Perceptron Multicapa

Introducción

- Minsky y Papert (1969) demostraron que el perceptron simple (y Adaline) no pueden resolver problemas no lineales (como XOR).
- La combinación de varios perceptrones podría resolver ciertos problemas no lineales pero no sabían como adaptar los pesos de las capas ocultas
- Rumelhart y otros autores (1986) presentaron la Regla Delta Generalizada para adaptar los pesos propagando los errores hacia atrás (retropropagación), para múltiples capas y funciones de activación no lineales
- (Applet: http://neuron.eng.wayne.edu/software.html)

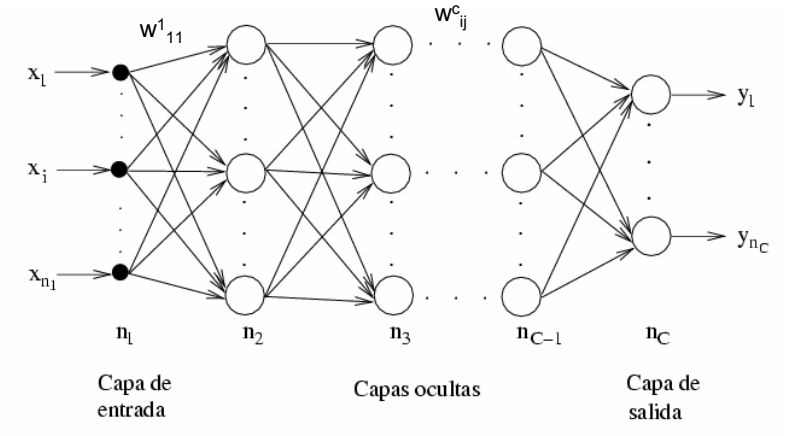
Introducción

- Se demuestra que el Perceptron Multicapa (MLP) es un APROXIMADOR UNIVERSAL
- Un MLP puede aproximar relaciones no lineales entre datos de entrada y de salida
- Es una de las arquitecturas más utilizadas en la resolución de problemas reales:
  - por ser aproximador universal
  - por su fácil uso y aplicabilidad
- Se ha aplicado con éxito en:
  - reconocimiento de voz
  - reconocimiento de imágenes
  - OCR
  - conducción de vehículos
  - diagnósticos médicos, etc…

Arquitectura

- Capa de entrada: sólo se encargan de recibir las señales de entrada y propagarlas a la siguiente capa
- Capa de salida: proporciona al exterior la respuesta de la red para cada patrón de entrada
- Capas ocultas: Realizan un procesamiento no lineal de los datos recibidos
- Son redes "feedforward": alimentadas hacia adelante
- Generalmente cada neurona está conectada a todas las neuronas de la siguiente capa (conectividad total)

Arquitectura



Propagación de los patrones de entrada

- El perceptron multicapa define una relación entre las variables de entrada y las variables de salida de la red
- Esta relación se obtiene propagando hacia adelante los valores de las variables de entrada
- Cada neurona de la red procesa la información recibida por sus entradas y produce una respuesta o activación que se propaga, a través de las conexiones correspondientes, hacia las neuronas de la siguiente capa.

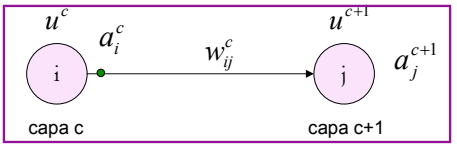
Propagación de los patrones de entrada. Notación

– C capas, n_c neuronas en la capa c=1, 2, …C

– Matriz de pesos y vector de umbrales de cada capa:

$$W^c = (w_{ij}^c) = \begin{pmatrix} w_{11}^c & w_{12}^c & \dots & w_{1n_{c+1}}^c \\ w_{21}^c & w_{22}^c & \dots & w_{2n_{c+1}}^c \\ \vdots & \vdots & & \vdots \\ w_{n_c 1}^c & w_{n_c 2}^c & \dots & w_{n_c n_{c+1}}^c \end{pmatrix} \qquad U^c = (u_i^c) = \begin{pmatrix} u_1^c \\ u_2^c \\ \vdots \\ u_{n_c}^c \end{pmatrix}$$

