uc3m | Universidad Carlos III de Madrid

OPENCOURSEWARE
REDES DE NEURONAS ARTIFICIALES
Inés M. Galván – José M. Valls

Tema 3

Perceptron Multicapa

Perceptron Multicapa

- Introducción
- Arquitectura
- Aprendizaje: Algoritmo de Retroprogación
- Proceso de Aprendizaje
- Algunos problemas con el uso del PM
- Clasificación y regresión no lineal

Introducción

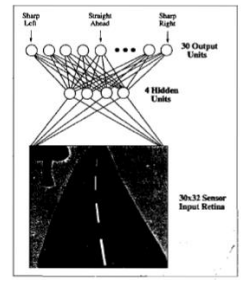
- Minsky y Papert (1969) demostraron que el perceptron simple (y Adaline) no pueden resolver problemas no lineales (como XOR).
- La combinación de varios perceptrones podría resolver ciertos problemas no lineales pero no existía un mecanismo automático para adaptar los pesos de las capas ocultas
- Rumelhart y otros autores (Rumelhart et al, 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

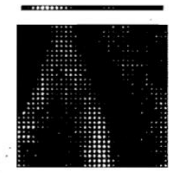
Introducción

- Se demuestra que el Perceptron Multicapa (MLP) es un Aproximador Universal (Cybenko, 1989), en el sentido que puede aproximar cualquier función continua.
- Un MLP puede aproximar relaciones no lineales entre variables 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
- El uso del PM es adecuado cuando:
  - Se permiten largos tiempos de entrenamiento
  - Se necesitan respuestas muy rápidas ante nuevas instancias
  - No es necesario entender o interpretar lo que ha aprendido el MLP
- Se ha aplicado con éxito en multiples aplicaciones

Ejemplo de aplicación: ALVINN

- Propuesto por (Pommerleau, 1991)
- Conducir automáticamente un vehículo a velocidad normal en autopista.
- Input: 30*32 pixels obtenidos por una cámara (960 inputs). Foto de la carretera
- Output: dirección que debe tomar el vehículo. Cada neurona de salida corresponde a una determinada dirección
- Entrenamiento: comportamiento de un conductor humano durante 5 minutos.
- ALVINN ha conducido con éxito a 70 mph, distancias de 90 millas en autopistas públicas americanas.

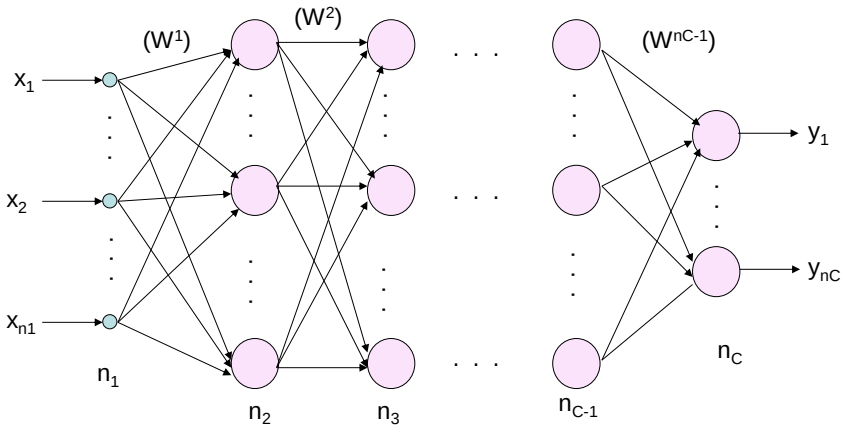




Arquitectura Perceptron Multicapa

- 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": conexiones hacia adelante
- Generalmente cada neurona está conectada a todas las neuronas de la siguiente capa (conectividad total)

Arquitectura Perceptron Multicapa



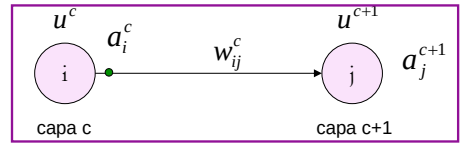
Propagación de los patrones de entrada. Notación

Considérese un PM con C capas y n_c neuronas en la capa c=1, 2, …C

– **W^c** es la matriz de pesos de la capa c a la capa c+1 y **U^c** es el vector de umbrales (bias) de la capa c

