Computación neuronal. Universidade de Santiago de Compostela, 1995 (181-218)

VII

MODELOS BÁSICOS DE REDES NEURONALES ARTIFICIALES

Carlos V. Regueiro
Universidad de A Coruña

Senén Barro; Eduardo Sánchez; Manuel Fernández-Delgado
Universidade de Santiago de Compostela

1. INTRODUCCIÓN

Las Redes Neuronales Artificiales (RNA) se están mostrando en los últimos años como un paradigma computacional válido y alternativo para tratar una serie de tareas, tales como la clasificación, reconstrucción y reconocimiento de patrones (visuales, sonoros, de radar, de ultrasonidos, etc.), mapeado y aproximación a funciones no lineales, optimización, control, etc., que son muy difíciles de implementar por métodos más convencionales. Estas tareas son de gran importancia en campos como la robótica [Torras, 1995], el procesado de imágenes [Cabello y col., 1995] o las telecomunicaciones [Díaz y col., 1995] por citar algunos.

Las principales ventajas de las RNA frente a otro tipo de estrategias se deben a su capacidad de adaptación o aprendizaje y a su paralelismo intrínseco. Las RNA no necesitan programarse, al menos en el sentido clásico del término, ya que son capaces de entrenarse en base a ejemplos y generalizar lo que han aprendido, aplicándolo a entradas para las que no han sido específicamente entrenadas. Su paralelismo intrínseco confiere a las RNA unas propiedades únicas en cuanto a resistencia frente a fallos estructurales y a ruido en las señales de entrada, a la vez que una muy alta capacidad de computación, al menos si se hace explícito dicho paralelismo en el proceso de síntesis.

Sin embargo, el desarrollo de las RNA topa con importantes escollos. Uno de los más importantes es la falta de unas bases matemáticas sólidas y amplias, capaces de soportar una teoría completa y global que profundice en la síntesis de las RNA. Por otra parte, y debido precisamente a tratarse de dispositivos intrínsecamente paralelos, con un elevado número de elementos profusamente conectados entre sí, su implementación *ad hoc* con las tecnologías actuales no es sencilla [Valderrama y Carrabina, 1995; Cabestany y col., 1995], mientras que su proyección sobre sistemas de propósito más general demanda un alto coste computacional.

En este trabajo se intentarán recoger, siguiendo una descripción o definición genérica de RNA, algunos de los modelos más difundidos que han aparecido en la literatura (otro tipo de revisiones se pueden ver en Wassermann [1989], Pao [1989], Hecht-Nielsen [1990], Lippmann [1987], Hush y Horne [1992a, 1992b, 1993]), aunque, debido a la gran cantidad y diversidad de arquitecturas de RNA que se han definido, muchas se escapan de dicha generalización.

Otro aspecto que trataremos de poner de manifiesto es que muchas de las operaciones asociadas a los modelos clásicos de RNA son productos matriz-vector y funciones aplicadas a vectores. De hecho, este uso intensivo de vectores hace que su proyección sobre arquitecturas de computación vectoriales alcance un rendimiento muy elevado, lo que favorece su utilización tanto para la simulación de nuevos modelos y algoritmos como para la ejecución eficiente de los ya existentes [Sánchez y col., 1994, 1995].

Este capítulo se ha estructurado en tres bloques principales. En el primero de ellos se tratarán de explicar las bases de las RNA, estableciendo los diferentes módulos o bloques constituyentes que componen una red básica (figura 1).

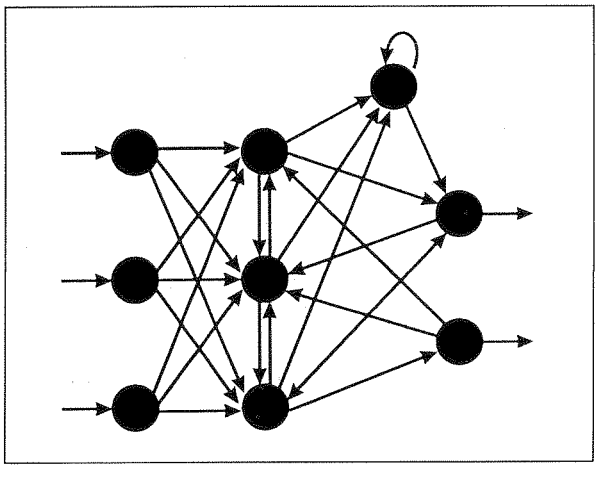

Figura 1. Ejemplo de topología de una Red Neuronal Artificial (RNA).

El segundo apartado se refiere exclusivamente a algoritmos de aprendizaje o entrenamiento. Realmente este apartado podría unirse al anterior pero debido a su especial interés se ha tratado como un bloque diferente.

Por último, se indicará cómo la mayor parte de los modelos de RNA actuales se pueden construir a partir de los módulos definidos en las dos primeras secciones. Finalmente se resumirán algunas conclusiones.

2. REDES NEURONALES ARTIFICIALES (RNA)

2.1. Introducción

Una definición general de RNA es la de un sistema de procesamiento de información compuesto por un gran número de Elementos de Procesamiento (EP), profusamente conectados entre sí a través de canales de comunicación normalmente unidireccionales, que operan sobre información local (interna y externa). Sin duda ésta es una definición muy vaga, ya que abarca no sólo a las RNA sino también a los computadores masivamente paralelos, por ejemplo.

Una definición más específica y propia, sin embargo, debe de hacer hincapié en aspectos como la motivación, orientación y autoorganización, entre otros.

La motivación de su diseño distingue a las RNA de otras técnicas computacionales. Las RNA son dispositivos de procesamiento, tanto un algoritmo como un dispositivo real, cuya realización ha sido motivada, aunque sea de lejos, por el diseño y funcionalidad del cerebro humano y los componentes del mismo. Esta es su principal razón de ser y la idea subyacente que da coherencia al campo que se da en denominar RNA. Esta es también la causa de que mucha de la terminología usada en RNA provenga en realidad de las ciencias de lo natural.

Otra característica importante de las RNA es su alto grado de paralelismo. Este paralelismo se asocia a una computación local por parte de cada EP, independiente del resto de los EP de la red. De todo ello se deduce que las RNA son altamente resistentes al ruido y muy robustas frente a fallos estructurales: la eliminación o mal funcionamiento de un porcentaje importante de unidades no provoca un «colapso», sino una disminución progresiva en el rendimiento de la red, características que, desde luego, posee el cerebro humano. Sin embargo, la característica más valorada de las RNA es su capacidad de «aprendizaje». En este sentido, no obstante, cabría pensar que las RNA modificasen de un modo autónomo su estructura en función de las entradas a la red a lo largo del tiempo, es decir, de los estímulos recibidos de su entorno. Sin embargo, en la mayoría de los modelos esta reestructuración se restringe a los pesos de las conexiones entre los elementos de la red, y se produce en base a un determinado algoritmo o regla ajeno a la propia RNA.

No es el objetivo de este trabajo, sin embargo, analizar o criticar las características, o, mejor dicho, la falta de unas características anatomo-fisiológicas más sólidas en la inmensa mayoría de modelos de RNA en uso (véase para ello el trabajo de Mira y Delgado [1995]). Por el contrario, nos limitaremos a analizar, intentando generalizar, los modelos más utilizados en las aplicaciones actuales de RNA.

En las siguientes secciones se definen y explican los elementos que componen una RNA: Elementos de Procesamiento (EP) o neuronas, y la topología u organización de todo el conjunto. Algunos autores [López y col., 1989] apuntan la existencia de niveles superiores de organización: sistemas de RNA en donde se combinan varios modelos de redes, y modelos de funcionamiento global, pero, en ambos casos, se trata de planteamientos todavía no bien estudiados.

2.2. Elementos de Procesamiento

Son los encargados de realizar las operaciones de la red. Según las entradas que tienen en cada momento determinan la salida correspondiente, también denominada estado de activación de la neurona. Como se puede ver en la figura 2, constan de tres partes o módulos diferenciados: las unidades de comunicación o conexiones, la función de combinación, f_j, y la función de activación, g_j.

A grandes rasgos, el procesamiento que realizan las neuronas o EP es el siguiente: las entradas son recibidas a través de las conexiones, en donde se realiza un primer tratamiento de las mismas. Posteriormente, la función de combinación indica cómo deben unirse o combinarse todas esas señales para obtener lo que llamamos potencial «postsináptico». Finalmente, la función de activación establece el estado o salida de la neurona a partir de ese potencial postsináptico. En las secciones siguientes estudiaremos con más detalle cada uno de estos componentes.

