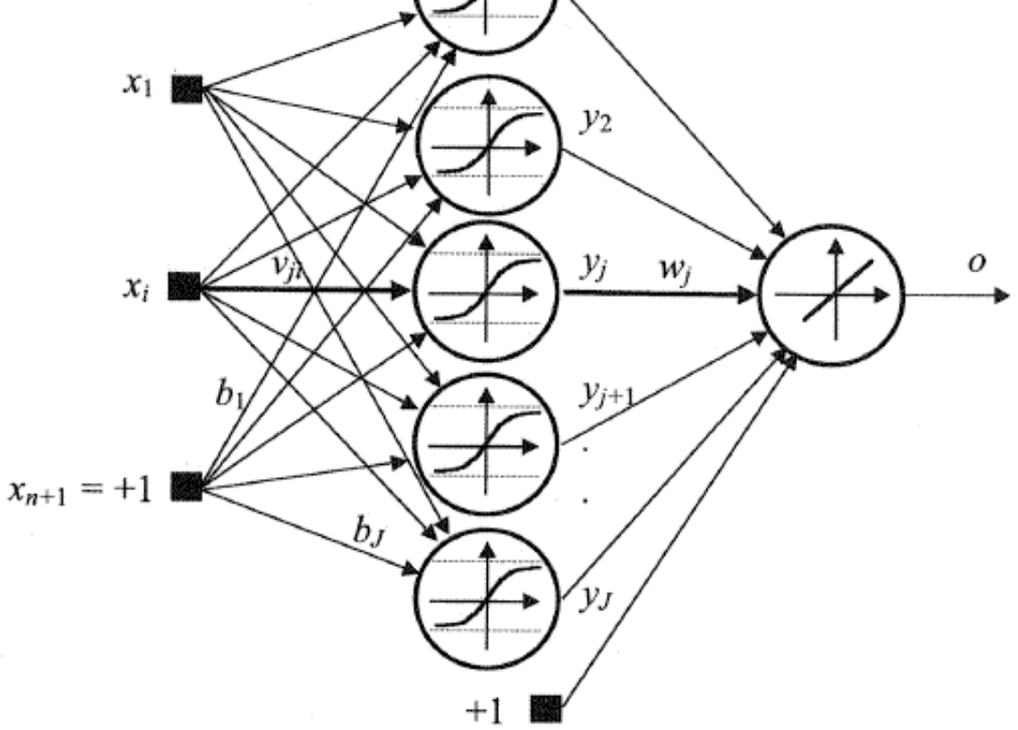
Elemento básico*: *neurona;

consta de un multiplicador (entradas-parámetros; ***pesos sinápticos***) seguido de un sumador y una función no lineal (normalmente la tangente hiperbólica).



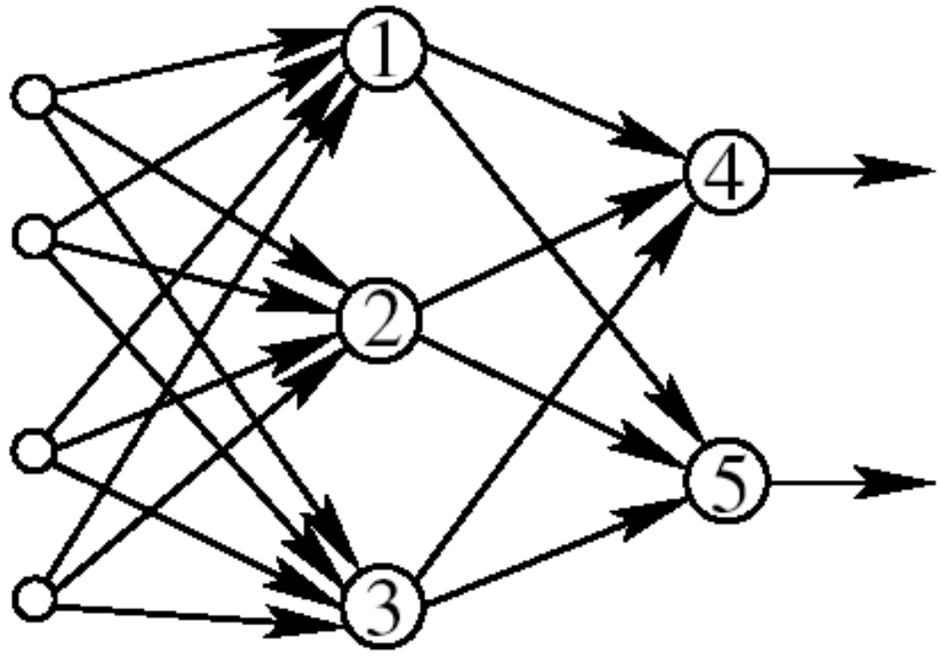
¡COMBINACIÓN DE REGRESIONES!

Arquitectura: se conoce bajo este nombre a la forma en la que se disponen las neuronas. En la red más extendida, ***el perceptrón multicapa (MLP)*** las neuronas se disponen en una serie de capas. La primera se conoce como capa de entrada y la última como capa de salida; las intermedias se conocen como capas ocultas. **Destacar su capacidad de aproximador universal; un MLP con dos capas ocultas es capaz de establecer cualquier mapeo entre dos conjuntos de datos (¡¡¡¡¡!!!!!).**



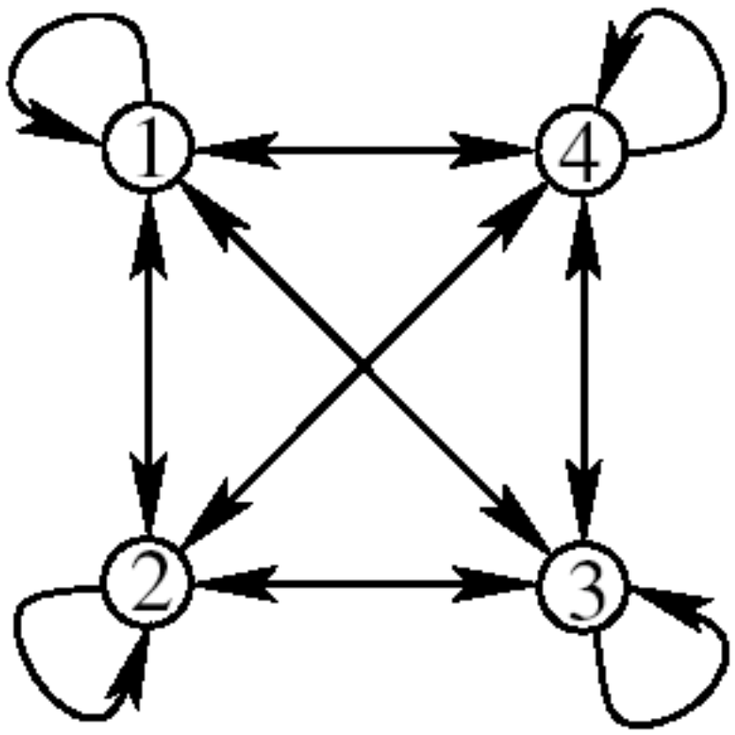
Tipos de estructuras multicapa

No recurrentes; en ese tipo de estructuras no se plantea ningún tipo de realimentación en la estructura. La señal sólo se propaga en un sentido; el modelo no tiene memoria. Este tipo será el que usaremos en este curso siendo el más extendido en cuanto a aplicaciones. Hay que destacar que, en aplicaciones temporales, podemos escoger como entradas del perceptrón la variable temporal en diferentes intervalos de tiempo, x(n), x(n-1), etc.



Neural Networks in a Softcomputing Framework, Springer

Recurrentes; aquí se tienen realimentaciones que pueden ser en una misma neurona, entre neuronas (pertenecientes a la misma capa o a diferentes capas) o entre la salida y la entrada del modelo. Este tipo de estructuras se utilizan para modelizar sistemas dinámicos (predicción de series temporales por ejemplo).



Perceptron multicapa.

La capacidad de modelización y flexibilidad del perceptron multicapa (MLP) es una gran ventaja y un serio inconveniente

Podemos establecer el modelo que buscamos entre dos conjuntos de variables SI EXISTE.

Podemos obtener auténticas burradas si no se maneja con cuidado el proceso de obtención de los parámetros

El MLP es un modelo no lineal que obtiene sus parámetros de métodos clásicos iterativos de optimización de funciones (métodos de gradiente, Newton, gradiente-conjugado, etc). La elección de la arquitectura sigue un método de prueba y error aunque existen métodos de poda y crecimiento).