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

– a^c_i es la activación de la neurona I de la capa c



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 las neuronas de la capa c-1; w_{ji}^{c-1} es el peso de la connexion de la neurona j de la capa c-1 a la neurona I de la capa c; u_i^c es el umbral o bias de la neurona i de la capa c; *f* es la función de activación

Propagación de los patrones de entrada. Activaciones

- Activación de las neuronas de la capa de salida

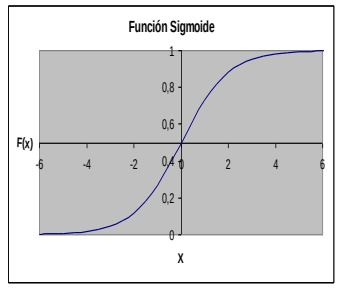
$$y_i = a_i^C = f\left(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C \right) \text{ para } i = 1, 2, ..., n_C$$

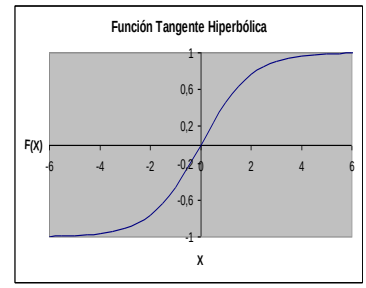
donde $Y = (y_1, y_2, ..., y_{n_C})$ es el vector de salida de la red

Función de activación

- Las funciones de activación más utilizadas son
  - **Función Sigmoidal** **- Tangente Hiperbólica**

$$f_1(x) = \frac{1}{1+e^{-x}} \qquad f_2(x) = \frac{1-e^{-x}}{1+e^{-x}}$$





Función de activación

- Ambas son crecientes con dos niveles de saturación
- Ambas funciones están relacionadas: $f_2(x) = 2 f_1(x) - 1$
- Normalmente la función de activación es común a todas las neuronas (función sigmoidal o tangent hiperbólica), salvo para las neuronas de salida, que en ocasiones se utiliza la función lineal f(x)=x.

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.
- **Por tanto, 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 $\mathbf{R}^{n_O}$**

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

siendo X las variables de entrada y W el conjunto de pesos y umbrales de la red.

Diseño de la arquitectura

Hay que decidir:

- Función de activación
  - Se basa 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 (Algoritmos de Selección de atributos)
- 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
  - Generalmente se prueban redes con diferente número de neuronas ocultas y se elige la mejor utilizando un conjunto de validación. Se podría realizar una búsqueda exhaustiva, pero generalmente no es necesario

Aprendizaje

El aprendizaje del Perceptron Multicapa es un proceso iterativo **supervisado:** modificación paulatina de los pesos y umbrales de la red hasta que la salida de la red sea lo más próxima posible a la salida deseada o esperada para cada patrón de entrenamiento

Dado

Conjunto de patrones o ejemplos

Vector de entrada: $x(n)=(x_1, x_2, ..., x_{n1})$

Vector de salida deseada: $s(n)$

Encontrar

Pesos W y umbrales U tales que

$s(n) \approx y(n) \quad \forall$ patrón n $\quad \Leftrightarrow |s(n)-y(n)| \approx 0 \ \forall n$

$\Leftrightarrow$ Minimizar $E=\Sigma |s(n)-y(n)|$

Aprendizaje

El aprendizaje es entonces equivalente a minimizar la función error E dada por:

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

N es el número de 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

El algoritmo de retropropagación utiliza el método del gradiente para encontrar un mínimo de la función

Aprendizaje

• El ajuste de los pesos se hace casi siempre por patrones: sucesiva minimización de los errores para cada patrón e(n), en lugar de minimizar el error global, utilizando método del gradiente

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

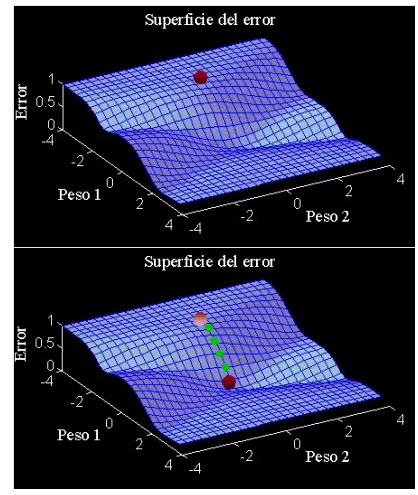
$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w}$$