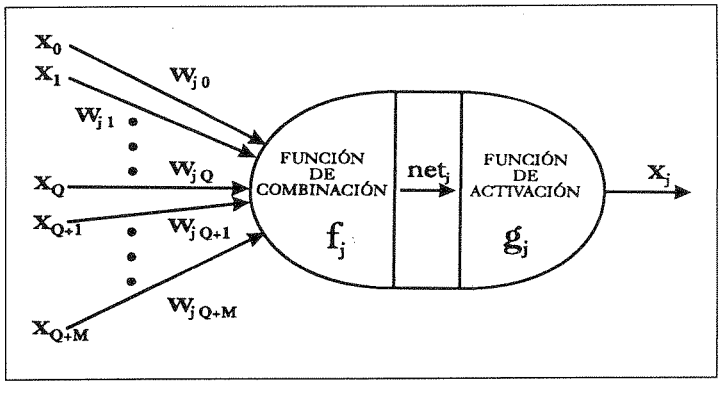

Figura 2: Neurona o Elemento de Procesamiento.

2.2.1. Conexiones

Son las vías o caminos de comunicación entre los diferentes EP, y entre éstos y las entradas del sistema. Su misión consiste en transmitir a otro EP bien la salida o estado de activación de un EP bien una entrada de la RNA. Las conexiones se establecen únicamente entre dos elementos y sólo pueden transmitir información en un sentido, es decir, son unidireccionales (aunque existen excepciones). A la forma en que están dispuestos los EP en cuanto a su esquema de conexionado se le suele conocer como **topología**.

Como hemos mencionado, podemos entender que las conexiones no se limitan exclusivamente a transmitir información entre EP, sino que también la procesan, aunque en la mayoría de los modelos sólo de un modo muy sencillo y restringido. Normalmente cada conexión lleva asociado un peso o ponderación w_{ji}, donde el índice i indica de que EP parte la conexión y j a cual llega. Podría pensarse, así, que el EP receptor no recibe exactamente la señal que ha enviado el emisor, sino que ésta es ponderada por el peso asociado a la conexión por la que se transmite (que pertenece al EP receptor).

La conexión w_{ji} se asocia al EP j-ésimo. Los pesos asociados a cada una de las conexiones de un EP se pueden representar y manipular como un único vector, con tantas componentes como número de entradas. Del mismo modo, el conjunto de todas las conexiones en una RNA se puede representar a través de matrices, aunque en el caso más general serán tensores de grado 2g+1 (esta cuestión la trataremos posteriormente con mayor profundidad).

2.2.2. Función de Combinación

Calcula el potencial postsináptico de la unidad o neurona j-ésima, net_j, en función del valor de las entradas de la RNA, $I = \{i_1, ..., i_M\}$, y del valor de las salidas o estados de activación de las otras neuronas que componen la red, x_q, en el instante actual o en otro anterior. La función de combinación puede ser diferente para cada EP. Su expresión general es

$$net_j(t+\Delta t) = f_j(x_0(t' \leq t), x_1(t' \leq t), ..., x_Q(t' \leq t), i_1(t' \leq t), ..., i_M(t' \leq t)) = f_j(X(t' \leq t)) \quad (1)$$

donde Q es el número total de neuronas o EP de la red y x_0 es una neurona especial que siempre está activa, es decir, su salida siempre es 1. El resultado será un vector de Q componentes: $Net = \{net_1, ..., net_Q\}$, una por cada EP de la RNA. En adelante nombraremos los vectores con letras mayúsculas y sus componentes con minúsculas.

Para simplificar la notación y las ecuaciones hemos reunido las salidas de los EP y las entradas de la RNA en un único vector de N+1 componentes (N = Q+M):

$$x_j = \begin{cases} 1 & j = 0 \\ x_j & j \in \{1,...,Q\} \\ i_{j-Q} & j \in \{Q+1,...,Q+M\} \end{cases} \qquad (2)$$

Existen muchas funciones de combinación definidas y, en principio, no existe ningún tipo de limitación para definir más. Sin embargo, nos restringiremos al estudio de las funciones más usuales. Por sencillez, y en la medida en que la simulación de sistemas con derivadas e integrales es muy costosa en tiempo de computación, nos restringiremos a reflejar expresiones de cálculo basadas en operaciones discretas (incrementos en lugar de derivadas, sumas en lugar de integrales, etc.), considerando el tiempo también discreto (una variable entera), de modo que los cálculos se condensan en momentos puntuales del eje de tiempo. En esta ejecución por ciclos consideraremos que un ciclo ha finalizado cuando volvemos a calcular otra vez todas las variables (potenciales y salidas de los EP, nuevos pesos durante el entrenamiento, etc.).

Las funciones de combinación más utilizadas en los modelos actuales de redes se acomodan a la siguiente expresión general:

$$net_j(t+\Delta t) = \sum_{i_1...i_g=0}^{N} \sum_{h_1...h_g=0}^{t} W_{ji_1...i_g}^{h_1...h_g}(t) \prod_{k=1}^{g} x_{i_k}(t-h_k) \qquad (3)$$

donde g es el orden de la neurona, o máximo número de multiplicandos que se permite, y los subíndices h_i recorren el eje temporal discreto. En este caso, W no sería una matriz, sino un tensor de grado 2g+1.

Net_j es una combinación lineal del producto de g entradas (de las N+1 posibles) en cualquier instante de tiempo t. Es decir, la RNA posee un matriz, X, de salidas en cada instante de tiempo, con dimensión teórica $(N+1)\cdot(t+1)$, aunque en la práctica t se substituye por una constante. Además, se permiten productos de cualesquiera g componentes de dicha matriz, con lo cual existen $(N+1)^{g+1}\cdot(t+1)^g$ posibilidades. Teniendo en cuenta la conmutatividad de la multiplicación, realmente existen $((N+g)!/((N-1)!\cdot(g+1)!))\cdot((t+g-1)!/((t-1)!\cdot g!))$ combinaciones diferentes.

La expresión general (3) se puede simplificar o particularizar de muchas formas diferentes, dando lugar a diversas funciones de combinación usuales. Veamos algunos casos.

- Si: a) $g = 1, \quad i_1 = i, \quad h_1 = h$

  b) $w^h_{ji} = 0 \qquad \forall\, h \neq 0$

En este caso obtenemos la **función de combinación lineal**,

$$net_j(t+\Delta t) = \sum_{i=0}^{N} w_{ji}(t)\, x_j(t) \tag{4}$$

Esta ecuación se corresponde con la de un hiperplano en un hiperespacio de n+1 dimensiones, que supondrá la división del espacio de entradas de la neurona en dos subespacios o regiones de decisión [Lippmann, 1987]. En la figura 3a se ve el caso de una neurona con dos 2 entradas.

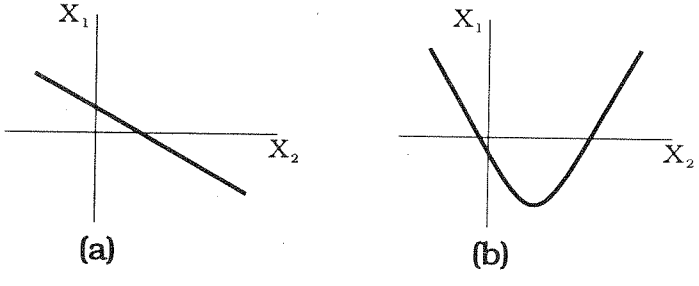

Figura 3. División de un espacio de dos entradas por una función de combinación lineal (a) y una función de combinación de orden dos (b).

- Si: a) $g = 1, \; i_1 = i, \; h_1 = h$

  b) $w^h_{ji} = 0 \qquad \forall\, h \neq \tau_{ji}$

Obtenemos así la **función de combinación lineal de entradas retardadas**, donde cada conexión w_{ji} lleva asociado un tiempo de propagación τ_{ji}

$$net_j(t+\Delta t) = \sum_{i=0}^{N} w_{ji}(t)\, x_i(t-\tau_{ji}) \tag{5}$$

La ventaja de este tipo de funciones es que permiten modelar un comportamiento secuencial en una RNA, aún en ausencia de realimentación de información entre EP.

- O, en general: a) $g = 1$, $i_1 = i$, $h_1 = h$

con lo que net_j es una combinación lineal del vector X a lo largo del tiempo, es decir,

$$net_j(t+\Delta t) = \sum_{h=0}^{t} \sum_{i=0}^{N} w^{h}_{ji}(t)\, x_i(t-h) \qquad (6)$$

O bien: a) $w^{h_1, h_2 \ldots h_g}_{j i_1 \ldots i_g} = 0 \qquad \forall\, h_k,\ k \in \{2,\ldots g\}$ y $\forall\, h_1 \neq 0$

Obtenemos entonces una **función de combinación de alto orden**, en donde el potencial postsináptico depende de las entradas y, además, de los productos cruzados de las mismas,

$$net_j(t+\Delta t) = \sum_{i_1 \ldots i_g = 0}^{N} w_{j i_1 \ldots i_g}(t) \prod_{k=1}^{g} x_{i_k}(t) \qquad (7)$$

Esta función de combinación también divide el espacio de entradas de la neurona en dos subespacios, pero con geometrías mucho más complejas que un hiperplano. Por ejemplo, una neurona de segundo orden posee regiones de decisión de tipo paraboloide (figura 3b), y, en este sentido, permite obviar ciertos problemas presentes en RNA de una capa con función de combinación lineal [Giles y Maxwell, 1987].

