Tema 8. Redes Neuronales

Pedro Larrañaga, Iñaki Inza, Abdelmalik Moujahid
Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad del País Vasco–Euskal Herriko Unibertsitatea

8.1 Introducción

En este tema vamos a introducir el paradigma de redes neuronales artificiales, muy popular dentro de la Inteligencia Computacional. El enfoque que se va a seguir en la introducción del paradigma es fundamentalmente como una herramienta para resolver problemas de clasificación supervisada. En ningún momento tratamos de plantear un modelo del sistema nervioso de los seres vivos.

A lo largo del tema se comienza en la Sección 8.2, introduciendo de manera muy resumida aquellos aspectos mas relevantes de un sistema neuronal artificial, tales como *neurona*, *dendrita*, *axón*, *sinapsis*, *función de activación*, finalizándose la sección con la exposición de distintos tipos de *arquitecturas de redes neuronales*. En la Sección 8.3 se presenta el modelo de red neuronal mas simple, el denominado *asociador lineal*, incluyéndose en la sección algunas ideas relacionadas con el aprendizaje Hebbiano. La Sección 8.4 presenta el *perceptrón simple* junto con el algoritmo de aprendizaje de pesos propuesto por Rosenblatt (1962) para este modelo de red neuronal. La Sección 8.5 introduce la denominada *adalina* junto con la regla de actualización de pesos propuesta para la misma por Widrow y Hoff (1960). El modelo *perceptrón multicapa* al igual que el *algoritmo de retropropagación de error* propuesto para el mismo por vez primera por Werboz (1974) se desarrollan en la Sección 8.6.

8.2 Sistema Neuronal Artificial

- Introducción biológica:
  - 1888 Ramón y Cajal demuestra que el sistema nervioso está compuesto por una red de células individuales, las neuronas, ampliamente interconectadas entre sí.

    La información fluye desde las dendritas hacia el axón atravesando el soma.
  - *Neurona*: $100 \cdot 10^6$ neuronas en el cerebro.
    - cuerpo celular o soma (de 10 a 80 micras de longitud)
    - del soma surge un denso árbol de ramificaciones (árbol dendrítico) formado por las dendritas
    - del soma parte una fibra tubular denominada axón (longitud de 100 micras hasta un metro)
    - el axón se ramifica en su extremo final para conectar con otras neuronas
  - Neuronas como procesadores de información sencillos. De manera simplista:

    - las dendritas constituyen el canal de entrada de la información
    - el soma es el órgano de cómputo
    - el axón corresponde al canal de salida, y a la vez envía información a otras neuronas. Cada neurona recibe información de aproximadamente 10,000 neuronas y envía impulsos a cientos de ellas
    - algunas neuronas reciben la información directamente del exterior
  - El cerebro se modela durante el desarrollo de un ser vivo. Algunas cualidades del ser humano no son innatas, sino adquiridas por la influencia de la información que del medio externo se proporciona a sus sensores.

    Diferentes maneras de modelar el sistema nervioso:

    - establecimiento de nuevas conexiones
    - ruptura de conexiones
    - modelado de las intensidades sinápticas (uniones entre neuronas)
    - muerte o reproducción neuronal

- Sistemas artificiales que van a copiar la estructura de las redes neuronales biológicas con el fin de alcanzar una funcionalidad similar.

- Tres conceptos clave a emular:

  - *procesamiento paralelo*, derivado de que los miles de millones de neuronas que intervienen, por ejemplo en el proceso de ver, están operando en paralelo sobre la totalidad de la imagen
  - *memoria distribuida*, mientras que en un computador la información está en posiciones de memoria bien definidas, en las redes neuronales biológicas dicha información está distribuida por la sinapsis de la red, existiendo una redundancia en el almacenamiento, para evitar la pérdida de información en caso de que una sinapsis resulte dañada.
  - *adaptabilidad al entorno*, por medio de la información de las sinapsis. Por medio de esta adaptabilidad se puede aprender de la experiencia y es posible generalizar conceptos a partir de casos particulares

El elemento básico de un sistema neuronal biológico es la neurona. Un sistema neuronal biológico está compuesto por millones de neuronas organizadas en capas. En la emulación de dicho sistema neuronal biológico, por medio de un sistema neuronal artificial, se puede establecer una estructura jerárquica similar a la existente en el cerebro. El elemento esencial será la neurona artificial, la cual se organizará en capas. Varias capas constituirán una red neuronal. Finalmente una red neuronal junto con los interfaces de entrada y salida constituirá el sistema global de proceso (véase Figura 1).

- *El modelo estándard de neurona artificial*

  Se va a introducir el denominado modelo estándard de neurona artificial según los principios descritos en Rumelhart y McClelland (1986) y McClelland y Rumelhart (1986). Siguiendo dichos principios, la i-ésima neurona artificial estándard consiste en:

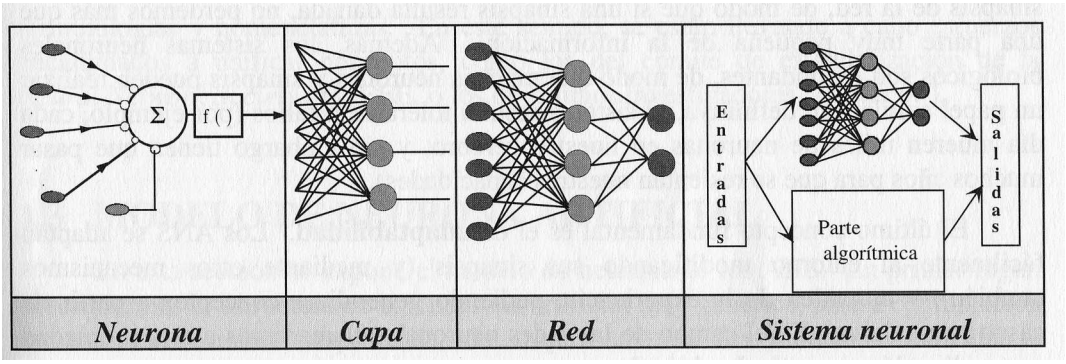

Figura 1: Sistema global de proceso de una red neuronal

- Un *conjunto de entradas* x_j y unos pesos sinápticos w_{ij}, con $j = 1, \ldots, n$
- Una *regla de propagación* h_i definida a partir del conjunto de entradas y los pesos sinápticos. Es decir:

$$h_i(x_1, \ldots, x_n, w_{i_1}, \ldots, w_{i_n})$$

La regla de propagación más comunmente utilizada consiste en combinar linealmente las entradas y los pesos sinápticos, obteniéndose:

$$h_i(x_1, \ldots, x_n, w_{i_1}, \ldots, w_{i_n}) = \sum_{i=1}^{n} w_{ij} x_j$$

Suele ser habitual añadir al conjunto de pesos de la neurona un parámetro adicional θ_i, que se denomina umbral, el cual se acostumbra a restar al potencial pos-sináptico. Es decir:

$$h_i(x_1, \ldots, x_n, w_{i_1}, \ldots, w_{i_n}) = \sum_{i=1}^{n} w_{ij} x_j - \theta_i$$

Si hacemos que los índices i y j comiencen en 0, y denotamos por $w_{i0} = \theta_i$ y $x_0 = -1$, podemos expresar la regla de propagación como:

$$h_i(x_1, \ldots, x_n, w_{i_1}, \ldots, w_{i_n}) = \sum_{j=0}^{n} w_{ij} x_j = \sum_{i=1}^{n} w_{ij} x_j - \theta_i$$

- Una *función de activación*, la cual representa simultáneamente la salida de la neurona y su estado de activación. Si denotamos por y_i dicha función de activación, se tiene $y_i = f_i(h_i) = f_i(\sum_{j=0}^{n} w_{ij} x_j)$

La Figura 2 muestra el modelo de neurona artificial estándar descrito previamente.