– Activación de la neurona i: a^c_i



Propagación de los patrones de entrada. Activaciones

- Activación de las neuronas de entrada

$$a_i^1 = x_i \text{ para } i = 1, 2, ..., n_1$$

donde $X = (x_1, x_2, ..., x_{n_1})$ representa el vector de entrada

- Activación de las neuronas de la capa oculta

$$a_i^c = f\left(\sum_{j=1}^{n_{c-1}} w_{ji}^{c-1} a_j^{c-1} + u_i^c \right) \text{ para } i = 1, 2, ..., n_c \text{ y } c = 2, 3..., C-1$$

donde a_j^{c-1} son las activaciones de la capa j-1

Propagación de los patrones de entrada. Activaciones

- Activación de las neuronas de la capa de salida

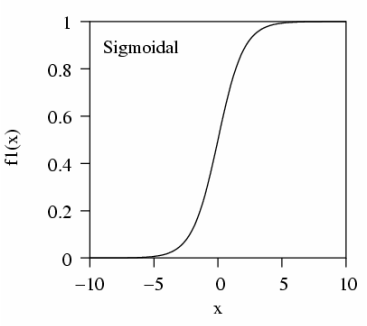
$$y_i = a_i^C = f\left(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C \right) \text{ para } i = 1, 2, ..., n_C$$

donde $Y = (y_1, y_2, ..., y_{n_C})$ es el vector de salida de la red

Función de activación

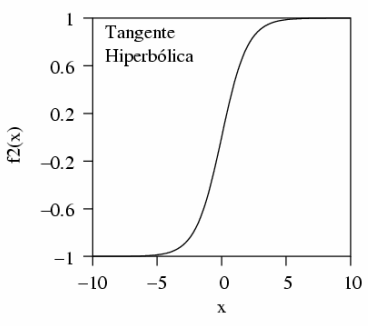
- Las funciones más utilizadas son
  - **Función Sigmoidal**

$$f_1(x) = \frac{1}{1 + e^{-x}}$$



- Tangente Hiperbólica

$$f_2(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$



Función de activación

- Ambas son crecientes con dos niveles de saturación
- Normalmente f es común a todas las neuronas
- La elige el diseñador según el valor de activación que se desee [-1,1] o [0,1]
- Ambas funciones están relacionadas: $f_2(x) = 2f_1(x) - 1$
- **El perceptron multicapa define, a través de sus conexiones y neuronas, una función continua no lineal del espacio $\mathbf{R}^{n_1}$ en el espacio $\mathbf{R}^{n_O}$**

$$Y = F(X, W)$$

Diseño de la arquitectura

Hay que decidir:

- Función de activación
  - Nos basamos en el recorrido deseado. No suele influir en la capacidad de la red para resolver un problema
- Número de neuronas de entrada y de salida
  - Vienen dados por las variables que definen el problema
  - A veces no se conoce el número de variables de entrada relevantes: es conveniente realizar un análisis previo de las variables de entrada para descartar las que no aportan información a la red (PCA, GA,..)
- Número de capas y neuronas ocultas
  - Debe ser elegido por el diseñador
  - No existe un método para determinar el número óptimo de neuronas ocultas para resolver un problema dado
  - Dado un problema, puede haber un gran número de arquitecturas capaces de resolverlo

MLP: Algoritmo de Retropropagación

Introduccón

Objetivo: Ajustar los pesos de la red para minimizar el error global

$$E = \frac{1}{N}\sum_{n=1}^{N} e(n)$$

Error medio para todos los patrones

Salida deseada, Salida

$$e(n) = \frac{1}{2}\sum_{i=1}^{n_C} (s_i(n) - y_i(n))^2$$

Error para el patrón n

Problema de minimización no lineal

Funciones de activación no lineales hacen que la respuesta de la red sea no lineal respecto a los pesos

Adaptación de los parámetros siguiendo una dirección de búsqueda

Dirección negativa del gradiente de la función de error E

Hay otros métodos de realizar la búsqueda: aleatoria, técnicas evolutivas, etc

Introduccón

- El ajuste de los pesos se hace casi siempre por patrones (métodos de gradiente estocástico):

  Sucesiva minimización de los errores para cada patrón e(n), en lugar de minimizar el error global

- Cada peso w se modifica para cada patrón de entrada n de acuerdo con la ley de aprendizaje:

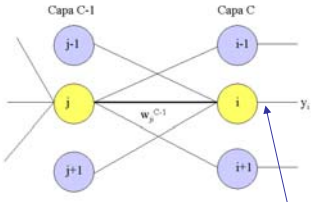
$$\Delta w = -\alpha \frac{\partial e(n)}{\partial w}$$

