Introducción a los modelos computacionales

Práctica 1. Implementación del perceptrón multicapa

Pedro Antonio Gutiérrez
pagutierrez@uco.es

Asignatura "Introducción a los modelos computacionales"
4º Curso Grado en Ingeniería Informática
Especialidad Computación
Escuela Politécnica Superior
(Universidad de Córdoba)

26 de septiembre de 2016

Objetivos de la práctica

- Familiarizar al alumno con los modelos computacionales de redes neuronales, en concreto, con el perceptrón multicapa.
- Implementar el algoritmo de retropropagación básico para el perceptrón multicapa.
- Comprobar el efecto de distintos parámetros:
  - Arquitectura de la red.
  - Factor de momento.
  - Utilización o no de sesgo.
  - etc.

El cerebro como un dispositivo computacional

- Motivación: Los algoritmos de aprendizaje desarrollados durante siglos no pueden abordar la complejidad de los problemas reales.
- Sin embargo, el cerebro humano es la más sofisticada computadora para resolver problemas extremadamente complejos.
  - Tamaño razonable: 10^{11} neuronas (células neuronales), donde solo una pequeña porción son utilizadas.
  - Nodos de procesamiento simples: las células no contienen demasiada información.
  - Masivamente paralelo: cada región del cerebro realiza tareas específicas.
  - Tolerante a fallos y robusto: la información se salva, principalmente, en las conexiones entre neuronas, que pueden llegar a regenerarse.

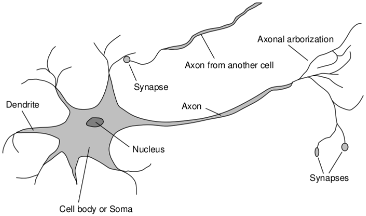
Comparando el cerebro contra un ordenador

	Ordenador	Cerebro humano
Unidades de computación	1 CPU, 10^5 puertas	10^{11} neuronas
Unidades de almacenamiento	10^9 bits RAM	10^{14} sinapsis
Tiempo por ciclo	10^{-8} segundos	10^{-3} segundos
Ancho de banda	10^{22} bits / segundo	10^{28} bits / segundo

Incluso siendo un millón de veces más rápido el ordenador que el cerebro en velocidad de computación, un cerebro termina siendo un millón de veces más rápido que el ordenador (gracias a la gran cantidad de elementos de computación).

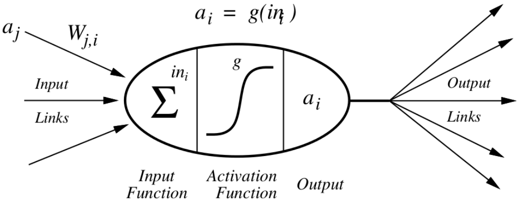
- Reconocer una cara:
  - Cerebro $< 1s$.
  - Computadora: billones de ciclos.

Neurona biológica



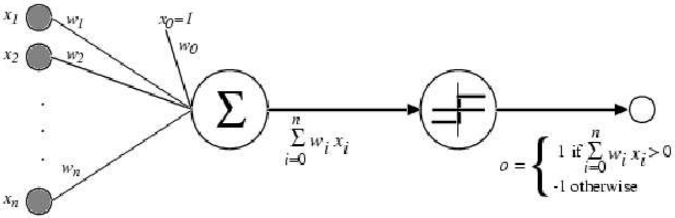
- Una neurona no hace nada hasta que la influencia de sus entradas alcanza un determinado nivel.
- La neurona produce una salida en la forma de pulso que parte del núcleo, baja por el axón y termina en sus ramificaciones.
- O se dispara o no hace nada (dispositivo "todo-o-nada").
- La salida causa la excitación o inhibición de otras neuronas.

Neurona artificial



- Las neuronas artificiales son nodos conectados a otros nodos mediante enlaces.
- A cada enlace se le asocia un peso.
- El peso determina la naturaleza (excitatoria $+$ o inhibitoria $-$) y la fuerza (valor absoluto) de la influencia entre nodos.
- Si la influencia de todos los los enlace de entrada es suficientemente alta, el nodo se activa.

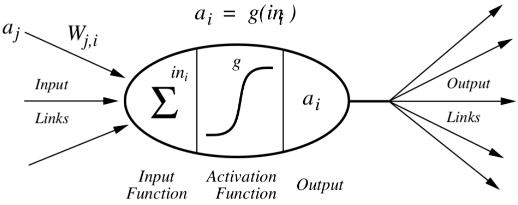
Neurona artificial: un posible ejemplo



$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Neurona artificial



- Cada nodo i tiene varias conexiones de entrada y de salida, cada una con sus pesos.
- La salida es una función de la suma ponderada de las entradas.

Neurona artificial

- Cada enlace de entrada a la neurona i le proporciona un valor de activación a_j que proviene de otra neurona.
- A menudo, la función de entrada es la suma ponderada de dichos valores de activación:

$$in_i(a_1, \ldots, a_{n_i}) = \sum_{j=1}^{n_i} W_{j,i}\, a_j$$

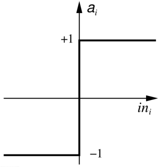
- La salida de la neurona es el resultado de aplicar la función de activación sobre la función de entrada.

$$out_i = g_i(in_i) = g_i\left(\sum_{j=1}^{n_i} W_{j,i}\, a_j\right)$$

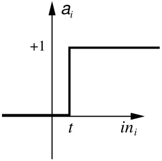
Neurona artificial

- Algunas funciones de activación:

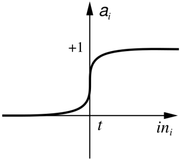
$$sign(x) = \begin{cases} +1, & \text{si } x \geq 0 \\ -1, & \text{si } x < 0 \end{cases}$$



$$step_t(x) = \frac{sign(x-t)+1}{2} = \begin{cases} 1, & \text{si } x \geq t \\ 0, & \text{si } x < t \end{cases}$$