No todas las funciones de combinación que se han definido y utilizado se acomodan a la expresión general (3). Un ejemplo es la función **media-varianza**. Se trata de una función de combinación atípica, en la medida en que cada conexión tiene asociados dos pesos w_{ji} y v_{ji}:

$$net_j(t+\Delta t) = \left[\sum_{i=0}^{N} \left(\frac{w_{ji}(t) - x_i(t)}{v_{ji}(t)} \right)^2 \right]^{-1/2} = \left[\left(\frac{W(t) - X(t)}{V(t)} \right)^T \left(\frac{W(t) - X(t)}{V(t)} \right) \right]^{-1/2} \qquad (8)$$

donde w_{ji} es el peso o ponderación de la conexión que parte de la neurona i y acaba en la neurona j, asumiéndose que la división entre vectores $A = B/C$ se definiese como $a_i = b_i/c_i$.

En algunos casos se han definido funciones de combinación más complejas que las reseñadas aquí, por ejemplo, jerarquizando y diferenciando las entradas en grupos, al igual que ocurre en una neurona biológica. En dicho caso la contrapartida biológica de la función de combinación la realizan las propias dendritas de la neurona, mediante una integración espacio temporal de las señales que reciben [Rumelhart y McClelland, 1986].

2.2.3. Función de activación

La otra función que caracteriza una neurona o procesador elemental es la función de activación. La misión de ésta es establecer el valor de la salida o estado del EP, x_j, a partir del potencial postsináptico calculado por la función de combinación, net_j:

$$x_j(t+\Delta t) = g_j(net_j(t)) \qquad \forall\, j \in \{1,...,N\} \tag{9}$$

Recordemos que x_0 no realiza ningún tipo de computación, y que las últimas M componentes del vector X son en realidad entradas del sistema.

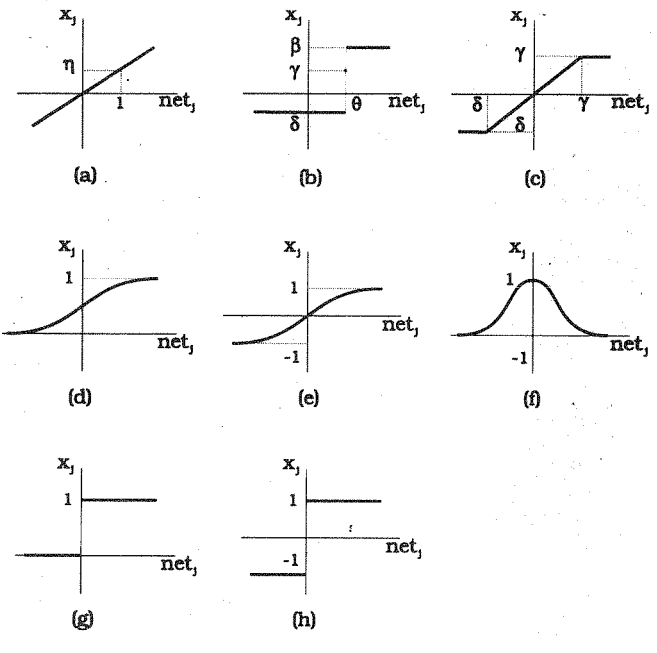

Figura 4. Gráficos de las funciones de activación clásicas (veánse los comentarios en el texto).

La elección de la función de activación establece límites en el rango de la salida de la neurona o EP. Al igual que en el caso anterior, existen muchas funciones diferentes, aunque no existe una expresión general, tal como ocurría con la función de combinación. Veamos los más usuales:

- **lineal** (figura 4a); no se realiza ningún tipo de operación sobre el potencial, salvo una ponderación por un valor constante η, por lo que, en principio, x_j podría tomar cualquier valor real ($x_j \in \mathbb{R}$)

$$x_j = \eta \, net_j \tag{10}$$

Si $\eta = 1$ la función es la identidad, usada, por ejemplo, en el perceptron [Rosenblatt, 1958; Minsky y Papert, 1969], Adaline y en la última capa de la Madaline [Widrow, 1962; Widrow y Hoff, 1960; Widrow y Lehr, 1990].

- **paso** (figura 4b);

$$x_j = \begin{cases} \beta & si \; net_j > \theta \\ \gamma & si \; net_j = \theta \\ \delta & si \; net_j < \theta \end{cases} \tag{11}$$

donde β, γ, δ y θ son constantes, normalmente iguales para todos los EP o neuronas de la RNA. Es la función utilizada en Redes de Hopfield de salidas discretas o ART-I [Carpenter y Grossberg, 1988], por ejemplo.

La función escalón (figura 4g) es una particularización de la función paso, tomando β=1, γ=δ=0 y θ=0, por lo que $x_j \in \{0,1\}$. Otra variante es la función signo (figura 4h), en donde β=1, γ=δ=-1 y θ=0, con lo que $x_j \in \{-1,+1\}$.

- **sigmoide** (figura 4d);

$$x_j = \left[1 + \exp\left(- \frac{net_j}{T} \right) \right]^{-1} \tag{12}$$

donde T es un parámetro que controla la pendiente de la función en el origen: si $T \to \infty$ la función sigmoide tiende a la función escalón (figura 4g). Con esta función, la salida del EP $x_j \in [0,1]$. Es la función más utilizada debido a que el rango de salida es continuo, aunque acotado, y es derivable. Ejemplos de modelos que utilizan esta función son Redes Perceptron Multicapa (RPM), Redes de Hopfield de salidas reales, Máquinas de Boltzmann [Ackley y col., 1985] y Cauchy [Szu, 1986], etc.

A veces se utiliza como variante de esta función la tangente hiperbólica (figura 4e):

$$x_j = \tanh(net_j) = 2\,\text{sigmoide}(net_j) - 1 = \frac{1 - e^{-net_j/T}}{1 + e^{-net_j/T}} \tag{13}$$

cuya salida, $x_j \in [-1,+1]$, mantiene una relación lineal con la función sigmoide. En este caso, cuando $T \to \infty$, la función tangente hiperbólica tiende a la función signo (figura 4h).

Una aproximación que se suele utilizar en vez de la función sigmoidal, por ser más fácil de calcular, derivable, con derivada continua, y monótonamente creciente (condiciones que se exigen a cualquier función de activación para poder aplicar un método del descenso del gradiente), es

- **rampa** (figura 4c);

$$x_j = \begin{cases} \dfrac{net_j^2}{net_j^2 + 1} & \text{si } net_j > 0 \\ 0 & \text{en otro caso} \end{cases} \tag{14}$$

$$x_j = \begin{cases} \gamma & \text{si } net_j \geq \gamma \\ net_j & \text{si } \delta < net_j < \gamma \\ \delta & \text{si } net_j \leq \delta \end{cases} \tag{15}$$

Si se eligen como parámetros $\gamma = -\delta = 1$ entonces $x_j \in [-1,+1]$ y la función rampa pasa a ser una linealización de la función $\tanh(net_j)$.

- **gaussiana** (figura 4f); utilizada con la función de combinación media-varianza en las RNA de funciones de base radial [Poggio y Girosi, 1990a; Powel, 1985], con $x_j \in [0,1]$:

$$x_j = \exp\left(-\frac{net_j^2}{2}\right) \tag{16}$$

- **máximo**; es una función muy utilizada cuando interesa obtener el máximo de un grupo de neuronas o EP,

$$x_i = \begin{cases} 1 & \text{si } \forall i \neq j \ net_j > net_i \\ 0 & \text{en otro caso} \end{cases} \qquad (17)$$

evidentemente, se pueden usar otros valores diferentes de 0 y 1 como valores de salida. El problema de esta función así definida es que requiere información no local, ya que cada EP necesita saber los potenciales de los otros EP sobre los que se está buscando el máximo. (Conviene recordar que cada EP sólo posee información de las salidas, x_i, de aquellos EP a los que está conectado, y, en principio, no puede conocer el potencial postsináptico, net_i, de los mismos, salvo que ambos coincidan, claro está).

