Redes neuronales artificiales para aprendizaje automático y minería de datos

Tratamiento Inteligente de la Información y Aplicaciones

Juan A. Botía

Departamento de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia

October 4, 2007

Guión de la clase

RNAs : Justificación

- Modelo aproximador de funciones reales, discretas y vectoriales en la salida.
- Aplicaciones
  - Reconocimiento de caracteres,
  - de rostros,
  - para pilotaje de vehículos, etc.
- Idóneas cuando...
  - Ejemplos del conjunto de aprendizaje formados por pares atributo-valor
  - La salida de la función que define los datos de entrada tiene como salida un valor real, un valor discreto o un vector de valores reales y/o discretos
  - Datos de entrenamiento contienen ruido (errores)
  - No es necesario interpretar, para comprender, la función objetivo aprendida por la RNA
  - No es relevante el tiempo consumido en la fase de aprendizaje

Ideas iniciales

- La regla de decisión óptima para minimizar la probabilidad de clasificar una nueva observación como ejemplar consiste en asignar a ese ejemplar nuevo la clase con mayor probabilidad a posteriori.
- El uso de redes neuronales para obtener clasificadores o modelos de regresión implica emplear una estrategia diferente

  $\Rightarrow$ *usar la idea de funciones discriminantes*

- una función discriminante se especifica en una determinada forma paramétrica
- el valor de los parámetros se determina (i.e. ajusta) mediante un algoritmo de aprendizaje sobre los datos de entrada.
- Representación básica: combinación lineal de variables de entrada

Discriminante más simple

- Una función discriminante, la denotaremos con $y(x)$ es tal que el vector x se asigna a la clase C_1 si $y(x) > 0$ y a la clase C_2 si $y(x) < 0$.
- La más simple es lineal

$$y(x) = w^T x + \omega_0,$$

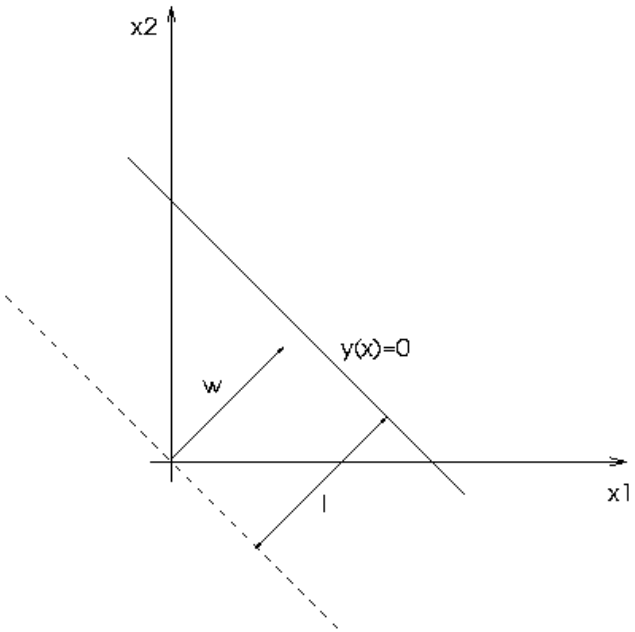
siendo w un vector d-dimensional y ω_0 el bias

Interpretación geométrica (ver [?])

- Si consideramos $y(x) = 0$ como el límite de decisión entre las dos clases, corresponde a u hiperlano $d - 1$ dimensional, en un espacio x d-dimensional
- Con dos variables de entrada x_1 y x_2, el límit decisión es una línea recta.
- Sean x^A y x^B del hiperplano citado, tenemos $y(x^A) = y(x^B) = 0$ y por la definición de $y(x$

$$w^T(x^B - x^A) = 0,$$

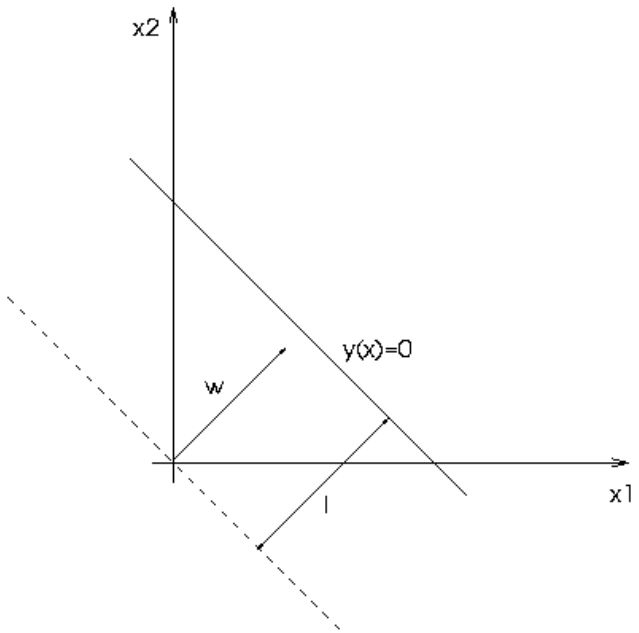
la orientación de w condiciona totalmente la orientación del discriminante.



Interpretación geométrica (y II)

Sea x un punto cualquier del hiperplano

- la distancia normal del origen al hiperplano viene dada por $l = \frac{w^T x}{||w||}$,
- como $w^T x = -\omega_0$, tenemos que $l = -\frac{\omega_0}{||w||}$
- ω_0 determina la posición con respecto al eje del discriminante, en el espacio de las variables de entrada.



Más de dos clases

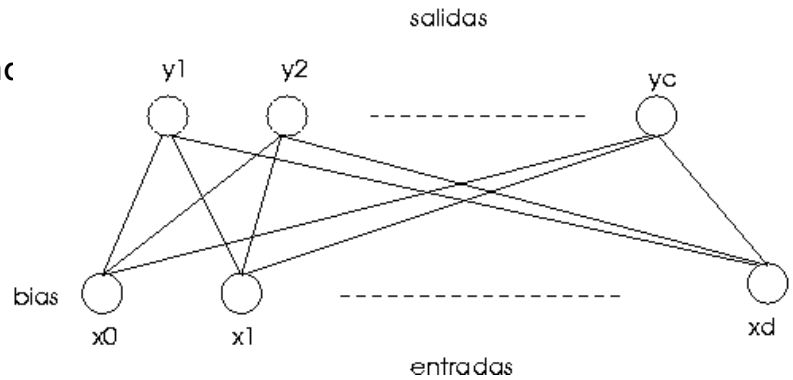
- Si hay c clases, usamos un discriminante $y_k(x)$ para cada clase C_k de la forma

$$y_k(x) = w_k^T + \omega_k k0,$$

el punto x se asignará a la clase C_k si se tiene que $y_k(x) > y_j(x) \forall j \neq k$

- La línea de separación entre dos clases C_k y C_j venc dada por el hiperplano $y_k(x) = y_j(x)$
- El hiperplano tendrá la forma

$$(w_k - w_j)^T x + (\omega_{k0} - \omega_{j0}) = 0.$$



- La orientación del discriminante vendrá dada por $(w_k - w_j)$
- La distancia al origen por el bias

$$l = -\frac{(\omega_{k0} - \omega_{j0})}{||w_k - wj||}.$$

Cálculo de w_i. Formulación del problema de los mínimos cuadrados

- Se basa en la optimización de una función de error de sumas cuadráticas (adecuada para regresión)
- transformamos el vector de entrada mediante M funciones $\phi_j(x)$, denominadas funciones base, y representamos la salida mediante una combinación lineal de esas funciones: $y_k(x) = \sum_{j=1}^{M} w_{kj}\phi_j(x) + w_{k0}$
- Podemos introducir una función $\phi_0 = 1$ adicional para que el discriminante quede: $y_k(x) = \sum_{j=0}^{M} w_{kj}\phi_j(x)$
- El error de sumas cuadráticas se define mediante la función:

$$E(x) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}(y_k(x^n; w) - t_k^n)^2,$$

$y_k(x^n; w)$ la salida para el nodo k como función del vector de entrada y el vector de pesos w
N es el número de patrones de entrenamiento
c el número de nodos de salida
t_k^n es el valor objetivo para el nodo k y el vector de entrada x^n

Función de error

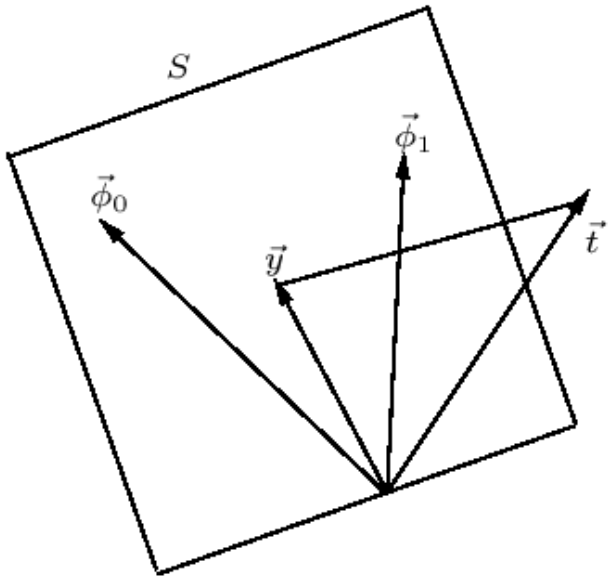
- Esta función de error compone una superficie de variaciones suaves a la salida
- Es función de w_{ij} y se puede minimizar con una gran cantidad de técnicas, e.g. mínimos cuadrados y gradiente descendente
- $E(w)$ es una función cuadrática con respecto w y por lo tanto las derivadas son funciones lineales en w
- Así, el valor de los pesos en el mínimo de la función de error puede calcularse

Interpretación geométrica

- Vamos a considerar una red de una única salida, i.e. $c = 1$
- para una observación de entrada x^n, la salida de la red será de la forma $y^n = \sum_{j=0}^{M} w_j \phi_j(x^n)$
- Agrupamos todos los valores objetivo de los datos de entrada para formar un vector N dimensional, $\vec{t}$
- Para cada una de las funciones base $\phi_j(x)$, agrupamos también sus correspondientes valores en $\vec{\phi}_j$ también de dimensión N
- Sea el número de funciones base, M tal que $M + 1 < N$.
- Los $M + 1$ vectores $\vec{\phi}_j$ componen una base S, $M + 1$ dimensional que engendra un supespacio vectorial S.
- Si agrupamos las salidas de la red y^n en un vector $\vec{y}$, este vector estará también en S ya que $\vec{y} = \sum_{j=0}^{M} w_j \vec{\phi}_j$

Interpretación geométrica (II)

Cambiando los valores de w_j cambiamos la dirección de $\vec{y}$



Interpretación geométrica (III)