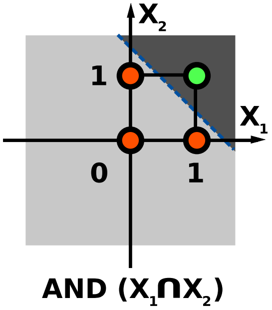


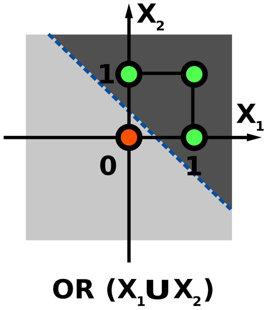
$$\sigma(x) = \frac{1}{1 + e^{-(x-t)}}$$



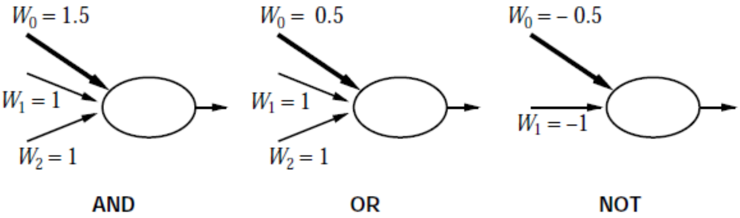
Neurona artificial

- Simular mediante una neurona funciones lógicas simples.
- Para el And y el Or nos basta una neurona con la función *step* y los pesos correctamente seleccionados:



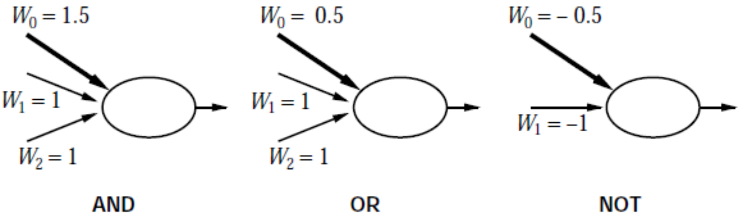


Neurona artificial



$$and(0,0) = step_{1,5}(1 \cdot 0 + 1 \cdot 0) = step_{1,5}(0) = 0$$
$$and(0,1) = step_{1,5}(1 \cdot 0 + 1 \cdot 1) = step_{1,5}(1) = 0$$
$$and(1,0) = step_{1,5}(1 \cdot 1 + 1 \cdot 0) = step_{1,5}(1) = 0$$
$$and(1,1) = step_{1,5}(1 \cdot 1 + 1 \cdot 1) = step_{1,5}(2) = 1$$

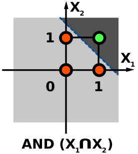
Neurona artificial

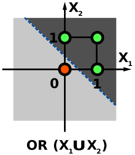


$$or(0,0) = step_{0,5}(1 \cdot 0 + 1 \cdot 0) = step_{0,5}(0) = 0$$
$$or(0,1) = step_{0,5}(1 \cdot 0 + 1 \cdot 1) = step_{0,5}(1) = 1$$
$$or(1,0) = step_{0,5}(1 \cdot 1 + 1 \cdot 0) = step_{0,5}(1) = 1$$
$$or(1,1) = step_{0,5}(1 \cdot 1 + 1 \cdot 1) = step_{0,5}(2) = 1$$

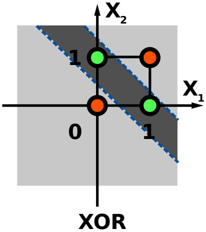
Neurona artificial

- Estos problemas se pueden resolver con una neurona porque son linealmente separables:



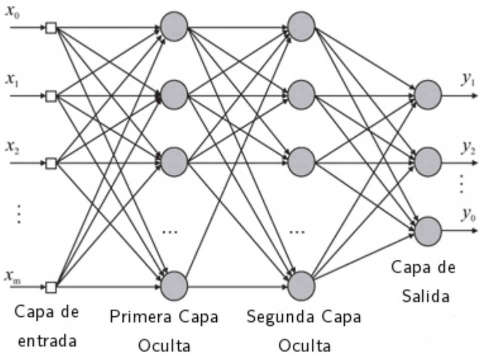


- Problema: una sola neurona no es capaz de resolver problemas más complejos, p.ej. el XOR.



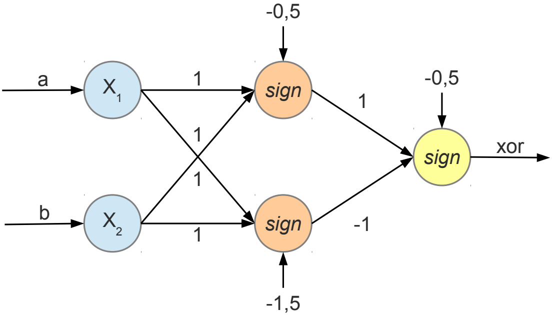
Redes neuronales artificiales

- Una Red Neuronal Artificial (RNA) es un grafo de nodos (o neuronas) conectados por enlaces.
- Las neuronas se organizan por capas, de manera que las salidas de las neuronas de una capa sirven como entradas para las neuronas de la siguiente capa.

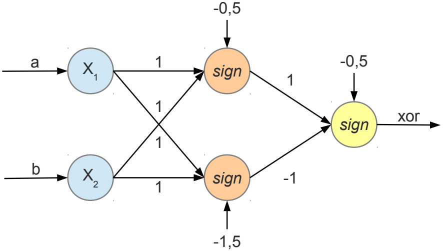


Redes neuronales artificiales

- El problema XOR se puede resolver con una RNA de una capa oculta y dos neuronas:

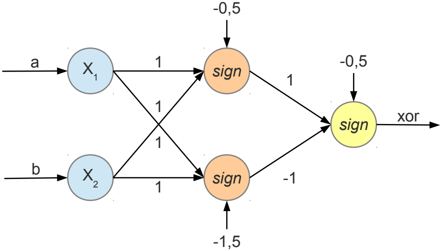


Redes neuronales artificiales



$$xor(0,0) = step_{0,5}(step_{0,5}(1 \cdot 0 + 1 \cdot 0) - step_{1,5}(1 \cdot 0 + 1 \cdot 0))$$
$$= step_{0,5}(step_{0,5}(0) - step_{1,5}(0)) = step_{0,5}(0 - 0) = step_{0,5}(0) = 0$$
$$xor(0,1) = step_{0,5}(step_{0,5}(1 \cdot 0 + 1 \cdot 1) - step_{1,5}(1 \cdot 0 + 1 \cdot 1))$$
$$= step_{0,5}(step_{0,5}(1) - step_{1,5}(1)) = step_{0,5}(1 - 0) = step_{0,5}(1) = 1$$