Algunos ejemplos de funciones de activación son los siguientes:

(i) *Neuronas todo-nada*

En este tipo de neuronas todo-nada, también llamadas dispositivos de

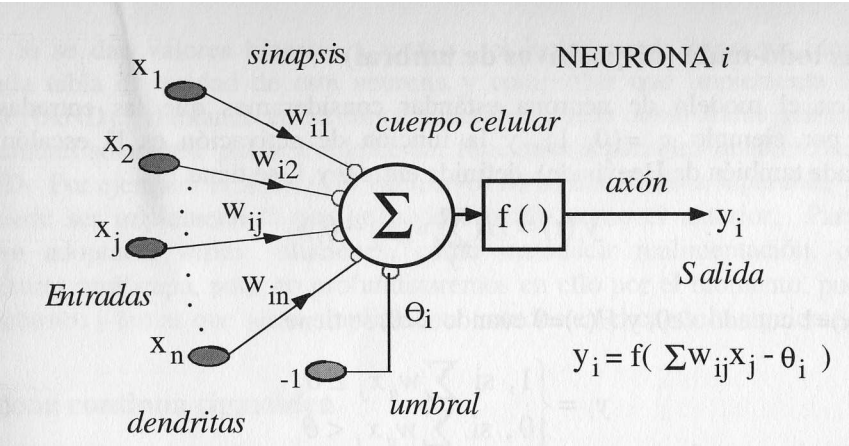

Figura 2: Modelo de neurona artificial standard

umbral, la función $f_i(\sum_{j=1}^{n} w_{ij}x_j - \theta_i)$ es una función escalonada. En tal caso, se tiene:

$$y_i = \begin{cases} 1 & \text{si } \sum_{j=1}^{n} w_{ij}x_j \geq \theta_i \\ 0 & \text{si } \sum_{j=1}^{n} w_{ij}x_j < \theta_i \end{cases}$$

Este tipo de función de activación es la usada tanto por el modelo de neurona del perceptrón original –como se verá más adelante– así como por el modelo de neurona introducido en el trabajo pionero de McCulloch y Pitts (1943). Nótese que en las neuronas todo-nada la salida es digital.

(ii) *Neurona continua sigmoidea*

Si queremos obtener una salida continua, es habitual el utilizar como función de activación una función sigmoidea. Las funciones sigmoideas más usadas son:

$$y_i = \frac{1}{1 + e^{-\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right)}}, \text{ con } y_i \in [0,1]$$

$$y_i = \frac{e^{\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right)} - e^{-\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right)}}{e^{\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right)} + e^{-\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right)}}, \text{ con } y_i \in [-1,1]$$

Ambas funciones de activación son continuas y diferenciables, y son usadas en el denominado perceptrón multicapa –véase Sección 8.6–. El requisito de trabajar con funciones diferenciables puede venir impuesto por la regla de aprendizaje, como sucede –tal y como se explicará posteriormente– con la regla de retropropagación del error (*back-propagation*).

- *Arquitecturas de redes neuronales*

  Se denomina arquitectura a la topología, estructura o patrón de conexionado de una red neuronal. En una red neuronal artificial los nodos se conectan por medio de sinapsis, estando el comportamiento de la red determinado por la estructura de conexiones sinápticas. Estas conexiones sinápticas son direccionales, es decir, la información solamente puede propagarse en un único sentido (desde la neurona presináptica a la pos-sináptica). En general las neuronas se suelen agrupar en unidades estructurales que denominaremos *capas*. El conjunto de una o más capas constituye la red neuronal.

  Se distinguen tres tipos de capas: de entrada, de salida y ocultas. Una *capa de entrada*, también denominada sensorial, está compuesta por neuronas que reciben datos o señales procedentes del entorno. Una *capa de salida* se compone de neuronas que proporcionan la respuesta de la red neuronal. Una *capa oculta* no tiene una conexión directa con el entorno, es decir, no se conecta directamente ni a órganos sensores ni a efectores. Este tipo de capa oculta proporciona grados de libertad a la red neuronal gracias a los cuales es capaz de representar más fehacientemente determinadas características del entorno que trata de modelar. Véase la Figura 3.

  Teniendo en cuenta diversos conceptos se pueden establecer diferentes tipos de

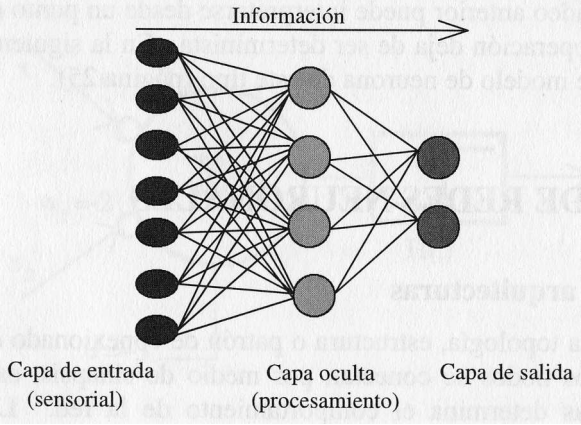

Figura 3: Arquitectura unidireccional con tres capas de neuronas: una capa de entrada, una capa oculta y una capa de salida

  arquitecturas neuronales. Así considerando su estructura podemos hablar de *redes monocapa* –compuestas por una única capa de neuronas– o *redes multicapa* –las neuronas se organizan en varias capas–. Teniendo en cuenta el flujo de datos, podemos distinguir entre *redes unidireccionales (feedforward)* y *redes recurrentes o realimentadas (feedback)*. Mientras que en las redes unidireccionales la información circula en un único sentido, en las redes recurrentes o realimentadas la información puede circular entre las distintas capas de neuronas en cualquier sentido, incluso en el de salida-entrada. La Figura 4 muestra dos ejemplos de arquitectura, uno corresponde a una red monocapa y recurrente y el otro a una red multicapa y unidireccional.

- *Definición de una red neuronal artificial*

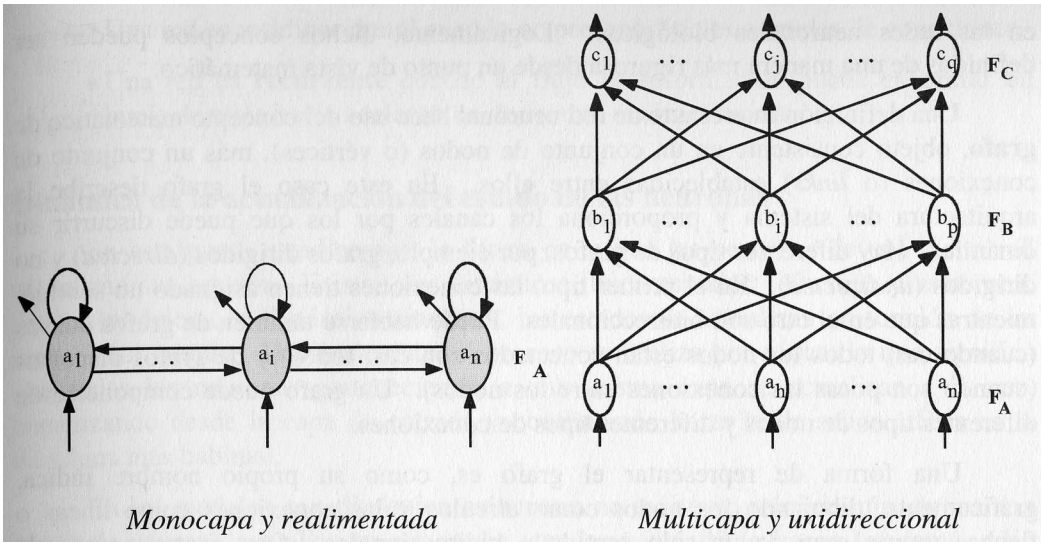

Figura 4: Diferentes arquitecturas de redes neuronales

Se puede definir una red neuronal artificial como un grafo dirigido, con las siguientes propiedades:

(i) A cada nodo (neurona) i se le asocia una variable de estado X_i.