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

Regla Delta Generalizada o **Algoritmo de Retropropagación**

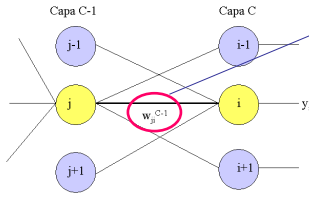
Aprendizaje

- Adaptación de los pesos siguiendo una dirección de búsqueda
  - Dirección negativa del gradiente de la función de error E (algoritmo de retropropagación)
  - Hay otros métodos para realizar la búsqueda: aleatoria, técnicas evolutivas, etc



Regla Delta Generalizada

•Pesos de la última capa



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

Sólo la derivada de la salida y_i es distinta de 0

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

La expresión del error es: $e(n) = \frac{1}{2}\sum_{i=1}^{n_C}(s_i(n) - y_i(n))^2 = \frac{1}{2}(s_1(n) - y_1(n))^2 + \ldots + \frac{1}{2}(s_i(n) - y_i(n))^2 + \ldots$

Dado que el peso w_{ji}^{C-1} sólo afecta a y_i, el resto de salidas no dependen de él, por tanto:

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

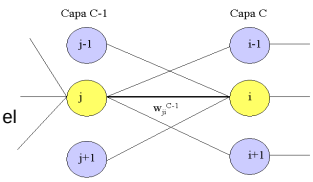
Por tanto, $\frac{\partial e(n)}{\partial w_{ji}^{C-1}} = -(s_i(n) - y_i(n)) \frac{\partial y_i(n)}{\partial w_{ji}^{C-1}}$ [2]

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:

$$y_i = a_i^C = f(\overbrace{\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C}^{[3]})$$



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

Por tanto, su derivada:

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

Reemplazando esta expresión en [2], se obtiene que:

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

δ_i

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

$$\delta_i^C(n) = (s_i(n) - y_i(n)) f'(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C) \quad [4]$$

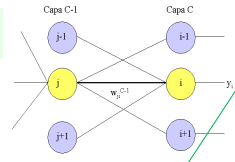
Por tanto [1] quedará: $\frac{\partial e(n)}{\partial w_{ji}^{C-1}} = -\delta_i^C(n) a_j^{C-1}$

Regla delta generalizada (pesos de la última capa)

Por tanto, el nuevo peso será $w_{ji}^{C-1}(n) = w_{ji}^{C-1}(n-1) + \alpha\delta_i^C(n)a_j^{C-1}(n)$

Teniendo en cuenta que el umbral (o bias) se considera como un peso más cuya entrada es siempre 1, la regla para modificar el umbral viene dada por:

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

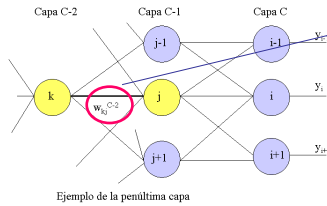


Para modificar el peso w^{C-1}_{ji} basta considerar:
- la activación de la neurona origen j
- y el término δ_i asociado a la neurona destino i.

El término δi contiene el error medido en la neurona i de salida y la derivada de la activación de dicha neurona

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.

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

Teniendo en cuenta la expresión del error y que todas las salidas depende del peso, se obtiene:

$$\frac{\partial e(n)}{\partial w_{kj}^{C-2}} = -\sum_{i=1}^{n_C}(s_i(n) - y_i(n))\frac{\partial y_i(n)}{\partial w_{kj}^{C-2}} \quad [5]$$

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

Ahora hay que calcular la derivada de y_i respecto al peso w^{C-2}_{kj}

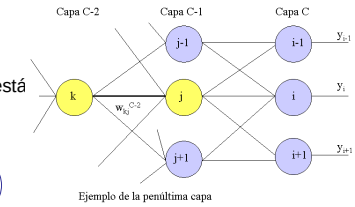
Regla delta generalizada (pesos de la penúltima capa)

Como y_i es:

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

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

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



Sustituyendo esta expresión en [5]:

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

δ_i

Regla delta generalizada (pesos de la penúltima capa)

Por tanto y de acuerdo a la definición δ^C_i de [4], queda:

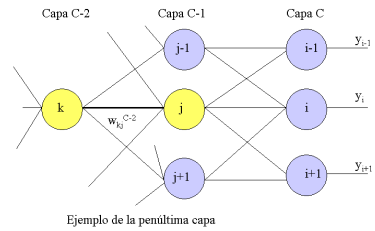
$$\frac{\partial e(n)}{\partial w_{kj}^{C-2}} = -\sum_{i=1}^{n_C} \delta_i^C w_{ji}^{C-1} \frac{\partial a_j^{C-1}}{\partial w_{kj}^{C-2}} \quad [6]$$