Redes neuronales artificiales

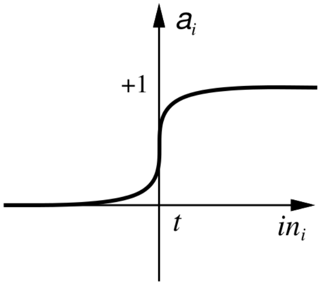


$$xor(1,0) = step_{0,5}(step_{0,5}(1\cdot 1 + 1\cdot 0) - step_{1,5}(1\cdot 1 + 1\cdot 0))$$
$$= step_{0,5}(step_{0,5}(1) - step_{1,5}(1)) = step_{0,5}(1-0) = step_{0,5}(1) = 1$$
$$xor(1,1) = step_{0,5}(step_{0,5}(1\cdot 1 + 1\cdot 1) - step_{1,5}(1\cdot 1 + 1\cdot 1))$$
$$= step_{0,5}(step_{0,5}(2) - step_{1,5}(2)) = step_{0,5}(1-1) = step_{0,5}(0) = 0$$

Redes neuronales artificiales

- La capa oculta pasa los datos a un espacio donde la tarea a realizar es más fácil.
- Si queremos optimizar los pesos de la red, la función *step* no es adecuada, ¿por qué?.
- Utilizamos una aproximación, la función sigmoide (con sesgo):

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



Redes neuronales artificiales

- Propiedad muy importante de la derivada de $\sigma(x)$:

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

$$\begin{aligned}
\sigma'(x) &= (-1)\cdot\left(1+e^{-x}\right)^{-2}\left(1+e^{-x}\right)' = \\
&= (-1)\cdot\left(1+e^{-x}\right)^{-2}\cdot\left(0+\left(e^{-x}\right)'\right) \\
&= (-1)\cdot\left(1+e^{-x}\right)^{-2}\cdot\left(e^{-x}\right)\cdot(-x)' \\
&= (-1)\cdot\left(1+e^{-x}\right)^{-2}\cdot\left(e^{-x}\right)\cdot(-1) = \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1}{(1+e^{-x})}\frac{e^{-x}}{(1+e^{-x})} = \sigma(x)\frac{1+e^{-x}-1}{(1+e^{-x})} \\
&= \sigma(x)\left(\frac{(1+e^{-x})}{(1+e^{-x})} - \frac{1}{(1+e^{-x})}\right) = \sigma(x)\cdot(1-\sigma(x))
\end{aligned}$$

Algoritmo de retropropagación

- Red neuronal *feedforward*: propagación hacia delante para obtener las salidas.
- Algoritmo *backpropagation*: propagación hacia detrás para obtener el error, las derivadas y ajustar los pesos.
- Dado un conjunto de entrenamiento, queremos ajustar los pesos de las conexiones para que el error cometido en dicho conjunto sea mínimo.
- La idea general es minimizar el error de entrenamiento y el procedimiento matemático está basado en derivar dicha función.

Algoritmo de retropropagación

- Notación:
  - Patrones de entrenamiento:
    - Vector de entradas: $\mathbf{x} = (x_1, \ldots, x_n)$.
    - Vector de salidas deseadas: $\mathbf{d} = (d_1, \ldots, d_k)$.
  - Arquitectura de la red: $\{n_0 : n_1 : \ldots : n_H\}$
    - n_h es el número de neuronas de la capa h.
    - $n_0 = n$, $n_H = k$.
    - $H - 1$ capas ocultas.
  - Pesos de la red. Para cada capa h, sin contar la capa de entrada:
    - Matriz con un vector de pesos por cada neurona: $\mathbf{W}^h = (\mathbf{w}_1^h, \ldots, \mathbf{w}_{n_h}^h)$.
    - Vector de pesos de la neurona j de la capa h (incluye sesgo): $\mathbf{w}_j^h = (w_{j0}^h, w_{j1}^h, \ldots, w_{jn_{(h-1)}}^h)$.
  - Salida de la red: $\mathbf{o} = (o_1, \ldots, o_k)$.

Algoritmo de retropropagación

Todas las neuronas, salvo las de la capa de entrada, serán de tipo sigmoide.

- Si las neuronas disponen de sesgo, su fórmula será:

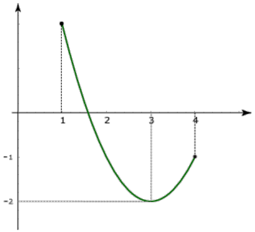
$$out_j^h = \frac{1}{1 + \exp(-w_{j0}^h - \sum_{i=1}^{n_{h-1}} w_{ji}^h out_i^{h-1})}$$

- Si no disponen de sesgo:

$$out_j^h = \frac{1}{1 + \exp(-\sum_{i=1}^{n_{h-1}} w_{ji}^h out_i^{h-1})}$$

Algoritmo de retropropagación

- Idea subyacente:
  - Minimizar la función $f(x) = x^2 - 6x + 7$.