- Usando notación vectorial $E(x) = \frac{1}{2} \left\| \sum_{j=1}^{M} w_i \vec{\phi}_j - \vec{t} \right\|^2$
- Si minimizamos con respecto a los pesos $\frac{\partial E}{\partial w_j} = 0 = \vec{\phi}_j^T (\vec{y} - \vec{t})$ (ver Desarrollo A)
- descompongamos $\vec{t} = \vec{t}_\perp + \vec{t}_\parallel$, siendo

  $\vec{t}_\parallel$ *proyeccion ortogonal de* $\vec{t}$ *en* S
  $\vec{t}_\perp$ *el resto*

- Como $\vec{\phi}_j^T \vec{t}_\perp = 0$ al ser perpendiculares, tenemos que

$$\frac{\partial E}{\partial w_j} = \vec{\phi}_j^T (\vec{y} - \vec{t}_\parallel) = 0, \; j = 1, \ldots, M.$$

- y como $\vec{\phi}_i$ componen la base que egendra el subespacio S, el único producto vectorial con esa base que puede ser nulo es el del vector nulo, con lo que tenemos $\vec{y} = \vec{t}_\parallel$
- el proceso de aprendizaje trata de modificar el vector $\vec{y}$ de tal forma que se minimice la distancia a $\vec{t}$

Solución basada en la pseudoinversa

- Reescribimos la función de error

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{c} \left\{ \sum_{j=0}^{M} w_{kj} \phi_j^n - t_k^n \right\}^2 .$$

- Al diferenciar con respecto a w e igualar las derivadas a cero

$$\sum_{n=1}^{N} \left\{ \sum_{j'=0}^{M} w_{kj'} \phi_{j'}^n - t_k^n \right\} \phi_j^n = 0$$

- En forma matricial $(\Phi^T \Phi) W^T = \Phi^T T$
  - Φ una matriz $N \times M$ con los elementos ϕ_j^n
  - W una matriz $c \times M$, con los elementos w_{kj}
  - T una matriz $N \times c$ con los elementos t_k^n

Solución basada en la pseudoinversa (II)

- La matriz $\Phi^T\Phi$ es cuadrada, de dimensiones $M \times M$
- Si el determinante de $\Phi^T\Phi$ es distinto de cero (i.e. es no singular), podemos invertirla para obtener la solución

$$W^T = \Phi^\dagger T$$

  - $\Phi^\dagger$ es una matriz $M \times N$ denominada pseudo inversa de Φ
  - $\Phi^\dagger = (\Phi^T\Phi)^{-1}\Phi^T$

Objeciones al método

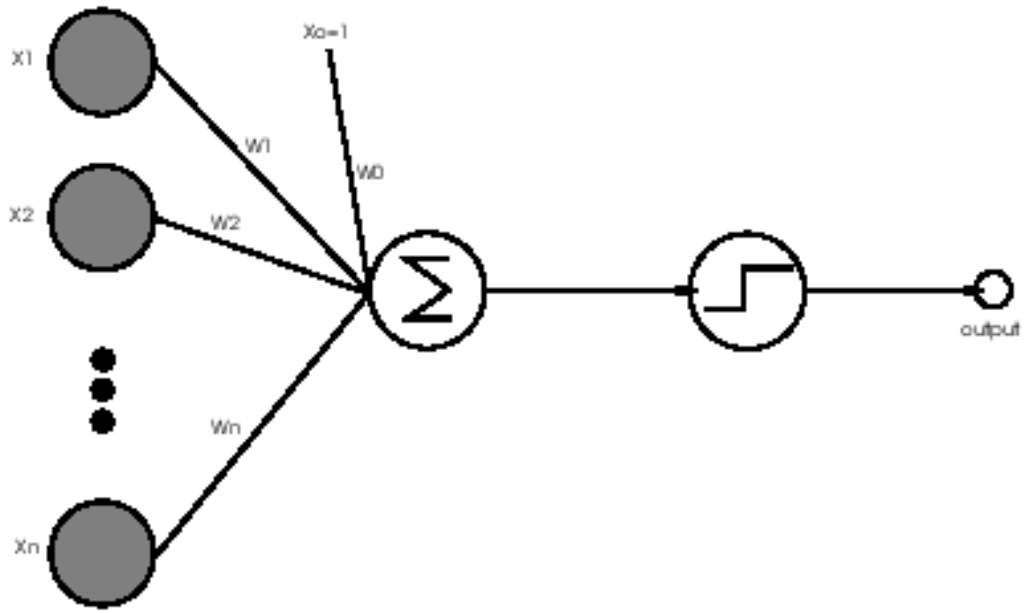
- Φ es, generalmente, una matriz no cuadrada y no tiene una inversa
- $\Phi^T\Phi$ suele tenerla
  - la solución directa de las ecuaciones normales puede presentar dificultades
    - ★ la posibilidad de que $\Phi^T\Phi$ sea singular o casi singular
    - ★ si dos de los vectores $\vec{\phi}_j$ son casi colineales
    - ★ los errores de redondeo típicos de una máquina las van a hacer casi linealmente dependientes. Si esto ocurre, el procedimiento numérico usado fallará
    - ★ adecuado un método del tipo SVD (*Singular Value Decomposition*)

El perceptrón (ver Minsky, 1969 [?])

- Es la red más sencilla
- Una única neurona, que puede tener varias entradas reales, y salida en $\{0,1\}$.
- Formalmente, dadas las entradas $x_1, x_2, \ldots, x_n$, la salida $0(x_1, x_2, \ldots, x_n)$ se define mediante

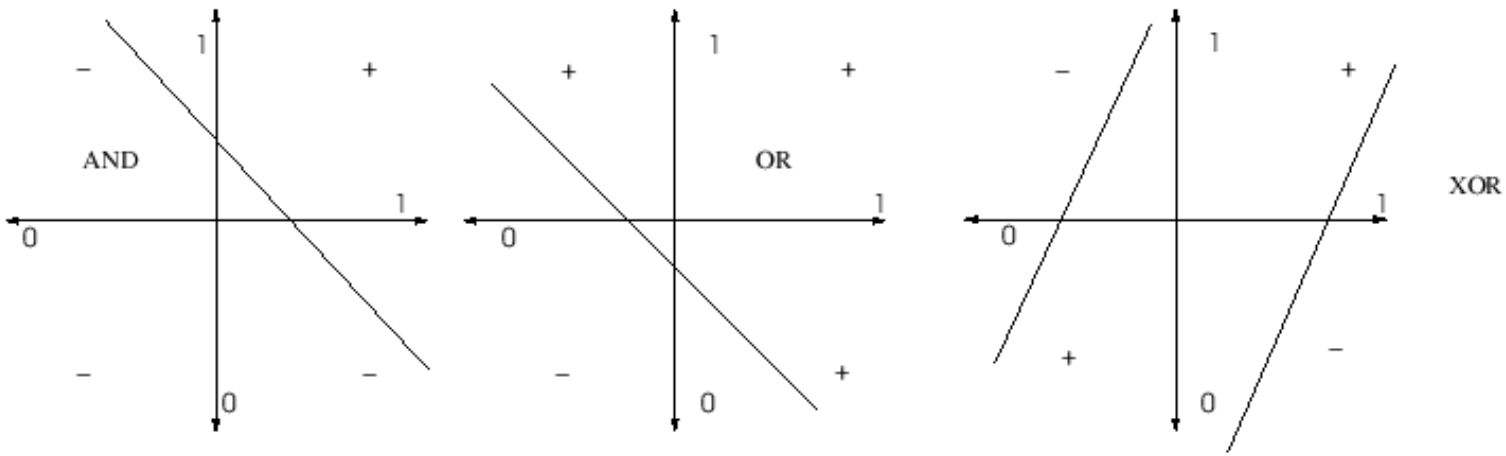
$$o(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{si } w_0 x_0 + w_1 x_1 + \ldots + w_n x_n > 0 \\ -1 & \textit{sino} \end{cases}$$

- Los w_i, $i = 1, 2, \ldots, n$ son los pesos asignados a cada entrada



El perceptrón. Capacidad de representación

- Con un perceptrón podemos representar funciones lógicas como `AND` y `OR`.
- Con un sólo nodo no podemos representar la función `XOR`.



Regla de entrenamiento del perceptrón [?]

- Mecanismo de aprendizaje de respuestas ± 1 adecuadas según el ejemplo de aprendizaje.
- Para cada ejemplo variar los w_i según la expresión

$$w_i \leftarrow w_i + \Delta w_i$$

en donde

$$\Delta w_i = \eta(t - o)x_i$$

- t es la salida esperada,
- o es la salida obtenida para el ejemplo
- η es el *ratio de aprendizaje*
- Problema: convergencia

Regla de entrenamiento delta

- Soluciona el problema de la convergencia anterior
- Basada en **gradiente descendente**
- Se ajusta asintóticamente a la representación deseada
- Medida de error

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

- D es el conjunto de ejemplos de entrenamiento,
- t_d es la salida de la función buscada, y
- o_d es la salida de la red.

Obtención de la regla

- El vector de gradiente será $\Delta E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$ y regla del gradiente será $\vec{w} \leftarrow +\Delta\vec{w}$, $\Delta\vec{w} = -\eta\Delta E(\vec{w})$
- ΔE se obtiene según

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\vec{x}_d) \\
&= \sum_{d\in D}(t_d - o_d)(-x_{id})
\end{aligned}$$

- Finalmente

$$\Delta w_i = \eta\sum_{d\in D}(t_d - o_d)x_{id}$$

Gradiente descendente: algoritmo

- Paso 0: sea el conjunto de entrenamiento un conjunto de ejemplos en la forma $< \vec{x}, t >$ en donde $\vec{x}$ es el vector de valores de entrada, y t es el valor de salida. η es el ratio de aprendizaje.
- Paso 1: Inicializar cada w_i a un valor aleatorio
- Paso 2: Hasta que se cumpla la condición de terminación
  - Inicializar los $\Delta w_i \leftarrow 0$
  - Para cada $< \vec{x}, t >$ hacer
    - ★ Realizar una pasada con $\vec{x}$ y calcular o
    - ★ Para cada w_i, hacer

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

  - Para cada w_i, hacer

$$w_i \leftarrow w_i + \Delta w_i$$

- El algoritmo anterior actualiza los pesos en cada pasada de **todo** el conjunto de entrenamiento

Versión estocástica

La versión estocástica lo hace para cada ejemplo:

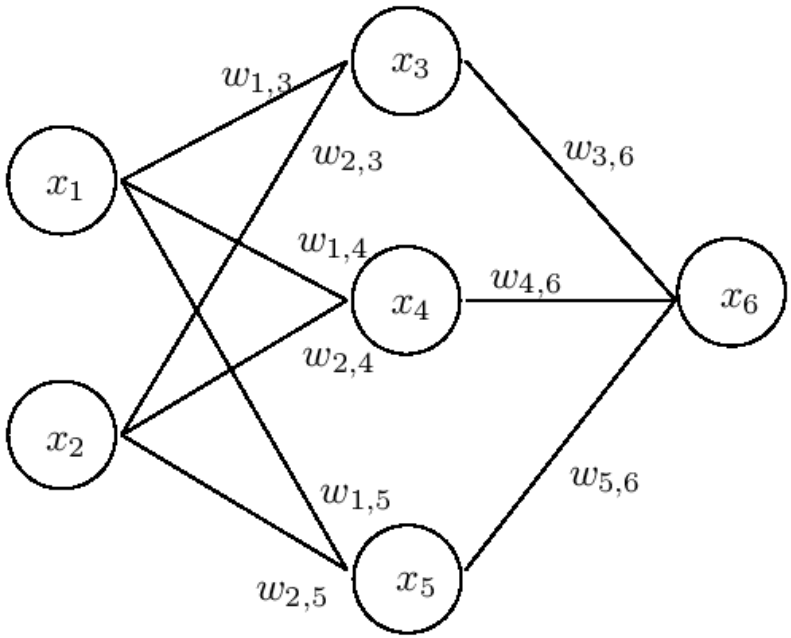
- Paso 0: sea el conjunto de entrenamiento un conjunto de ejemplos en la forma $< \vec{x}, t >$ en donde $\vec{x}$ es el vector de valores de entrada, y t es el valor de salida. η es el ratio de aprendizaje.
- Paso 1: Inicializar cada w_i a un valor aleatorio
- Paso 2: Hasta que se cumpla la condición de terminación
  - Inicializar los $\Delta w_i \leftarrow 0$
  - Para cada $< \vec{x}, t >$ hacer Realizar una pasada con $\vec{x}$ y calcular o Para cada w_i, hacer

$$w_i \leftarrow w_i + \eta(t - o)x_i$$

Comentarios a las versiones

- La versión estándar es más costosa computacionalmente.
- La versión estocástica es más *segura*, un vector de error por cada ejemplo.
- La regla de entrenamiento se denomina **regla delta**, LMS (*least-mean-square*), regla ADALINE (*ADAptative LINEar unit*) y Widrow-Hoff.

- ¿Cómo podemos aumentar la capacidad de representación de un perceptrón?
- Ejemplo
  - Podríamos intentar formar una red con varios perceptrones lineales como ...



Nodos

- Su expresión sería

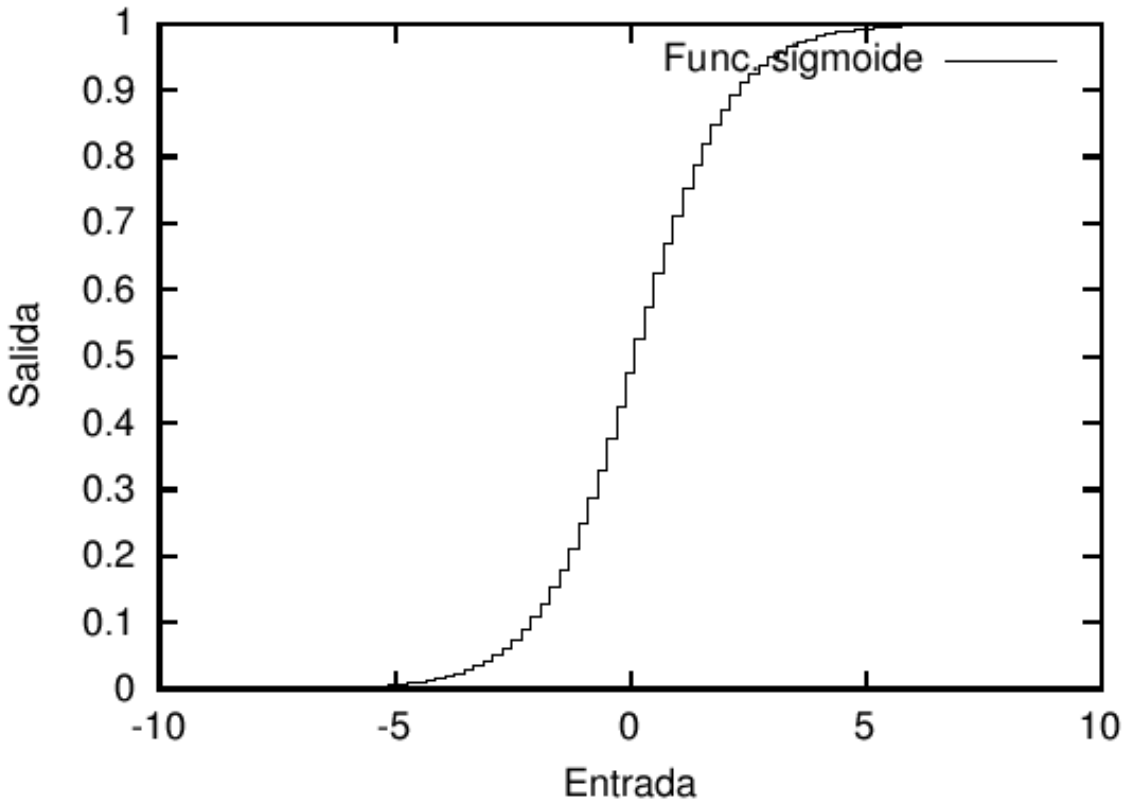
$$
\begin{aligned}
o(x_1, x_2) &= x_3 w_{3,6} + x_4 w_{4,6} + x_5 w_{5,6} \\
&= (x_1 w_{1,3} + x_2 w_{2,3}) w_{3,6} + (x_1 w_{1,4} + x_2 w_{2,4}) w_{4,6} + \\
&\quad (x_1 w_{1,5} + x_2 w_{2,5}) w_{5,6} \\
&= x_1 (w_{1,3} w_{3,6} + w_{1,4} w_{4,6} + w_{1,5} w_{5,6}) + \\
&\quad x_2 (w_{2,3} w_{3,6} + w_{2,4} w_{4,6} + w_{2,5} w_{5,6})
\end{aligned}
$$

- Podríamos usar perceptrones con umbral (i.e. salidas ± 1)

  PROBLEMA $\longrightarrow$ *es necesario poder derivar las funciones de salida*

Nodos (II)

Solución: sigmoide $o = \sigma(\vec{w}\vec{x}) = \frac{1}{1+e^{-\vec{w}\vec{x}}}$ con derivada $\frac{d\sigma(y)}{dy} = \sigma(y)(1 - \sigma(y))$

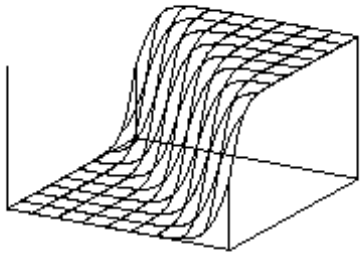


Nodos (III)

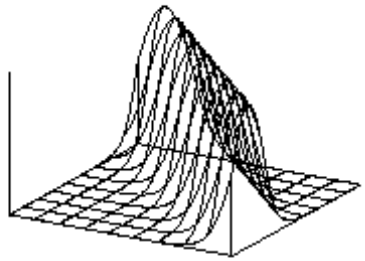
- Podríamos obtener la expresión resultande de aplicar la sigmoide a todos los nodos, para la red anterior
- Como puede verse, esta superficie ya es mucho más compleja que una superficie lineal.

$$
\begin{aligned}
o(x_1, x_2) &= \sigma(\sigma(x_3)w_{3,6} + \sigma(x_4)w_{4,6} + \sigma(x_5)w_{5,6}) \\
&= \sigma(\sigma(x_1 w_{1,3} + x_2 w_{2,3})w_{3,6}) + \\
&\quad \sigma(\sigma(x_1 w_{1,4} + x_2 w_{2,4})w_{4,6}) + \\
&\quad \sigma(\sigma(x_1 w_{1,5} + x_2 w_{2,5})w_{5,6})
\end{aligned}
$$

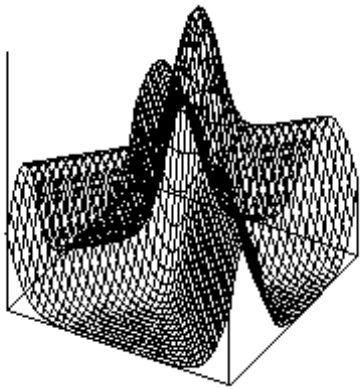
Capacidad de representación



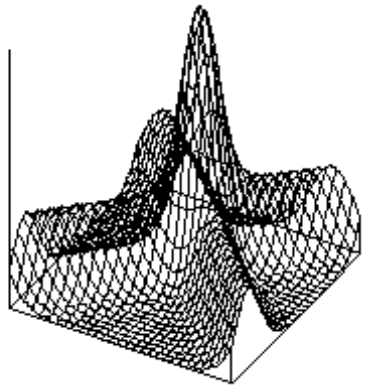
sigmoide de dos variables



suma de sigmoides



sigmoide de suma de sigmoides



combinación lineal de sigmoides

Algoritmo *Backpropagation*

- Aprende los w_i para una red neuronal multi-capa, con funciones de activación derivables
- Versión *fully-connected* y conexión estricta
- Función de error

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in O} (t_{kd} - o_{kd})^2$$

  en donde D es el conjunto de ejemplos y O el conjunto de nodos de salida.
- t_{kd} será la salida esperada para el ejemplo d en el nodo k,
- o_{kd} la salida obtenida para el nodo k y el ejemplo d.
- Convergencia a mínimos locales

Algoritmo *Backpropagation*, versión estocástica

Paso 1: Inicializar cada w_{ij} a un valor aleatorio
Paso 2: Hasta que se cumpla la condición de terminación
Paso 3: Para cada $< \vec{x}, \vec{t} >$ hacer

- Propagar la entrada por la red hacia adelante inicializando los nodos de entrada con $\vec{x}$ y calcular los o empezando por todos los nodos de la capa oculta, y a continuación los de la capa de salida.
- Propagar el error por la red, hacia atrás

(1) Para cada nodo de salida k, calcular su error, δ_k según

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

(2) Para cada nodo oculto, h, calcular su error, δ_h según

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in O} w_{hk} \delta_k$$

(3) Actualizar los w_{ij} según $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$ donde $\Delta w_{ij} = \eta \delta_i x_{ij}$

Algoritmo *Backpropagation*

Similitud con regla delta

- En regla delta, w_i se actualizan con $\eta x_i(t - o)$.
- En *backpropagation* se suma a los w_{ij} el producto $\eta x_{ij}\delta_i$.
- En realidad δ_i es el producto de $(t - o)$ por $\sigma()'$ en los nodos de salida
- En los nodos de la capa oculta el producto es de $\sigma()'$ y $\sum_{k \in O} w_{hk}\delta_k$

Algoritmo *Backpropagation* (y II)

Condiciones de parada

- En un problema "real" se realizan usualmente varios miles de pasadas (i.e. *epochs*) por todo el conjunto de entrenamiento.
- A veces la condición de parada del algoritmo es simplemente llegar a un número límite de esas pasadas
- Otras es no alcanzar una cantidad mínima de error por pasada.
- Se ha de evitar el sobreaprendizaje u *overfitting*.