Una vez se ha realizado correctamente el entrenamiento y validación de un MLP éste puede proporcionar:

a) Información sobre el ajuste obtenido

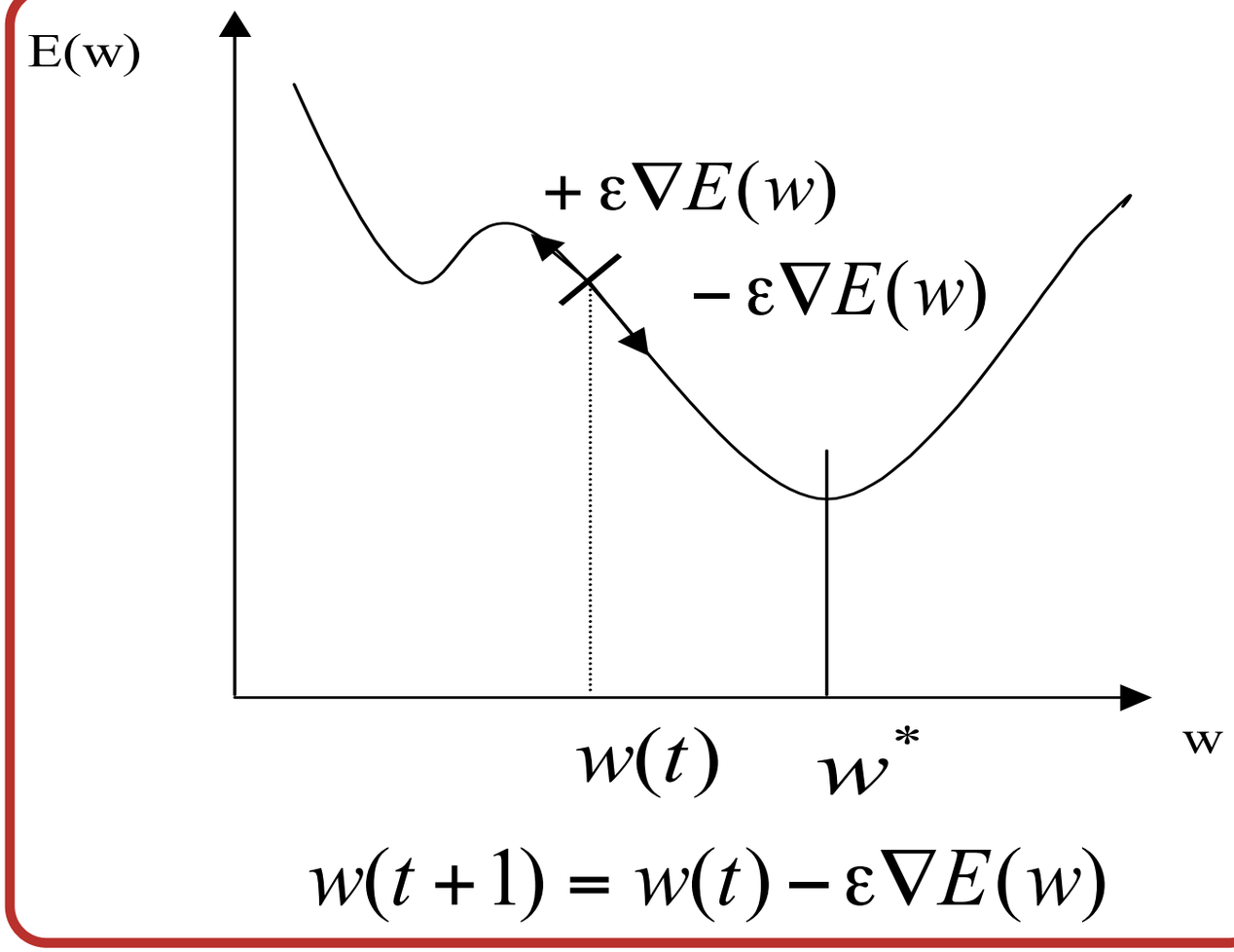
b) Información sobre la importancia relativa de las variables de entrada.

c) Podemos establecer dicho MLP como modelo del problema (puedo variar concentraciones de fármacos en el modelo sin tener que perjudicar al paciente).

d) Podemos obtener información cualitativa al usarlo junto con el SOM.

Regla Delta

El problema que se plantea en una red neuronal es exactamente el mismo que en un modelo lineal; hay unos parámetros a determinar usando la minimización del error cuadrático. Una posible forma es resolverlo de forma iterativa mediante lo que se conoce como **regla delta**. El problema, consiste en que el valor inicial de los parámetros nos puede conducir a un mínimo local; solución: ¡¡probamos diferentes valores iniciales con diferentes entrenamientos!!



$$w(t+1) = w(t) - \varepsilon\nabla E(w)$$

¿Dónde aplicamos la regla delta?

Esta regla se aplica en aquellos problemas en los que la solución directa no conduce a ecuaciones lineales en los parámetros. En el métodos de mínimos cuadrados se llegaba a ecuaciones lineales al derivar parcialmente e igualar a cero esas derivadas.

¿Cómo aplicamos la regla delta?

La forma de aplicarla es siempre la misma; los parámetros se inicializan de forma aleatoria; se determina la salida del modelo y el error cometido se usa (junto con otros factores) para modificar esos parámetros y obtener un mejor funcionamiento.....es un procedimiento iterativo

Repaso de mínimos cuadrados

En plan de repaso; en el método de mínimos cuadrados se planteaba una función de coste que dependía de los parámetros del modelo y el objetivo era minimizar esa función de coste.

$$J_1[e(n)] = e^2(n)$$

$$J(n) = \sum_{k=1}^{N} J_1(k) = \sum_{k=1}^{N} e^2(k)$$

Donde el error era la diferencia entre el valor que se quiere ajustar y el que da el modelo.

$$e(n) = [y_R(n) - y_M(n)]$$

Si planteamos una regresión simple

$$e(n) = [y_R(n) - \{b_o + b_1 \cdot x(n)\}]$$

A nivel matemático se deriva con respecto a cada parámetro y se iguala a cero.

$$\frac{\partial J(n)}{\partial b_0} = 0 \Leftrightarrow A \cdot b_o + B \cdot b_1 = C$$

$$\frac{\partial J(n)}{\partial b_1} = 0 \Leftrightarrow D \cdot b_o + E \cdot b_1 = F$$

Se llega a un sistema de dos ecuaciones con dos incógnitas en las que hay que obtener b_0 y b_1 siendo A,B,C, D,E y F las siguientes cantidades (B=D).

AHORA ES INMEDIATO OBTENER LOS PARÁMETROS DEL MODELO

$$A = N \Leftrightarrow B = \sum_{k=1}^{N} x(k) \Leftrightarrow C = \sum_{k=1}^{N} y_R(k)$$

$$E = \sum_{k=1}^{N} x^2(k) \Leftrightarrow F = \sum_{k=1}^{N} x(k) \cdot y_R(k)$$

Regla delta y regresión robusta (I).

Queremos implementar una regresión simple robusta usando otra función de coste en este caso la del módulo del error

$$J(n)=\sum_{k=1}^{N} J_2(k)=\sum_{k=1}^{N}\left|e(k)\right|$$

Derivamos parcialmente con respecto a los parámetros; ahora hay que derivar un módulo (derivada, función signo)

$$\frac{\partial J(n)}{\partial b_0}=0 \Leftrightarrow \sum_{k=1}^{N} signo\left[e(k)\right]=0$$

$$\frac{\partial J(n)}{\partial b_1}=0 \Leftrightarrow \sum_{k=1}^{N} signo\left[e(k)\right]\cdot x(k)=0$$

Las últimas ecuaciones no tienen una resolución directa; hay que encontrar los parámetros que cumplan.

$$\sum_{k=1}^{N} signo\left\{y_R(k)-\left[b_0+b_1\cdot x(k)\right]\right\}=0$$

Aplicando la regla delta

$$b_0=b_0-\alpha\cdot\frac{\partial J(n)}{\partial b_0}$$

$$b_1=b_1-\alpha\cdot\frac{\partial J(n)}{\partial b_1}$$

Esa regla conduce a la siguientes reglas de actualización.