  - Función derivada $f'(x) = 2x - 6$
  - Como la función es muy simple (un solo mínimo global) y solo depende de una variable (x), para minimizar podemos igualar a cero esta derivada:

$$2x - 6 = 0 \rightarrow x = 6/2 = 3 \tag{1}$$

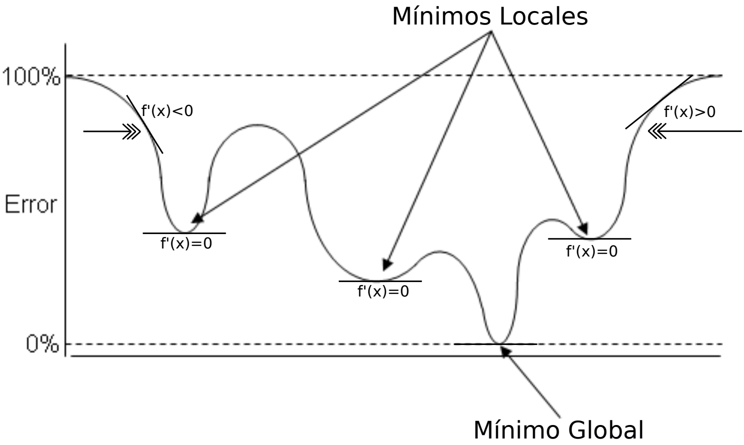
Algoritmo de retropropagación

- La filosofía es la misma, vamos a intentar minimizar el error cometido por la red neuronal, tomando los pesos de la red como variables. Para un patrón concreto, el error medio es:

$$E = \frac{1}{k} \sum_{i=1}^{k} (d_i - o_i)^2 \tag{2}$$

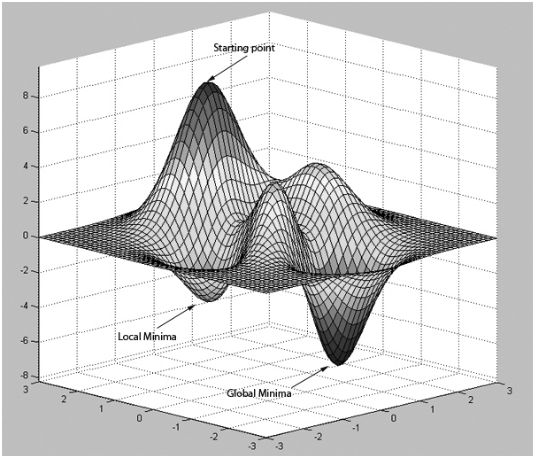
- El valor de d_i es fijo (según el patrón de entrenamiento), pero el valor de o_i depende del valor de los pesos.
- Realizamos un proceso iterativo, donde, dado un valor para los pesos actuales, movemos esos pesos intentando minimizar E.
- Evaluamos la derivada en el punto en el que estamos:
  - Si $f'(x) > 0$, nos movemos para la izquierda.
  - Si $f'(x) < 0$, nos movemos para la derecha.

Algoritmo de retropropagación



Algoritmo de retropropagación

- Imaginemos que la red solo tiene dos pesos, entonces, según el valor de los pesos, podemos representar su superficie de error:



Algoritmo de retropropagación

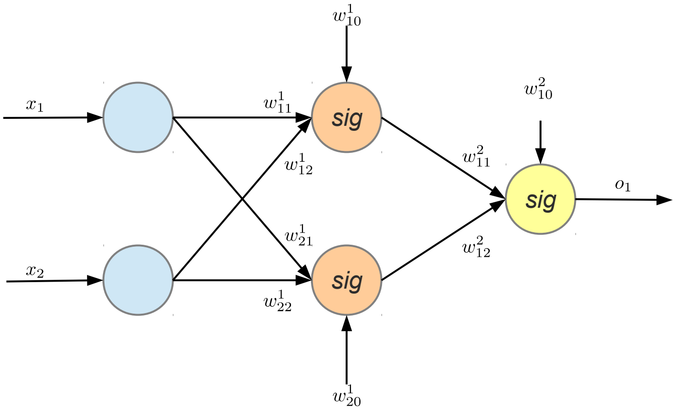
- En el caso de tener múltiples pesos, necesitamos un vector de derivadas, donde cada componente es la derivada del error respecto a cada uno de los pesos.
- Esto es lo que se conoce como el vector gradiente.

$$\nabla E = \left\{ \frac{\partial E}{\partial w_{10}^{1}}, \frac{\partial E}{\partial w_{11}^{1}}, \ldots, \frac{\partial E}{\partial w_{kn_{(H-1)}}^{H}} \right\} \tag{3}$$

- La estructuración en capas de la red neuronal hace que estas derivadas se puedan calcular de forma recursiva.

Algoritmo de retropropagación

- Vamos a calcular las derivadas para un ejemplo simple y luego veremos como se calculan de forma general.



Algoritmo de retropropagación

Fase 1: Propagación hacia delante.

- Llamamos out_j^h a la salida de la neurona j en la capa h.
- Dados dos valores x_1 y x_2 de entrada, calcular la salida de cada neurona.
  - Primera capa:

$$out_1^0 = x_1; out_2^0 = x_2$$

  - Segunda capa:

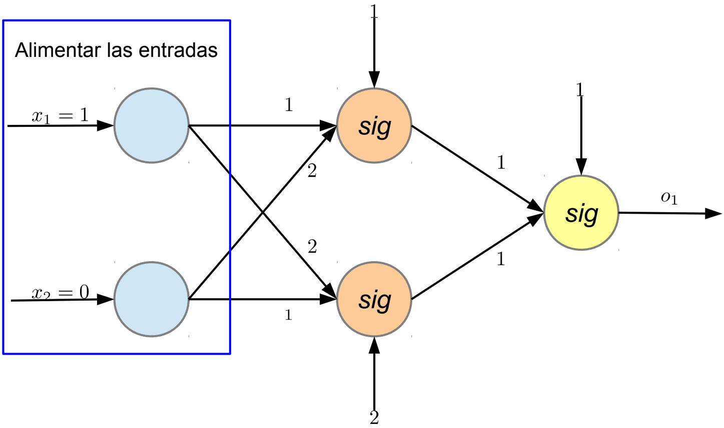
$$\begin{aligned} out_1^1 &= \sigma(w_{10}^1 + w_{11}^1 out_1^0 + w_{12}^1 out_2^0) = \sigma(w_{10}^1 + w_{11}^1 x_1 + w_{12}^1 x_2); \\ out_2^1 &= \sigma(w_{20}^1 + w_{21}^1 out_2^0 + w_{11}^1 out_2^0) = \sigma(w_{20}^1 + w_{21}^1 x_1 + w_{22}^1 x_2); \end{aligned}$$