Momentum

- El algoritmo actualiza gradualmente los pesos de la red en la dirección de máxima variación del error mediante la expresión

$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\partial E}{\partial w_{ij}}(t)$$

- Si el valor de η es excesivamente grande, entonces la búsqueda hacia un error pequeño sufrirá oscilaciones
- Si es muy pequeño, la convergencia al mínimo será lenta y se necesitarán muchas iteraciones para llegar a un error aceptable $\rightarrow$ *momentum*

$$\Delta w_{ij}(t) = -\eta \frac{\partial E}{\partial w_{ij}}(t) + \mu \Delta w_{ij}(t-1)$$

- Marca una inercia, en el avance hasta el punto óptimo, desde la última actualización hasta la actual, lo cual evitaba las oscilaciones.

RPROP (ver [?])

- Idea: variar el valor de η, conforme se avanzaba en el aprendizaje
- Ejemplos de algoritmos son Delta-Bar-Delta, SuperSAB y Quickprop.
- RPROP tiene en cuenta, en cada paso, la derivada parcial de cada peso, y su expresión general para la cantidad con la que actualizar los pesos es

$$\Delta_{ij}(t) = \begin{cases} \eta^+ \times \Delta_{ij}(t-1) & , \quad \text{si } \frac{\partial E(t-1)}{\partial w_{ij}} \times \frac{\partial E(t)}{\partial wij} > 0 \\ \eta^- \times \Delta_{ij}(t-1) & , \quad \text{si } \frac{\partial E(t-1)}{\partial w_{ij}} \times \frac{\partial E(t)}{\partial wij} < 0 \\ \Delta_{ij}(t-1) & , \quad \text{si no} \end{cases}$$

- Si de una iteración a otra, el valor de la derivada parcial del peso w_{ij} cambia su signo $\longrightarrow$ la actualización anterior fue demasiado elevada
- Si mantiene el signo lo que hacemos es aumentar el valor de actualización para aumentar la velocidad de convergencia.
- La ecuación de actualización queda

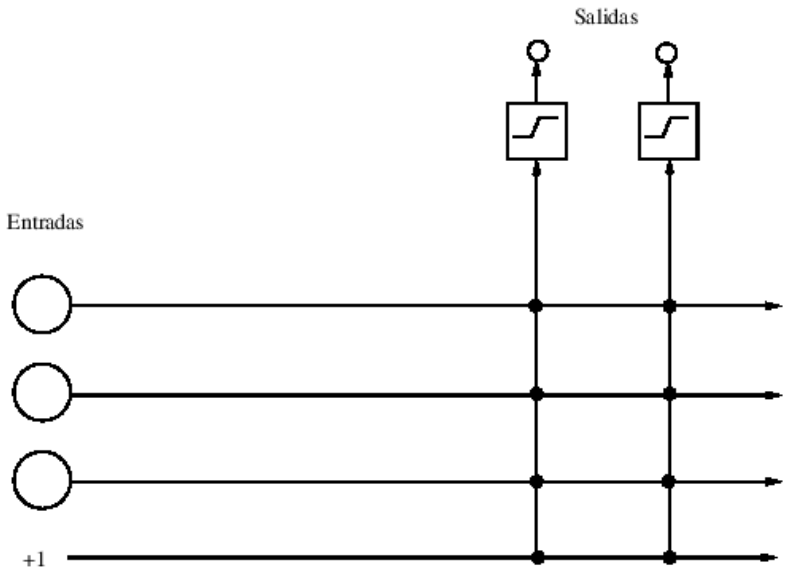
$$w_{ij}(t+1) = w_{ij}(t) - \Delta_{ij}(t) * sign\left(\frac{\partial E(t)}{\partial w_{ij}}\right)$$

Algoritmo de cascada-correlación (ver [?])

- Responde al problema del objetivo móvil del backpropagation
- Cada nodo en la red intenta jugar un rol determinado en el cálculo global
- El resto también, y no se comunican entre sí
- Es dificil, en esas condiciones, que cada nodo encuentre su lugar
- Manifestación $\longrightarrow$ el "efecto manada"
- Supóngase dos tareas A y B a realizar por los nodos ocultos
- Cada nodo deberá decidir a cuál de las dos dedicarse
- Si A genera un error mayor que el de B entonces todos los nodos se concentrarán en resolver la tarea de A
- Una vez que A está resuelta, se comenzará a centrar la atención en B, y el problema en A reaparecerá
- Al final, la distribución de los nodos será correcta, pero existe un largo período de indecisión
- Solución $\longrightarrow$ permitir que únicamente un conjunto reducido de nodos cambie cada vez, manteniendo el resto constantes
- Dos ideas: arquitectura en cascada y algoritmo de aprendizaje

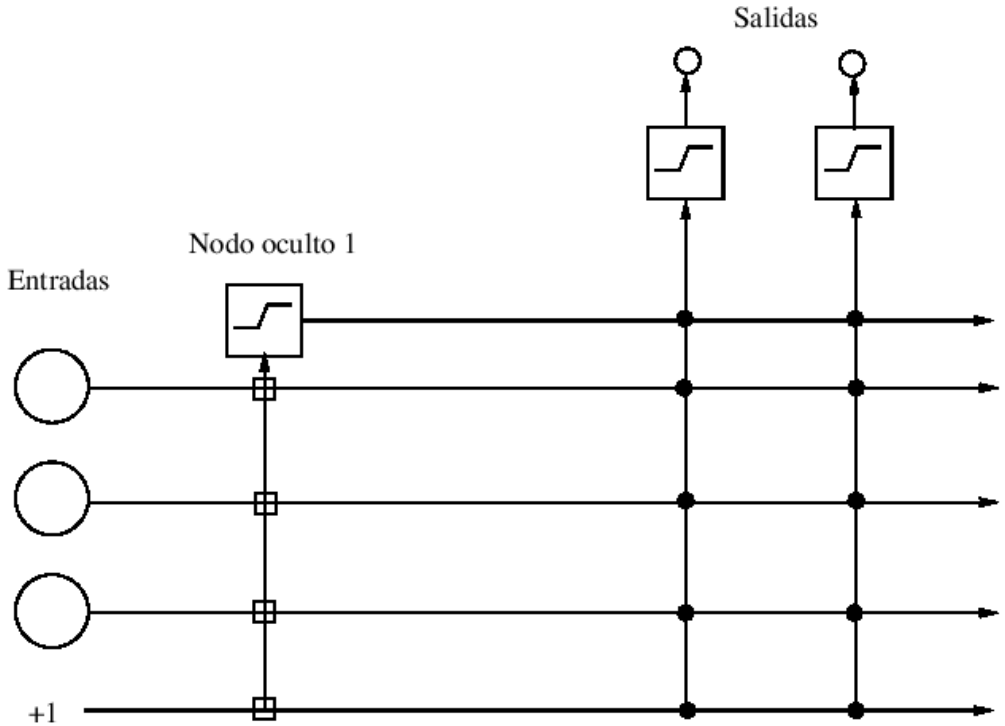
Correlación en cascada - Arquitectura

Inicialmente se parte de una red sin nodos ocultos



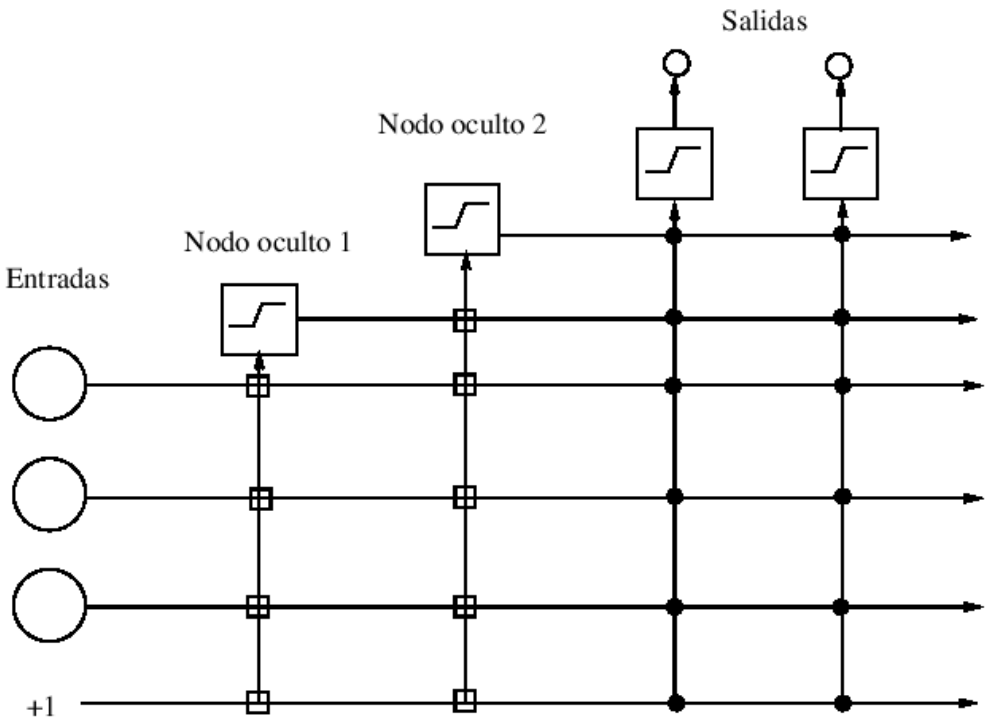
Correlación en cascada - Arquitectura (II)

Cada nuevo nodo oculto recibe una conexión de la entrada y se conecta a todas las salidas



Correlación en cascada - Arquitectura (III)

Un nuevo nodo oculto recibe conexiones de la entrada y todos los nodos ocultos preexistentes



Correlación en cascada - Aprendizaje

- Los pesos de los arcos que van desde la entrada a los nodos ocultos se congelan una vez se añaden
- Los pesos de los arcos que van desde los nodos ocultos a los de salida se entrenan repetidamente
- Inicialmente (sin nodos ocultos) se entrenan los pesos de las entradas a las salidas (regla delta, regla de entrenamiento del perceptron, etc)
- Una vez no se tiene una reducción significante del error, se evalua la red sobre D.
  - Si el error obtenido es aceptable, paramos
  - Si no, aun queda un resto de error que necesitamos reducir
    - ★ Intentamos reducirlo añadiendo un nuevo nodo (usamos el algoritmo de creación de nodos)
    - ★ Congelamos todos los pesos de entrada, incluido el del nuevo nodo
    - ★ Entrenamos los pesos de salida de nuevo
    - ★ Repetimos el ciclo hasta que el error es aceptable