- **probabilística**; en vez de obtener la salida del EP, en realidad se obtiene la probabilidad de que el EP esté activado (1) o no (0). Por ejemplo, se puede utilizar una distribución del tipo:

$$P[x_i = 1] = \left[1 + \exp\left(- \frac{net_i}{T_i} \right) \right]^{-1} \qquad (18)$$

donde T_i es una constante denominada «temperatura», que regula la pendiente de la curva. Cuando T tiende a cero, la curva resultante es muy parecida a la función signo: si net_i es positivo entonces la salida de la neurona, x_i, será 1 con una probabilidad 1, mientras que si net_i es negativo, la salida será 0 con toda probabilidad. La diferencia con la función signo es que, si net_i es 0, entonces la probabilidad de que la salida sea 1 ó 0 es la misma (0.5). Si T tiende a más o menos infinito tendremos la probabilidad constante P = 0.5, es decir, independientemente del valor de net_i, x_i será 1 ó 0 con igual probabilidad. Para que exista convergencia es necesario reducir la temperatura con el tiempo siguiendo, por ejemplo, la siguiente ecuación:

$$T(t) = \frac{T_0}{\ln (1+t)} \qquad (19)$$

donde T_0 es la temperatura inicial y t el tiempo.

2.3. Topología

La topología de una RNA señala la forma en que se disponen y organizan espacialmente todos los elementos de que se compone una red (EP e interconexiones). Quizás la forma más genérica de considerar las diferentes arquitecturas es suponer que las entradas de la RNA son EP identidad y que todos los EP forman un único vector, X, de dimensión N+1. Del mismo modo, podemos suponer que todas las conexiones entre EP forman una única matriz de pesos, W, de dimensión $(N+1)^2$. Esta visión se puede simplificar bastante si consideramos que muchas veces se agrupa a un determinado número de EP, en función de un interconexionado específico, en lo que denominamos **capas**.

Cada capa será pues un vector de menor dimensión que X, y tanto las conexiones que salen de ella como las que entran serán submatrices de W. Dos capas que siempre son fáciles de identificar son la de entrada, X^0, y la de salida, X^L (donde L es el número de capas), que se corresponden con las entradas (I) y salidas (O) de la RNA, respectivamente. Considerando las entradas a la RNA como una capa, ésta se entenderá compuesta por neuronas sin capacidad de procesamiento de información, ya que actúan como meras repetidoras o distribuidoras de las entradas de la RNA al resto de las capas. Sin embargo, por regla general las entradas no se contabilizan como una capa (de ahí que asumamos L capas y no L+1).

Como cada neurona o EP sólo puede pertenecer a una única capa, la concatenación de los vectores de las diferentes capas compondrá el vector X, al igual que la ordenación de las diferentes submatrices de interconexiones entre capas nos dará la matriz original W. Esto se verá mejor con alguno de los ejemplos que expondremos más adelante.

El hecho de dividir los EP en capas también implica que las actualizaciones de las salidas de los EP no se realizan simultáneamente sino en fases: una para cada capa, permitiendo que la información fluya secuencialmente, de una capa a otra, desde las entradas de la red hasta las salidas, por lo que se necesitan tantas fases como capas posea la red. Por otra parte, dentro de cada capa los EP se pueden actualizar de dos modos diferentes: o bien de forma síncrona (todos los EP recalculan su salida a la vez), o bien de forma asíncrona. En el último caso se elige aleatoriamente un EP que actualiza su salida, después se elige otro para la reactualización y así sucesivamente. La diferencia con el primer método radica en que una vez que un EP cambia su salida, los siguientes EP utilizan esa nueva salida y no la anterior, por lo tanto los resultados son distintos que con el método síncrono. Además, el resultado final depende fuertemente del orden en el que se actualizan o eligen los EP. Las redes no recurrentes utilizan, como norma habitual, un funcionamiento, o mejor dicho, una actualización síncrona de los EP de cada capa. Por otro lado, existen ejemplos de redes recurrentes para cada tipo de funcionamiento síncrono y asíncrono.

2.3.1. RNA recurrentes

Una red se denomina recurrente cuando en el flujo de datos entre las neuronas que consideramos de entrada y las que consideramos de salida existen realimentaciones (figura 5). La posibilidad de realimentación hace que la red sea mucho más potente, computacionalmente hablando, ya que le permite tener «memoria» aún en el caso de que se asuman EP sin memoria local y no se considere integración temporal. De hecho, se puede demostrar que una red neuronal recurrente operando sobre un eje temporal discreto, puede aprender a emular a cualquier autómata determinístico de número de estados finito [Siegelman y Sontag, 1991], y, en general, a cualquier máquina de Turing [Minsky, 1967; Alon y col., 1991].

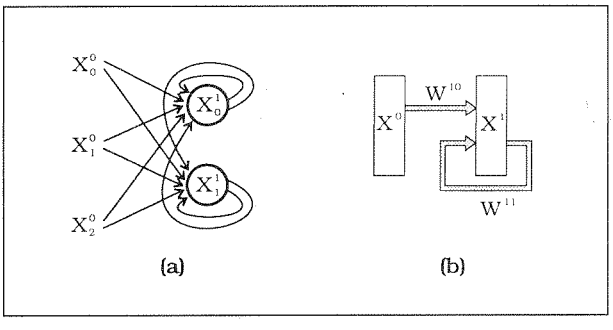

Figura 5. Ejemplo de red recurrente por elementos de procesamiento (a) y por bloques (b).

Pertenecen a este grupo las RPM Recurrentes en sus diferentes versiones [Elman, 1990; Jordan, 1986], Redes de Kohonen o Mapas de Características Autoorganizados [Kohonen, 1982, 1990], Contrapropagación [Hetch-Nielsen, 1987, 1988], etc.. Un caso particular de redes recurrentes son las Redes de Hopfield [Hopfield, 1982, 1984; Hopfield y Tank, 1985] y WTA (*'Winner Take All'* o MAXNET) [Lippmann, 1987], ya que sólo poseen una capa (aparte de la de entrada), formando parte todas las neuronas de la misma capa.

2.3.2. RNA no recurrentes

Al contrario que en el caso anterior, la información fluye siempre en un determinado sentido, sin realimentaciones, con lo que las únicas entradas a los elementos de una capa sólo pueden proceder de las salidas de las neuronas que le preceden en el sentido de avance o flujo de información. En la figura 6 se puede ver un ejemplo de red no recurrente junto a la representación por bloques de la misma que utilizaremos a partir de ahora.

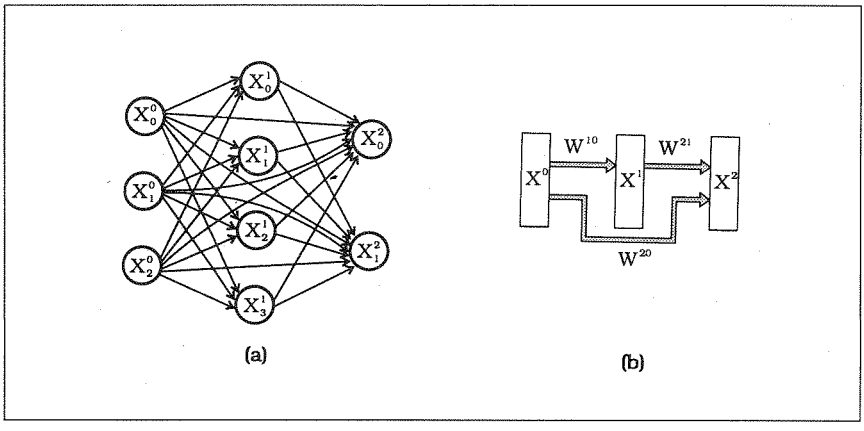

Figura 6. Ejemplo de red no recurrente por elementos de procesamiento (a) y por bloques (b).

En una representación matricial de la estructura de pesos de la RNA, en la que las capas estén ordenadas desde las entradas a las salidas, se caracterizan por poseer el triángulo superior izquierdo de la matriz W, incluida la diagonal principal, formado por ceros. La representante por excelencia de este grupo es la RPM, aunque también pertenecen a este tipo de redes la Madaline, el Cognitron, el Neocognitron [Fukushima y col., 1983] y las Máquinas de Cauchy y Boltzmann. En este grupo también se pueden integrar los primeros modelos como el Perceptron y la Adaline [Widrow y Lehr, 1990].

Un resultado importante que se puede aplicar a este tipo de redes es que un perceptron con dos capas y función de activación no lineal puede generar cualquier función con la aproximación que se desee, es decir, es un aproximador universal [Kolmogorov, 1957; Kůrková, 1992]. Para ello, sólo hay que aumentar el número de neuronas en la capa intermedia. La dificultad estriba en que la demostración no es constructiva, por lo que no se conoce a priori, o teóricamente, el número de EP ocultos ni el valor de los pesos necesarios para implementar una determinada función.