  - Tercera capa:

$$out_1^2 = o_1 = \sigma(w_{10}^2 + w_{11}^2 out_1^1 + w_{12}^2 out_2^1)$$

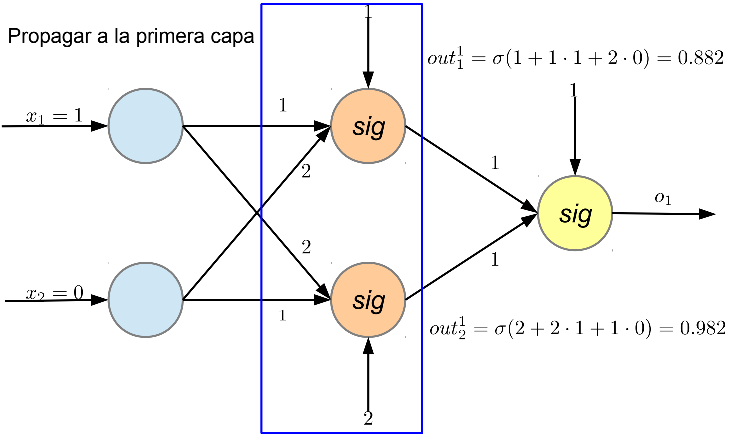
Algoritmo de retropropagación

Fase 1: Propagación hacia delante.



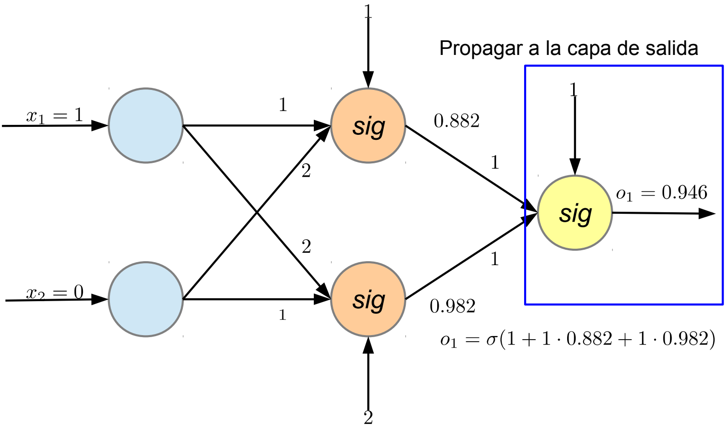
Algoritmo de retropropagación

Fase 1: Propagación hacia delante.



Algoritmo de retropropagación

Fase 1: Propagación hacia delante.



Algoritmo de retropropagación

Fase 2: Cálculo del error y de las derivadas.

- Obtenemos el valor de error cometido por la red:

$$E = (d_1 - o_1)^2$$

- Ahora derivamos ese error respecto a cada uno de los pesos:

$$\nabla E = \left\{ \frac{\partial E}{\partial w_{10}^1}, \frac{\partial E}{\partial w_{11}^1}, \frac{\partial E}{\partial w_{12}^1}, \frac{\partial E}{\partial w_{20}^1}, \frac{\partial E}{\partial w_{21}^1}, \frac{\partial E}{\partial w_{22}^1}, \frac{\partial E}{\partial w_{20}^2}, \frac{\partial E}{\partial w_{21}^2}, \frac{\partial E}{\partial w_{22}^2} \right\}$$

- De la expresión $(d_1 - o_1)^2$, los pesos solo influyen en o_1 (o_1 es una función de cada uno de los pesos). En estos casos, la regla de la cadena nos permite realizar estas derivadas de forma recursiva (lo siguiente se cumple para todos los pesos):

$$\frac{\partial E}{\partial w_{10}^2} = \frac{\partial E}{\partial o_1} \frac{\partial o_1}{\partial w_{10}^2} = -2(d_1 - o_1) \frac{\partial o_1}{\partial w_{10}^2}$$

Algoritmo de retropropagación

- Llamamos net_j^h a la suma ponderada de las entradas de la neurona j en la capa h, es decir, a la salida antes de aplicar la función sigmoide:

$$out_j^h = \sigma(net_j^h)$$

Algoritmo de retropropagación

- Recordamos:

$$o_1 = \sigma(net_1^2)$$
$$\sigma(x)' = \sigma(x)(1-\sigma(x))$$

- Continuamos derivando con respecto a o_1:

$$\frac{\partial E}{\partial w_{10}^2} = -2(d_1 - o_1)\frac{\partial o_1}{\partial w_{10}^2} = -2(d_1 - o_1) \cdot o_1(1-o_1)\frac{\partial net_1^2}{\partial w_{10}^2}$$
$$\frac{\partial E}{\partial w_{11}^2} = -2(d_1 - o_1)\frac{\partial o_1}{\partial w_{11}^2} = -2(d_1 - o_1) \cdot o_1(1-o_1)\frac{\partial net_1^2}{\partial w_{11}^2}$$
$$\frac{\partial E}{\partial w_{12}^2} = -2(d_1 - o_1)\frac{\partial o_1}{\partial w_{12}^2} = -2(d_1 - o_1) \cdot o_1(1-o_1)\frac{\partial net_1^2}{\partial w_{12}^2}$$

Algoritmo de retropropagación

- Y ahora ya podemos escribir las derivadas completas para los pesos de la capa de salida (w^2_{1j}):

$$
\begin{aligned}
net^2_1 &= w^2_{10} + w^2_{11} out^1_1 + w^2_{12} out^1_2 \\
\frac{\partial E}{\partial w^2_{10}} &= -2(d_1 - o_1)o_1(1 - o_1)\frac{\partial net^2_1}{\partial w^2_{10}} = \\
&= -2(d_1 - o_1)o_1(1 - o_1)1 \\
\frac{\partial E}{\partial w^2_{11}} &= -2(d_1 - o_1)o_1(1 - o_1)\frac{\partial net^2_1}{\partial w^2_{11}} = \\
&= -2(d_1 - o_1)o_1(1 - o_1)out^1_1 \\
\frac{\partial E}{\partial w^2_{12}} &= -2(d_1 - o_1)o_1(1 - o_1)\frac{\partial net^2_1}{\partial w^2_{12}} = \\
&= -2(d_1 - o_1)o_1(1 - o_1)out^1_2
\end{aligned}
$$