Correlación en cascada - Alg. Creación de nodos

- Se comienza con un nodo candidato que recibe conexiones de
  1. todo nodo de entrada
  2. todo nodo oculto preexistente

  sin conectar su salida
- Se realizan varias pasadas de entrenamiento ajustando los pesos de entrada al nuevo nodo, intentando maximizar $S \rightarrow$ correlación entre V y el error residual E_o que se observa en la unidad de salida o

$$S = \sum_o \left| \sum_p (V_p - \bar{V})(E_{p,o} - \bar{E_o}) \right|$$

en donde

- V es el valor de salida del nodo candidato
- o es la salida de la red en la que medimos el error
- p es la instancia de entrada
- $\bar{V}$ la media de los valores de V sobre todas las entradas
- $\bar{E_o}$ la media de los valores de E_o sobre todas las entradas

Correlación en cascada - Maximizando S

Para maximizar S necesitamos calcular $\partial S/\partial w_i$

$$\partial S/\partial w_i = \sum_{p,o} \sigma_o (E_{p,o} - \bar{E}_o) f'_p I_{i,p}$$

en donde

- σ_o es el signo de S
- f'_p es la derivada de la función de activación del nodo candidato, para la instancia p, con respecto a la suma de las entradas
- $I_{i,p}$ es la entrada al nodo candidato desde el nodo de entrada i, con la instancia p

Una vez calculada $\partial S/\partial w_i$ para todo arco de entrada al nodo candidato realizamos gradiente ascendente para maximizar S

Redes RBF (ver [?])

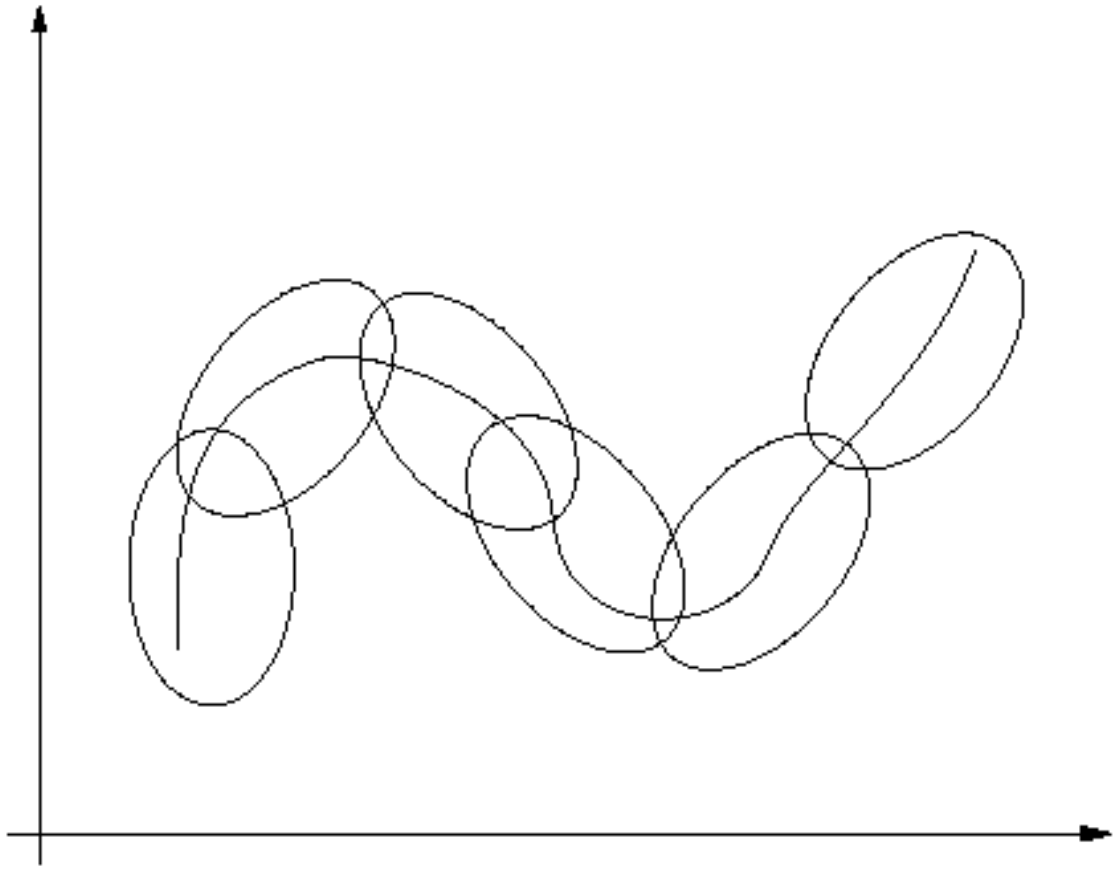
- Las redes RBF (*Radial Basis Functions*)
- Se trata de aproximar globalmente una función mediante el uso de varias RBF que realizan una aproximación local.
- La hipótesis inducida con este tipo de métodos va a tener la forma

$$\hat{f}_r(x) = w_0 + w_1 K_1(d(\mu_1, x)) + w_2 K_2(d(\mu_2, x)) + \cdots + w_k K_k(d(\mu_k, x)) \quad (1)$$

en donde

  - cada μ_i es un punto del espacio $\Re^n$ y
  - cada una de las funciones $K_i(d(\mu_i, x))$ está definida de tal forma que decrece conforme la distancia de x a μ_i crece ($K_i \rightarrow 0$ cuando $|x| \rightarrow \infty$)
  - k es una constante determinada por el usuario.

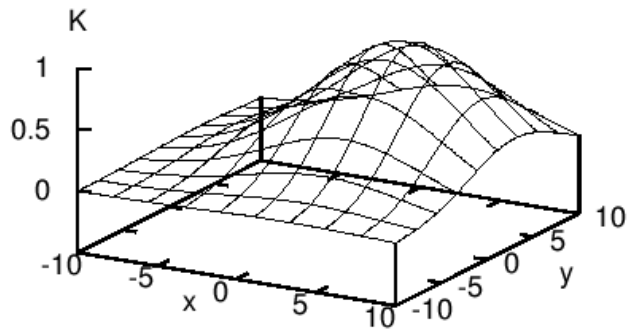
Estrategia de aproximación



Redes RBF

- Aunque $\hat{f}_r(x)$ es una aproximación global, las $K_i(d(\mu_i, x))$ contribuyen localmente, en una región cercana a μ_i.
- Se suele definir usando una función de tipo gausiana, centrada en un punto μ_i, con una varianza σ_i^2 tal que

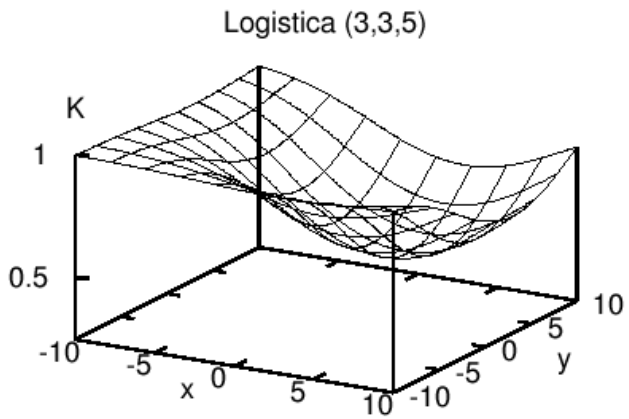
$$K_i(d(\mu_i, x)) = e^{-\frac{d^2(\mu_i, x)}{2\sigma_i^2}}$$



Redes RBF (II)

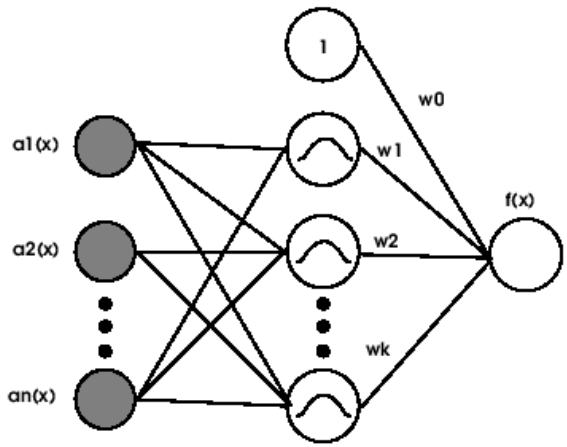
- O bien una función logística, como

$$K_i(d(\mu_i, x)) = \frac{1}{1 + e^{d^2(\mu_i, x)/\sigma_i^2}}$$



Redes RBF (III)

- La aproximación local que ofrecen estas funciones es muy potente.
- La función de la ecuación 1 puede aproximar cualquier función con un error arbitrariamente pequeño si usa un número k suficientemente grande de funciones gausianas y el ancho σ^2 de cada función K se puede especificar de forma separada.
- Puede verse como una red de dos capas en la cual en la primera se calculan los centros x_i y las desviaciones estándares σ^2 de cada una de las funciones K_i, y la segunda capa realiza la combinación lineal de las mismas según los pesos w_i.