La derivada de la activación de la neurona j es:

$$\frac{\partial a_j^{C-1}}{\partial w_{kj}^{C-2}} = f'(\sum_{k=1}^{n_{C-2}} w_{kj}^{C-2} a_k^{C-2} + u_j^{C-1}) a_k^{C-2} \quad [7]$$

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

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

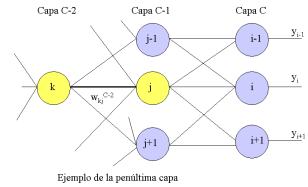


δ^{C-1}_j

Regla delta generalizada (pesos de la penúltima capa)

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

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



Ejemplo de la penúltima capa

Sustituyendo [7] en [6] y de acuerdo con la definición de δ asociado a la neurona j de la capa C-1, se obtiene:

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

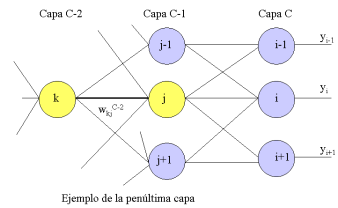
Regla delta generalizada (pesos de la penúltima capa)

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^{C-2}_{kj} basta considerar
- la activación de la neurona origen k
- y el término δ asociado a la neurona destino j

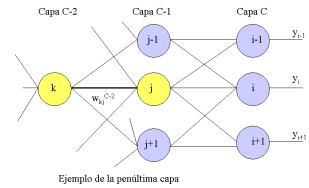
Regla delta generalizada (pesos de la penúltima capa)

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

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



Regla delta generalizada. Capa oculta c

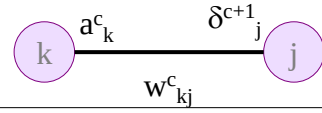
Las expresiones obtenidas se pueden generalizar para los pesos de cualquier capa c a la capa c+1 (c=1,2,.... C-2)

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

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

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

Basta tener en cuenta la activación de la neurona 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 ponderados por los pesos correspondientes

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

Regla delta generalizada. Derivada de f

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

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

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

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

Como δ de la última capa es: $\delta_i^C = (s_i - y_i) f'(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C)$

y f'( )=f( )(1-f( )) resulta:

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

Última capa

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

Regla delta generalizada. Derivada de f

Para el resto de las capas:

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

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. 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 :

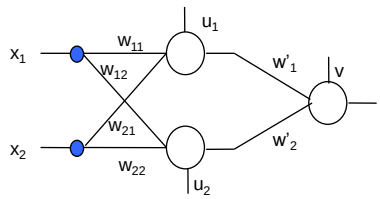
$$\delta_j^{c+1} = a_j^{c+1}(1 - a_j^{c+1}) \sum_{i=1}^{n_{c+2}} \delta_i^{c+2} w_{ji}^{c+1}$$

Resumen de la regla delta generalizada

- Para modificar un peso basta tener en cuenta activación de la neurona de la que parte la conexión y el término δ de la neurona a la que llega la conexión
- El término δ de una neurona de salida se calcula utilizando la derivada de la función de activación de dicha neurona y el error medido en dicha neurona de salida
- El término δ de cualquier otra neurona de la red se calcula utilizando la derivada de la función de activación de dicha neurona y la suma (ponderada por los pesos correspondiente) de los términos δ de las neuronas de la siguiente capa
- Por tanto, 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 del peso 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

Regla delta generalizada: Ejemplo

Considérese la siguiente arquitectura



Las activaciones de las neuronas son (1):