$$b_0=b_0+\alpha\cdot\sum_{k=1}^{N} signo\left[e(k)\right]$$

$$b_1=b_1+\alpha\cdot\sum_{k=1}^{N} signo\left[e(k)\right]\cdot x(k)$$

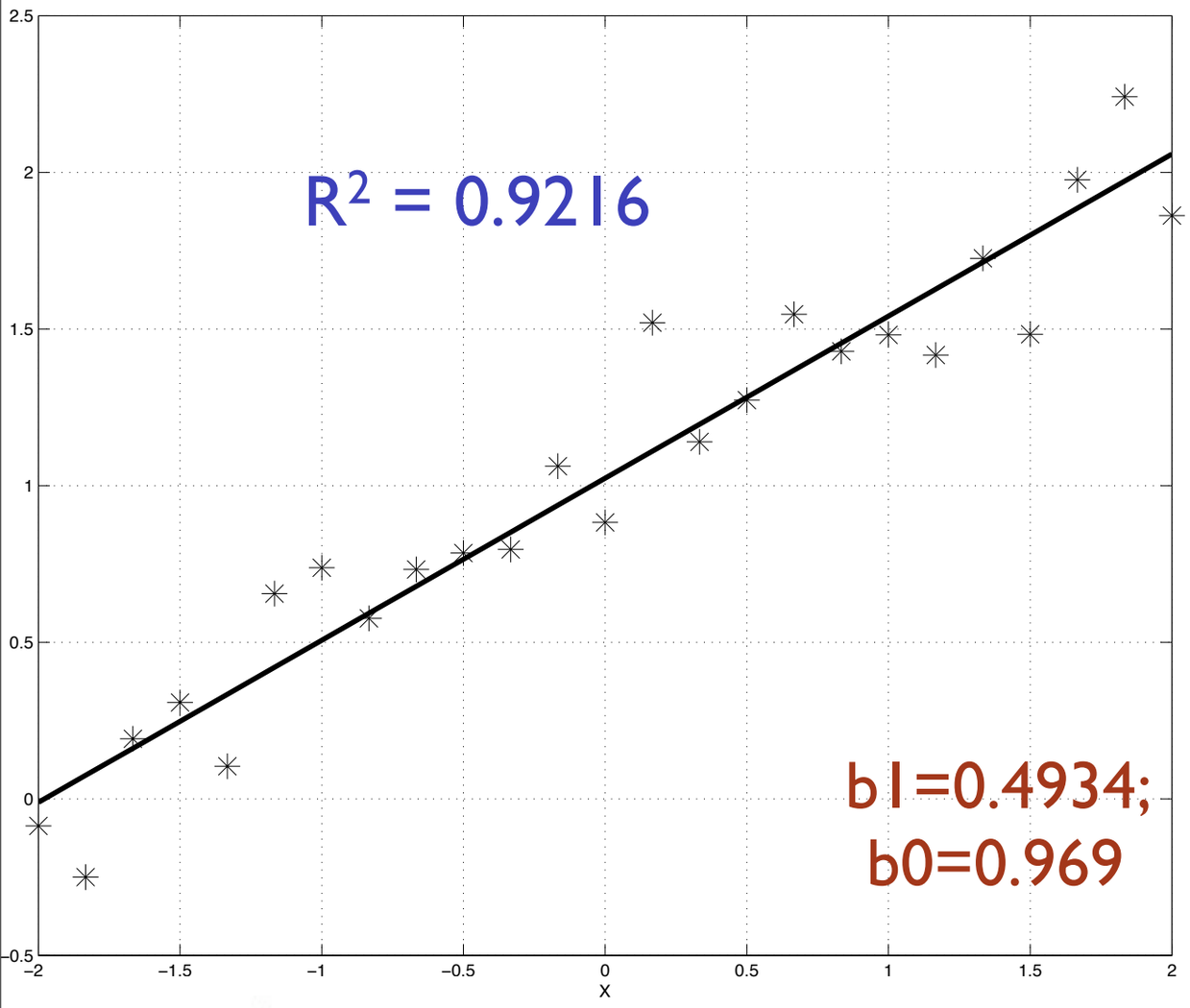
Regla delta y regresión robusta. MATLAB (I)

El ejemplo que vamos a plantear es el siguiente; generamos 25 datos de acuerdo a

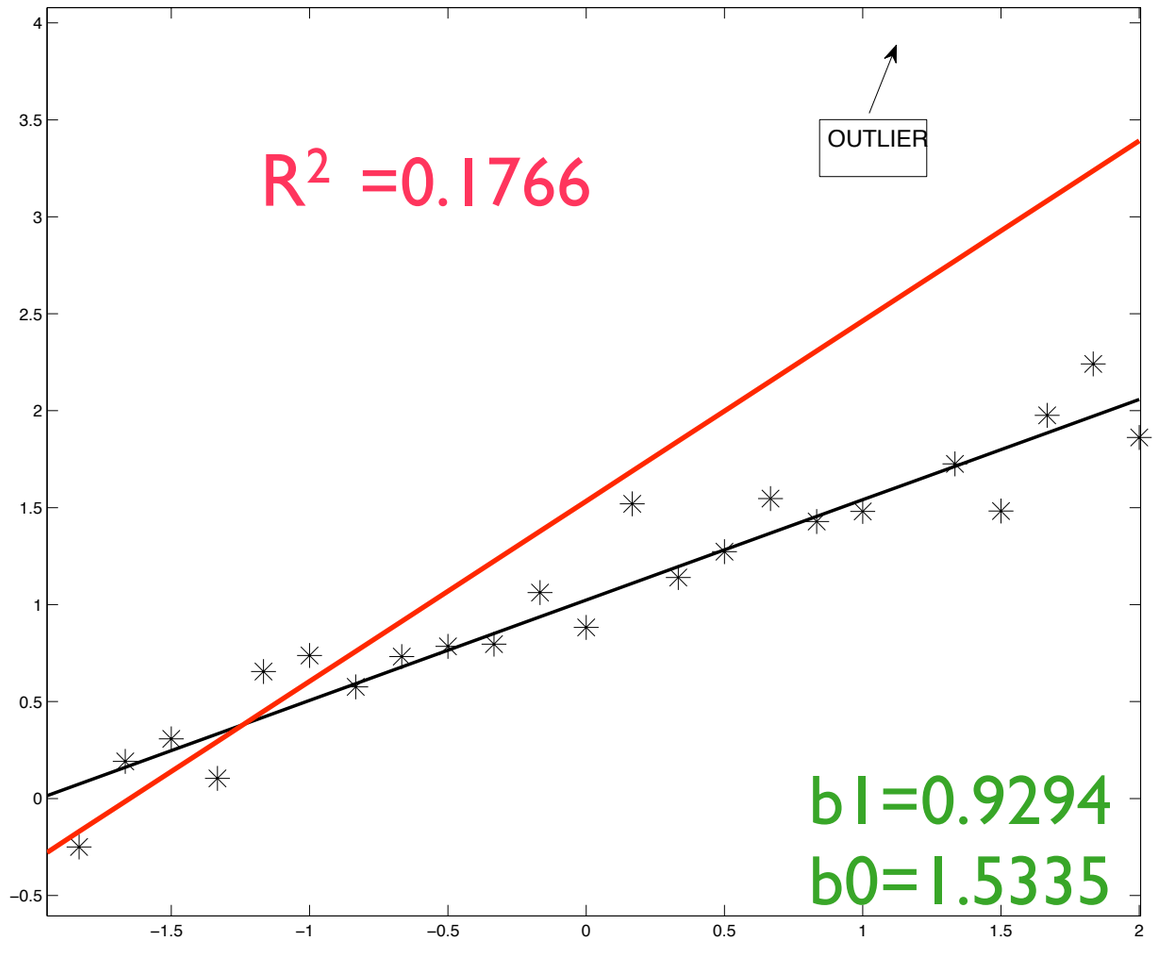
$$y_i = \frac{1}{2} \cdot x_i + 1 + \varepsilon_i$$

Aquí la variable x está en el intervalo [-2 2] y la variable ε_i es de tipo normal, media cero y desv. estándar 0.2.

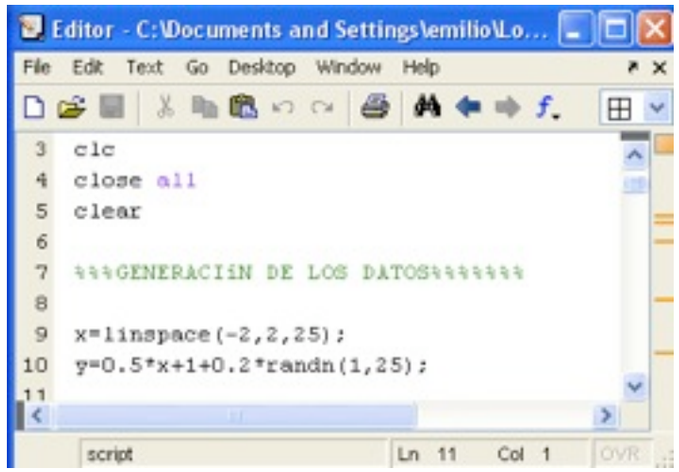
Al hacer un ajuste de mínimos cuadrados (se puede hacer con las instrucciones *polyfit y regress*) obtenemos los siguientes resultados.