Algoritmo de retropropagación

- Pasamos a analizar las derivadas de los pesos de la capa oculta (w^1_{1i} y w^1_{2i}):
  - Los pesos w^1_{1i} influyen en out^1_1.
  - Los pesos w^1_{2i} influyen en out^1_2.
- Por tanto, su derivada será similar a las otras derivadas, pero cuando lleguemos a $\frac{\partial net^2_1}{\partial w^1_{ji}}$ tendremos que continuar derivando.
- Para el peso w^1_{10} (sesgo de la primera neurona en la primera capa):

$$\begin{aligned}
net^2_1 &= w^2_{10} + w^2_{11}\color{red}{out^1_1}\color{black} + w^2_{12}out^1_2 \\
\frac{\partial E}{\partial w^1_{10}} &= -2(d_1 - o_1)o_1(1 - o_1)\frac{\partial net^2_1}{\partial w^2_{10}} = \\
&= -2(d_1 - o_1)o_1(1 - o_1)\color{red}{w^2_{11}\frac{\partial out^1_1}{\partial w^2_{10}}}
\end{aligned}$$

Algoritmo de retropropagación

- Recordamos:

$$
\begin{aligned}
out_1^1 &= \sigma(net_1^1) \\
net_1^1 &= w_{10}^1 + w_{11}^1 x_1 + w_{12}^1 x_2 \\
\sigma(x)' &= \sigma(x)(1-\sigma(x))
\end{aligned}
$$

- Continuamos derivando con respecto a out_1^1:

$$
\frac{\partial E}{\partial w_{10}^1} = -2(d_1 - o_1)o_1(1-o_1)w_{11}^2 \frac{\partial out_1^1}{\partial w_{10}^2} =
$$

$$
= -2(d_1 - o_1)o_1(1-o_1)w_{11}^2 out_1^1(1-out_1^1)\frac{\partial net_1^1}{\partial w_{10}^2} =
$$

$$
= -2(d_1 - o_1)o_1(1-o_1)w_{11}^2 out_1^1(1-out_1^1)1
$$

Algoritmo de retropropagación

- Repetimos este proceso para todos los pesos de capa oculta:

$$\frac{\partial E}{\partial w_{10}^1} = -2(d_1 - o_1)o_1(1 - o_1)w_{11}^2 out_1^1(1 - out_1^1)1$$

$$\frac{\partial E}{\partial w_{11}^1} = -2(d_1 - o_1)o_1(1 - o_1)w_{11}^2 out_1^1(1 - out_1^1)x_1$$

$$\frac{\partial E}{\partial w_{12}^1} = -2(d_1 - o_1)o_1(1 - o_1)w_{11}^2 out_1^1(1 - out_1^1)x_2$$

$$\frac{\partial E}{\partial w_{20}^1} = -2(d_1 - o_1)o_1(1 - o_1)w_{12}^2 out_2^1(1 - out_2^1)1$$

$$\frac{\partial E}{\partial w_{21}^1} = -2(d_1 - o_1)o_1(1 - o_1)w_{12}^2 out_2^1(1 - out_2^1)x_1$$

$$\frac{\partial E}{\partial w_{22}^1} = -2(d_1 - o_1)o_1(1 - o_1)w_{12}^2 out_2^1(1 - out_2^1)x_2$$

Algoritmo de retropropagación

- Recapitulando:
  - Capa de salida:

$$\frac{\partial E}{\partial w_{ji}^{2}} = \begin{cases} -2(d_1 - o_1)o_1(1 - o_1)1, & \text{si } i = 0, \\ -2(d_1 - o_1)o_1(1 - o_1)out_i^1, & \text{si } i \neq 0. \end{cases}$$

  - Capa oculta:

$$\frac{\partial E}{\partial w_{ji}^{1}} = \begin{cases} -2(d_1 - o_1)o_1(1 - o_1)w_{1j}^2 out_j^1(1 - out_j^1)1, & \text{si } i = 0, \\ -2(d_1 - o_1)o_1(1 - o_1)w_{1j}^2 out_j^1(1 - out_j^1)x_i, & \text{si } i \neq 0. \end{cases}$$

Algoritmo de retropropagación

- Ojo, hay partes comunes:
  - Capa de salida:

$$\frac{\partial E}{\partial w_{ji}^2} = \begin{cases} -2(d_1 - o_1)o_1(1 - o_1)1, & \text{si } i = 0, \\ -2(d_1 - o_1)o_1(1 - o_1)out_i^1, & \text{si } i \neq 0. \end{cases}$$

  - Capa oculta:

$$\frac{\partial E}{\partial w_{ji}^1} = \begin{cases} -2(d_1 - o_1)o_1(1 - o_1)w_{1j}^2 out_j^1(1 - out_j^1)1, & \text{si } i = 0, \\ -2(d_1 - o_1)o_1(1 - o_1)w_{1j}^2 out_j^1(1 - out_j^1)x_i, & \text{si } i \neq 0. \end{cases}$$

Algoritmo de retropropagación

- Muchas partes son comunes, el cálculo de derivadas se puede realizar de manera recursiva.
- Llamamos δ_j^h a la derivada de la neurona j de la capa oculta h con respecto al error ("cuánta culpa tiene sobre el error esa neurona").

$$\begin{aligned}\delta_1^2 &= -2(d_1 - o_1)o_1(1 - o_1)\\ \delta_1^1 &= w_{11}^2\delta_1^2 out_1^1(1 - out_1^1)\\ \delta_2^1 &= w_{12}^2\delta_1^2 out_2^1(1 - out_2^1)\end{aligned}$$

- A efectos de la actualización de pesos, la constante (2) puede obviarse.