(1) Nótese que se ha cambiado la notación para las activaciones de las neuronas de la red

$$b_1 = f(w_{11} \cdot x_1 + w_{21} \cdot x_2 + u_1)$$

$$b_2 = f(w_{12} \cdot x_1 + w_{22} \cdot x_2 + u_2)$$

$$y = f(w'_1 \cdot b_1 + w'_2 \cdot b_2 + v)$$

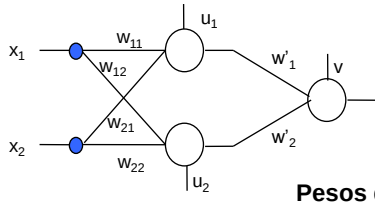
➡ **Pesos de la capa oculta a la de salida: w'$_1$, w'$_2$**

$$w'_1{}^{nuevo} = w'_1{}^{anterior} + \alpha \cdot \delta^3(n) \cdot b_1 \qquad w'_2{}^{nuevo} = w'_2{}^{anterior} + \alpha \cdot \delta^3(n) \cdot b_2$$

$$v^{nuevo} = v^{anterior} + \alpha \cdot \delta^3(n)$$

$$\delta^3(n) = (s(n) - y) \cdot y \cdot (1 - y)$$

Regla delta generalizada: Ejemplo



Pesos de la capa de entrada a la oculta: w_{ij}

$$w_{11}^{nuevo} = w_{11}^{anterior} + \alpha \cdot \delta_1^2(n) \cdot x_1 \qquad w_{12}^{nuevo} = w_{12}^{anterior} + \alpha \cdot \delta_2^2(n) \cdot x_1$$

$$w_{21}^{nuevo} = w_{21}^{anterior} + \alpha \cdot \delta_1^2(n) \cdot x_2 \qquad w_{22}^{nuevo} = w_{22}^{anterior} + \alpha \cdot \delta_2^2(n) \cdot x_2$$

$$u_1^{nuevo} = u_1^{anterior} + \alpha \cdot \delta_1^2(n) \qquad u_2^{nuevo} = u_2^{anterior} + \alpha \cdot \delta_2^2(n)$$

$$\delta_1^2(n) = \delta^3(n) \cdot b_1 \cdot (1 - b_1) \cdot w'_1 \qquad \delta_2^2(x) = \delta^3(n) \cdot b_2 \cdot (1 - b_2) \cdot w'_2$$

Proceso de aprendizaje en el MLP

1. Se inicializan los pesos y umbrales (bias) 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 16).*
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*

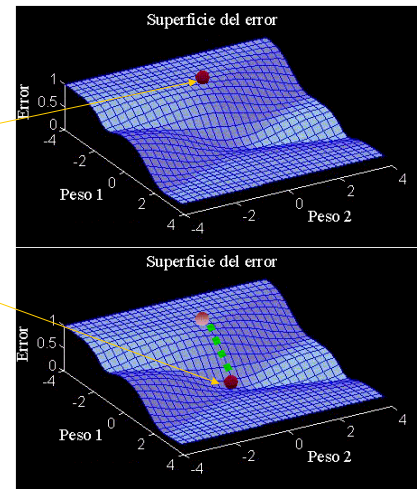
Proceso de aprendizaje en el MLP

5. Se repiten los pasos 2, 3 y 4 para todos los patrones de entrenamiento, completando así un ciclo de aprendizaje
6. Se evalúa el error total E (diapositiva 16: error cuadrático medio) cometido por la red. Es el error de entrenamiento. También se puede evaluar el error medio
7. Se presentan los patrones de test (o validación), calculando únicamente la salida de la red (no se modifican los pesos) y se evalúa el error total en el conjunto de test (o validación).
8. Se repiten los pasos 2, 3, 4, 5, 6 y 7 hasta alcanzar un mínimo del error de entrenamiento, para lo cual se realizan m ciclos de aprendizaje. Puede utilizarse otro criterio de parada:
    1. El error de entrenamiento permanece estable
    2. El error de validación permanece estable
    3. El error de validación empieza a aumentar

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



Razón de aprendizaje y Momento

- El cambio en el peso es proporcional al gradiente del error, siendo α (razón de aprendizaje: valor entre 0 y 1) la constante de proporcionalidad
  - Si α es grande, los desplazamientos son más bruscos y el error puede oscilar alrededor del mínimo. La convergencia generalmente es más rápida, aunque con el riesgo de saltar un buen mínimo
  - Si α es pequeña, los desplazamientos son más suaves, implicando una convergencia más lenta del algoritmo.
  - Es un parámetro del aprendizaje que se determinará experimentalmente.