$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w}$$

Obtención de la regla delta generalizada

•Pesos de la última capa

$$w_{ji}^{C-1}(n) = w_{ji}^{C-1}(n-1) - \alpha \frac{\partial e(n)}{\partial w_{ji}^{C-1}}$$



$$\frac{\partial e(n)}{\partial w_{ji}^{C-1}} = \frac{\partial e(n)}{\partial y_i(n)} \frac{\partial y_i(n)}{\partial w_{ji}^{C-1}} \quad [1]$$

El peso w_{ji}^{C-1} sólo afecta a y_i, el resto de salidas no dependen de él

$$e(n) = \frac{1}{2}\sum_{i=1}^{n_C}(s_i(n) - y_i(n))^2 = \frac{1}{2}(s_1(n) - y_1(n))^2 + \ldots + \frac{1}{2}(s_i(n) - y_i(n))^2 + \ldots$$

$$\frac{\partial e(n)}{\partial y_i(n)} = -(s_i(n) - y_i(n))$$

Por tanto,

$$\frac{\partial e(n)}{\partial w_{ji}^{C-1}} = -(s_i(n) - y_i(n)) \frac{\partial y_i(n)}{\partial w_{ji}^{C-1}}$$

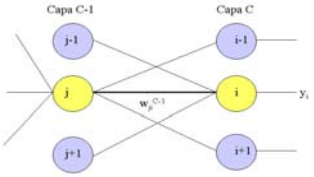
Sólo la derivada de la salida y_i es distinta de 0

Ahora hay que calcular: $\frac{\partial y_i(n)}{\partial w_{ji}^{C-1}}$

Regla delta generalizada (pesos de la última capa)

La salida y_i es:

$$y_i = a_i^C = f(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C) \quad [2]$$



Es la derivada de [2], sólo el sumando j depende de w_{ij}

Por tanto, su derivada:

$$\frac{\partial y_i(n)}{\partial w_{ji}^{C-1}} = f'(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C) a_j^{C-1}$$

Por tanto, la derivada del error será:

$$\frac{\partial e(n)}{\partial w_{ji}^{C-1}} = -(s_i(n) - y_i(n)) f'(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C) a_j^{C-1}$$

$\rightarrow \boldsymbol{\delta_i}$

Se define el término δ asociado a la neurona i de la capa C:

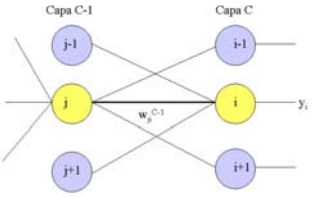
$$\delta_i^C(n) = (s_i(n) - y_i(n)) f'(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C) \quad [3]$$

Por tanto [1] quedará: $\frac{\partial e(n)}{\partial w_{ji}^{C-1}} = -\delta_i^C(n) a_j^{C-1}$

Regla delta generalizada (pesos de la última capa)

Por tanto, el nuevo peso será

$$w_{ji}^{C-1}(n) = w_{ji}^{C-1}(n-1) + \alpha \delta_i^C(n) a_j^{C-1}(n)$$

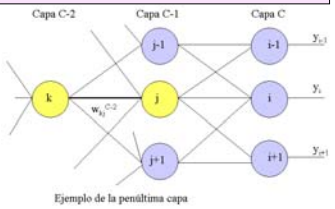


Lo mismo ocurre con los umbrales de la última capa:

$$u_i^C(n) = u_i^C(n-1) + \alpha \delta_i^C(n)$$

Para modificar el peso w_{ji} basta considerar
- la activación de la neurona origen j
- y el término δ asociado a la neurona destino i

Regla delta generalizada (pesos de la penúltima capa)



•Pesos de la capa C-2 a la capa C-1

$$w_{kj}^{C-2}(n) = w_{kj}^{C-2}(n-1) - \alpha \frac{\partial e(n)}{\partial w_{kj}^{C-2}}$$

Ahora este peso afecta a todas las salidas

$$e(n) = \frac{1}{2}\sum_{i=1}^{n_C}(s_i(n) - y_i(n))^2 = \frac{1}{2}(s_1(n) - y_1(n))^2 + \ldots + \frac{1}{2}(s_i(n) - y_i(n))^2 + \ldots$$