Algoritmo de retropropagación

- Redefinimos las derivadas en función de estos valores δ_j^h:
  - Capa de salida:

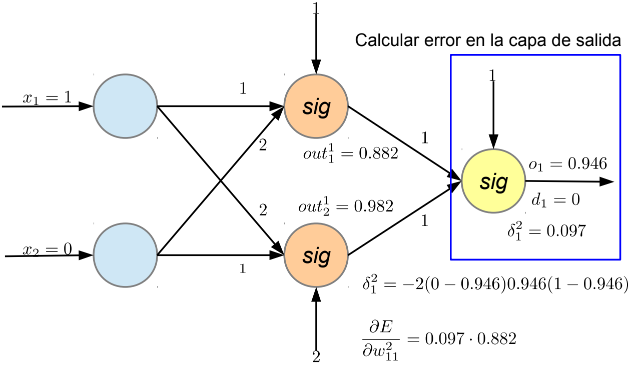
$$\frac{\partial E}{\partial w_{ji}^2} = \begin{cases} \delta_j^2 1, & \text{si } i = 0, \\ \delta_j^2 out_i^1, & \text{si } i \neq 0. \end{cases}$$

  - Capa oculta:

$$\frac{\partial E}{\partial w_{ji}^1} = \begin{cases} \delta_j^1 1, & \text{si } i = 0, \\ \delta_j^1 x_i, & \text{si } i \neq 0. \end{cases}$$

Algoritmo de retropropagación

Fase 2: Propagación hacia detrás.



$$\delta_1^2 = -2(0 - 0.946)0.946(1 - 0.946)$$

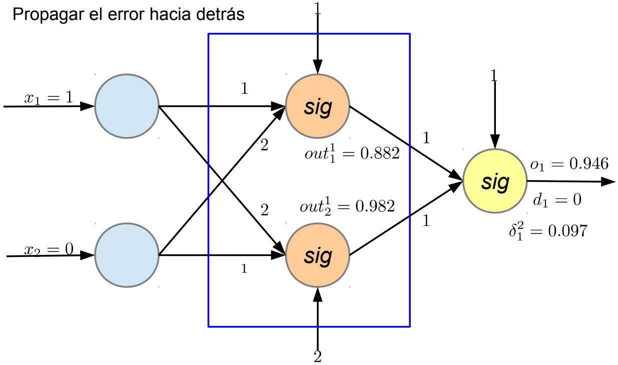
$$\frac{\partial E}{\partial w_{11}^2} = 0.097 \cdot 0.882$$

$$\frac{\partial E}{\partial w_{12}^2} = 0.097 \cdot 0.982$$

$$\frac{\partial E}{\partial w_{10}^2} = 0.097 \cdot 1$$

Algoritmo de retropropagación

Fase 2: Propagación hacia detrás.



$$\delta_1^1 = 1 \cdot 0.097 \cdot 0.882 \cdot (1 - 0.882) = 0.0101$$
$$\delta_2^1 = 1 \cdot 0.097 \cdot 0.982 \cdot (1 - 0.982) = 0.0017$$

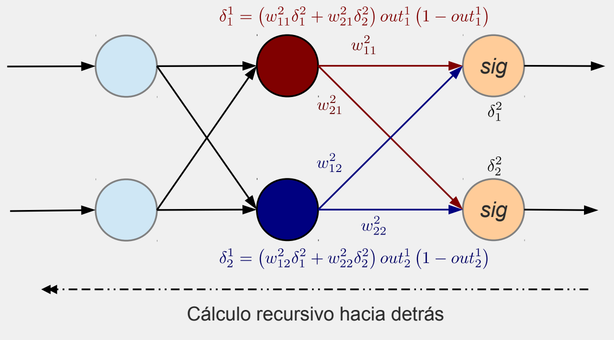
$$\frac{\partial E}{\partial w_{10}^1} = 0.0101 \cdot 1 \quad \frac{\partial E}{\partial w_{11}^1} = 0.0101 \cdot 1 \quad \frac{\partial E}{\partial w_{12}^1} = 0.0101 \cdot 0$$
$$\frac{\partial E}{\partial w_{20}^1} = 0.0017 \cdot 1 \quad \frac{\partial E}{\partial w_{21}^1} = 0.0017 \cdot 1 \quad \frac{\partial E}{\partial w_{22}^1} = 0.0017 \cdot 0$$

Algoritmo de retropropagación

Fase 2: Propagación hacia detrás.

Si hay varias neuronas en la siguiente capa, cada δ_j^h recibe su valor a partir de las neuronas con las que esté conectado:

$$\delta_j^h \leftarrow \left(\sum_{i=1}^{n_{h+1}} w_{ij}^{h+1} \delta_i^{h+1} \right) \cdot out_j^h \cdot (1 - out_j^h)$$



Algoritmo de retropropagación

Fase 3: Actualización de pesos.

- Una vez hemos obtenido el vector gradiente, debemos actualizar los pesos.
- Se utiliza el propio valor de la derivada (sobre todo por su signo), multiplicado por una constante *eta* (η) que controla que los pasos que se van dando no sean muy pequeños o muy grandes (tasa de aprendizaje).
- Fórmula general:

$$w_{ji}^h = w_{ji}^h - \eta \Delta w_{ji}^h$$

$$\Delta w_{ji}^h = \frac{\partial E}{\partial w_{ji}^h} = \begin{cases} \delta_j^h \cdot 1, & \text{si } i = 0 \\ \delta_j^h \cdot out_i^{h-1}, & \text{si } i \neq 0 \end{cases}$$

Algoritmo de retropropagación

Fase 3: Actualización de pesos.

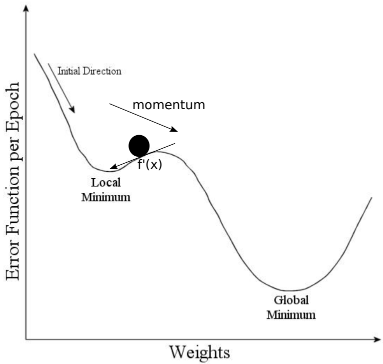
- Ajustar el valor de η es difícil:
  - Si es demasiado grande, podemos provocar oscilaciones.
  - Si es demasiado pequeño, necesitaremos muchas iteraciones.
- Utilizaremos el concepto de momento, para mejorar la convergencia:
  - Los cambios anteriores deberían influir en la dirección de movimiento actual:

$$w_{ji}^{h} = w_{ji}^{h} - \eta \Delta w_{ji}^{h} - \mu \left(\eta \Delta w_{ji}^{h}(t-1) \right)$$

  - Así, los pesos que empiezan a moverse en una determinada dirección, tienden a moverse en ese dirección.
  - La constante *mu* (μ) controla el efecto del momento.