- Se puede modificar la ley de aprendizaje añadiendo un término llamado *momento* (Phansalkar and Sastry, 1994)

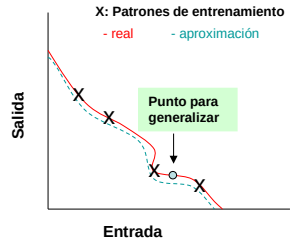
$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w} + \eta \Delta w(n-1)$$

donde Δw(n-1)=w(n-1)-w(n-2)

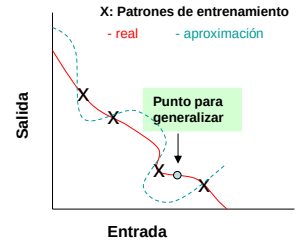
Esta nueva ley conserva las propiedades de la ley anterior, pero incorpora cierta inercia que puede evitar oscilaciones en el error

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** con un conjunto de datos diferente al utilizado para el entrenamiento



PM con buena capacidad de generalización



PM con mala capacidad de generalización

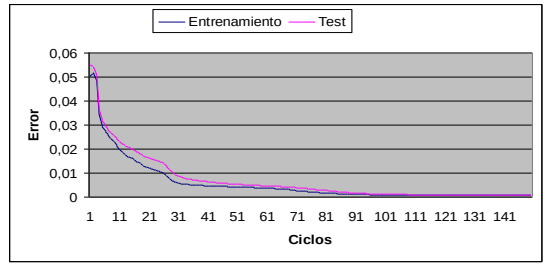
Capacidad de generalización

- 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 del proceso de entrenamiento
- Cada cierto número de ciclos de entrenamiento se pasa el conjunto de test o (validación) (sin ajustar pesos) para medir la capacidad de generalización
- Así puede medirse tanto la evolución del error de entrenamiento como la de test (o validación)

Capacidad de generalización

Situación 1: Ambos errores (entrenamiento y test (o validación)) permanecen estables después de cierto número de ciclos

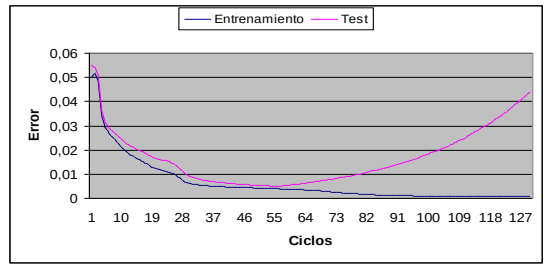
- El aprendizaje ha terminado con éxito
- Nivel de generalización bueno



Capacidad de generalización. Sobreaprendizaje

Situación 2: Después de cierto número de ciclos el error de test (o 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 55 aprox.

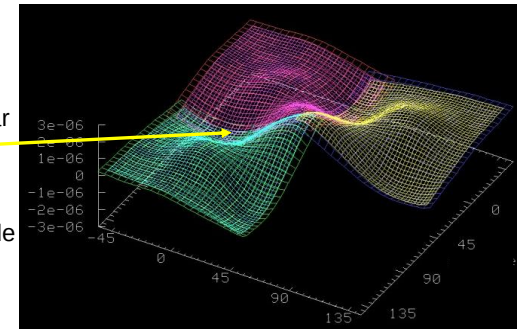


Capacidad de generalización

- El sobreaprendizaje en el Perceptron Muticapa puede deberse a:
  - Un número excesivo de ciclos de aprendizaje: Los pesos se especializan en los datos de entrenamiento consiguiendo un error cercano a cero para estos datos
  - Un excesivo número de pesos o neuronas ocultas: se tiende a ajustar con mucha exactitud los patrones de entrenamiento, porque la función tiene muchos grados de libertad (muchos parámetros):
- Complejidad del modelo alta en relación con el número de patrones de entrenamiento
- Si se disponen de pocos patrones de entrenamiento, la función podría ajustarse a estos pocos patrones y perder capacidad de generalización.

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

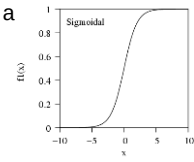
• Partir de diferentes inicializaciones aleatorias

• Aumentar el número de neuronas ocultas

Atención: En ocasiones un mínimo local puede ser una buena solución a un problema y por tanto no ser necesario salir de él.

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

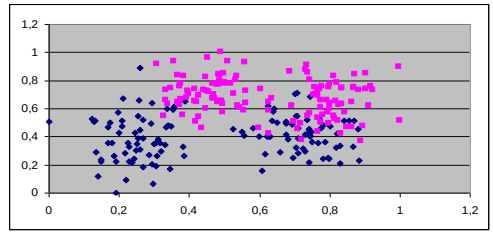


  - Cuando hay parálisis, puede ocurrir que después de un tiempo el error siga bajando
  - Para evitar la saturación conviene comenzar con valores de pesos próximos a 0
  - Es convenietne también normalizar las variables de entrada y salida deseada de la red

Clasificación no Lineal

El problema de clasificación consiste en establecer una correspondencia entre un conjunto de elementos dados y un conjunto clases.