(ii) A cada conexión (i, j) entre los nodos (neuronas) i y j se le asocia un peso $w_{ij} \in \mathbb{R}$.

(iii) A cada nodo (neurona) i se le asocia un umbral $\theta_i \in \mathbb{R}$.

(iv) Para cada nodo i se define una función $f_i(x_1, \ldots, x_n, w_{i_1}, \ldots, w_{i_n}, \theta_i)$ que depende de los pesos de sus conexiones, del umbral y de los estados de los nodos j que estén conectados con el nodo i. El valor de esta función proporciona el nuevo estado del nodo.

Por lo que respecta a la terminología habitual en redes neuronales artificiales, tal y como se ha comentado previamente los nodos del grafo representan a las neuronas y las conexiones a las *sinapsis*. Se denominan *neuronas de entrada* a aquellas neuronas sin sinapsis entrantes. A las neuronas sin sinapsis salientes se las denomina *neuronas de salida*, y finalmente a aquellas neuronas que no son ni de entrada ni de salida se les denomina *neuronas ocultas*. Una red es *unidireccional* cuando no presenta bucles cerrados o conexiones, mientras que una red se dice *recurrente* o *realimentada* cuando el flujo de información puede tener un bucle de atrás hacia adelante, es decir, una realimentación. En la relación con la manera en la que las neuronas de una red actualizan sus estados, podemos distinguir entre *dinámica síncrona* –en la cual todas las neuronas pertenecientes a una misma capa se actualizan a la vez, comenzando en la capa de entrada y continuando hasta la de salida– y *dinámica asíncrona* –en la cual cada neurona actualiza su estado sin atender a cuando lo hacen las demás neuronas–. Si bien el tipo de dinámica presente en los sistemas neuronales biológicos es asíncrono, lo habitual en las redes neuronales artificiales es que la dinámica sea síncrona.

8.3 El Asociador Lineal

En este tema vamos a presentar varios modelos de redes neuronales unidireccionales organizados en capas y cuya finalidad sea tratar un problema de aprendizaje supervisado, es decir, vamos a presentar modelos estándar de redes neuronales para reconocer patrones. Textos en los que profundizar los conocimientos aquí presentados incluyen los de Bishop (1995), Haykin (1999) y Principe y col. (2000).

Ha de tenerse en cuenta que en el asociador lineal las variables (neuronas) de salida son continuas, de ahí que el interés de este paradigma para la resolución de problemas de clasificación supervisada sea limitado. Nótese igualmente que la notación usada en esta sección no es coherente con la del resto del tema, ya que en esta sección se denota por $y_1, \dots y_m$ a los valores de las variables respuesta a predecir, mientras que el valor predicho por la red neuronal se denota por $W\mathbf{x}$.

- *El asociador lineal. Aprendizaje Hebbiano*

  El denominado asociador lineal –véase Figura 5– consta únicamente de una capa

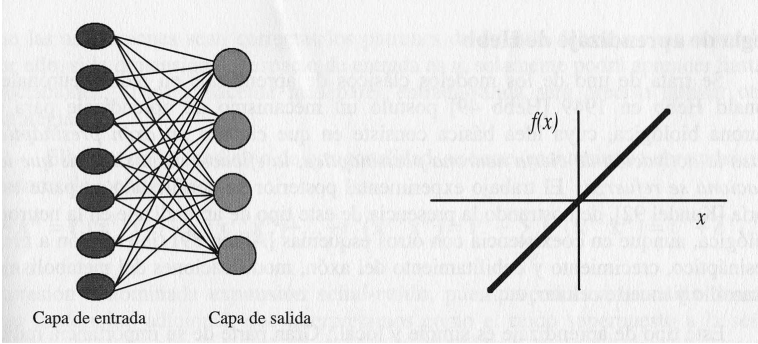

Figura 5: Arquitectura (izquierda) y función de activación (derecha) del asociador lineal

de neuronas lineales cuyas entradas denotamos por $x_1, \dots, x_n$ y cuyas salidas se denotan por $y_1, \dots, y_m$. Denotamos por $W \in M(m, n)$ a la matriz de pesos sinápticos, cuyos elementos se expresan por medio de w_{ij} con $i = 1, \dots, m$ y $j = 1, \dots, n$.

La operación efectuada por el asociador lineal es:

$\mathbf{y} = (y_1, \dots, y_m) == W(x_1, \dots, x_n)$

o bien

$y_i = \sum_{j=1}^{n} w_{ij} x_j$, con $i = 1, \dots, m$.

Dentro del marco de neurona estándar presentado en la sección anterior, el asociador lineal calcula el potencial pos-sináptico por medio de la convencional suma ponderada, cantidad a la que aplica posteriormente una función de activación de tipo identidad.

El asociador lineal debe aprender a asociar N pares entrada-salida

$$D = \{(\mathbf{x}^r, \mathbf{y}^r), r = 1, \dots, N\}$$

ajustando la matriz de pesos W, de tal manera que ante entradas similares a $\mathbf{x}^r$ responda con salidas similares a $\mathbf{y}^r$. El problema radica en encontrar la matriz de pesos W óptima en el sentido anterior. Para ello, en el campo de las redes

neuronales se hace uso de una regla de aprendizaje, que a partir de las entradas y de las salidas deseadas proporcione el conjunto óptimo de pesos W.

Uno de los modelos clásicos de aprendizaje de redes neuronales es el propuesto por Hebb (1949), el cual postuló un mecanismo de aprendizaje para una neurona biológica, cuya idea básica consiste en que cuando un axón presináptico causa la activación de cierta neurona pos-sináptica, la eficacia de la sinapsis que las relaciona se refuerza.

Si bien este tipo de aprendizaje es simple y local, su importancia radica en que fue pionero tanto en neurociencias como en neurocomputación, de ahí que otros algoritmos más complejos lo tomen como punto de partida.

De manera general se denomina *aprendizaje Hebbiano* a un aprendizaje que involucra una modificación en los pesos, Δw_{ij}, proporcional al producto de una entrada x_j y de una salida y_i de la neurona. Es decir, $\Delta w_{ij} = \varepsilon y_i x_j$, donde a $0 < \varepsilon < 1$ se le denomina *ritmo de aprendizaje*. En concreto al considerar el ejemplo $(\mathbf{x}^r, \mathbf{y}^r)$ la regla de actualización de pesos resulta ser:

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}^r$$

$$\text{con } \Delta w_{ij}^r = \varepsilon y_i^r x_j^r$$

Es habitual el tratar de deducir los algoritmos de aprendizaje a partir de un cierto criterio a optimizar, es decir, se debe de proponer un criterio que mida el rendimiento de la red neuronal para encontrar una regla de actualización de pesos que la optimice. Una manera extendida de definir el rendimiento es a partir del error cuadrático medio de las salidas actuales de la red respecto de las deseadas. Es decir:

$$\frac{1}{N}\sum_{r=1}^{N} ||\mathbf{y}^r - W\mathbf{x}^r|| = \frac{1}{N}\sum_{r=1}^{N}\sum_{i=1}^{m}\sum_{j=1}^{n}(y_i^r - Wx_j^r)^2$$

De esta manera el problema del aprendizaje de los pesos de la red neuronal se transforma en el de obtener un conjunto de pesos que minimicen la expresión anterior. Si denotamos por $\mathbf{X} = (\mathbf{x}^1, \ldots, \mathbf{x}^N)$, una matriz $n \times N$ que tiene por columnas los vectores de entrada, y por $\mathbf{Y} = (\mathbf{y}^1, \ldots, \mathbf{y}^N)$ una matriz $m \times N$ cuyas columnas son los vectores de salida, la anterior ecuación se puede expresar como:

$$\frac{1}{N}\|\mathbf{Y} - W\mathbf{X}\|$$

La minimización de la expresión anterior se obtiene al hacer $W = \mathbf{Y}\mathbf{X}^{-1}$, de ahí que una regla de aprendizaje basada en la matriz pseudoinversa se puede escribir como $W = \mathbf{Y}\mathbf{X}^{+}$, donde $\mathbf{X}^{+}$ denota la matriz pseudoinversa de $\mathbf{X}$.