Si introducimos un outlier obtenemos



Regla delta y regresión robusta. MATLAB (II)



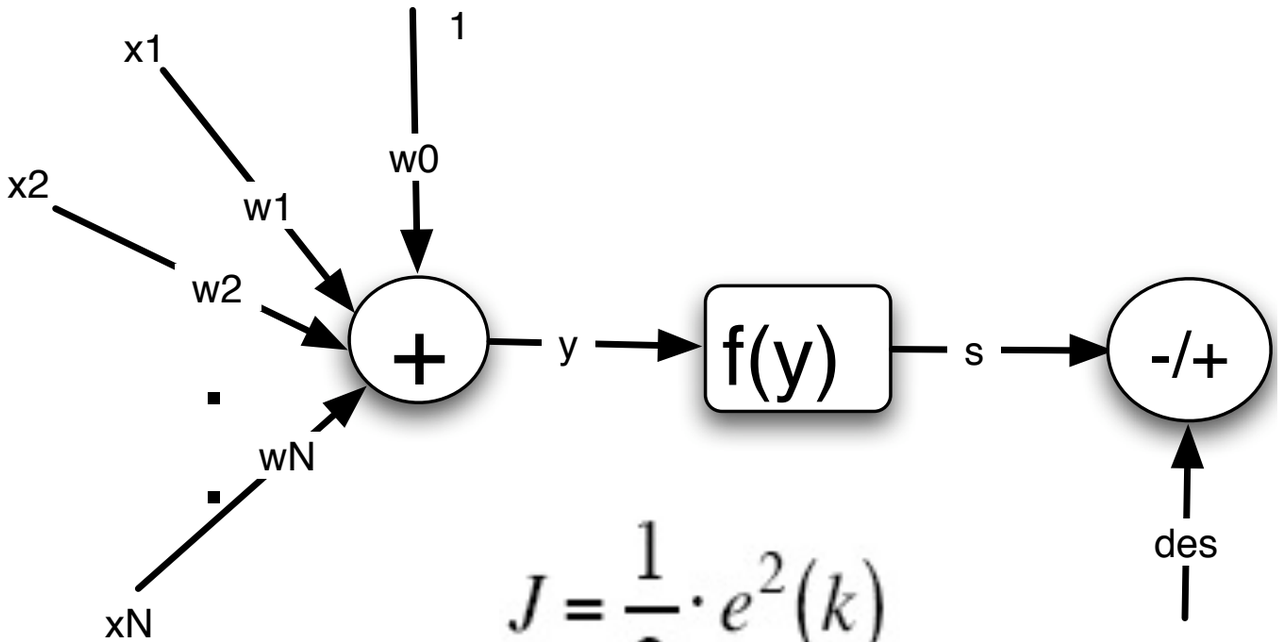

```
%%%%AñADIMOS UN"OUTLIER";
y(20)=factor*y(20);
[A,B,C,R,stats]=regress(y',xx);

A
stats(1)
pause
clc
```

```
%%%REGREsIÓN ROBUSTA %%%%%%%%%%%
%%%INICIALIZAMOS LOS COEFIC.%%%
b1=0.1*rand(1,1);
b0=0.1*rand(1,1);
coef=[b1 b0];
alpha=0.01;
%%%PLANTEAMOS UN BUCLE PARA CALCULAR
%%%%%%LOS PARÁMETROS %%%%%%%%%%%%%%

XX=[x' ones(25,1)];

for k=1:5000,

%%%calculamos los errores

yy=coef*XX';
e=y-yy;
aux=sign([e' e']);
coef=coef+alpha*sum(aux.*XX);

end

coef
pause
```

```
%%%%EXISTE OTRA OPCIÓN MÁS FÁCIL%%%%%%
%%%%%%%%%TIPO ONLINE %%%%%%%%%%%%%%%%%%
b1=0.1*rand(1,1);
b0=0.1*rand(1,1);
coef=[b1 b0];

for t=1:1000

for k=1:25,

%%%calculamos los errores

yy=coef*[x(k) 1]';
e=y(k)-yy;
aux=sign(e);
coef=coef+alpha*aux*[x(k) 1];

end

end
coef
```

Regla delta en una neurona (I)

Se plantea el modelo (neurona) así como la función de coste a minimizar.



$$J = \frac{1}{2} \cdot e^2(k)$$

Donde el error es la diferencia entre la señal deseada, des, y la señal de salida de la neurona, De acuerdo a la actuación de la neurona se tiene lo siguiente:

$$y = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots\ldots + w_N \cdot x_N$$

$$s = f(y) \Leftrightarrow s = \frac{1 - e^{-\beta \cdot y}}{1 + e^{-\beta \cdot y}} \quad -1<s<1$$

Si se aplica la regla delta a un determinado coeficiente

$$w_1 = w_1 - \alpha \cdot \frac{\partial J}{\partial w_1}$$

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial e} \cdot \frac{\partial e}{\partial s} \cdot \frac{\partial s}{\partial y} \cdot \frac{\partial y}{\partial w_1}$$

$$\frac{\partial J}{\partial e} = 2 \cdot e \qquad \frac{\partial e}{\partial s} = -1$$

$$\frac{\partial s}{\partial y} = \frac{1}{2} \cdot \left(1 - s^2\right) \qquad \frac{\partial y}{\partial w_1} = x_1$$

Por lo tanto la actualización quedará

$$w_1 = w_1 + \frac{1}{2} \cdot \alpha \cdot e \cdot \left[1 - s^2\right] \cdot x_1$$

Regla delta en una neurona (I)

ALGORITMO
0-Se inicializan de forma aleatoria los coeficientes y se le da un valor a la constante de adaptación.
1-Se coge otro patrón de entrada y se determina la salida de la neurona.
2-Se determina el error cometido por la neurona.
3-Se actualizan los coeficientes de la neurona.
4- Si no se da la condición de parada volvemos al paso 1.

Algunas cuestiones importantes.

No es necesario conocer nada del problema para resolverlo NO HAY QUE DESPEJAR NADA; según este tipo de procedimientos lo fundamental es hacer una buena selección de entradas.

La inicialización en una neurona da igual, sólo tenemos un mínimo global, esto no ocurrirá al considerar estructuras multicapa .

El cambiar una función de coste SÓLO supone cambiar un término de la actualización; el correspondiente a la derivada de la función de coste con respecto del error.

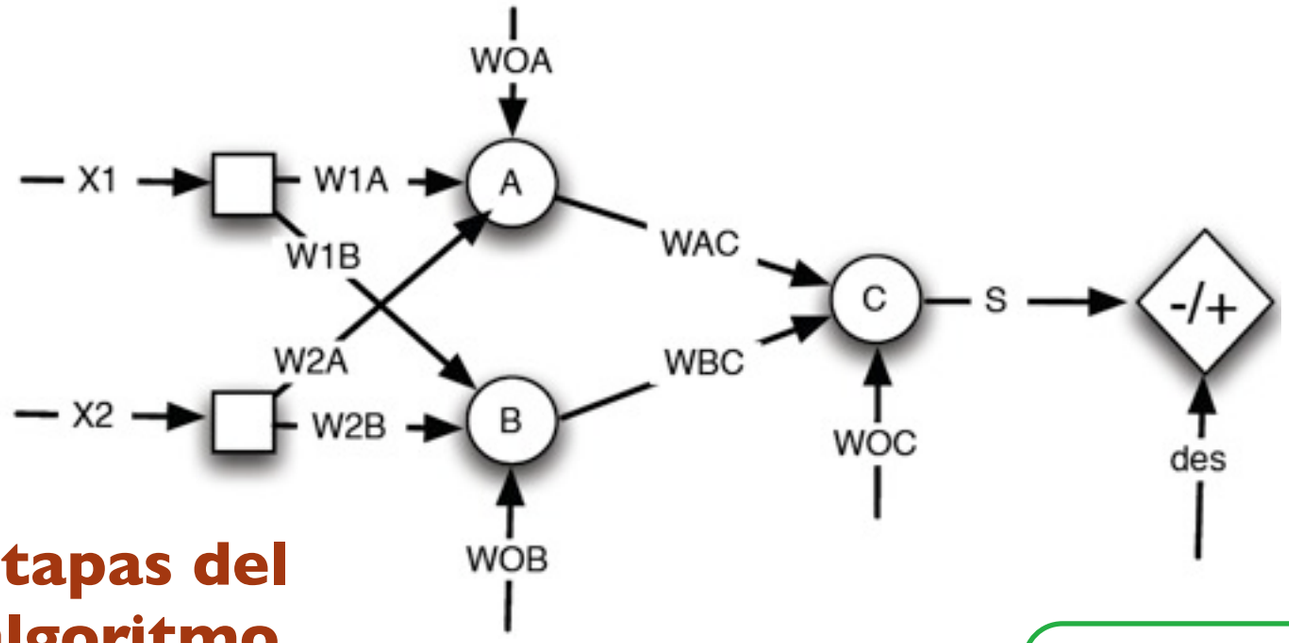
Este algoritmo es fácilmente factorizable y, por lo tanto, directamente implementable en Matlab.