Por tanto la derivada del error será

todas las salidas *y* dependen del peso w_{kj}

$$\frac{\partial e(n)}{\partial w_{kj}^{C-2}} = -\sum_{i=1}^{n_C}(s_i(n) - y_i(n))\frac{\partial y_i(n)}{\partial w_{kj}^{C-2}} \quad [4]$$

suma de las derivadas de cada una de las salidas de la red

Ahora hay que calcular la derivada de yi respecto al peso wkj

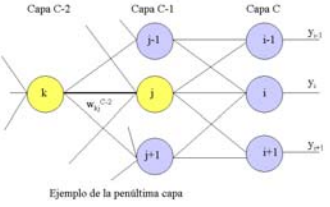
Regla delta generalizada (pesos de la penúltima capa)

Como y_i es:

$$y_i = a_i^C = f\left(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C\right)$$

ahora la variable w_{kj}^{C-2} está dentro de 1 término a_j

$$\frac{\partial y_i(n)}{\partial w_{kj}^{C-2}} = f'\left(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C\right)\omega_{ji}^{C-1}\frac{\partial a_j^{C-1}}{\partial w_{kj}^{C-2}}$$



Se sustituye este valor en [4]:

$$\frac{\partial e(n)}{\partial w_{kj}^{C-2}} = -\sum_{i=1}^{n_C}(s_i(n) - y_i(n))f'\left(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C\right)w_{ji}^{C-1}\frac{\partial a_j^{C-1}}{\partial w_{kj}^{C-2}}$$

δ_i

Regla delta generalizada (pesos de la penúltima capa)

Por tanto y de acuerdo a la definición δ de [3], queda:

$$\frac{\partial e(n)}{\partial w_{kj}^{C-2}} = -\sum_{i=1}^{n_C} \delta_i^C w_{ji}^{C-1} \frac{\partial a_j^{C-1}}{\partial w_{kj}^{C-2}}$$

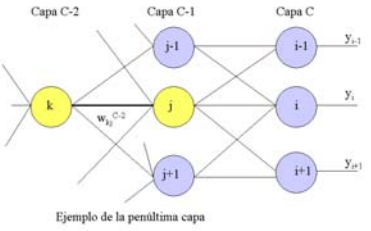
[5]

La derivada de la activación de la neurona j es:

$$\frac{\partial a_j^{C-1}}{\partial w_{kj}^{C-2}} = f'(\sum_{k=1}^{n_{C-2}} w_{kj}^{C-2} a_k^{C-2} + u_j^{C-1}) a_k^{C-2}$$

[6]

El único término que depende de w_{kj} es el que sale de k



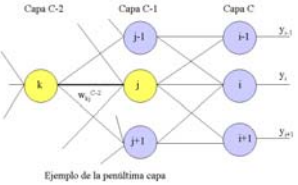
$$\frac{\partial e(n)}{\partial w_{kj}^{C-2}} = -\sum_{i=1}^{n_C} \delta_i^C w_{ji}^{C-1} f'(\sum_{k=1}^{n_{C-2}} w_{kj}^{C-2} a_k^{C-2} + u_j^{C-1}) a_k^{C-2}$$

$\boldsymbol{\delta_j}$

Regla delta generalizada (pesos de la penúltima capa)

Se define el término δ asociado a la neurona j de la capa C-1

$$\delta_j^{C-1} = f'(\sum_{k=1}^{n_{C-2}} w_{kj}^{C-2} a_k^{C-2} + u_j^{C-1}) \sum_{i=1}^{n_C} \delta_i^C w_{ji}^{C-1}$$



Sustituyendo [6] en [5] y de acuerdo con la definición de δ asociado a la neurona j de la capa C-1:

$$\frac{\partial e(n)}{\partial w_{kj}^{C-2}} = -\delta_j^{C-1} a_k^{C-2}$$

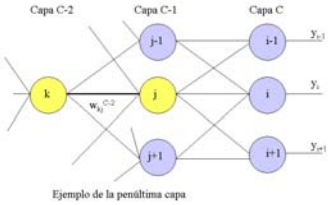
Regla delta generalizada (pesos de la penúltima capa)

Por tanto, el nuevo peso será

$$w_{kj}^{C-2}(n) = w_{kj}^{C-2}(n-1) + \alpha\delta_j^{C-1}(n)a_k^{C-2}(n)$$

Lo mismo ocurre con los umbrales de la penúltima capa:

$$u_j^{C-1}(n) = u_j^{C-1}(n-1) + \alpha\delta_j^{C-1}(n)$$



Ejemplo de la penúltima capa

Para modificar el peso w_{kj} basta considerar
- la activación de la neurona origen k
- y el término δ asociado a la neurona destino j