8.4 El Perceptrón Simple

El perceptrón simple fue introducido por Rosenblatt (1962) y es un modelo unidireccional compuesto por dos capas de neuronas, una de entrada y otra de salida. La operación en un perceptrón simple que consta de n neuronas de entrada y m neuronas de salida se puede expresar como:

$$y_i = f\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right)$$

con $i = 1, \ldots, m$.

Las neuronas de entrada son discretas y la función de activación de las neuronas de la capa de salida es de tipo escalón. Véase la Figura 6.

El perceptrón simple puede utilizarse como clasificador, radicando su importancia

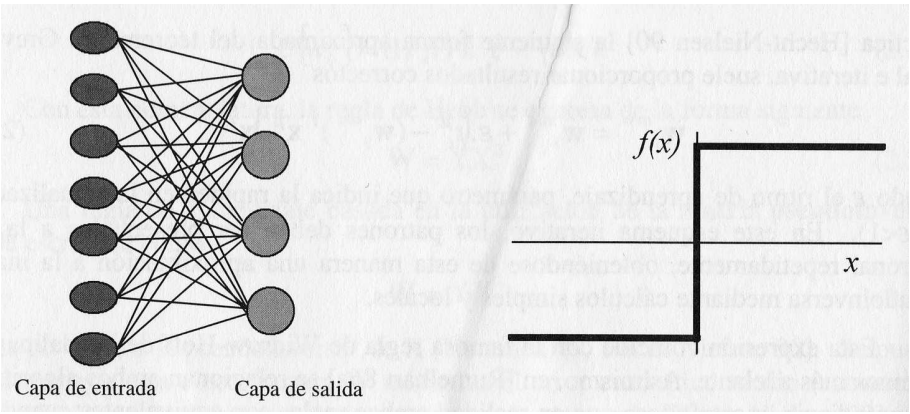

Figura 6: Arquitectura (izquierda) y función de transferencia (derecha) de un perceptrón simple

histórica en su carácter de dispositivo entrenable, ya que el algoritmo de aprendizaje del modelo introducido por Rosenblatt (1962) permite determinar automáticamente los pesos sinápticos que clasifican un conjunto de patrones a partir de un conjunto de ejemplos etiquetados.

Veamos con un ejemplo sencillo, que contiene dos neuronas de entrada, que el perceptrón simple tan sólo puede discriminar entre dos clases linealmente separables, es decir, clases cuyas regiones de decisión pueden ser separadas mediante una única condición lineal o hiperplano.

Si denotamos por x_1 y x_2 a las dos neuronas de entrada, la operación efectuada por el perceptrón simple consiste en:

$$y = \begin{cases} 1 & \text{si } w_1 x_1 + w_2 x_2 \geq \theta \\ 0 & \text{si } w_1 x_1 + w_2 x_2 < \theta \end{cases}$$

Si consideramos x_1 y x_2 situadas sobre los ejes de abcisas y ordenadas respectivamente, la condición

$$w_1 x_1 + w_2 x_2 - \theta = 0$$

es equivalente a

$$x_2 = -\frac{w_1}{w_2} x_1 + \frac{\theta}{w_2}$$

y representa una recta que define la región de decisión determinada por el perceptrón simple. Es por ello que dicho perceptrón simple representa un discriminador lineal, al implementar una condición lineal que separa dos regiones en el espacio que representan dos clases diferentes de patrones. Véase la Figura 7.

Por tanto, el perceptrón simple presenta grandes limitaciones, ya que tan sólo es capaz de representar funciones linealmente separables. Basándose en este hecho, Minsky y Papert (1969) publicaron un trabajo exponiendo las limitaciones del perceptrón simple, como consecuencia del cual muchos de los recursos que se venían dedicando a las

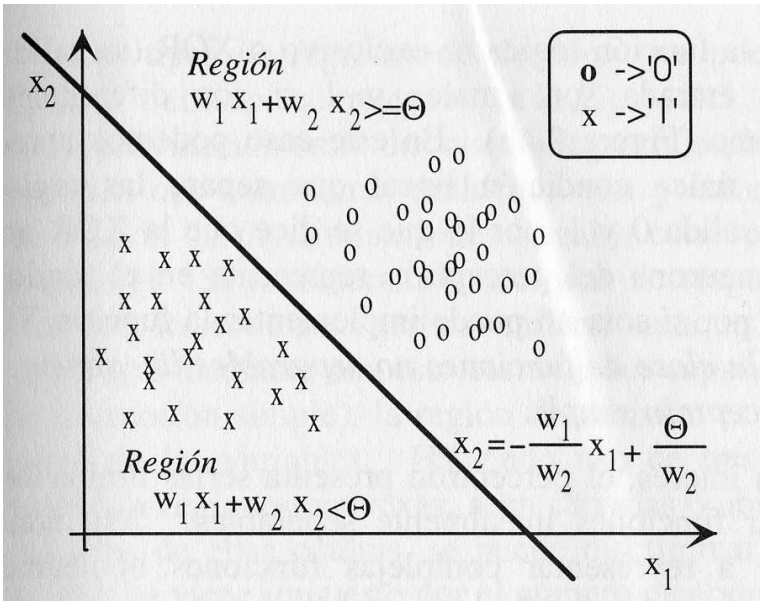

Figura 7: Región de decisión correspondiente a un perceptrón simple con dos neuronas de entrada

redes neuronales se desviaron a otros campos de la inteligencia artificial.

Tal y como se ha comentado previamente, la importancia del perceptrón simple radica en el hecho de su carácter entrenable, ya que el algoritmo introducido por Rosenblatt (1962) permite que el perceptrón simple determine automáticamente los pesos sinápticos que clasifican un conjunto de patrones etiquetados.

El algoritmo de aprendizaje del perceptrón simple pertenece al grupo de los algoritmos que se fundamentan en la corrección de errores. Los algoritmos de este tipo ajustan los pesos de manera proporcional a la diferencia existente entre la salida actual de la red neuronal y la salida deseada, con el objetivo de minimizar el error actual de la red.

Veamos el funcionamiento del algoritmo de aprendizaje de pesos para el perceptrón simple propuesto por Rosenblatt (1962). Vamos a denotar por $\mathbf{x}^r$ al conjunto de patrones de entrada y por c^r a sus salidas respectivas, con $r = 1, \ldots, N$. Supongamos que tanto las variables de entrada como las de salida toman dos posibles valores: -1 y $+1$. Se parte de una arquitectura de red neuronal que es un perceptrón simple y se requiere que clasifique correctamente todos los patrones de los que partimos, para llevar a cabo el entrenamiento de la red neuronal. La manera en la que actualizaremos los pesos es la siguiente: si ante la presentación de r-ésimo patrón la respuesta que proporciona el perceptrón simple es correcta, no actualizaremos los pesos, mientras que si la respuesta es incorrecta los pesos se modificarán según la regla de Hebb, es decir

$$\Delta w_{ij}^r(t) = \begin{cases} 2\varepsilon {c_i}^r {x_j}^r & \text{si } {y_i}^r \neq {c_i}^r \\ 0 & \text{si } {y_i}^r = {c_i}^r \end{cases}$$

La regla anterior se puede reescribir de la siguiente manera

$$\Delta w_{ij}^r(t) = \varepsilon({c_i}^r - {y_i}^r){x_j}^r$$

que es la forma habitual de expresar la regla de actualización de pesos del perceptrón simple, cuando las entradas y las salidas son discretas y toman valores -1 y $+1$. Puede

comprobarse que en este caso la actualización de pesos únicamente podrá tomar los valores -2ε, 0 y $+2\varepsilon$. A nivel práctico se debe llegar a un compromiso para el valor del ritmo de aprendizaje, ε, ya que un valor pequeño del mismo implica un aprendizaje lento, mientras que uno excesivamente grande puede conducir a oscilaciones excesivas de los pesos no aconsejables en el proceso de entrenamiento.