Regla delta en una neurona. MATLAB

```
%%% PROGRAMA QUE IMPLEMENTA LA REGLA DELTA EN %%
%%%% UNA NEURONA %%%%%%%%%%%%%%%%%%%%
close all
clc
clear
%%%EN ESTE PROBLEMA ES UNO DE CLASIFICACIîN 2-D%
%%%%%%PLANTEAMOS UNA PUERTA AND %%%%%%%%%%%%%%%%
XX=[-1 1; -1 1; 1 -1; 1 1];
des=[-1 -1 -1 1];
%%%PIDO LA CONSTANTE DE ADAPTACIîN %%%%%%%%%%
alpha=input('Constante de adaptaci—n ');
%%%%Inicializo los coeficientes %%%%%%
%%%de forma aleatoria %%%%%%%%%%%%%%%%
W=randn(1,3);
contador=zeros(1,100);
```

```
for m=1:100 % le pasamos 100 vecesel conjunto de datos%%
    for k=1:4,
        %%%CçLCULO DE LA SALIDA %%%%%%%%%%%
        %%%CONSTRUYO EL VECTOR DE ENTRADA%%
        x=[1 XX(k,:)];
        %%determino la salida de la neurona%%%
        y=W*x';
        s=tanh(y);
        %%%determino el error %%%%%%%%%%%%%%%%
        e(k)=des(k)-s;
       %%%actualizo lo coeficientes %%%%%%%%
        W=W+alpha*e(k)*0.5*(1-s^2)*x;
       %%%almaceno el sumatorio de los errores^2 (À?).
        contador(m)=contador(m)+e(k)^2;
    end
end

plot(contador)
```

Regla delta y perceptrón multicapa (MLP)



Ahora se plantea el mismo procedimiento, regla delta, pero se nos complica porque tenemos una estructura multicapa. Sólo hay que tener en cuenta la salida de cada una de las neuronas.

Etapas del algoritmo

Inicialización aleatoria de los coeficientes; se fijan las constantes de adaptación.

1- Se coge un patrón de entrada y se determina su salida

2- Se determina el error cometido

3- Actualización de los coeficientes del MLP.

4- Si no se cumple el criterio de parada se vuelve al paso 1.

Determinación de la salida de la red.

Neurona A

$$y_A = w_{0A} + w_{1A} \cdot x_1 + w_{2A} \cdot x_2$$

$$s_A = f(y_A) \Leftrightarrow s_A = \frac{1 - e^{-\beta \cdot y_A}}{1 + e^{-\beta \cdot y_A}}$$

Neurona B

$$y_B = w_{0B} + w_{1B} \cdot x_1 + w_{2B} \cdot x_2$$

$$s_B = f(y_B) \Leftrightarrow s_B = \frac{1 - e^{-\beta \cdot y_B}}{1 + e^{-\beta \cdot y_B}}$$

Neurona C

$$y_C = w_{0C} + w_{AC} \cdot s_A + w_{BC} \cdot s_B$$

$$s = f(y_C) \Leftrightarrow s = \frac{1 - e^{-\beta \cdot y_C}}{1 + e^{-\beta \cdot y_C}}$$

Regla delta y perceptrón multicapa (II)

Cálculo del error

$$e = des - s$$

Actualización de los coeficientes (capa de salida).

Aplicamos la regla delta a los coeficientes (suponemos una función de coste cuadrática).

$$w_{AC} = w_{AC} - \alpha \cdot \frac{\partial J}{\partial w_{AC}}$$

$$\frac{\partial J}{\partial w_{AC}} = \frac{\partial J}{\partial e} \cdot \frac{\partial e}{\partial s} \cdot \frac{\partial s}{\partial y_C} \cdot \frac{\partial y_C}{\partial w_{AC}}$$

Determinando todas las derivadas parciales se obtiene lo siguiente:

$$w_{AC} = w_{AC} + \alpha \cdot e \cdot \left[1 - s^2\right] \cdot o_A$$

donde todas las constantes quedan englobadas en α

Actualización de los coeficientes (capa oculta).

En esta capa es donde aparecen problemas en la actualización. Si se plantea la regla delta:

$$\frac{\partial J}{\partial w_{1A}} = \frac{\partial J}{\partial e} \cdot \frac{\partial e}{\partial s} \cdot \frac{\partial s}{\partial y_C} \cdot \frac{\partial y_C}{\partial o_A} \cdot \frac{\partial o_A}{\partial y_A} \cdot \frac{\partial y_A}{\partial w_{1A}}$$

Determinando cada derivada se llega a:

$$w_{1A} = w_{1A} + \alpha \cdot e \cdot \left[1 - s^2\right] \cdot w_{AC} \cdot \left[1 - s_A^2\right] \cdot x_1$$

El resto de coeficientes se calcularían de forma similar. **El algoritmo obtenido se conoce con el nombre de algoritmo de retropropagación (BP)**. Se ha obtenido un modelo matemático no lineal en el que sus parámetros se obtienen de forma iterativa sin necesidad de un conocimiento previo del problema. Existen dos tipos de algoritmos on-line (se actualiza patrón a patrón) y batch (se actualiza al final).

Perceptrón multicapa y Matlab (I).

```
%%% PROGRAMA PARA RESOLVER UN PROBLEMA DE UNA PUERTA XOR %%%'
%%%%% USANDO UN PERCEPTRIN MULTICAPA %%%%%%%%%%%%%%%%%%%%%%%%%'

clear
clc
close all

%%%DEFINIMOS EL PROBLEMA A RESOLVER %%%%%%%%%%%%%%%%
X=[-1 -1;-1 1;1 -1;1 1];
des=[-1 1 1 -1];

%%%%INICIALIZAMOS LOS PESOS%%%%%%%%
%%NEURONA A %%%%%%%%%%%%%%%%%%%%%%%
w1a=randn(1,1);
w2a=randn(1,1);
woa=randn(1,1);

%%NEURONA B %%%%%%%%%%%%%%%%%%%%%%%
w1b=randn(1,1);
w2b=randn(1,1);
wob=randn(1,1);

%%NEURONA C %%%%%%%%%%%%%%%%%%%%%%%
wac=randn(1,1);
wbc=randn(1,1);
woc=randn(1,1);

%%%CONSTANTE DE ADAPTACIÓN %%%%%%
alpha=0.2;
```

```
for k=1:100 %%BUCLE DE ƒPOCAS %%%

    for t=1:4,
        %%%DATOS%%%%%%%%%%%%%%%%
        x1=X(t,1);
        x2=X(t,2);
        %SALIDA DE LA NEURONA A
        ya=woa+w1a*x1+w2a*x2;
        oa=tanh(ya);
        %SALIDA DE LA NEURONA B
        yb=wob+w1b*x1+w2b*x2;
        ob=tanh(yb);
        %SALIDA DE LA NEURONA C
        yc=woc+wac*oa+wbc*ob;
        oc=tanh(yc);

        %%%DETERMINACIÓN DEL ERROR %%%%
        e=des(t)-oc;
        contador(k)=contador(k)+(e^2);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%ACTUALZIACIÓN DE LOS PESOS %%%%%
        %%%CAPA OCULTA%%%%%%%%%%%%%%%%%%%%%%%

        %%%%%NEURONA A %%%%%%%%%%%%%%%%%%%%%%
        woa=woa+alpha*e*(1-oc^2)*wac*(1-oa^2);
        w1a=w1a+alpha*e*(1-oc^2)*wac*(1-oa^2)*x1;
        w2a=w2a+alpha*e*(1-oc^2)*wac*(1-oa^2)*x2;

        %%%%%NEURONA B %%%%%%%%%%%%%%%%%%%%%%
        wob=wob+alpha*e*(1-oc^2)*wbc*(1-ob^2);
        w1b=w1b+alpha*e*(1-oc^2)*wbc*(1-ob^2)*x1;
        w2b=w2b+alpha*e*(1-oc^2)*wbc*(1-ob^2)*x2;

        %%%%CAPA DE SALIDA %%%%%%%%%%%%%%%%%%%%

        woc=woc+alpha*e*(1-oc^2);
        wac=wac+alpha*e*(1-oc^2)*oa;
        wbc=wbc+alpha*e*(1-oc^2)*ob;

    end

end
```

Perceptrón multicapa y Matlab (II).