3. ENTRENAMIENTO

La principal característica de las RNA es su capacidad de aprendizaje o entrenamiento. Se entiende por entrenamiento de una RNA la variación o modificación de la estructura y/o función de la misma como consecuencia de la historia de estímulos-respuestas de la red, y con la intención de mejorar el comportamiento de

la misma. A pesar de las grandes posibilidades que se abren con esta definición, la mayoría de los modelos desarrollados hasta el momento se limitan a modificar los pesos asociados con cada una de las conexiones, y, como mucho, modifican alguno de los parámetros de la función de combinación o activación de cada EP. Bien es verdad que con la estrategia de variar los pesos, también podemos modificar indirectamente la topología de la RNA (por la eliminación de conexiones de la red).

El entrenamiento usual en las RNA consistirá pues en modificar los pesos según un determinado algoritmo de entrenamiento o aprendizaje hasta obtener la respuesta conveniente. Si los pesos son constantes en el tiempo entonces estamos ante un sistema con conexiones fijas (suelen usarse en optimización de problemas una vez calculados los pesos adecuados), o bien la red está en la fase de operación posterior a su entrenamiento.

Si los pesos varían con el tiempo entonces estamos en la fase de aprendizaje donde, en general, las modificaciones de los pesos se realizan en función de todos los valores, tanto actuales como anteriores, de los propios pesos, las entradas de la RNA y de todas las salidas de los EP (figura 7), es decir,

$$W(t+\Delta t) = \Gamma\,(W(t' \leq t),\ X(t' \leq t),\ A(t' \leq t)) \qquad (20)$$

Recordamos de nuevo que hemos incorporado las entradas de la RNA (I) en el vector X.

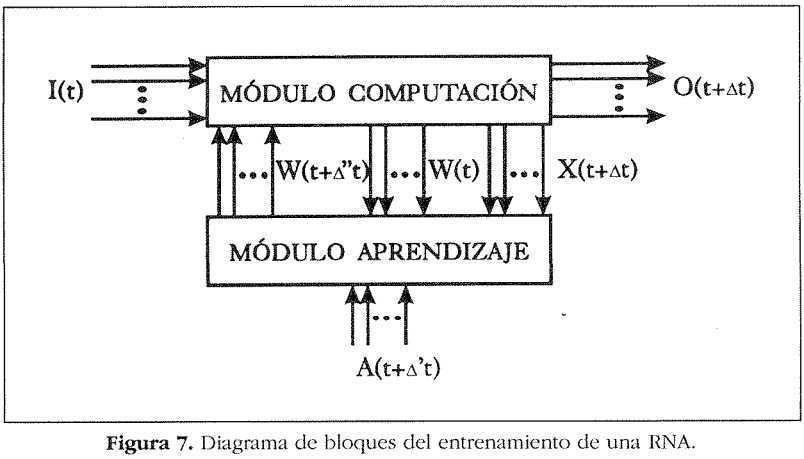

Figura 7. Diagrama de bloques del entrenamiento de una RNA.

El vector A recoge un conjunto adicional de entradas al módulo de aprendizaje, utilizadas exclusivamente por el algoritmo de entrenamiento. Por lo tanto, A sólo afecta a las salidas de la RNA de forma indirecta: al modificar la respuesta del

sistema. La naturaleza de A nos permite distinguir básicamente tres tipos de entrenamiento:

- **Supervisado**. A(t) es el vector de salida que se desea tenga el sistema frente a las entradas I(t).

- **No supervisado**. A es nulo; no existe más información sobre la evolución que sufre el sistema que las propias entradas. Es el extremo opuesto al entrenamiento supervisado.

- **Con refuerzo**. Está en un punto intermedio entre los dos anteriores. En este caso, A es un vector de entradas adicionales que permite al algoritmo de entrenamiento calcular el grado de acierto o error de la red (refuerzo), pero sin indicar los valores concretos deseables para cada salida. En algunos casos puede ocurrir que el refuerzo, r, sea una entrada única al sistema, con lo que A sólo poseerá una componente. En cualquier caso, si r se puede calcular en función de las entradas, I, entonces consideraremos que el entrenamiento es no supervisado.

Además de esta clasificación, puede pensarse en otras clasificaciones de los algoritmos de entrenamiento. Así, por ejemplo, éstos se pueden dividir en entrenamientos de tipo determinístico o estocástico, según sea el proceso de cálculo de nuevos pesos.

3.1. Entrenamiento Determinístico

Si la función $\Gamma(\cdot)$ es determinística se dice que el entrenamiento también lo es. No debe llevarnos a engaño el ver que en prácticamente todos los algoritmos que analizaremos existe cierta componente aleatoria como, por ejemplo, la inicialización de los pesos. Aún en ese caso los resultados son totalmente deterministicos si partimos de los mismos pesos iniciales y entrenamos la RNA con los mismos ejemplos de entrada y con idénticos parámetros y condiciones de partida. Es decir, los resultados que obtendremos, pesos finales de la RNA ya entrenada, serán siempre los mismos. Lo que se elige al azar es el punto de partida en el espacio W donde queremos que empiece a buscar el algoritmo la mejor combinación de pesos que resuelva el problema, pero el proceso es intrínsecamente determinístico.

Como también se ha mencionado, los algoritmos de entrenamiento se clasifican, asimismo, por la naturaleza de A. Según sea un vector que almacena la salida esperada del sistema para una entrada I (X^0), un vector nulo, o un vector con entradas adicionales (necesarias para calcular el refuerzo r), el entrenamiento será supervisado, no supervisado o con refuerzo, respectivamente.

3.1.1. Entrenamiento Determinístico No Supervisado

En este tipo de sistemas la RNA no posee ninguna información de lo bien o mal que lo está haciendo, no tiene ningún elemento que evalúe desde el exterior su rendimiento, salvo, quizás, que las nuevas entradas pueden estar condicionadas por las propias salidas o respuestas del sistema. En este caso las modificaciones se realizan única y exclusivamente en función de las entradas.

Existen diferentes «familias» de algoritmos no supervisados que siguen principios comunes con alguna modificación. Las más usadas se explican a continuación.

- **Algoritmos Hebbianos**. Se basan todos en una hipótesis formulada en la década de los 40 por Hebb [Hebb, 1949]: las conexiones que unen diferentes neuronas que se encuentran activas en un instante determinado se fortalecen. La idea es que si la actividad de una neurona excita, y, por lo tanto, activa a otra, lo normal es pensar que existe cierta relación entre ambas, con lo que su relación (conexión) se debe mantener y reforzar. Por lo tanto, w_{ij} se modificará en función de x_i y xj. La ecuación básica es

$$W_{ij}(t+\Delta t) = w_{ij}(t) + \eta\, x_j(t)\, x_i(t) \qquad (21)$$

donde η es una constante positiva que determina la velocidad de aprendizaje.

Si las salidas de los EP son binarias ($x_i \in \{0,1\}$), entonces la operación producto se transforma en la operación AND y la suma en OR. A veces se prefiere usar otro tipo de salidas, por ejemplo $x_i \in \{-1,+1\}$, aunque esta ecuación es válida para las tres posibilidades de algoritmos *hebbianos*:

- salidas y pesos ilimitados, números reales.
- pesos ilimitados pero salidas discretas (generalmente binarias).
- salidas y pesos limitados a ciertos valores discretos.

- **Algoritmos Hebbianos de Diferencias**. Al igual que el anterior, se basa en establecer correlaciones entre las entradas y salidas de una neurona, pero, en vez de hacerlo con respecto a los valores instantáneos, se establecen entre *variaciones o tendencias* de dichos valores.

- la ecuación del algoritmo básico es

$$W_{ij}(t+\Delta t) = w_{ij}(t) + \eta\, \Delta x_i(t)\, \Delta x_j(t) \qquad (22)$$

donde $\Delta x_i(t) = x_i(t) - x_i(t-\Delta' t)$.

· la ecuación de la variante impulso-refuerzo es

$$W_{ij}(t+\Delta t) = w_{ij}(t) + \Delta x_i(t) \sum_{h=1}^{H} \alpha(t-h) \left| w_{ij}(t-h) \right| \Delta x_j(t-h) \qquad (23)$$

donde la idea es extender las correlaciones de tendencias sobre el intervalo asociado a H. El nombre proviene de considerar el término $\Delta x_i(t)$ como un refuerzo y a $\Delta x_j(t-h)$ como un impulso.

- **Aprendizaje Competitivo**, se basa en reforzar aquel vector de pesos que más se «parezca» a la entrada actual. En general el «parecido» se calcula a través de una función de combinación lineal, que no es más que el producto escalar de dos vectores: el vector de pesos del EP y el vector de sus entradas. La unidad con potencial postsináptico mayor será aquella que mejor codifique la entrada en sus pesos, y, por lo tanto, la que hay que actualizar para que aún se parezca más. El proceso de encontrar el EP con el potencial postsináptico más alto se realiza, bien con una función de activación no local de cálculo de máximo, bien estableciendo una inhibición lateral entre las neuronas.