Conviene aclarar que el proceso de aprendizaje es iterativo. Se parte de una configuración sináptica inicial –habitualmente pesos pequeños inicializados aleatoriamente– presentándose los patrones una y otra vez, con objeto de que los pesos se ajusten iterativamente según la regla anterior, $\Delta w_{ij}^r(t) = \varepsilon(c_i{}^r - y_i{}^r)x_j{}^r$, hasta que todos los patrones queden bien clasificados si es posible-. El hiperplano que establece el límite entre dos clases tal y como se aprecia en la Figura 8 se desplaza lentamente hasta conseguir separarlas por completo –si esto fuera posible–. El ajuste de los pesos en la iteración t debido a todo el conjunto de aprendizaje será

$$w_{ij}(t+1) = w_{ij}(t) + \sum_{r=1}^{N} \Delta w_{ij}{}^r(t)$$

Rosenblatt (1962) demostró que si la función a representar es linealmente separable,

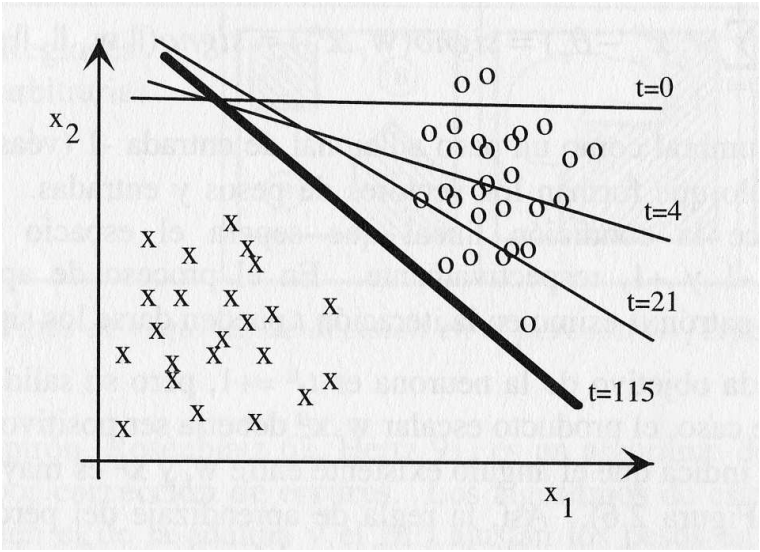

Figura 8: Evolución de las regiones de decisión establecidas por el perceptrón simple

el algoritmo anterior siempre converge en un tiempo finito y con independencia de los pesos de partida. Si la función a representar no es linealmente separable, el proceso de entrenamiento oscilará. Tal y como puede en la Figura 8 el algoritmo de entrenamiento del perceptrón simple se detiene tan pronto como consigue clasificar correctamente todos los ejemplos de los que consta la base de datos inicial, de ahí que ocurra con frecuencia que la línea de discriminación quede muy cerca de las muestras de uno de los grupos.

8.5 La Adalina

Otro modelo de red neuronal artificial clásico es la Adalina, introducida por Widrow y Hoff (1960), cuyo nombre proviene de ADAptative LInear Neuron. En la Adalina las entradas pueden ser continuas y se utiliza una neurona similar a la del perceptrón

simple, pero en este caso de respuesta lineal –véase la Figura 9–. A diferencia del asociador lineal la adalina incorpora un parámetro adicional denominado *bias*, el cual no debe de ser considerado como un umbral de disparo, sino como un parámetro que proporciona un grado de libertad adicional al modelo.

Teniendo en cuenta lo anterior, la ecuación de la adalina resulta ser

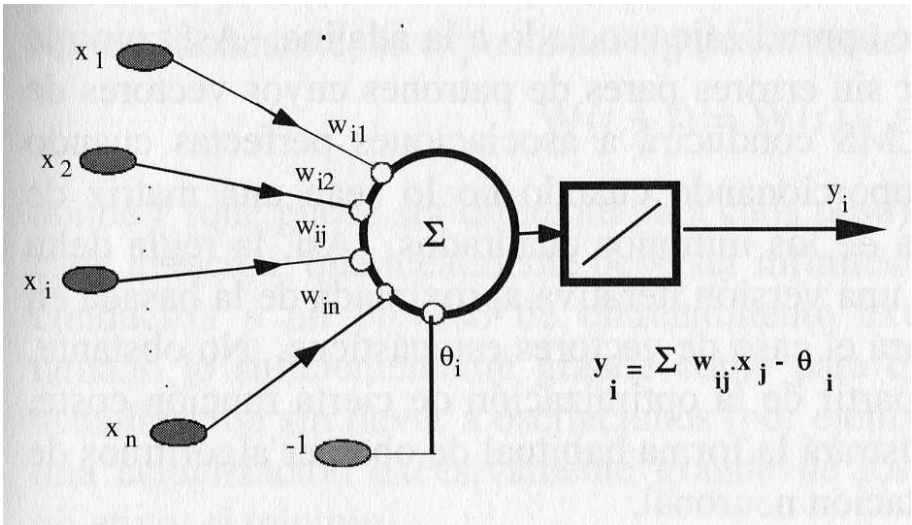

Figura 9: Neurona lineal de la Adalina

$$y_i(t) = \sum_{j=1}^{n} w_{ij} x_j - \theta_i$$

con $i = 1, \ldots, m$.

Otra diferencia fundamental de la adalina con respecto del asociador lineal y el perceptrón simple radica en la regla de aprendizaje. En la adalina se utiliza la regla de Widrow-Hoff, también conocida como LMS (*Least Mean Square*) o regla de mínimos cuadrados. Esta regla permite actualizaciones de los pesos proporcionales al error cometido por la neurona.

La adalina se viene utilizando desde comienzos de los años sesenta como filtro adaptativo, por ejemplo en aplicaciones relacionadas con la reducción del ruido en la transmisión de señales (Widrow y Winter, 1988). De este modo, y desde hace años, millones de modems de todo el mundo incluyen una adalina.

La regla de actualización de pesos LMS que se lleva a cabo en la adalina se fundamenta en considerar el aprendizaje de dichos pesos como un problema de optimización de una determinada función de coste. Dicha función de coste va a medir el rendimiento actual de la red y dependerá de los pesos sinápticos de la misma. Vamos a ver cómo un método de optimización aplicado a dicha función de coste va a proporcionar una regla de actualización de los pesos, que a partir de los patrones de aprendizaje modifique iterativamente los pesos hasta alcanzar la configuración "óptima" de los mismos.