Aprendizaje con RBFs

- El diseño de una red RBF implica escoger
  1. El tipo y número de RBFs
  2. Sus correspondientes μ_j y σ_j
  3. y los w_j
- Aprendizaje de dos tipos
  - En una sola fase mediante gradiente
  - En dos fases mediante clustering+cálculo matricial

Aprendizaje en una fase

- Aprendizaje supervisado, con gradiente descendente
  - Definición del error

$$E = \frac{1}{2} \sum_n \sum_k (t_k^n - y_k(x^n))^2,$$

siendo $y_k(x) = \sum_{j=0}^{M} w_{kj} K_j(x)$ y t_k^n el valor objetivo para la unidad k y el ejemplar n

  - Ecuaciones de actualización

$$\begin{aligned}
\Delta w_{kj} &= \eta_1 (t_k^n - y_k(x^n)) K_j(x^n) \\
\Delta \mu_j &= \eta_2 K_j(x^n) \frac{||x^n - \mu_j||}{\sigma_j^2} \sum_k (y_k(t_k^n) - x^n) w_{kj} \\
\Delta \sigma_j &= \eta_3 K_j(x^n) \frac{||x^n - \mu_j||^2}{\sigma_j^3} \sum_k (y_k(t_k^n) - x^n) w_{kj}
\end{aligned}$$

Aprendizaje con RBFs

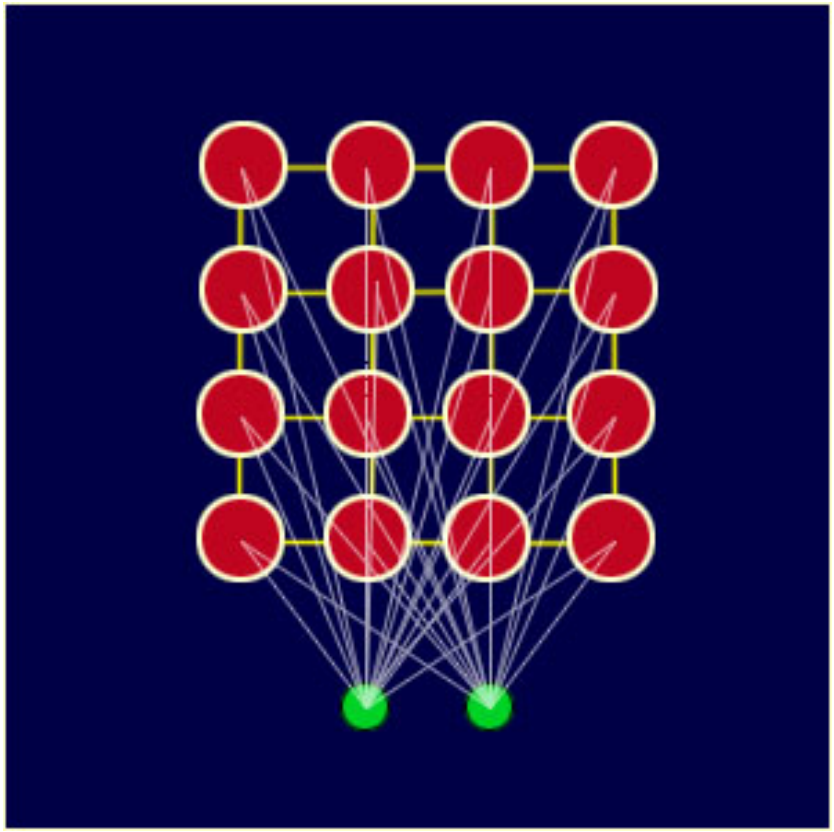
- Aprendizaje en dos fases
  1. Determinar los μ_j y los σ_j
     - ★ Aprendizaje no supervisado (solamente tenemos en cuenta los valores de entrada)
     - ★ Las RBFs deben representar las densidades de los datos de entrada (colocar al menos una RBF en zonas en donde existe presencia de datos)
     - ★ Selección aleatoria de subconjunto
     - ★ Clustering
  2. Determinar después los w_{ij}
     - ★ La RBF puede verse ahora como una red de una sola capa
     - ★ Determinación de los pesos mediante la pseudo inversa

$$W^T = \Phi^t T$$

en donde $(T)_{nk} = t_k^n$ y $(\Phi)_{nj} = K_j(x^n)$ y $\Phi^t = (\Phi^T \Phi)^{-1} \Phi^T$

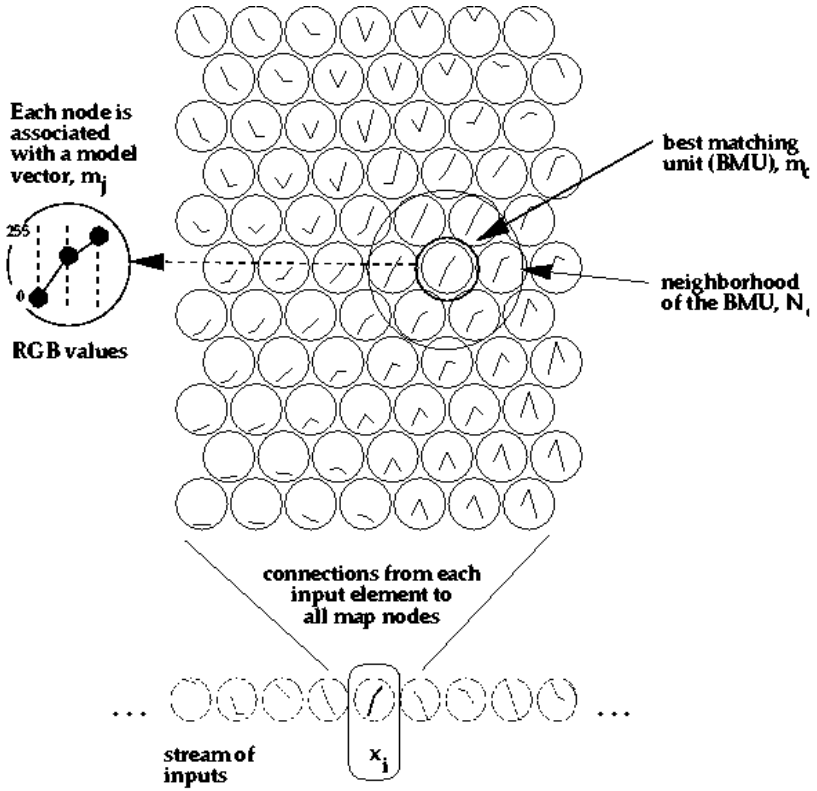
SOM - Mapas auto organizativos (ver [?])

- Los SOM (*Self-Organizing Map*) pueden verse como un array de neuronas, dispuestas espacialmente en un array de pocas dimensiones, típicamente 1 o 2


[1]

[1] http://www.ai-junkie.com/ann/som/images/Figure2.jpg

SOM - Mapas auto organizativos (otro ejemplo)



SOM - Organización

- La entrada está totalmente conectada (*fully connected*) al array
- Cada nodo del mapa, representado mediante un círculo en la malla, sirve como un modelo o prototipo de una clase de entradas similares
- los nodos del mismo se ajustan a determinadas señales de entrada (i.e. los patrones o grupos de patrones) de una manera más o menos ordenada
- El proceso de aprendizaje es competitivo
  - cada vez, solamente una neurona produce la señal de activación mayor y, por lo tanto, gana al resto
- es no supervisado ya que no es necesario un instructor para que el SOM detecte regularidades y las refleje en el array
- también llamados mapas de Kohonen, fueron creados por el finlandés Teuvo Kohonen a principios de los ochenta
- detalles en http://websom.hut.fi/websom/

SOM - El algoritmo básico de aprendizaje

Vamos a explicar con un ejemplo cómo se ajustan las neuronas de este tipo de mapa. Supongamos el conjunto de datos siguiente:

Rojo	Verde	Azul	Color
250	235	215	antique white
165	042	042	brown
222	184	135	burlywood
210	105	30	chocolate
255	127	80	coral
184	134	11	dark goldenrod
189	183	107	dark khaki
255	140		dark orange
233	150	122	dark salmon
...	...	...	...

El algoritmo básico de aprendizaje (II)

- cada entrada representa un color concreto, expresado en base al sistema RGB(*Red, Green, Blue*)
- denotamos cada uno de los ejemplares del conjunto de datos de entrada con $x(t) \in R^n$ siendo t el índice del ejemplar (o bien una coordenada temporal discreta)
- cada nodo i contiene un vector modelo $m_i(t) \in R^n$, que tiene el mismo número de elementos (i.e. dimensiones) que los ejemplares de entrada

Versión estocástica

- estocástica $\equiv$ una iteración por ejemplar
- el vector de modelos $m_i(t)$ se puede inicializar de manera aleatoria
- cada ejemplar de entrada $x(t)$ se ha de mapear a la localización o modelo $m_i(t)$ en el array en la cual se encuentra mejor
- Este modelo se denominará entonces *best matching unit* (BMU). Y se seleccionará para que cumpla

$$||x(t) - m_{BMU}(t)|| = \min_i ||x(t) - m_i(t)||,$$

1. El vector de entrada $x(t)$ se compara con todos los modelos $m_i(t)$ del array para identificar el modelo $m_k(t)$ que más se parece (e.g. mediante distancia euclídea si los componentes del vector de cada característica son continuos). A $m_k(t)$ lo denominamos el modelo ganador.
2. Tanto el ganador como ciertos nodos localizados en su vecindad se ajustan para hacerlos similares al ejemplar de entrada $x(t)$ mediante un proceso de aprendizaje que especificamos posteriormente.

El algoritmo básico de aprendizaje (IV)

El método de aprendizaje al que aludíamos arriba se puede resumir en las siguientes ecuaciones de actualización de modelos:

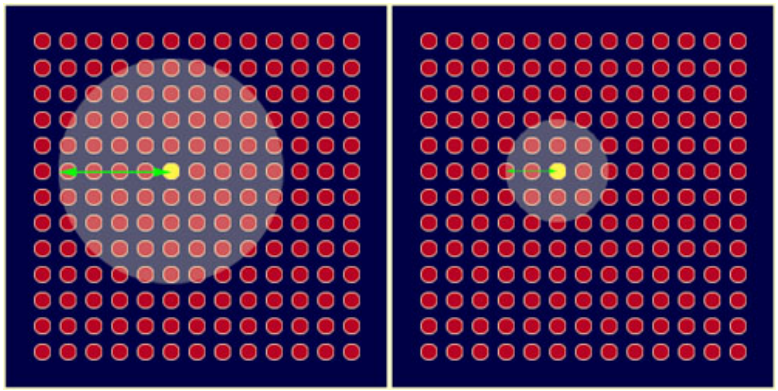
$$\begin{cases} m_i(t+1) = m_i(t) + \alpha[x_i(t) - m_i(t)] & \forall i \in N_c(t), \\ m_i(t+1) = m_i(t) & \text{si no} \end{cases} \quad (2)$$

Donde

- t es el índice de la iteración
- $\alpha \in [0, 1]$ es el ratio de aprendizaje
- $N_c(t)$ especifica la vecindad al rededor del nodo c ganador, en la iteración $t-$ésima

El algoritmo básico de aprendizaje (V)

- El radio de vecindad decrece lentamente
  - el orden global se consigue en una fase temprana y las correcciones locales se hacen posteriormente, cuando la vecindad es más estrecha
- se decrementa, por la misma razón, el valor del parámetro α con lo que, en realidad, tenemos una serie de valores que nos da una función $\alpha(t)$



El algoritmo básico de aprendizaje (V)