Si en la implementación anterior se introducen más entradas/neuronas entonces hay que aumentar el número de ecuaciones por lo que se hace necesario otra implementación más óptima.....lo que se hace es expresar ese algoritmo en forma matricial.

```
%%%%PROGRAMA QUE RESUELVE EL PROBLEMA DE LA XOR%%%%%
%%%MATRICIALMENTE Y CON EL ALGORITMO DE MOMENTO %%%%
close all; clear; clc
%%% Introducción de variables%%%%%%%%%%%%%%%%%
numepoch=input('Número de épocas ');
NN=input('Número de neuronas ocultas ');
ca=input('Constante de adaptación ');
cm=input('Constante de momento ');
nout=1;
%%%%%%%%%%Inicialización de variables%%%%%%%%%%%%%%%%%%%%%
W1=0.5*randn(NN,2);
inc1=zeros(NN,2);
W2=0.5*randn(1,NN);
inc2=zeros(1,NN);
s1=0.05*randn(NN,1);
incc1=zeros(NN,1);
s2=0.05*randn(1,1);
incc2=zeros(1,1);
e=zeros(1,4);
error=zeros(1,numepoch);
%%%DATOS DE LA XOR%%%%%%%%%%%%
xx=[-1 -1;-1 1;1 -1;1 1];
des=[-1 1 1 -1];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%Propagación de la señal hacia adelante%%%%%%%%%%%%%%%
for t=1:numepoch,
    for k=1:4
    x=(xx(k,:))';
    %%%%Primera Capa%%%%%%%%%
    y=W1*x+s1;
    yy=tanh(y);
    %%%%Capa de salida%%%%%%%%%
    z=W2*yy+s2;
    zz=tanh(z);
    %%%%%Cálculo del error%%%%%%%%%%%%
    e(k)=des(k)-zz;
    %%%Cálculo de los deltas%%%%%%%
    delta1=e(k)*(1-zz.^2);
    delta2=W2'*delta1.*((1-yy.^2));
    %%%%Actualización de los pesos capa salida%%%%%%%%%%
    aux2=W2;
    auxx2=s2;
    W2=W2+ca*delta1*yy'+cm*inc2;
    s2=s2+ca*delta1+cm*incc2;
    inc2=W2-aux2;
    incc2=s2-auxx2;
    %%%%%Actualización de los coeficientes de la capa entrada%%
    aux1=W1;
    auxx1=s1;
    W1=W1+ca*delta2*x'+cm*inc1;
    s1=s1+ca*delta2+cm*incc1;
    inc1=W1-aux1;
    incc1=s1-auxx1;
    end
error(t)=mean(e.^2);
end
```

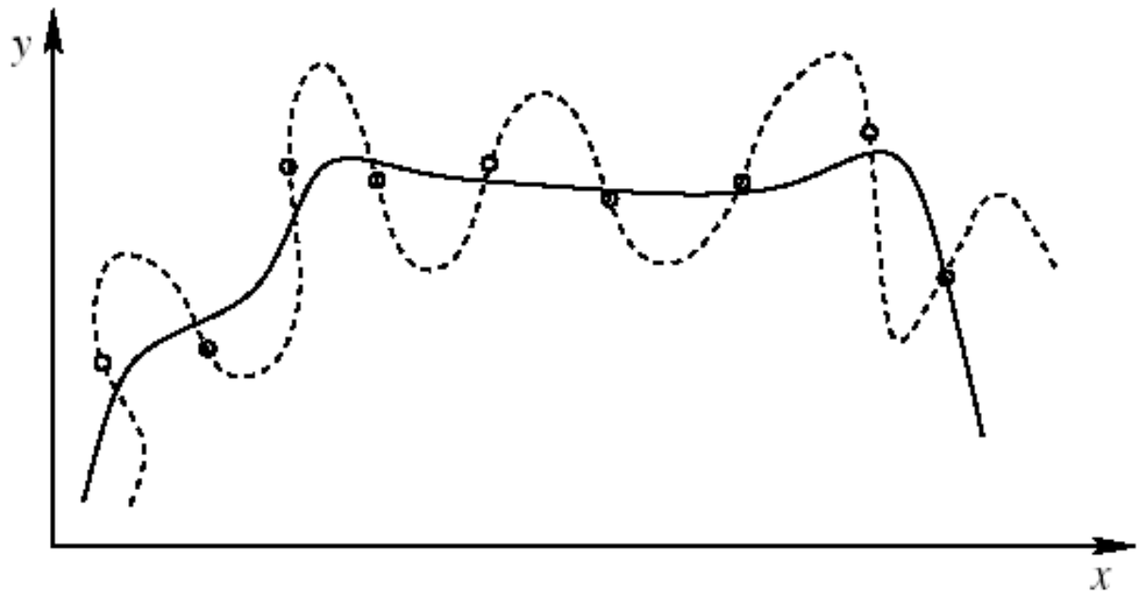
Validación.

Una red neuronal es capaz de establecer cualquier relación entre dos conjuntos

Ventaja: Se tiene un modelizador universal

Inconveniente: Se tiene un modelizador demasiado flexible

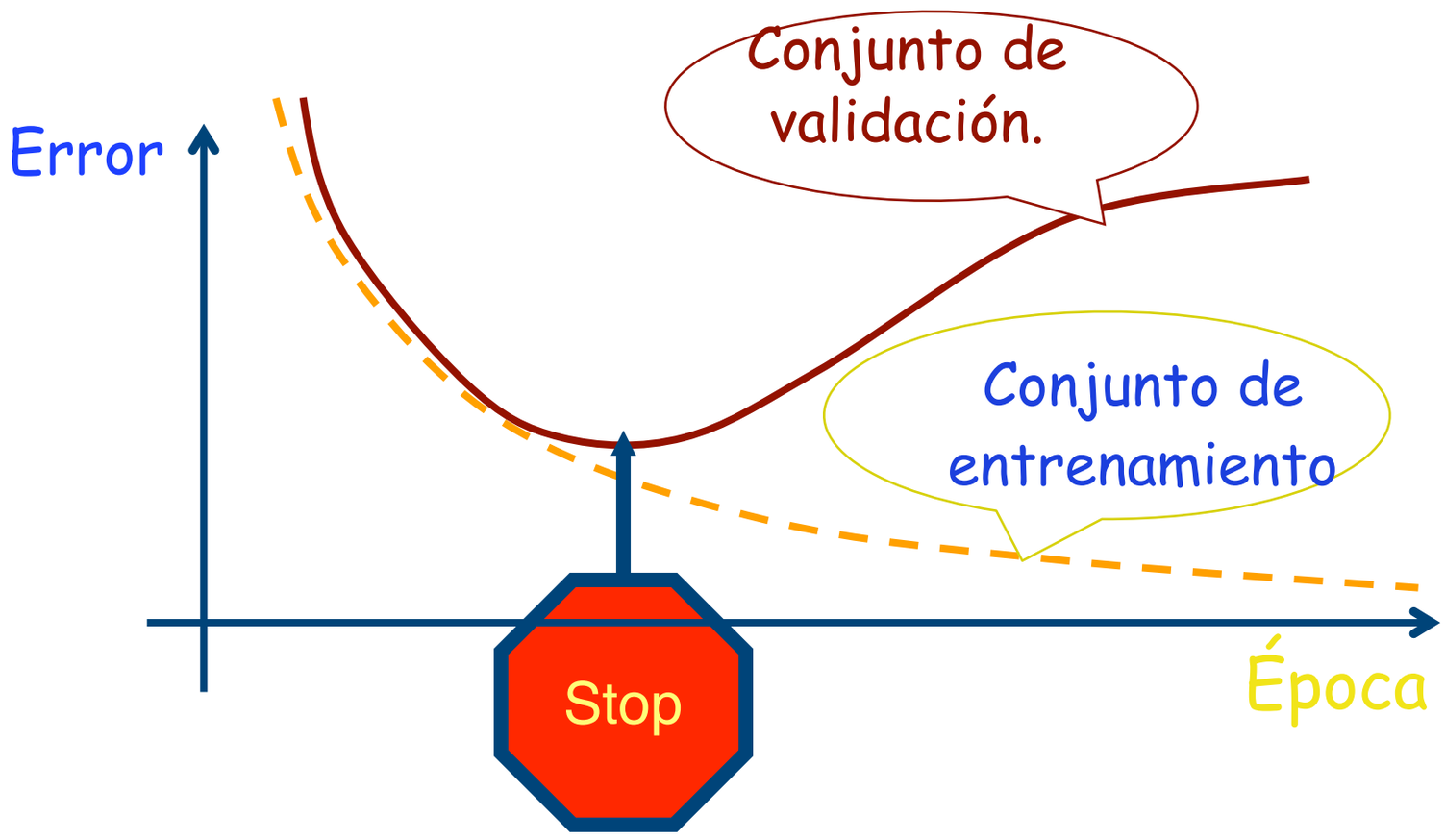
Neural Networks in a Softcomputing Framework, Springer



Es necesario definir dos conjuntos dentro de nuestros datos; conjunto de entrenamiento y validación

El conjunto de validación sirve para controlar el **overfitting** del modelo. Existen muchas estrategias para realizar esta validación.