El método de optimización utilizado por la regla LMS es el denominado *descenso por el gradiente*, el cual puede ser visto como un optimizador local en un espacio de búsqueda continuo. Se parte de una función de coste E que proporciona el error

actual cometido por la red neuronal. Dicha función de coste será una función de los pesos sinápticos. Es decir:

$$\begin{array}{rl} E : \mathbb{R}^{(n\times m)+m} & \longrightarrow \mathbb{R} \\ (w_{11},\ldots,w_{1n},\ldots,w_{m1},\ldots,w_{mn},\theta_1,\ldots,\theta_m) & \longrightarrow E(w_{11},\ldots,w_{1n},\ldots,w_{m1},\ldots,w_{mn},\theta_1,\ldots,\theta_m) \end{array}$$

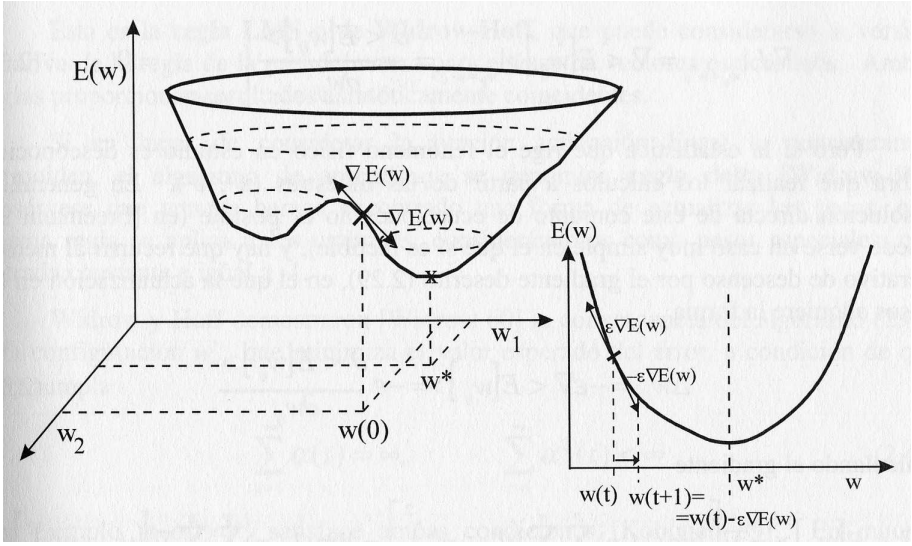

Figura 10: Superficie de error $E(w)$ en el espacio de pesos (izquierda) y método de descenso por el gradiente (derecha) en el que se basa la regla de aprendizaje de la Adalina

Nos podemos imaginar la representación gráfica de esta función como una hipersuperficie con montañas y valles (véase Figura 10) en la que la posición ocupada por un valle corresponde con una configuración de pesos localmente óptima. El objetivo del aprendizaje, es decir del proceso de actualización de pesos, consiste en encontrar la configuración de los mismos que corresponde al mínimo global de la función de error o función de coste definida. Con frecuencia en una red neuronal genérica nos vamos a tener que conformar con un mínimo local suficientemente bueno.

Para encontrar la configuración de pesos óptima mediante el método del descenso por el gradiente se opera del siguiente modo. Se parte en $t = 0$ de una cierta configuración de peso $\mathbf{w}^0$, y se calcula la dirección de la máxima variación de la función $E(\mathbf{w})$ en $\mathbf{w}^0$, la cual vendrá dada por su gradiente en $\mathbf{w}^0$. El sentido de la máxima variación apuntará hacia una colina en la hipersuperficie que representa a la función $E(\mathbf{w})$. A continuación se modifican los parámetros $\mathbf{w}$ siguiendo el sentido opuesto al indicado por el gradiente de la función de error. De esta manera se lleva a cabo un descenso por la hipersuperficie de error, aproximándose en una cierta cantidad al valle –mínimo local–. El proceso anterior de descenso por el gradiente se itera hasta que dicho mínimo local es alcanzado.

Matemáticamente se expresaría

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

donde ε (que puede ser diferente para cada paso) indica el tamaño del paso tomado en cada iteración. Si bien idealmente ε debería de ser infinitesimal, una elección de

este tipo llevaría a un proceso de entrenamiento extremadamente lento, de ahí que ε debe ser lo suficientemente grande como para que cumpla el compromiso de rápida actualización sin llevar a oscilaciones.

Teniendo en cuenta que la función de error de la red neuronal, o función de coste a minimizar, si contemplamos el problema como un problema de optimización, es de la manera siguiente:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{r=1}^{N}\sum_{i=1}^{m}(c_i^r - y_i^r)^2$$

Es decir, la función de error mide el error cuadrático correspondiente a las salidas actuales de la red respecto de los objetivos y calculando el gradiente de dicha función con respecto de cada variable (peso) w_{ij}, y teniendo en cuenta que $y_i^r = \sum_{j=1}^{n} w_{ij}x_j^r - \theta_i$, se obtiene:

$$\frac{\partial E(w_{ij})}{\partial w_{ij}} = -\left(\frac{1}{2}\right)2\sum_{r=1}^{N}(c_i^r - y_i^r)\frac{dy_i^r}{dw_{ij}} = -\sum_{r=1}^{N}(c_i^r - y_i^r)\,x_j^r$$

De ahí que el incremento de los pesos en la Adalina, según la regla de adaptación LMS resulte ser:

$$\Delta w_{ij} = -\varepsilon\frac{\partial E(w_{ij})}{\partial w_{ij}} = \varepsilon\sum_{r=1}^{N}(c_i^r - y_i^r)\,x_j^r$$

Conviene resaltar que mientras que en la regla del perceptrón simple se llevan a cabo actualizaciones discretas en los pesos, en la adalina la regla LMS produce actualizaciones de tipo continuo, de manera que un mayor error produce una actualización mayor. Otra diferencia entre ambos, derivada de la naturaleza del tipo de entradas, es que la regla del perceptrón converge en un número finito de iteraciones –si es capaz de clasificar correctamente todos los patrones–, mientras que la regla LMS se acerca asintóticamente a una solución, ya que el tamaño de los incrementos se hace cada vez menor.

8.6 El Perceptrón Multicapa

En la sección 8.4 hemos visto las limitaciones del perceptrón simple, ya que con él tan sólo podemos discriminar patrones que pueden ser separados por un hiperplano –una recta en el caso de dos neuronas de entrada–. Una manera de solventar estas limitaciones del perceptrón simple es por medio de la inclusión de capas ocultas, obteniendo de esta forma una red neuronal que se denomina perceptrón multicapa.

La Figura 11 muestra las regiones de decisión que se obtienen para distintas arquitecturas de redes neuronales considerando dos neuronas en la capa inicial. Así por ejemplo para una arquitectura de perceptrón simple la región de decisión es una recta, mientras que el perceptrón multicapa con una única capa de neuronas ocultas puede discriminar regiones convexas. Por otra parte el perceptrón multicapa con dos capas de neuronas ocultas es capaz de discriminar regiones de forma arbitraria.

El perceptrón multicapa o MLP (*Multi-Layer Perceptron*) se suele entrenar por medio de un algoritmo de retropropagación de errores o BP (*Back Propagation*) de ahí que dicha arquitectura se conozca también bajo el nombre de red de retropropagación.