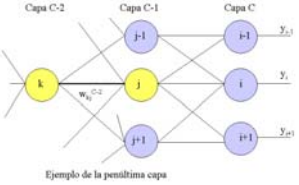
Regla delta generalizada (pesos de la penúltima capa)

El término δ de la neurona j capa C-1 viene dado por:

Derivada de la función de activación de j

Suma de los términos δ de la siguiente capa multiplicados por los correspondientes pesos

$$\delta_j^{C-1} = f'(\sum_{k=1}^{n_{C-2}} w_{kj}^{C-2}a_k^{C-2} + u_j^{C-1})\sum_{i=1}^{n_C}\delta_i^C w_{ji}^{C-1}$$



Ejemplo de la penúltima capa

Regla delta generalizada. Capa oculta c

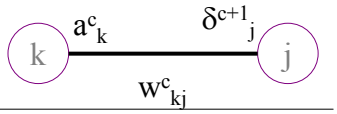
Se generaliza para los pesos de cualquier capa c a la capa c+1 (c=1,2,.... C-2)

$$w_{kj}^{c}(n) = w_{kj}^{c}(n-1) + \alpha\delta_{j}^{c+1}(n)a_{k}^{c}(n)$$

$$u_{j}^{c+1}(n) = u_{j}^{c+1}(n-1) + \alpha\delta_{j}^{c+1}(n)$$

k=1,2,....n_c
j=1,2,.....n_{c+1}
c=1,2,.... C-2

Basta tener en cuenta la activación de la que parte la conexión y el término δ de la neurona a la que llega la conexión.



Regla delta generalizada. Capa oculta c

El término δ de una neurona j se calcula utilizando la derivada de su función de activación y la suma de los términos δ de las neuronas de la siguiente capa

$$\delta_{j}^{c+1} = f'(\sum_{k=1}^{n_c} w_{kj}^{c}a_{k}^{c} + u_{j}^{c+1})\sum_{i=1}^{nc+2}\delta_{i}^{c+2}w_{ji}^{c+1}$$

Regla delta generalizada. Derivada de f

- **Derivada de la función de activación**
  - **Función sigmoidal**

$$f_1(x) = \frac{1}{1+e^{-x}} \qquad f'_1(x) = \frac{-1}{(1+e^{-x})^2}(-e^{-x})$$

$$f_1'(x) = \frac{1}{1+e^{-x}}\frac{e^{-x}}{1+e^{-x}} = \frac{1}{1+e^{-x}}\left(\frac{1+e^{-x}-1}{1+e^{-x}}\right) = \frac{1}{1+e^{-x}}\left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right)$$

$$f_1'(x) = f_1(x)(1-f_1(x))$$

Por tanto, como la δ de la última capa es:

Para simplificar la notación, suprimimos la referencia al patrón n, (n)

$$\delta_i^C = (s_i - y_i) f'\left(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C\right)$$

resulta:

$$\delta_i^C = (s_i - y_i) y_i (1 - y_i)$$

Ultima capa

Regla delta generalizada. Derivada de f

Para el resto de las capas:

$$\delta_j^{c+1} = f'\left(\sum_{k=1}^{n_c} w_{kj}^c a_k^c + u_j^{c+1}\right)\sum_{i=1}^{n_{c+2}} \delta_i^{c+2} w_{ji}^{c+1}$$

por tanto:

Resto de capas

$$\delta_j^{c+1} = a_j^{c+1}(1 - a_j^{c+1})\sum_{i=1}^{n_{c+2}} \delta_i^{c+2} w_{ji}^{c+1}$$

Resumen de la regla delta generalizada

- Cada neurona de salida distribuye hacia atrás su error (valor δ) a todas las neuronas ocultas que se conectan a ella ponderado por el valor de la conexión.

- Así, cada neurona oculta recibe un cierto error (δ) de cada neurona de salida, siendo la suma el valor δ de la neurona oculta.