Evolución del aprendizaje



Elección de la arquitectura de la red neuronal.

El teorema de Cybenko asegura que un percetrón multicapa (MLP) con dos capas ocultas puede establecer una relación entre dos conjuntos pero no especifica el número de neuronas que debe tener cada capa.

Hay tres tipos de aproximaciones en este problema; una aproximación "constructiva" otra "destructiva" y, por último el método de prueba/error. La primera consiste en partir de pocas neuronas y añadir éstas si se comprueba que el funcionamiento de la red ha mejorado. La segunda aproximación parte de redes de gran tamaño y elimina, o bien pesos sinápticos,o bien neuronas; esta última aproximación también se conoce como métodos de poda.

Dada la capacidad de los ordenadores el método de prueba y error, a día de hoy, puede ofrecer mejores resultados que las otras dos aproximaciones.

Entre las aproximaciones constructivas y las de poda están más extendidas las de poda y, entre ellas las que más se suelen utilizar son aquellas que suman un término a la función de coste que penaliza los pesos de alto valor. Se hace tender entonces los pesos a cero; eliminándose aquellos que están por debajo de un cierto umbral.

$$J = \frac{1}{2} \cdot e^2 + \lambda \cdot \sum_{i,j} w_{ij}^2$$

$$J = \frac{1}{2} \cdot e^2 + \lambda \cdot \sum_{i,j} \frac{w_{ij}^2}{w_0^2 + w_{ij}^2}$$

Variantes del algoritmo básico de aprendizaje.

Existen gran cantidad de algoritmos de aprendizaje (algoritmos de búsqueda de mínimos en funciones multidimensionales). Algunos de ellos usan sólo información de la primera derivada de la función de coste, como el BP, otros usan información de la segunda derivada y, por último, existen otros de búsqueda de mínimos globales (genéticos, simulated annealing y swarm intelligence). Aquí solo comentaremos, por su sencillez, los basados en la primera derivada.

Algoritmo de momento

Aumenta la velocidad y reduce el problema de convergencia en la zonas planas.

$$w_{n+1} = w_n - \alpha \cdot \frac{\partial J}{\partial w_n} + \mu \cdot \Delta w_{n-1}$$

Algoritmo Silva-Almeida

Se usa la regla delta pero la constante cambia con el tiempo; 0<d<1, u>1. Mejora la velocidad de convergencia; el conocer si se está lejos/cerca del mínimo nos lo da los cambios de signo en dos valores consecutivos el gradiente.

$$\alpha_n = \begin{cases} d \cdot \alpha_{n-1} & si \quad [\nabla J]_n \cdot [\nabla J]_{n-1} < 0 \\ u \cdot \alpha_{n-1} & si \quad [\nabla J]_n \cdot [\nabla J]_{n-1} > 0 \end{cases}$$

Algoritmo Delta-Bar-Delta.

Ahora se tiene lo mismo que en el algoritmo de Silva-Almeida (0<d<1, u>1) pero, ahora, se compara el gradiente actual y un promedio de los anteriores

$$\alpha_n = \begin{cases} d \cdot \alpha_{n-1} & si \quad [\nabla J]_n \cdot P_{n-1} < 0 \\ u \cdot \alpha_{n-1} & si \quad [\nabla J]_n \cdot P_{n-1} > 0 \end{cases}$$

Todos los algoritmos que usan la primera derivada se basan en lo mismo; lejos del mínimo la constante tiene que tomar un valor alto y, cerca del mínimo, la constante debe ser baja.

Problemas que puedo tener en el aprendizaje.

Efecto	Caída en un mínimo local	Saturación de las neuronas (derivada de la función de activación igual a cero)	Derivada de la función de coste igual a cero	Algoritmo inestable, no se converge a ninguna solución.	El algoritmo no evoluciona. Análisis de sensibilidad erróneo.
Causa	Mala inicialización de los pesos	Mala inicialización de los pesos	Zonas planas de la función de error	Malas elecciones de las constantes	Mal procesado de las entradas
Solución.	Uso de algoritmos de búsqueda global; realización de múltiples pruebas	Inicialización de los pesos con una distribución normal de media cero y varianza de bajo valor.	Uso de otros algoritmos distintos al BP (algoritmos de segundo orden)	Se cambian los parámetros o bien se recurre a algoritmos de segundo orden.	HAY QUE REALIZAR UN PROCESADO CUIDADOSO Y CORRECTO

Análisis de sensibilidad.

Una vez obtenido un perceptrón multicapa, que proporciona buenos resultados en los conjuntos de entrenamiento y validación puede proporcionar más información realizando un **análisis de sensibilidad**

Mediante este análisis se determina la importancia relativa de las variables de entrada. Esta información se puede utilizar en primer lugar para obtener información cualitativa del problema que se intenta resolver y, en segundo lugar, para eliminar posibles entradas de cara a obtener un modelo neuronal más sencillo.

En un modelo neuronal se hace algo parecido, se determina la salida de la red para los patrones que se tienen y la salida tomando esa variable igual a cero. Después se calcula una función creciente de la diferencia entre esas salidas

$$S_{x_i}^1 = \sum_{k=1}^{N} [o_k - \varphi_k]^2 \qquad S_{x_i}^2 = \sum_{k=1}^{N} |o_k - \varphi_k|$$

En cualquier modelo este tipo de análisis se hace mediante la siguiente derivada

$$S_{x_i} = \frac{\partial [Modelo]}{\partial x_i}$$

Donde S designa la sensibilidad de una determinada variable. En un modelo lineal esta sensibilidad es proporcional al coeficiente que acompaña a esa variable.

El último paso es ordenar de mayor a menor esas cantidades; ese es el orden de la importancia de la entradas a la red (¿sabrías razonar por qué?). Es un método sencillo pero que funciona muy bien.

Pasos a la hora de aplicar perceptrón multicapa

1- Identificar el tipo de problema a resolver; ¿clasificación, predicción?

2- Tratamiento de los datos: eliminación de valores extremos, normalización de variables, filtrado e interpolado y reducción de características si es posible.

3-División de los datos en dos conjuntos; entrenamiento y validación. Hay que tener especial cuidado que al dividir los conjuntos se tenga la misma distribución.

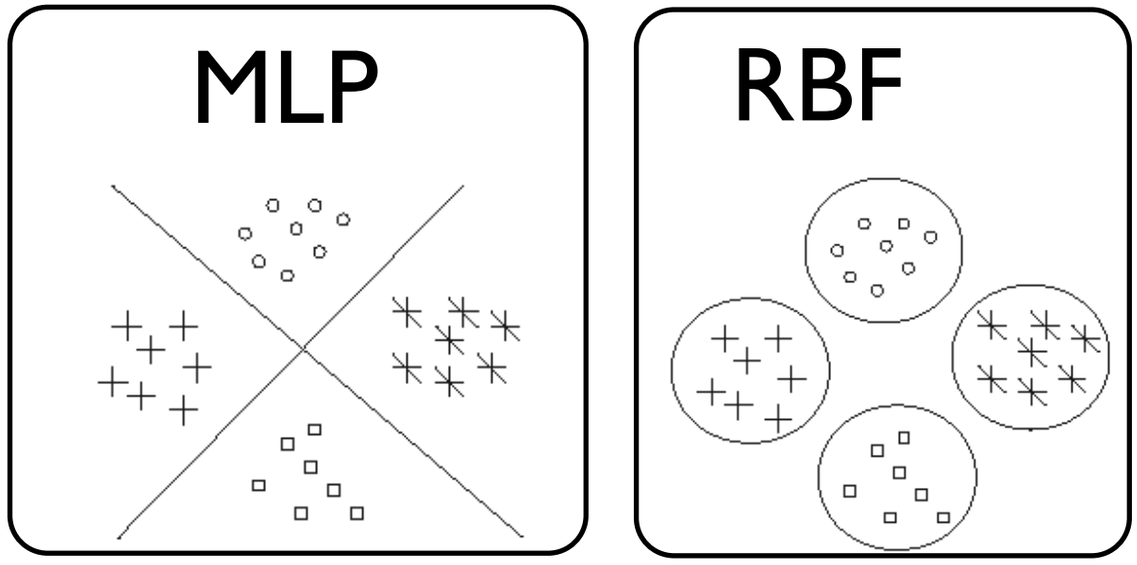
4- Elección de la arquitectura de la red....hasta ahora el mejor método es prueba y error (esto lo podemos hacer por la capacidad actual de los ordenadores).