La modificación de los pesos se realiza según la siguiente ecuación

$$W_{ij}(t+\Delta t) = w_{ij}(t) + \eta(t)\, x_i(t)\, \left(x_j(t) - w_{ij}(t)\right) \qquad (24)$$

- **Aprendizaje de Kohonen** para los mapas de características auto-organizados. Es muy similar al aprendizaje competitivo y comparte su misma filosofía. La principal diferencia radica en que no sólo se modifican los pesos de la neurona de mayor potencial postsináptico, sino que también se modifican los pesos de sus EP vecinos. El factor de vecindad se puede definir en una, dos o más dimensiones, en espacios de bases canónicas o no, y suele ser un parámetro dinámico.

La ecuación que modifica los pesos es ligeramente diferente a la ecuación que utiliza el aprendizaje competitivo, ya que no tiene en cuenta directamente el valor de la salida de la neurona que se está entrenando, x_i. Para que el algoritmo funcione, la constante de proporcionalidad debe pertenecer al intervalo [0,1] y decrecer con el tiempo, por ejemplo según la razón 1/t. Otra condición que se debe verificar es que la constante de proporcionalidad o aprendizaje de la neurona j (η_{ij}) decrezca con la distancia a la neurona i ganadora, de forma que cuanto más alejada se encuentre, menos entrenamiento recibe.

$$W_{ij}(t+\Delta t) = w_{ij}(t) + \eta_{ij}(t)\, x_i(t)\, \left(x_j(t) - w_{ij}(t)\right) \qquad (25)$$

Se trata, por tanto, de una particularización del aprendizaje competitivo con $x_i(t) \in \{0, 1\}$.

3.1.2. Entrenamiento Determinístico con Refuerzo

Al contrario que con los algoritmos no supervisados, en este tipo de entrenamiento la RNA sí obtiene «ayuda» del exterior para cuantificar su rendimiento. Esta ayuda se materializa a través de un refuerzo (r) establecido por un proceso supervisor o por el propio entorno de la red, mediante el cual se valora la actuación de la RNA (básicamente el éxito o el fracaso). Otra alternativa consiste en aportar los datos necesarios para que la propia red calcule el refuerzo (en dicho caso, en vez de conocer directamente r, se conoce un conjunto de entradas, A, a partir de las cuales la red puede calcular el refuerzo: r = f(A)). Sin embargo, el refuerzo no da ninguna indicación sobre cual es o debería ser la salida. Simplemente valora la actuación conjunta o el resultado global de todas las salidas de la red. Un algoritmo de aprendizaje con refuerzo suele tener como objetivo el maximizar el refuerzo, r(t), que obtiene la RNA a lo largo del tiempo.

Por ejemplo, si el resultado es aceptable se puede indicar con un valor positivo del refuerzo (entendido, por tanto, como una «recompensa» o «premio»). Si, por el contrario, el resultado es inaceptable, el refuerzo puede tomar un valor negativo («castigo»), codificando así de una forma escalar los distintos resultados. Es decir, sabemos si la RNA lo hace más o menos bien o mal, pero no sabemos exactamente qué o cómo debería cambiar para hacerlo aún mejor. Además de ese problema de la asignación espacial del refuerzo está el problema de la asignación temporal, es decir, cómo se distribuye el refuerzo entre toda la secuencia de acciones que han dado lugar a la última salida. Este problema es aún más grave en el caso de que los refuerzos no sean inmediatos o locales (uno para cada salida), sino globales o retardados (no todas las salidas generan refuerzo).

Un entrenamiento con refuerzo modifica los pesos de los EP en función del refuerzo obtenido por la red. La principal dificultad de este procedimiento es la asignación de créditos, es decir, como repartir a cada neurona la recompensa o castigo que recibe la red en su conjunto. Este problema no existe en aquellos casos en donde el refuerzo se da a nivel de neurona.

Quizás el algoritmo más simple y conocido sea el **aprendizaje por búsqueda asociativa** (*associative search learning rule*). Este algoritmo es la contrapartida de las reglas correlacionales y su ecuación es la del aprendizaje de Hebb con el refuerzo ponderando la modificación de los pesos [Torras, 1991]. Su expresión más simple es

$$W_{ij}(t+\Delta t) = w_{ij}(t) + \eta\, x_j(t)\, x_i(t)\, r(t) \qquad (26)$$

Esta regla tiende a maximizar r(t) para cada situación. Sin embargo, posee el mismo problema que el Perceptron: sólo puede implementar funciones linealmente separables, lo cual reduce en gran medida su rango de aplicaciones.

Otros algoritmos de entrenamiento con refuerzo son la Crítica Heurística Adaptativa, AHC [Barto y col., 1983] y el de Recompensa-Penalización Asociativa, A_{RP} [Barto y Anandan, 1985]. También es muy utilizado el algoritmo de Diferencias Temporales, TD(λ) [Sutton, 1988] ya que es uno de los pocos capaces de utilizar refuerzos no continuos sino a intervalos irregulares (refuerzos globales o retardados).

3.1.3. Entrenamiento Determinístico Supervisado

El último caso que hemos contemplado es el aprendizaje supervisado. La diferencia identificativa de estos algoritmos es que sí se conoce cuál debería ser la salida correcta o deseada para los elementos de entrada, bien de forma individual o, por ejemplo, al final de una secuencia. Hay que dejar claro que cuando se dice conocer, se quiere decir saber el valor exacto de cada uno de los EP que conforman la capa de salida.

Estos algoritmos son los más utilizados. Se basan, en general, en una definición del error cometido por la red (diferencia entre las salidas calculadas por la red, $X^L = O$, y las deseadas A) a través de una expresión del tipo

$$E = \phi\,(X^L, A) \tag{27}$$

La función del algoritmo es minimizar dicho error a través de la modificación de los pesos de la RNA siguiendo métodos clásicos de minimización; por ejemplo, métodos de descenso del gradiente, en donde la modificación de cada peso es proporcional a su contribución al error global, E, del sistema:

$$\Delta w_{ij} \propto -\partial E / \partial w_{ij} \tag{28}$$

Veamos las principales variantes.

- **regla delta, LMS (*'Least Mean Square'*) o Widrow-Hoff**, típica de las Redes Adaline y Madaline. Se basa en minimizar el error definido como la distancia euclídea entre el vector de salida de la red (X) y el vector de salida deseado (A), $E = \|A - X\|^2 = 1/2 \sum_i (a_i - x_i)^2$. Es un método basado en el descenso del gradiente, es decir, los pesos se modifican siguiendo la dirección contraria al gradiente $\partial E/\partial w_{ij} = (a_i - x_i).x_j$, o sea

$$W_{ij}(t+\Delta t) = w_{ij}(t) + \eta\,(a_i(t) - x_i(t))\,x_j(t) \tag{29}$$

donde, de nuevo, x_j es una entrada y x_i una salida de la red, mientras que a_i indica la salida correcta de la salida i-ésima de la red. Al igual que en el algoritmo anterior sólo hay una capa con pesos modificables, lo cual significa que sólo puede aprender funciones linealmente separables [Minsky y Papert, 1969].

- **regla delta generalizada**, comúnmente llamada '*Backpropagation*' y muy usada en la Red Perceptron Multicapa (RPM) [Rumelhart y col., 1986; Werbos, 1974; Parker, 1985]. Es una generalización del método anterior para aplicarlo a redes con más de una capa y funciones de activación no lineales. El problema principal consiste en calcular o definir el error que cometen las neuronas de las capas ocultas. Para ese cálculo se utilizó la regla de la cadena a la hora de calcular el gradiente del error, definido igual que en el caso anterior: $E = ||A - X||^2$.

$$W^{\alpha\beta}_{ij}(t+\Delta t) = w^{\alpha\beta}_{ij}(t) + \eta \ \lambda^{\alpha}_{i}(t)\, x^{\beta}_{j}(t) \qquad (30)$$

donde los índices griegos indican la capa; β es de donde parte la conexión (en concreto de su neurona j-ésima) y α es la capa destino de la conexión, en concreto su neurona i-ésima. λ^{α}_{i} es la contribución al error final del EP^{α}_{i}. Este error se calcula siguiendo dos ecuaciones.

• Si la neurona está en la capa final, en la que conocemos los valores de activación deseados a_i, entonces

$$\lambda^{L}_{i}(t) = (a_i(t) - x^{L}_{i}(t))\, f'(net^{L}_{i}(t)) \qquad (31)$$

donde L indica la última capa, net^{L}_{i} es el potencial postsináptico de la neurona i-ésima de la última capa, y $f'(\cdot)$ es la derivada de la función de activación, normalmente una sigmoide.