- para especificarlo más correctamente, también podríamos usar la expresión

$$m_i(t+1) = m_i(t) + \alpha(t) h_{bi}(t)[x_i(t) - m_i(t)], \ \forall i,$$

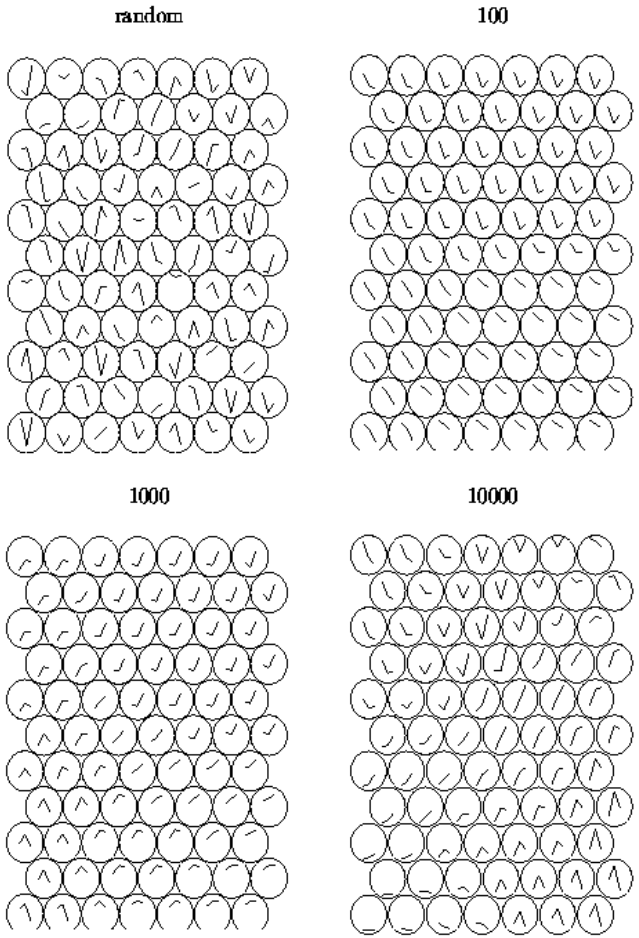
donde

  - h_{bi} es una función de base radial, centrada en el nodo ganador cuya salida decrece con la distancia al mismo, y que podría ser

$$h_{bi}(t) = e^{-\frac{||r_b - x_i||^2}{2\sigma^2(t)}},$$

con r_b y r_i son las posiciones de las neuronas b e i respectivamente, y $\sigma(t)$ es el radio de vecindad

Evolución del aprendizaje



El papel de los SOM en el análisis inteligente de datos

- SOM realizan una operación de representación, mediante un grupo de vectores, de un sistema continuo
- Algo así como una compresión con pérdida bastante grosera
- Lo que realmente está realizando es un clustering!!!

Podemos utilizar los SOM en

- Preparación de datos
- Visualización
- Haciendo uso de los datos de entrada en el mapa

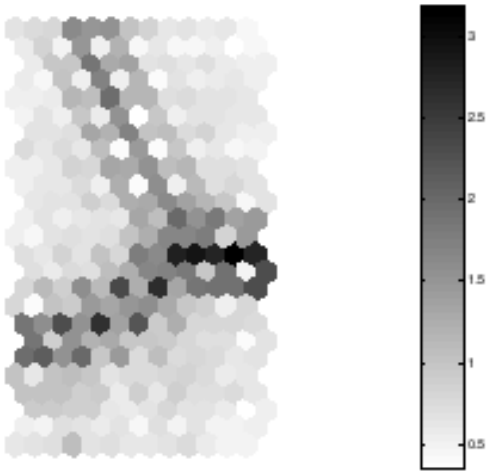
SOM - Preparación de datos

- Reducción de datos
  - se podría usar un pequeño porcentaje de los datos, no todos, seleccionados separadamente en cada uno de los diagramas de Voronoi de cada modelo del mapa
- Discretización: hacemos corresponder cada modelo con una salida discreta con lo que para valores continuos podemos obtener valores a la salida discretizados (discretización por histogramas)
- De símbolos a números
  - los SOM no pueden utilizar símbolos a la entrada
  - es posible localizar su posición (la de los símbolos) en el mapa una vez se ha aprendido el mapa haciendo uso de las variables continuas
  - Solamente en el caso de que símbolos diferentes tengan posiciones diferentes en el mapa, se pueden utilizar las coordenadas para sustituir a dichos símbolos
- Valores nulos
  - al calcular el BMU de un ejemplar con valores nulos, solamente usamos los valores conocidos
  - asumimos que para los valores nulos, éstos son parecidos (o iguales, en realidad) a los que sí están presentes en los modelos

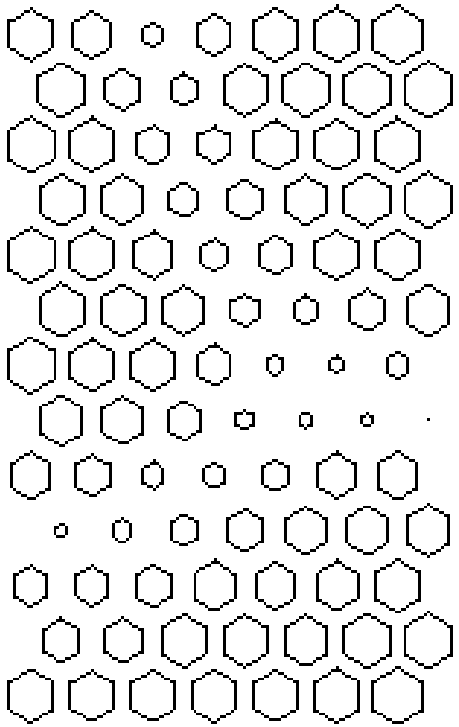
Visualización

- SOM es en realidad una representación de los datos de entrenamiento mediante 1 o dos dimensiones
- El SOM, convenientemente ordenado puede utilizarse para visualizar diferentes propiedades del mismo y, por lo tanto, de los datos.
- Técnicas
  1. Las matrices de distancias
  2. Mapas coloreados

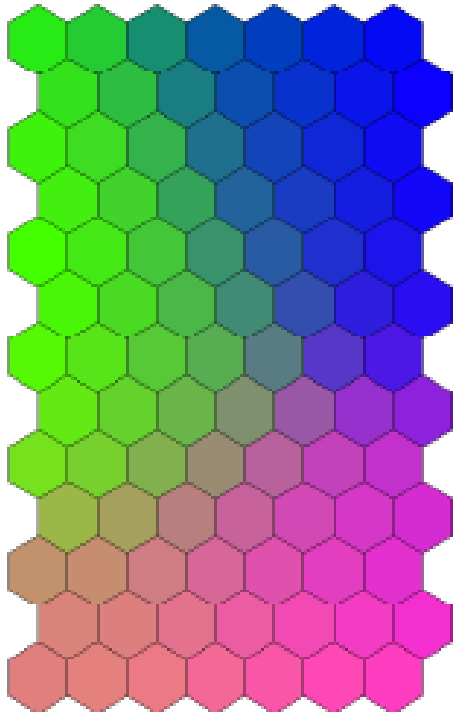
Visualización (II)



(a)



(b)



(c)

Dimensiones de los datos aisladas en el mapa

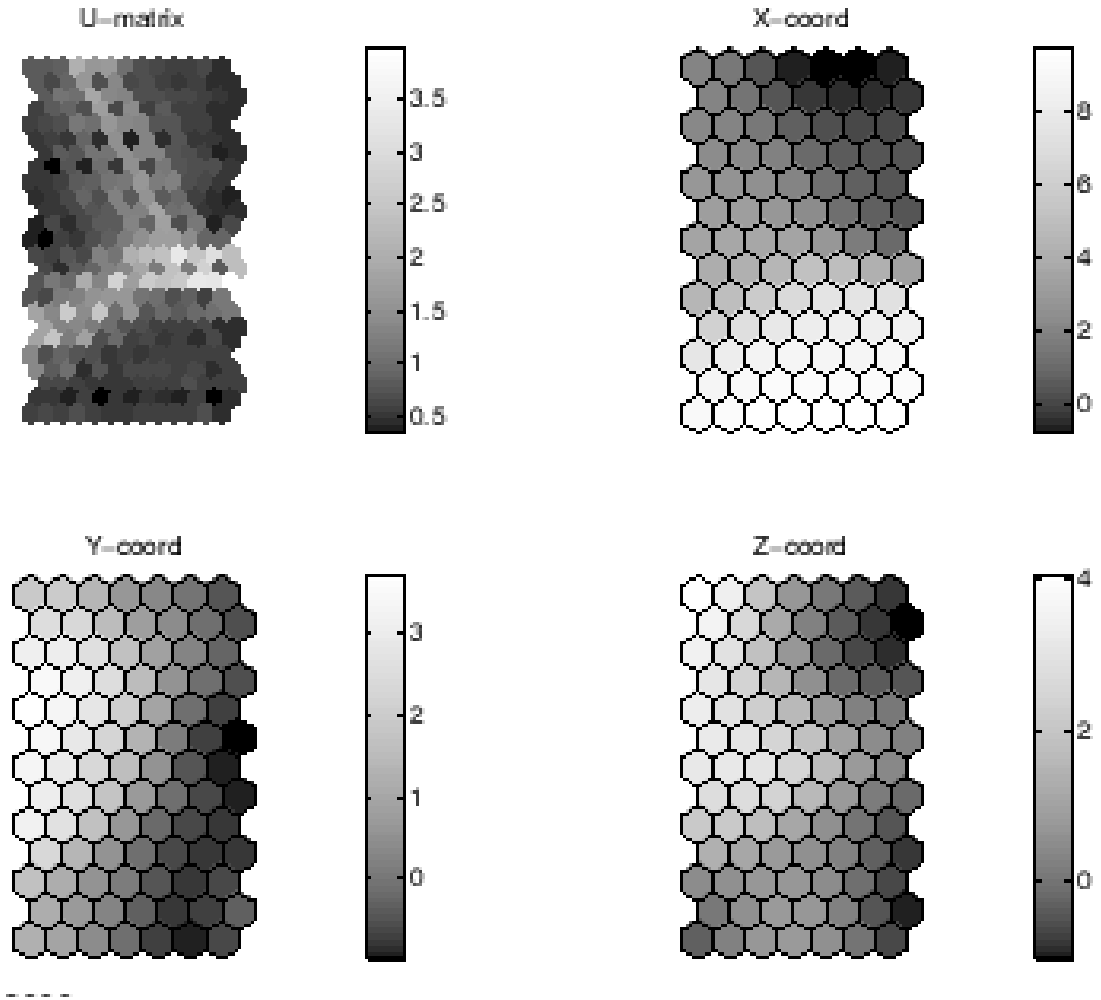
Separando las dimensiones podemos responder a las preguntas

- qué tipos de valores tienen las variables de entrada?
- qué tipos de valores o combinación de los mismos son típicos en los diferentes clusters?
- Existen dependencias significantes entre las variables?