Clasificación no lineal: cuando la separación entre las clases es no lineal, es decir no se puede realizar mediante hiperplanos



Dado un conjunto de ejemplos, no siempre es posible conocer cuántas clases tienen que ser consideradas.
Para poder afrontar un problema de clasificación con redes de neuronas supervisadas es condición necesaria y suficiente conocer el número de clases
Si el número de clases es desconocido es necesario utilizar técnicas no supervisadas

Clasificación no Lineal

Sean $X_i=(X_{i1},...,X_{in})$ i=1,...K, patrones de entrada y C_1, C_2,..., C_m, m clases diferentes

- La red tendrá n neuronas de entrada que reciben los patrones $X_i=(X_{i1},...,X_{in})$
- **Dos opciones** para el número de neuronas de salida
  - m neuronas de salida que representan las m clases C_1, C_2,..., C_m. En este caso, la salida deseada para cada patrón de entrada X_i es una m-tupla $(a_i,....,a_m)$ donde

    a_j=1 si X_i pertenece a la clase C_j

    a_k=0 para todo k distinto de j
  - 1 salida de la red. En este caso, la salida deseada será un número real en el intervalo [0,1]. Dicho intervalo se divide en m trozos. Es decir:

    2 clases: C1=0 y C2=1

    3 clases: C1=0, C2=0.5 y C3=1

    4 clases: C1=0, C2=0.4, C3=0.7 y C4=1

Para muchas clases es aconsejable utilizar tantas salidas como clases

Clasificación no Lineal

Interpretación de las salidas de la red

La(s) salida(s) de la red es (son) número(s) reales. Para decidir qué clase representa es necesario introducir un criterio:

- **Red con m salidas**: Se calcula el índice para el cual se obtiene el máximo de (y_1, y_2, …y_m). Ese índice será la clase
- **Red con 1 salida**: Se estable un umbral para decidir la clase. Por ejemplo:

  2 clases: si y <= 0.5, entonces C1 y si y>0.5 entonces C2

  3 clases: si y <=0.4 entonces C1, si 0.4<y<=0.7 entonces C2 y si y>=0.7 entonces C3

Codificación de los patrones de entrada

Los ejemplos a clasificar pueden ser de cualquier naturaleza, imágenes, juegos, letras, etc. y tienen que ser codificados en un vector de R^n. La codificación de los patrones juega un papel muy importante en el problema de clasificación hasta el punto que una codificación no adecuada puede producir resultados no satisfactorios.

Regresión no Lineal

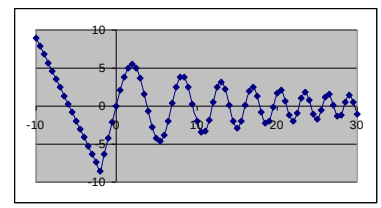
Dado

$(x^1=(x^1_1, ..., x^1_n), y^1)$

$(x^2=(x^2_1, ..., x^2_n), y^2)$

…

$(x^m=(x^m_1, ..., x^m_n), y^m)$



Encontrar

Una función no lineal F(x) tal que $F(x^p) \approx y^p$

La relación entre la entrada y salida es no lineal: Red de neuronas no lineales (neuronas ocultas) y supervisadas: Percpetron Multicapa, Redes de Base Radial

Entradas de la red: variables de entrada de la función
Salidas de la red: variables de salida de la función
La función no lineal F viene dada por la red (neuronas ocultas y pesos)

Referencias

(Minsky, and Papert, 1969): Minsky, M., & Papert, S. Perceptrons, 1969

(Rumelhart et al, 1986): Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. "Learning representations by back-propagating errors". Nature. **323** (6088): 533–536, 1986

(Cybenko 1989): Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, *2*(4), 303-314, 1989

(Pommerleau , 1991): D. A. Pommerleau. Efficient Training of Artificial Neural Networks for Autonomous Navigation, Neural Computation 3, 1991

(Phansalkar and Sastry, 1994): Phansalkar, V. V., & Sastry, P. S. Analysis of the back-propagation algorithm with momentum. *IEEE Transactions on Neural Networks*, *5*(3), 505-506, 1994