- Estos errores se van propagando hacia atrás, llegando a la primera capa

Resumen de la regla delta generalizada. Expresiones

Para la última capa:

$$w_{ji}^{C-1}(n) = w_{ji}^{C-1}(n-1) + \alpha\delta_i^C(n)a_j^{C-1}(n)$$
$$u_i^C(n) = u_i^C(n-1) + \alpha\delta_i^C(n)$$

donde :

$$\delta_i^C = (s_i - y_i)y_i(1-y_i)$$

Para el resto de capas:

$$w_{kj}^c(n) = w_{kj}^c(n-1) + \alpha\delta_j^{c+1}(n)a_k^c(n)$$
$$u_j^{c+1}(n) = u_j^{c+1}(n-1) + \alpha\delta_j^{c+1}(n)$$

donde :

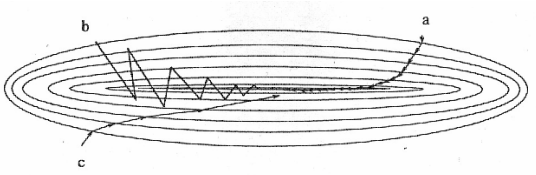
$$\delta_j^{c+1} = a_j^{c+1}(1-a_j^{c+1})\sum_{i=1}^{n_{c+2}}\delta_i^{c+2}w_{ji}^{c+1}$$

Tasa de aprendizaje y momento

- **El cambio en el peso es proporcional al gradiente del error, siendo α (tasa de aprendizaje) la constante de proporcionalidad**
  - **Si α es grande, el error puede oscilar alrededor del mínimo**
  - **Si α es pequeña, la convergencia de más lenta**
- **Se puede modificar la ley de aprendizaje añadiendo un término llamado *momento***

$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w} + \eta \Delta w(n-1)$$

- **Así, se añade cierta *inercia* a los cambios y se evitan oscilaciones**



The descent in weight space. a) for small learning rate; b) for large learning rate: note the oscillations, and c) with large learning rate and momentum term added.

Proceso de aprendizaje en el MLP

1. Se inicializan los pesos y umbrales de la red (valores aleatorios próximos a 0)
2. Se presenta un patrón *n* de entrenamiento, *(X(n),S(n)),* y se propaga hacia la salida, obteniéndose la respuesta de la red *Y(n)*
3. Se evalúa el error cuadrático, *e(n)*, cometido por la red para el patrón *n. (ver diapositiva 14)*
4. Se aplica la regla delta generalizada para modificar los pesos y umbrales de la red:
   1. *Se calculan los valores δ para todas las neuronas de la capa de salida*
   2. *Se calculan los valores δ para el resto de las neuronas de la red, empezando desde la última capa oculta y retropropagando dichos valores hacia la capa de entrada*
   3. *Se modifican pesos y umbrales*

Proceso de aprendizaje en el MLP

1. Se repiten los pasos 2, 3 y 4 para todos los patrones de entrenamiento, completando así un ciclo de aprendizaje

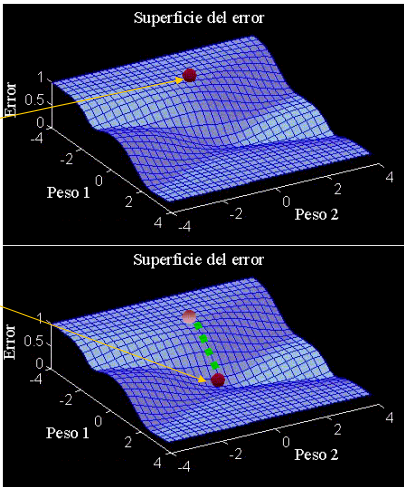
2. Se evalúa el error total E (diapositiva 2) cometido por la red. Es el error de entrenamiento. (puede medirse el error medio o el error acumulado para todos los patrones)

3. Se repiten los pasos 2, 3, 4, 5 y 6 hasta alcanzar un mínimo del error de entrenamiento, para lo cual se realizan m ciclos de aprendizaje. Puede utilizarse otro criterio de parada (pequeña variación del error de entrenamiento, aumento del error del conjunto de test, etc…)

Proceso de aprendizaje en el MLP

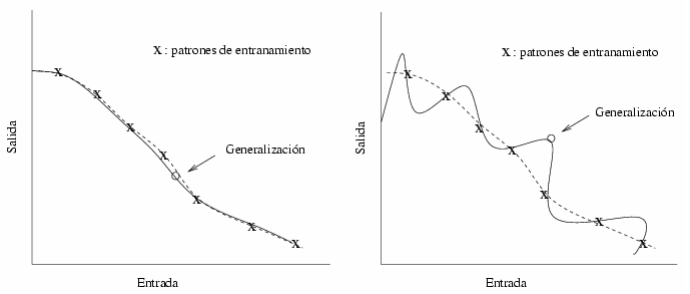
Idea intuitiva:

- Partiendo de un punto aleatorio W(0) del espacio de pesos, el proceso de aprendizaje desplaza el vector de pesos W(n-1) siguiendo la dirección negativa del gradiente del error en dicho punto, alcanzando un nuevo punto W(n) que estará más próximo del mínimo del error que el anterior



Capacidad de generalización

- No sólo es importante saber si la red se adapta a los patrones de entrenamiento sino que hay que conocer cómo se comportará ante patrones no utilizados en el entrenamiento
- Hay que evaluar la **capacidad de generalización**



Buena capacidad de generalización

Mala capacidad de generalización

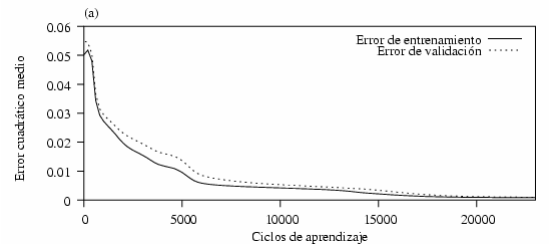
Capacidad de generalización

- Necesitaremos dos conjuntos:
  - **Conjunto de entrenamiento**: para entrenar la red (ajustar pesos y umbrales)
  - **Conjunto de validación o test**: para medir la capacidad de la red ante patrones no utilizados en el entrenamiento
  - Ambos conjuntos deben ser representativos del problema
- A veces, es necesario exigir menor ajuste a los datos de entrenamiento para obtener mejor generalización
- Para poder hacer esto, es necesario evaluar la capacidad de generalización a la vez que se realiza el entrenamiento, no sólo al final
- Cada cierto número de ciclos de entrenamiento se pasa el conjunto de validación (sin ajustar pesos) para medir la capacidad de generalización
- Así puede medirse tanto la evoluciòn del error de entrenamiento como la de validación

Capacidad de generalización

Situación 1: Ambos errores permanecen estables después de cierto número de ciclos

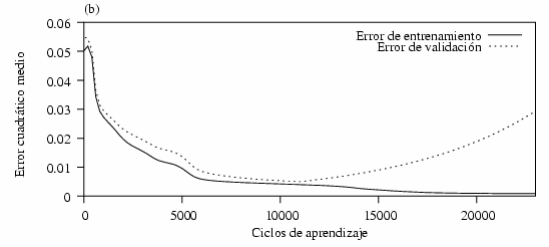
- El aprendizaje ha terminado con éxito
- Nivel de generalización bueno



Capacidad de generalización. Sobreaprendizaje

Situación 2: Después de cierto número de ciclos el error de validación empieza a aumentar

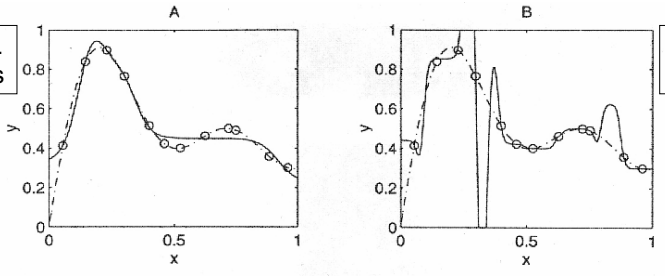
- Se ha conseguido un error de entrenamiento muy bajo, pero a costa de perder generalización
- Se ha producido **sobreaprendizaje**
- Hubiera sido conveniente detener el entrenamiento en el ciclo 10000



Sobreaprendizaje

- El sobreaprendizaje puede deberse a un número excesivo de ciclos de aprendizaje
- También puede deberse a un excesivo número de pesos o parámetros (de neuronas): se tiende a ajustar con mucha exactitud los patrones de entrenamiento, porque la función tiene muchos grados de libertad (muchos parámetros)
- Si los datos de entrenamiento tienen ruido, la función se ajusta al ruido impidiendo generalizar bien

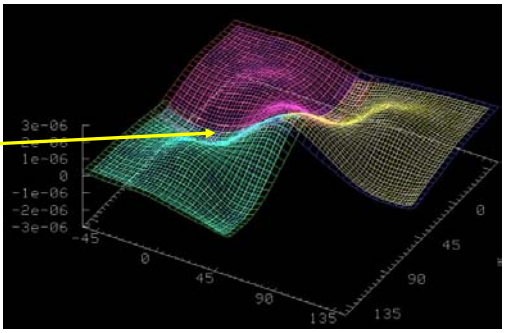
5 neur. ocultas



20 neur. ocultas

Deficiencias del algoritmo de aprendizaje

Mínimos locales: La superficie del error es compleja. El método del gradiente nos puede llevar a un mínimo local