Dimensiones de los datos aisladas en el mapa

- Cada dimensión puede verse como una rodaja o sección del mapa
- Cada componente visualiza la dispersión de los valores de cada dimensión en el mapa y como tales, son muy similares a los histogramas
- si usamos cada mapa individualmente, podemos detectar acumulaciones de valores de cada variable en determinadas regiones del mapa
- Si usamos varios mapas, podemos detectar correlaciones de varias variables

Dimensiones de los datos aisladas en el mapa (y II)



Normalización de la entrada

- Es interesante rescalar las variables de entrada en cuando diferentes variables difieren significativamente en la magnitud de sus valores
- Incluso puede que la diferencia no refleje la importancia relativa en la salida de la red
- Si tratamos cada variable de manera independiente
  - Para cada variable x_i, calculamos

$$\overline{x}_i = \frac{1}{N}\sum_{i=1}^{N} x_i^n \text{ y } \sigma_i^2 = \frac{1}{N-1}\sum_{n=1}^{N}(x_i^n - \overline{x}_i)^2,$$

  - Las nuevas variables rescaladas serán

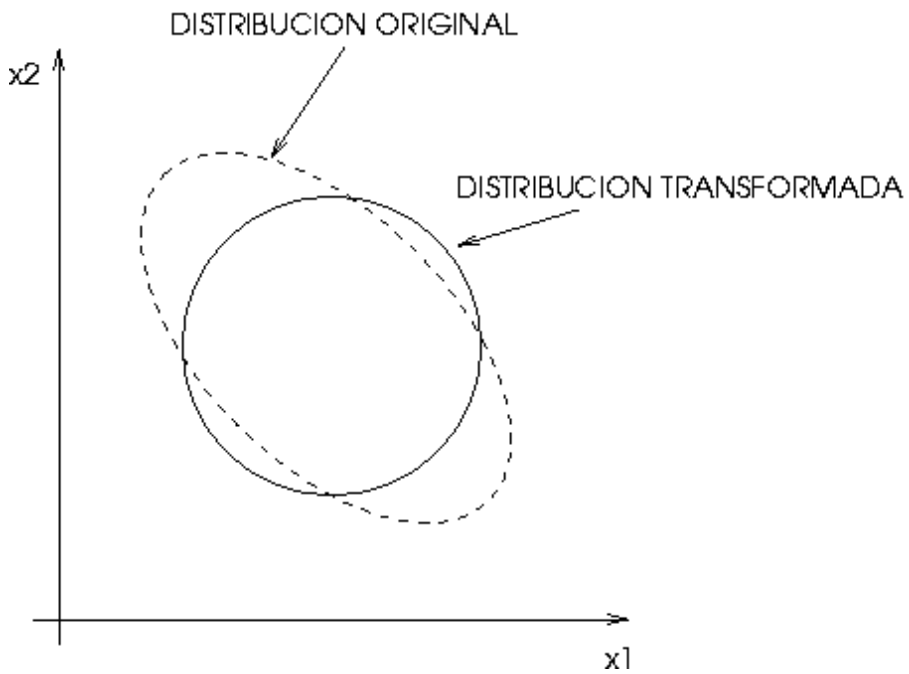
$$\hat{x}_i^n = \frac{x_i^n - \overline{x}_i}{\sigma_i}.$$

  - Las nuevas variables tienen media cero y varianza 1
  - Si el problema es de regresión se debe transformar la salida
- El proceso de inicialización aleatoria tiene ahora más sentido

Normalización teniendo en cuenta covarianzas

- Para que una RBF trabaje bien, todas las variables deben expandirse en el mismo rango y su covarianza debe ser mínima ya que su activación depende de

$$||x - \mu_j||^2 = \sum_{i=1}^{d} \{x_i - \mu_{ji}\}^2$$



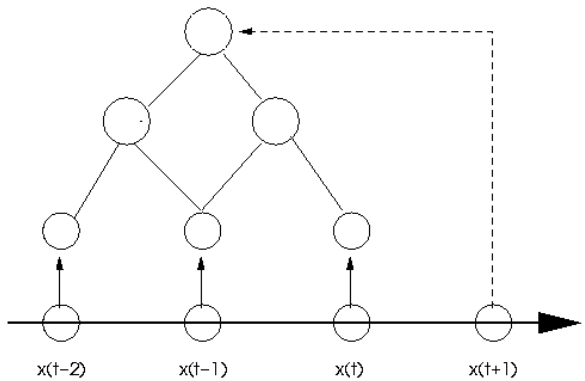
Datos discretos

- ¿Cómo usamos datos discretos en la entrada de una red?
- Si son ordinales es fácil (e.g. edad de una persona, fecha)
- Si son categóricos, no tienen un orden asociado
  - color de un objeto, con valores posibles $\{$rojo, verde, azul$\}$
  - el hacerles corresponder los valores $\{0,0.5,1.0\}$ induce un orden artificial
  - codificación 1-de-c: tres variables de entrada, una para cada color, de tal forma que la codificación será $\{(1,0,0),(0,1,0),(0,0,1)\}$

Series temporales

- el objetivo en este tipo de problemas es el de predecir el valor de x en un corto período de tiempo en el futuro
- Adecuado aplicar redes feed-forward, siempre que los datos se hayan preprocesado correctamente
- Se muestrea $x(t)$ en intervalos de tiempo iguales para generar una serie discreta de valores x_{t-1}, x_t, x_{t+1} y con d valores creamos un ejemplar

$$(x, y) = ((x_{t-d+1}, x_{t-d+2}, \ldots, x_t), x_{t+1})$$

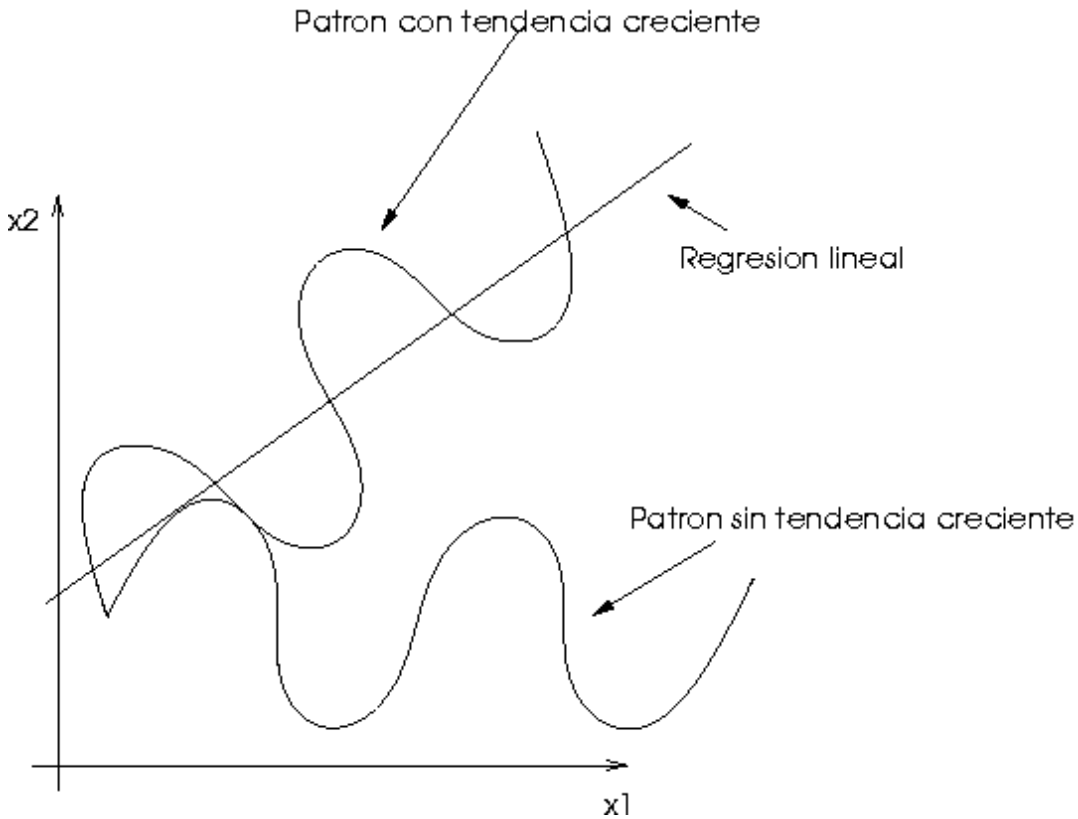


Predicciones n-steps ahead

- Podemos predecir x_{t+1} apartir de $x_{t-d+1}, x_{t-d+2}, \ldots, x_t$ (1-step ahead)
- ¿Y para predecir x_{t+n}, $n \geq 2$?
  - predecimos primero x_{t+1}
  - para la siguiente predicción usamos como entrada $x_{t-d}, x_{t-d+2}, \ldots, x_t, x_{t+1}$ y obtendremos x_{t+2}
- los errores se van acumulando
- la tendencia real se aleja conforme avanzamos en el horizonte de predicción, de la modelada por la red

Series temporales: problemas

- ¿Cómo elegimos el intervalo de muestreo?
- Trending




Michael Berthold and David J. Hand.
Intelligent Data Analysis. An Introduction.
Springer, 2003.
Second edition.


S. E. Fahlman and C. Lebiere.
The cascade-correlation learning architecture.
In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, Denver 1989, 1990. Morgan Kaufmann, San Mateo.


M. Minsky and S. Papert.
Perceptrons.
MIT Press, 1969.


Tom M. Mitchell.
Machine Learning.
McGraw-Hill, 1997.


J. Moody and C. Darken.
Fast learning in networks of locally tuned processing units.

Neural Computation, 1:281–294, 1989.


Martin Riedmiller and Heinrich Braun.
A direct adaptive method for faster backpropagation learning: The RPROP algorithm.
In *Proc. of the IEEE Intl. Conf. on Neural Networks*, pages 586–591, San Francisco, CA, 1993.


Juha Vesanto.
Using the SOM and local models in time-series prediction.
In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4–6*, pages 209–214. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland, 1997.

Desarrollo A

Si hacemos el desarrollo por componentes vectoriales,

$$\sum_{j=0}^{M} w_j \vec{\phi}_j - \vec{t} = (w_1\phi_{11} + \ldots + w_m\phi_{m1}, \ldots, w_1\phi_{1N} + \ldots + w_M\phi_{MN}) - (t_1, \ldots, t_N),$$

y entonces sea

$$v = (w_1\phi_{11} + \ldots + w_m\phi_{m1} - t_1, \ldots, w_1\phi_{1N} + \ldots + w_M\phi_{MN} - t_N),$$

y como

$$\|v\|^2 = v' = \left((w_1\phi_{11} + \ldots + w_m\phi_{m1} - t_1)^2, \ldots, (w_1\phi_{1N} + \ldots + w_M\phi_{MN} - t_N)^2\right),$$

tenemos que

$$\frac{\partial 1/2v'}{\partial w_j} = ((w_1\phi_{11} + \ldots + w_m\phi_{m1} - t_1)\phi_{j1}, \ldots, (w_1\phi_{1N} + \ldots + w_M\phi_{MN} - t_N)\phi_{jN}) = \vec{\phi}_j^T(\vec{y} - \vec{t})$$