Evidentemente, el error no se puede calcular a la vez que las salidas (en el mismo tiempo t). Sin embargo, si consideramos que el tiempo no es continuo sino discreto, entonces podemos suponer que sí son simultáneos; la idea va más allá, en la línea apuntada previamente de operar en ciclos y que dentro de cada ciclo los resultados y operaciones sean instantáneos. Por otra parte, comenzará un nuevo ciclo cuando necesitemos calcular nuevos valores para las mismas variables.

• Si la neurona está en una capa intermedia, λ se calcula retropropagando los errores desde la capa de salida a la de entrada según la ecuación

$$\lambda^{\alpha}_{i}(t) = f'(net^{\alpha}_{i}(t)) \sum_{\beta} \sum_{j=0}^{n^{\beta}} w^{\beta\alpha}_{ji}(t)\, \lambda^{\beta}_{j}(t) \qquad \forall\, \alpha \in \{1,...,L-1\},\ \beta \in \{\alpha+1,...,L\} \qquad (32)$$

donde n^{β} señala el número de neuronas que hay en la capa β (siempre posterior a α). Es decir, el error que comete una neurona j-ésima de una capa oculta es una combinación lineal de los errores cometidos en la capa de salida y en las capas anteriores. La ponderación de esos errores depende de lo que ha influido la neurona j-ésima en la generación de cada salida y por lo tanto de cada error.

Podemos poner todas estas ecuaciones en forma matricial

$$W^{\alpha\beta}(t+\Delta t) = W^{\alpha\beta}(t) + \eta\, \Lambda^{\alpha}(t)\, (X^{\beta}(t))^{T} \tag{33}$$

$$\Lambda^{L}(t) = (A(t) - X^{L}(t)) \odot f'(Net^{L}(t)) \tag{34}$$

donde el operador $\odot$ indica un producto de vectores componente a componente, y

$$\Lambda^{\alpha}(t) = f'(Net^{\alpha}(t)) \odot \left[\sum_{\beta} (W^{\beta\alpha}(t))^{T}\, \Lambda^{\beta}(t) \right] \qquad \forall\, \alpha \in \{1,...,L-1\},\ \beta \in \{\alpha+1,...,L\} \tag{35}$$

La regla delta generalizada también se puede aplicar en redes neuronales que posean funciones combinacionales de alto orden (con multiplicaciones). Las ecuaciones resultantes se pueden ver en Rumelhart y col. [1986].

- **regla delta generalizada a través del tiempo, (*'Backpropagation Through Time'*)**, aplicada en algunas RPM recurrentes. Las ecuaciones de propagación del error se mantienen. Ésta es la forma más intuitiva de generalizar el *'Backpropagation'* para entrenar redes recurrentes. Se basa en hacer tantos duplicados de la red como pasos o realimentaciones haya que realizar antes de conocer la salida correcta o deseada. El objetivo es sustituir la red recurrente por otra no recurrente, obtener las salidas, los errores y después retropropagarlos por todos y cada uno de los duplicados de la red inicial, modificando los pesos.

Así, la expresión (30) para una secuencia de H entradas quedaría de la forma siguiente:

$$W^{\alpha\beta}{}_{ij}(t+\Delta t) = w^{\alpha\beta}{}_{ij}(t) + \eta \sum_{h=0}^{H} \lambda^{\alpha}_{i}(t-h)\, x^{\beta}_{j}(t-h) \tag{36}$$

siendo H el número de realimentaciones que tienen lugar antes de conocer la salida deseada. En notación vectorial esta expresión sería de la forma:

$$W^{\alpha\beta}(t+\Delta t) = W^{\alpha\beta}(t) + \eta \sum_{h=0}^{H} \Lambda^{\alpha}(t-h)\, (X^{\beta}(t-h))^{T} \tag{37}$$

De esta manera, cada peso cambia, con un sólo ejemplo de entrenamiento, tantas veces como duplicados realizados.

- **aprendizaje recurrente en tiempo real (*'Real-Time Recurrent Learning'*, RTRL)** de Williams y Zipser [1989a, 1989b]. Es una generalización de la regla delta

en donde el gradiente del error se calcula o aproxima en tiempo real de forma iterativa, es decir, no hace falta esperar a que acabe toda la secuencia. Vamos a suponer, sin pérdida de generalidad, que sólo tenemos una capa de unidades (además de la capa de entrada) y que sólo unas pocas unidades poseen salida al exterior. A ese conjunto de salida lo denominaremos S. Conviene tener presente que S puede variar en el tiempo. Siguiendo a Williams y Zipser [1989a, 1989b; Hush y Horne, 1993; Haykin, 1994]

$$W_{ij}(t+\Delta t) = w_{ij}(t) + \eta \sum_{l \in S} (a_l(t) - x_l(t))\, \pi_{ijl} \tag{38}$$

donde se define $p_{ijl} = \partial x_l / \partial w_{ij}$. Estas variables sirven para conocer el gradiente y se calculan iterativamente siguiendo la ecuación

$$\pi_{ijl}(t+\Delta t) = f'(net_l(t)) + \left[\delta_{il} x_j(t) + \sum_{k \in S} w_{lk}(t)\, \pi_{ijk}(t) \right] \tag{39}$$

y sabiendo que las condiciones iniciales son $\pi_{ijl}(0) = 0 \quad \forall i,j,l$. En esta ecuación, δ_{il} es la delta de Kronecker, igual a 1 si i=l y 0 en otro caso.

Este algoritmo es más eficiente que el BPTT, pero tiene el inconveniente de que es computacionalmente más costoso debido al proceso recursivo. En concreto, la complejidad de este algoritmo es $O(n^4)$, siendo n el número de unidades [Hertz y col., 1991].

Existen otros algoritmos para entrenar RPM recurrentes, entre los que podemos destacar el aprendizaje de retropropagación recurrente (*'Real-Dependent Recurrent Backpropagation'*) de Pearlmutter [1989a, 1989b] y el aprendizaje de retropropagación temporal (*'Temporal Backpropagation'*) de Wan [1990a, 1990b], así como otros métodos para calcular el gradiente del error [Almeida, 1987; Pineda, 1988a, 1988b]. Sin embargo, todos estos algoritmos están definidos a partir de funciones continuas en el tiempo y utilizan sistemas de ecuaciones en derivadas parciales fuertemente acopladas que se deben resolver por métodos numéricos. No estudiaremos estos algoritmos porque se escapan de los objetivos que nos hemos marcado en la realización de este capítulo. De todos modos, los lectores interesados pueden acudir a la bibliografía referenciada para ahondar más en el tema.

3.2. Entrenamiento Estocástico

Si la función $\Gamma(\cdot)$ es una función de probabilidad, entonces se dice que el entrenamiento es estocástico o estadístico. Casi todos los algoritmos de este tipo son supervisados. Ejemplos de este tipo son los algoritmos de aprendizaje de la Máquina

de Boltzmann y la de Cauchy, que, en realidad, son iguales, salvo que usan distintas funciones de probabilidad para calcular las modificaciones de los pesos.

El principio básico de cualquier algoritmo estadístico es que, al contrario de lo que ocurre con los métodos determinísticos, la magnitud de las modificaciones de los pesos se calcula al **azar**. Cada vez que se cambia un peso se recalculan las salidas de la red y se observa la evolución del error, definido como $E = ||X - A||^2 = 1/2 \Sigma_i (x_i - a_i)^2$ (igual que en los algoritmos determinísticos supervisados). Evidentemente, la mayoría de las modificaciones de las conexiones no mejoran el rendimiento de la RNA. El objetivo de un algoritmo estocástico es el de permitir únicamente cambios que reduzcan el error y deshacer aquellos que lo incrementen. Con objeto de evitar mínimos locales, no obstante, se permite que un porcentaje muy pequeño de «malos» cambios en los pesos permanezca, según una determinada distribución de probabilidad.

Los pasos de un algoritmo de este tipo serían los siguientes:

- Primero, aplicar un vector de entrada y calcular el vector de salida correspondiente.
- Segundo, comparar las respuestas obtenidas con las deseadas y calcular el rendimiento o error asociado a la respuesta de la red. El objetivo del entrenamiento será, evidentemente, minimizar ese error o función objetivo.
- Tercero, escoger aleatoriamente (siguiendo una distribución normal) un peso cualquiera de la red y modificarlo en una pequeña cantidad aleatoria, a determinar según el tipo de entrenamiento (Boltzmann o Cauchy).
- Cuarto, volver a calcular las salidas y el error cometido esta vez. Si éste ha disminuido, entonces se guarda la modificación y se actualiza el peso. Si, por el contrario, el error aumenta, entonces se debe calcular la probabilidad de mantener dicho cambio.
- Quinto y último, repetir los pasos anteriores hasta que se alcance la convergencia o se obtenga la precisión deseada en el funcionamiento de la RNA.