Posibles soluciones:

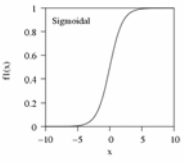
• Añadir ruido al método de descenso del gradiente

• Partir de diferentes inicializaciones aleatorias

• Aumentar el número de neuronas ocultas

Deficiencias del algoritmo de aprendizaje

- Parálisis o saturación
  - Se produce cuando la **entrada total a una neurona toma valores muy altos** (+/-): la neurona se satura y alcanza un valor próximo a 1 o a
  - El ajuste de los pesos es proporcional a $y_k(1-y_k)$: es prácticamente 0
  - La red se estanca
  - No es lo mismo que un mínimo local, aunque puede confundirse con esta situación
  - Cuando hay parálisis, puede ocurrir que después de un tiempo el error siga bajando
  - Para evitar la saturación conviene comenzar con valores de pesos próximos a 0

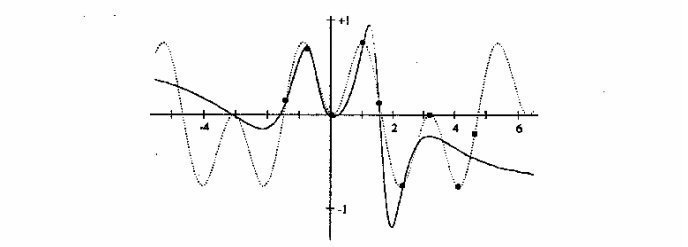


Otras funciones de activación

La sigmoide no es muy apropiada para aproximar funciones periódicas.

Ejemplo: Red con f.a. sigmoide y 8 neuronas ocultas para aproximar la función

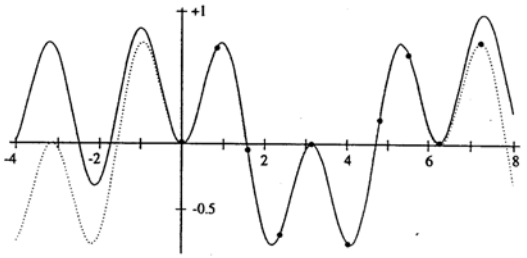
$$f(x) = \sin(2x)\sin(x)$$



The periodic function $f(x) = \sin(2x)\sin(x)$ approximated with sigmoid activation functions.
(Adapted from (Dastani, 1991).)

Otras funciones de activación

La aproximación es mucho mejor con una función de activación sin(x). Con sólo 4 neuronas ocultas se aproxima mejor

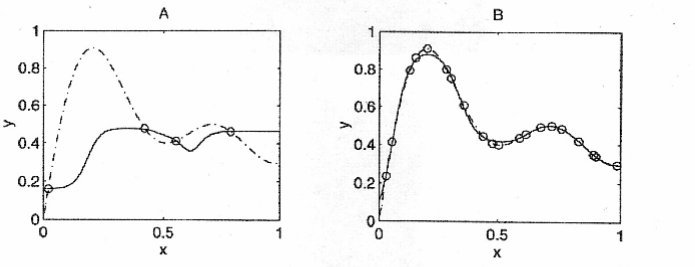


The periodic function $f(x) = \sin(2x)\sin(x)$ approximated with sine activation functions. (Adapted from (Dastani, 1991).)

Efecto del tamaño del conjunto de entrenamiento

El conjunto de entrenamiento debe tener un número suficiente de muestras para representar adecuadamente el problema

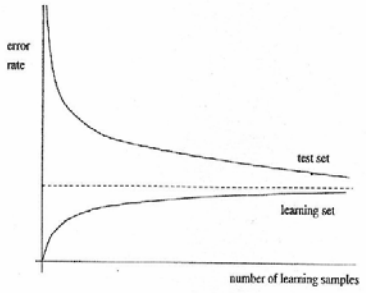
Ejemplo: Aproximación de una función con 4 muestras y con 20:



Effect of the learning set size on the generalization. The dashed line gives the desired function, the learning samples are depicted as circles and the approximation by the network is shown by the drawn line. 5 hidden units are used. a) 4 learning samples. b) 20 learning samples.

Efecto del tamaño del conjunto de entrenamiento

Cuando el número de ejemplos de entrenamiento crece, los errores de test y de entrenamiento convergen



Effect of the learning set size on the error rate. The average error rate and the average test error rate as a function of the number of learning samples.