Algoritmo de retropropagación

- La idea es que en un ejemplo como el siguiente, el momento haga que se pueda escapar del óptimo local.
- Pese a que la derivada te diga que vayas a la izquierda, la "*inercia*" puede llevarte a seguir para la derecha:



Pseudocódigo del algoritmo a implementar en la práctica:
Algoritmo de retropropagación *on-line*

Inicio

1. $w_{ji}^h \leftarrow U[-1,1]$ *// Aleatorios entre -1 y $+1$*
2. **Repetir**
   1. **Para** cada patrón con entradas **x**, y salidas **d**
      1. $\Delta w_{ji}^h \leftarrow 0$ *// Se aplicarán cambios por cada patrón*
      2. $out_j^0 \leftarrow x_j$ *// Alimentar entradas*
      3. propagarEntradas() *// Propagar las entradas ($\Rightarrow\Rightarrow$)*
      4. retropropagarError() *// Retropropagar el error ($\Leftarrow\Leftarrow$)*
      5. acumularCambio() *// Calcular ajuste de pesos*
      6. ajustarPesos() *// Aplicar el ajuste calculado*

      Fin Para

   Hasta (CondicionParada)
3. **Devolver** matrices de pesos.

Fin

Pseudocódigo de la versión *off-line* (no se pide):

Algoritmo de retropropagación *off-line*

Inicio

1. $w_{ji}^{h} \leftarrow U[-1,1]$ *// Aleatorios entre -1 y $+1$*
2. **Repetir**
   1. $\Delta w_{ji}^{h} \leftarrow 0$ *// Se aplicarán cambios al final*
   2. **Para** cada patrón con entradas $\mathbf{x}$, y salidas $\mathbf{d}$
      1. $out_j^0 \leftarrow x_j$ *// Alimentar entradas*
      2. propagarEntradas() *// Propagar las entradas ($\Rightarrow\Rightarrow$)*
      3. retropropagarError() *// Retropropagar el error ($\Leftarrow\Leftarrow$)*
      4. acumularCambio() *// Acumular ajuste de pesos*

      Fin Para
   3. ajustarPesos() *// Aplicar el ajuste calculado*

   Hasta (CondicionParada)
3. **Devolver** matrices de pesos.

Fin

Algoritmo de retropropagación: funciones

propagarEntradas()

Inicio

1. **Para** h de 1 a H *// Para cada capa (⇒⇒)*
   1. **Para** j de 1 a n_h *// Para cada neurona de la capa h*
      1. $net_j^h \leftarrow w_{j0}^h + \sum_{i=1}^{n_{h-1}} w_{ji}^h out_i^{h-1}$
      2. $out_j^h \leftarrow \sigma\left(net_j^h\right)$

      Fin Para

   Fin Para

Fin

Algoritmo de retropropagación: funciones

retropropagarError()

Inicio

1. **Para** j de 1 a n_H *// Para cada neurona de salida*
   1. $\delta_j^H \leftarrow -\left(d_j - out_j^H\right) \cdot out_j^H \cdot \left(1 - out_j^H\right)$ *// Hemos obviado la constante (2), ya que el resultado será el mismo*

   Fin Para
2. **Para** h de $H-1$ a 1 *// Para cada capa ($\Leftarrow\Leftarrow$)*
   1. **Para** j de 1 a n_h *// Para cada neurona de la capa h*
      1. $\delta_j^h \leftarrow \left(\sum_{i=1}^{n_{h+1}} w_{ij}^{h+1}\delta_i^{h+1}\right) \cdot out_j^h \cdot (1 - out_j^h)$ *// Pasa por todas las neuronas de la capa $h+1$ conectadas con j*

      Fin Para

   Fin Para

Fin

Algoritmo de retropropagación: funciones

acumularCambio()

Inicio

1. **Para** h de 1 a H *// Para cada capa* $(\Rightarrow\Rightarrow)$
   1. **Para** j de 1 a n_h *// Para cada neurona de la capa h*
      1. **Para** i de 1 a n_{h-1} *// Para cada neurona de la capa* $h-1$
         $\Delta w_{ji}^h \leftarrow \Delta w_{ji}^h + \delta_j^h \cdot out_i^{h-1}$
         Fin Para
      2. $\Delta w_{j0}^h \leftarrow \Delta w_{j0}^h + \delta_j^h \cdot 1$ *// Sesgo*

      Fin Para

   Fin Para

Fin

Algoritmo de retropropagación: funciones

ajustarPesos()

Inicio

1. **Para** h de 1 a H *// Para cada capa ($\Rightarrow\Rightarrow$)*
   1. **Para** j de 1 a n_h *// Para cada neurona de la capa h*
      1. **Para** i de 1 a n_{h-1} *// Para cada neurona de la capa $h-1$*
         $w_{ji}^h \leftarrow w_{ji}^h - \eta \Delta w_{ji}^h - \mu \left(\eta \Delta w_{ji}^h(t-1) \right)$
         Fin Para
      2. $w_{j0}^h \leftarrow w_{j0}^h - \eta \Delta w_{j0}^h - \mu \left(\eta \Delta w_{j0}^h(t-1) \right)$ *// Sesgo*

      Fin Para

   Fin Para

Fin

Introducción a los modelos computacionales

Práctica 1. Implementación del perceptrón multicapa

Pedro Antonio Gutiérrez
pagutierrez@uco.es

Asignatura "Introducción a los modelos computacionales"
4º Curso Grado en Ingeniería Informática
Especialidad Computación
Escuela Politécnica Superior
(Universidad de Córdoba)

26 de septiembre de 2016