5-Elección del algoritmo de obtención de los parámetros, algoritmo de aprendizaje, así como el tipo de aprendizaje (online o batch). Si se escogen algoritmos de búsqueda local (que son casi todos!!!) hay que repetir el proceso de entrenamiento con diferentes valores iniciales de los parámetros de la red.

6- La etapa final es un **análisis en profundidad** de los modelos para evitar conclusiones erróneas.

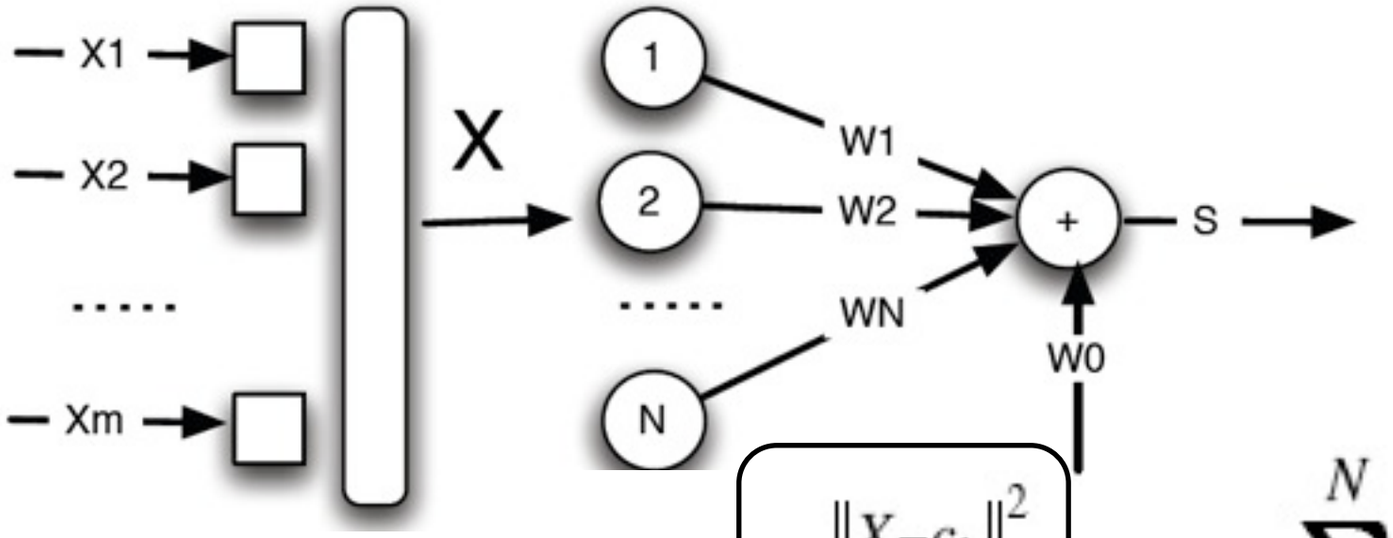
Funciones de Base Radial (RBF).

Data Mining: A Knowledge Discovery Approach, Springer



Suponen una aproximación diferente a la solución de problemas respecto de los perceptrones multicapa (MLP). En los MLP la solución obtenida es global mientras que, con este tipo de redes la solución es local.

Arquitectura de la red.



Aquí se tiene la capa oculta y la de salida; la de salida realiza una combinación lineal de sus entradas y la oculta realiza un **clustering** de los datos de entrada.

$$X = [x_1 ..... x_m] \qquad o_k = e^{\frac{\|X - c_k\|^2}{2\sigma_k^2}} \qquad s = \sum_{k=0}^{N} w_k \cdot o_k$$

$$c_k = [c_{k1} ..... c_{km}] \qquad \|X - c_k\|^2 = \sum_{s=1}^{m} (x_s - c_{ks})^2$$

Como siempre el problema es determinar los parámetros de la RBF

$o_0 = 1$

Algoritmo de aprendizaje de la RBF (I)

Funcionamiento de la RBF.

0- Se inicializan aleatoriamente los parámetros de la RBF (c, σ^2 y w).

1- Se escoge un patrón de entrada y se determina la salida de la RBF.

2- Se calcula el error cometido.

3- Se actualizan los parámetros de la RBF.

4- Si no se cumple a condición de parada se vuelve al paso 1.

La salida de la RBF se obtiene como:

$$u_k = \|X - c_k\|^2 = \sum_{s=1}^{m} (x_s - c_{ks})^2$$

$$o_k = e^{-\frac{u_k}{2 \cdot \sigma_k^2}} \qquad s = \sum_{k=0}^{N} w_k \cdot o_k$$

Para la actualización de los coeficientes se usará la regla delta; se plantea una función de coste que determine el funcionamiento de la red; la más usual es la cuadrática. En el caso de los coeficientes w

$$J = \frac{1}{2} \cdot e^2 = \frac{1}{2} \cdot (des - s)^2$$

$$w_j = w_j - \alpha \cdot \frac{\partial J}{\partial w_j} \qquad \frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial e} \cdot \frac{\partial e}{\partial s} \cdot \frac{\partial s}{\partial w_j}$$

$$w_j = w_j + \alpha \cdot e \cdot o_j$$

Algoritmo de aprendizaje de la RBF (II)

Si se plantea el mismo procedimiento para los centros

$$c_l = c_l - \alpha \cdot \frac{\partial J}{\partial c_l}$$

CUIDADO QUE ESTA ECUACIÓN ES VECTORIAL.

$$\frac{\partial J}{\partial c_l} = \frac{\partial J}{\partial e} \cdot \frac{\partial e}{\partial s} \cdot \frac{\partial s}{\partial o_l} \cdot \frac{\partial o_l}{\partial u_l} \cdot \frac{\partial u_l}{\partial c_l}$$

$$c_l = c_l + \frac{\alpha \cdot e \cdot w_l \cdot o_l}{\sigma_l^2} \cdot \left(X - c_l\right)$$

Aplicando ahora para las varianzas se obtiene

$$\sigma_t^2 = \sigma_t^2 - \alpha \cdot \frac{\partial J}{\partial \sigma_t^2}$$

$$\frac{\partial J}{\partial \sigma_t^2} = \frac{\partial J}{\partial e} \cdot \frac{\partial e}{\partial s} \cdot \frac{\partial s}{\partial o_t} \cdot \frac{\partial o_t}{\partial u_t} \cdot \frac{\partial u_t}{\partial \sigma_t^2}$$

$$\sigma_t^2 = \sigma_t^2 + \alpha \cdot \frac{e \cdot w_t \cdot o_t \cdot \left\|X - c_t\right\|^2}{2 \cdot \left[\sigma_t^2\right]^2}$$

Tenemos un procedimiento iterativo para calcular los parámetros del sistema neuronal. Al igual que pasaba con el perceptrón multicapa el principal problema de esta aproximación son los mínimos locales.

Existen otros procedimientos para calcular los parámetros de la RBF y que se pueden combinar con el comentado aquí. Por ejemplo, la capa intermedia determina un clustering de los datos; podemos aplicar aquí el algoritmo HCM para determinar los centros (o el algoritmo FCM). Además la capa de salida es lineal respecto de los parámetros w por lo que se pueden plantear mínimos cuadrados para estos parámetros.

Algunas cuestiones a tener en cuenta.

Se han visto tres estructuras neuronales; SOM, MLP y RBF pero existen muchas más (incluso existen híbridos de éstas).

Mientras el principal uso del SOM es el análisis exploratorio de datos la RBF se utiliza para problemas de modelización/predicción y el MLP se aplica en dichos problemas y en problemas de clasificación.

La regla delta se puede aplicar en todo modelo matemático. IMPRESCINDIBLE CONOCERLA. Proporciona un método general para obtener los parámetros de cualquier modelo. El problema son los mínimos locales.

El Perceptrón Multicapa permite establecer cualquier relación entre dos conjuntos de datos POR ESO MISMO ES NECESARIO Y FUNDAMENTAL ANALIZAR LOS RESULTADOS.

Los resultados obtenidos con el MLP se pueden analizar de la misma forma que los modelos lineales; podemos establecer ANOVAS, intervalos de confianza, etc; lo único que hacemos es aumentar la POTENCIA DEL MODELO MATEMÁTICO; si el problema es complicado.....¿la solución va a ser un modelo sencillo?

VNIVERSITAT DE VALÈNCIA

MASTER DE INGENIERÍA BIOMÉDICA.

Métodos de ayuda al diagnóstico clínico.

Tema 5: Redes neuronales.