El desarrollo del algoritmo BP resulta una curiosa historia de redescubrimientos y olvidos. Si bien fue Werboz (1974) quien en su tesis doctoral lo introdujo por vez

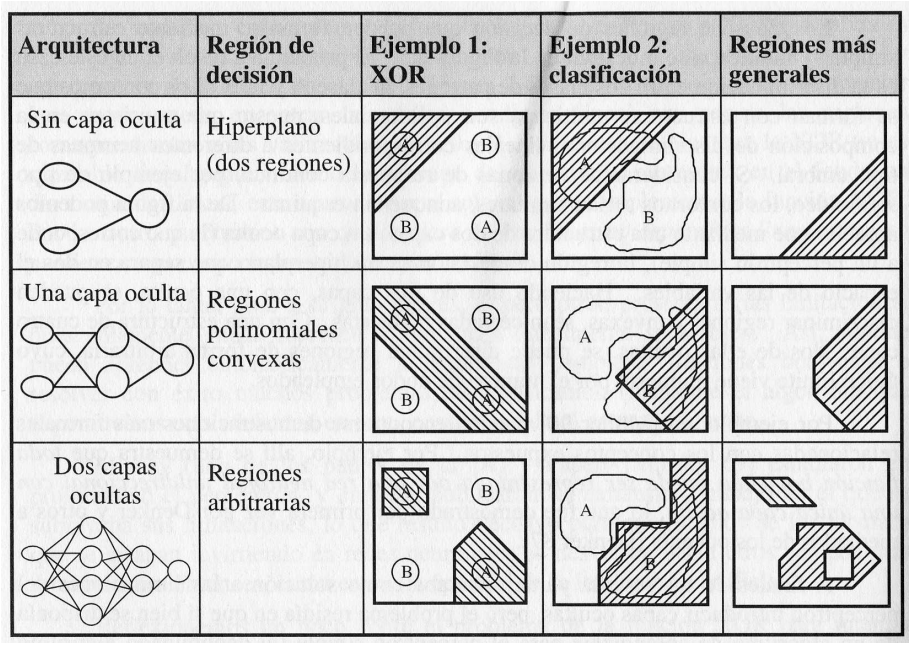

Figura 11: Regiones de decisión obtenidas para el perceptrón simple (arriba), el perceptrón multicapa con una capa oculta (en medio) y el perceptrón multicapa cn dos capas ocultas (abajo)

primera, el hecho no tuvo repercusión en su época hasta que Rumelhart y col. (1986) lo redescubrieron de manera independiente y comenzaron a popularizarlo ayudados por los avances en computación existentes en la época, los cuales permitían satisfacer los requisitos de computación que el algoritmo BP requiere.

La estructura del MLP con una única capa oculta se muestra en las Figuras 12 y 13.

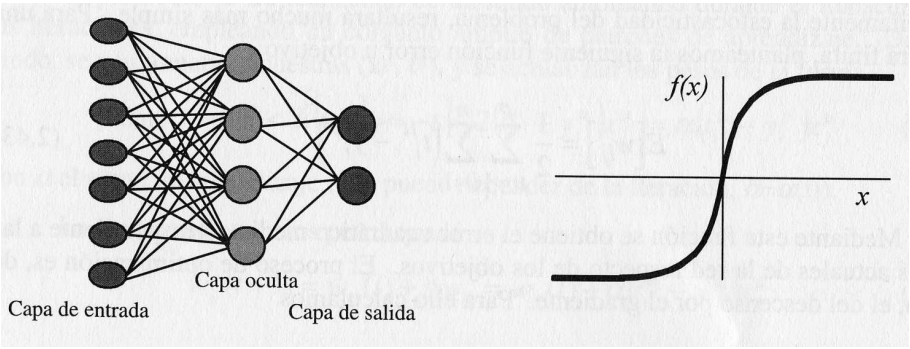

Figura 12: Arquitectura (izquierda) y función de activación (derecha) para el perceptrón multicapa

Denotamos por x_i a las n entradas de la red, y_j a las o salidas de la capa oculta y z_k a las s salidas de la capa final –por tanto a las salidas de la red– las cuales deben de ser comparadas con las salidas objetivo c_k. Además, w_{ij} representarán los pesos de la capa oculta, θ_j sus umbrales correspondientes, w'_{kj} los pesos de la capa de salida y θ'_k sus umbrales respectivos

Las operaciones efectuadas por un MLP con una única capa oculta y con funciones de activación para la capa oculta y capa final de tipo sigmoide y lineal respectivamente,

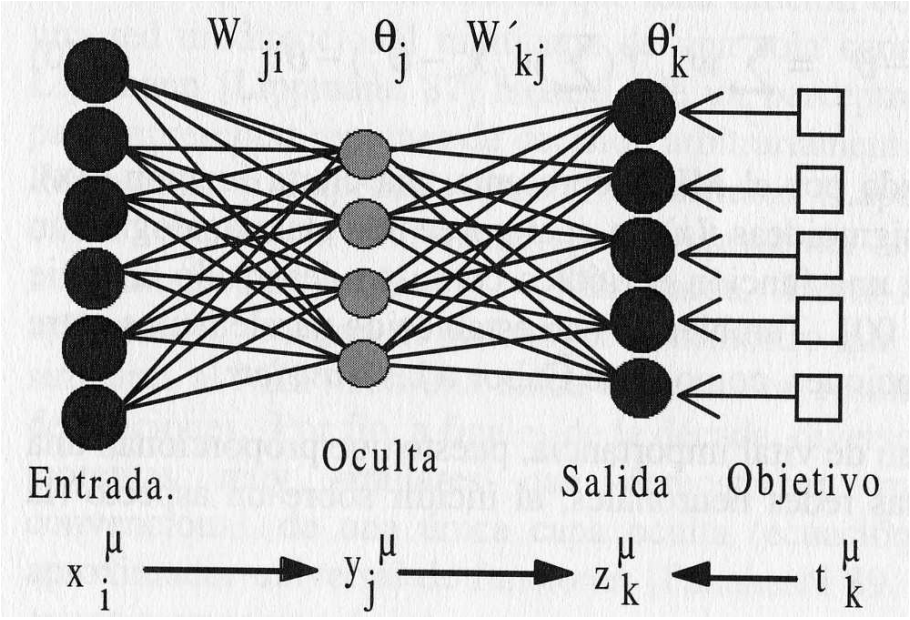

Figura 13: Arquitectura del perceptrón multicapa

son las siguientes:

$$z_k = \sum_{j=1}^{o} w'_{kj} y_j - \theta'_k = \sum_{j=1}^{o} w'_{kj} f\left(\sum_{i=1}^{n} w_{ji} x_i - \theta_j\right) - \theta'_k$$

siendo $f(x)$ una función de tipo sigmoideo.

La popularidad de la arquitectura MLP se debe al hecho de que un MLP con una única capa oculta puede aproximar cualquier función continua en un intervalo hasta el nivel deseado, cuestión demostrada por Funahaski (1989) y que proporciona una base sólida al campo de las redes neuronales, aunque el resultado no informa sobre el número de nodos ocultos necesarios para llevar a cabo la aproximación.

Veamos a continuación cómo entrenar un MLP con una única capa de neuronas oculta por medio del algoritmo BP de retropropagación de los errores. Dicho algoritmo BP puede verse como una extensión del algoritmo LMS a las redes multicapa. Para ello se planteará una función de error similar a la utilizada para obtener la regla de actualización de pesos LMS, y se obtendrán las fórmulas correspondientes al algoritmo BP tanto en función de los pesos de la capa de salida como de los pesos de la capa oculta. Se utilizará la regla de la cadena y para ello se necesitará que las funciones de transferencia de las neuronas sean derivables.

Sea un MLP de tres capas –es decir, con una capa oculta– cuya arquitectura se presenta en la Figura 13-, con las entradas, salidas, pesos y umbrales de las neuronas definidos anteriormente. Dado un patrón de entrada $\mathbf{x}^r (r = 1, \ldots, N)$ la operación global de esta arquitectura se representa del siguiente modo, para cada una de las k con $(k = 1, \ldots, s)$ neuronas de salida:

$$z_k^r = \sum_{j=1}^{o} w'_{kj} y_j^r - \theta'_k = \sum_{j=1}^{o} w'_{kj} f\left(\sum_{i=1}^{n} w_{ji} x_i^r - \theta_j\right) - \theta'_k$$

Al igual que en el caso de la adalina, la función de costo de la que se parte es el

error cuadrático medio, $E(\mathbf{w}, \mathbf{w}', \boldsymbol{\theta}, \boldsymbol{\theta}') = \frac{1}{2}\sum_{r=1}^{N}\sum_{k=1}^{m}(c_k^r - z_k^r)^2$, siendo la función $E(\mathbf{w}, \mathbf{w}', \boldsymbol{\theta}, \boldsymbol{\theta}')$

$$\begin{array}{rl} E : \mathbb{R}^{n\times o+o\times s+o+s} & \longrightarrow \mathbb{R} \\ (w_{11},\ldots,w_{on},w'_{1s},\ldots,w'_{os},\theta_1,\ldots,\theta_o,\theta'_1,\ldots,\theta'_s) & \longrightarrow E(w_{11},\ldots,w_{on},w'_{1s},\ldots,w'_{os},\theta_1,\ldots,\theta_o,\theta'_1,\ldots,\theta'_s) \end{array}$$

La minimización se llevará a cabo por el descenso por el gradiente, existiendo en este caso un gradiente respecto de los pesos de la capa de salida $\Delta w'_{kj} = -\varepsilon\frac{\partial E}{\partial w'_{kj}}$ y otro respecto de los pesos de la capa oculta $\Delta w_{ji} = -\varepsilon\frac{\partial E}{\partial w_{ji}}$.

Las expresiones de actualización de los pesos se obtienen derivando, teniendo en cuenta las dependencias funcionales y aplicando la regla de la cadena.

$$\Delta w'_{kj} = \varepsilon\sum_{r=1}^{N}\left(c_k^r - \left(\sum_{j=1}^{o} w'_{kj}y_j^r - \theta'_k\right)\right)^2 y_j^r$$

$$\Delta w_{ji} = \varepsilon\sum_{r=1}^{N}\Delta_j^r x_i^r$$

$$\text{con } \Delta_j^r = \left(\sum_{k=1}^{s}\left(\sum_{j=1}^{o} w'_{kj}y_j^r - \theta'_k\right)w'_{kj}\right)\frac{\partial f\left(\sum_{i=1}^{n} w_{ji}x_i^r - \theta_j\right)}{\partial\left(\sum_{i=1}^{n} w_{ji}x_i^r - \theta_j\right)}$$

La actualización de los umbrales (*bias*) se realiza por medio de las anteriores expresiones, considerando que el umbral es un caso particular de un peso sináptico, cuya entrada es una constante igual a -1.

En las expresiones anteriores está implícito el concepto de propagación hacia atrás de los errores que da el nombre al algoritmo. En primer lugar se calcula la expresión $\left(c_k^r - \left(\sum_{j=1}^{o} w'_{kj}y_j^r - \theta'_k\right)\right)$ que se denomina *señal de error*, por ser proporcional al error de la salida actual de la red, con el que se calcula la actualización $\Delta w'_{kj}$ de los pesos de la capa de salida. A continuación se propagan hacia atrás los errores $\left(c_k^r - \left(\sum_{j=1}^{o} w'_{kj}y_j^r - \theta'_k\right)\right)$ a través de la sinapsis, obteniéndose las señales de error $\left(\sum_{k=1}^{s}\left(\sum_{j=1}^{o} w'_{kj}y_j^r - \theta'_k\right)w'_{kj}\right)\frac{\partial f\left(\sum_{i=1}^{n} w_{ji}x_i^r - \theta_j\right)}{\partial\left(\sum_{i=1}^{n} w_{ji}x_i^r - \theta_j\right)}$ correspondientes a las sinapsis de la capa oculta. Con estas señales de error se calcula la actualización de Δw_{ji} de las sinapis ocultas. El algoritmo puede además ser extendido a arquitecturas con más de una capa oculta siguiendo este esquema de retropropagación del error.

Según lo visto, el procedimiento para entrenar mediante el algoritmo BP una arquitectura MLP dada es el siguiente:

Paso 1. Establecer aleatoriamente los pesos y umbrales iniciales ($t := 0$).

Paso 2. Para cada patrón r del conjunto de entrenamiento

2.1 Llevar a cabo una fase de ejecución para obtener la respuesta de la red frente al patrón r-ésimo

2.2 Calcular las señales de error asociadas $\left(c_k^r - \left(\sum_{j=1}^{o} w'_{kj} y_j^r - \theta'_k\right)\right)$ y

$$\left(\sum_{k=1}^{s}\left(\sum_{j=1}^{o} w'_{kj} y_j^r - \theta'_k\right) w'_{kj}\right) \frac{\partial f\left(\sum_{i=1}^{n} w_{ji} x_i^r - \theta_j\right)}{\partial\left(\sum_{i=1}^{n} w_{ji} x_i^r - \theta_j\right)}$$

2.3 Calcular el incremento parcial de los pesos y umbrales debidos a cada patrón r.

Paso 3. Calcular el incremento total actual, extendido a todos los patrones, de los pesos $\Delta w'_{kj}$ y Δw_{ji}. Hacer lo mismo con los umbrales.

Paso 4. Actualizar pesos y umbrales.

Paso 5. Calcular el error total.
Hacer $t := t + 1$ y volver al Paso 2 si todavía el error total no es satisfactorio.

Se debe comenzar siempre con pesos iniciales aleatorios pequeños, tanto positivos como negativos. En el esquema presentado se lleva a cabo una fase de ejecución para todos y cada uno de los patrones del conjunto de entrenamiento, se calcula la variación en los pesos debida a cada patrón, se acumulan, y a continuación se efectúa la actualización de los pesos. Este esquema se acostumbra a denominar *aprendizaje por lotes* o *en batch*.

El algoritmo BP es un método de aprendizaje general que presenta como ventaja principal el hecho de que se puede aplicar a gran número de problemas distintos, proporcionando buenas soluciones con no demasiado tiempo de desarrollo. Sin embargo si se pretende afinar más y obtener una solución excelente habría que ser cuidadoso en cuestiones adicionales que se encuentran fuera del alcance de estos apuntes, tales como la determinación de una arquitectura óptima, la selección de los pesos iniciales, el preprocesamiento de los datos de entrada, la utilización de técnicas que eviten el sobreajuste, etc.

Referencias

1. S. Ramón y Cajal (1899). *Textura del Sistema Nervioso del Hombre y de los Vertebrados*, N. Moya.

2. D.E. Rumelhart, J.L. MacClelland (eds.) (1986). *Parallel Distributed Processing. Vol 1. Foundations*, MIT Press.

3. J.L. McClelland, E. Rumelhart (eds.) (1986). *Parallel Distributed Processing. Vol 2. Psychological and Biological Models*, MIT Press

4. W.S. McCulloch, W. Pitts (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**, 115-133

5. C.M. Bishop (1995). *Neural Networks for Pattern Recognition*, Oxford University Press

6. S. Haykin (1999). *Neural Networks. A Comprehensive Foundation*, Prentice-hall

7. J.C. Principe, N.R. Eulalio, W.C. Lefebvre (2000). *Neural and Adaptive Systems. Foundamentals Through Simulations*, John Wiley

8. D. Hebb (1949). *Organization of the Behaviour*, Wiley

9. F. Rosenblatt (1962). *Principles of Neurodynamics*, Spartan Books

10. M. Minsky, S. Papert (1969). *Perceptrons: An Introduction to Computational Geometry*, MIT Press

11. B. Widrow, M.E. Hoff (1960). Adaptive switching circuits. *IRE WESCON Convention Record*, **4**, 96-104

12. B. Widrow, R. Winter (1988). Neural nets for adaptive filtering and adaptive patterns recognition. *IEEE Computer*, 25-39

13. P.J. Werboz (1974). *Beyond Regression: New Tools for Prediction and Analysis in Behavioral Sciences*. Tesis Doctoral. Universidad de Harvard

14. K.I. Funahaski (1989). On the approximate realization of continuous mappings by neural networks. *Neural networks*, **2**, 183-192

15. D.E. Rumelhart, G.E. Hinton, R.J. Williams (1986). Learning representations by backpropagation errors. *Nature*, **323**, 533-536