Como se ha mencionado, se pueden elegir diversas funciones de probabilidad y, en cada caso, obtendremos métodos diferentes. Sin embargo, las distribuciones usuales son dos.

- **Entrenamiento estocástico de Boltzmann**, desarrollado por Hinton y Sejnowski [1983, 1986; Ackley y col., 1985]. En este entrenamiento la probabilidad de mantener cambios que aumenten la función de energía definida como

$$E = -1/2 \sum_i \sum_j w_{ij} x_i x_j \qquad (40)$$

se elige según una distribución de Boltzmann

$$P(y) = \exp\left(-\frac{y}{kT}\right) \tag{41}$$

donde k es una constante y T es una variable denominada temperatura artificial. La misión de T es la de simular un '*annealing*', es decir, cuando la temperatura es alta se permiten muchos cambios «malos», con la idea de ir explorando grandes porciones del espacio de pesos. Según va avanzando el proceso, se reduce la temperatura y se mantienen menos modificaciones «erróneas», con lo que el sistema se va estabilizando. La razón de esa estabilización es que la modificación de un peso es aleatoria, pero sigue una distribución Gaussiana

$$P(\Delta w) = \exp\left(-\frac{\Delta w^2}{(T(t))^2}\right) \tag{42}$$

Para que el algoritmo funcione correctamente y converja a una solución aceptable o cuasi-óptima, la reducción de la temperatura debe ser logarítmica con el tiempo,

$$T(t) = \frac{T_0}{\ln(1+t)} \tag{43}$$

siendo T_0 una temperatura inicial arbitraria. El principal problema de este método es la lentitud de aprendizaje, es decir, el tiempo de convergencia es extremadamente grande.

- **Entrenamiento estocástico de Cauchy**. Es muy parecido al método de Boltzmann, salvo que la magnitud de la modificación de los pesos, Δw, se elige según una distribución de Cauchy en vez de seguir una distribución Gaussiana [Szu, 1986; Szu y Hartley, 1987]

$$P(x) = \frac{T(t)}{T(t)^2 + x^2} \tag{44}$$

Esta ecuación se puede integrar por métodos usuales y resolviendo para x obtenemos la expresión [Wassermann, 1989]

$$\Delta w = \eta\, T(t) \tan[x] \tag{45}$$

De esta forma, el método Monte Carlo se vuelve muy simple: se selecciona un número aleatorio x a partir de una función de distribución normal, sobre el intervalo $(-\pi/2,+\pi/2)$ (para acotar la tangente) y se calcula Δw (y por tanto, el nuevo valor de w) a partir de la ecuación anterior, utilizando la temperatura actual.

Con este cambio, la temperatura puede decrecer según

$$T(t) = \frac{T_0}{1 + t} \tag{46}$$

Esta dependencia de la temperatura artificial, de forma linealmente inversa al tiempo, hace que el entrenamiento de Cauchy sea mucho más rápido que el de Boltzmann.

4. RESUMEN DE MODELOS DE REDES NEURONALES ARTIFICIALES

Hasta ahora hemos hablado de los distintos bloques y procesos constituyentes de una RNA, sin hacer referencia al conjunto. En este apartado expresaremos, sin pretender ser exhaustivos, algunos de los modelos existentes, los más comunes, como combinación de los diferentes módulos que se han expuesto hasta ahora.

4.1. Redes recurrentes

Red recurrente genérica (figura 5b)
- Función de combinación, $f(\cdot)$: lineal
- Función de activación, $g(\cdot)$: paso (signo), sigmoide, tanh, rampa.
- Topología:

$$\left(\mathrm{Net}^1\right) = \left(W^{10} W^{11}\right)\begin{pmatrix} X^0 \\ X^1 \end{pmatrix} \qquad X^1 = g\left(\mathrm{Net}^1\right)$$

- Observación: la submatriz W^{11} permite conexión total entre todas las neuronas de la primera capa. Si W^{11} (o la matriz resultado de permutar el orden de los EP) posee ceros en toda su parte superior derecha, entonces se puede reducir a una RPM (no recurrente); si en la diagonal de W^{11} sólo hay ceros, entonces ningún EP posee realimentación, y si la parte superior derecha de la matriz de conexiones también son todo ceros, entonces ningún EP está conectado a otro EP de índice superior, por lo tanto, la información fluirá de los EP de menor a mayor índice. Se pueden distinguir

entre varios tipos de redes recurrentes atendiendo a la forma de la matriz W^{11}: redes de Hopfield, BAM, Kohonen, Elman y Jordan. A continuación expondremos las principales diferencias y utilidades de cada una de ellas.

- Aplicaciones: es equivalente a una máquina de Turing y, en particular, puede emular cualquier autómata determinístico de estados finitos. [Williams y Zipser, 1989a; Alon y col., 1990]

Red de Hopfield (figura 5b) [Hopfield, 1982, 1984; Hopfield y Tank, 1985]

- Topología:

$$\left(\mathrm{Net}^1\right)=\left(W^{10}W^{11}\right)\begin{pmatrix}X^0\\X^1\end{pmatrix} \qquad X^1=g\left(\mathrm{Net}^1\right) \quad \text{con} \quad \begin{aligned} x_j^1 &= x_j = o_j && \forall j\in\{1,\ldots,Q\}\\ x_j^0 &= x_j = i_{j-Q} && \forall j\in\{Q+1,\ldots,Q+M\}\\ x_0^0 &= 0 \end{aligned}$$

- Observaciones: $w_{ij}^{1,0} = 0$, $\forall\, i \neq j$. La actualización de los EP debe ser asíncrona para evitar oscilaciones.
- Algoritmo de entrenamiento: *hebbiano* o ninguno ('*hard-wired*')
- Aplicaciones: reconstrucción y asociación de patrones, optimización, eliminación de ruido, mejora de contraste, etc.

Mapa autoorganizado de Kohonen (figura 5b) [Kohonen, 1982, 1990]

- Función de activación, $g(\cdot)$:
  - + máximo ($W^{11} = 0$); paso, rampa, sigmoide o tanh
  - + inhibición lateral, tipo WTA: $w_{ii} = 1$ y $w_{ij} = -\varepsilon\ \forall\, i \neq j$, con $\varepsilon << 1$
  - + función de inhibición lateral de tipo sombrero mejicano
- Topología:

$$\left(\mathrm{Net}^1\right)=\left(W^{10}W^{11}\right)\begin{pmatrix}X^0\\X^1\end{pmatrix} \qquad X^1=g\left(\mathrm{Net}^1\right)$$

- Algoritmo de entrenamiento: de Kohonen
- Aplicaciones: mapeado no lineal, vectorización, etc.

Contrapropagación (figura 8) [Hecht-Nielsen, 1987, 1988]

1ª capa Red de Kohonen, *2ª capa* Red de Grossberg

- Topología:

$$\begin{pmatrix}\mathrm{Net}^1\\ \mathrm{Net}^2\end{pmatrix}=\begin{pmatrix}W^{10} & W^{11} & 0\\ 0 & W^{21} & 0\end{pmatrix}\begin{pmatrix}X^0\\X^1\\X^2\end{pmatrix} \qquad \begin{aligned} X^1 &= g_1\left(\mathrm{Net}^1\right)\\ X^2 &= g_2\left(\mathrm{Net}^2\right)\end{aligned}$$

- Aplicaciones: reconocimiento y reconstrucción de patrones, mejora de la señal, etc.

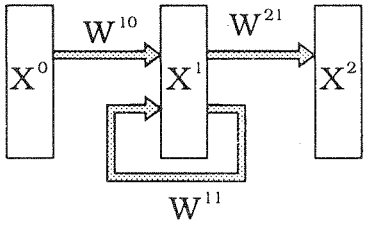

Figura 8. Red de Contrapropagación.

BAM 'Bidirectional Associative Memory' (figura 9) [Kosko, 1987]
- Topología:

$$\begin{pmatrix} Net^1 \\ Net^2 \end{pmatrix} = \begin{pmatrix} w^{10} & 0 & 0 & w^{12} \\ 0 & w^{20} & w^{21} & 0 \end{pmatrix} \begin{pmatrix} X^0{}_1 \\ X^0{}_2 \\ X^1 \\ X^2 \end{pmatrix} \qquad \begin{pmatrix} X^1 \\ X^2 \end{pmatrix} = paso \begin{pmatrix} Net^1 \\ Net^2 \end{pmatrix}$$

- Observación: cada capa posee un grupo de entradas específicas y diferenciadas. $w_{ij}^{10} = w_{ij}^{20} = 0 \ \forall \ i \neq j$.
- Algoritmo de entrenamiento: *hebbiano*
- Aplicaciones: reconstrucción y asociación de pares de patrones. Es una generalización de la red de Hopfield.

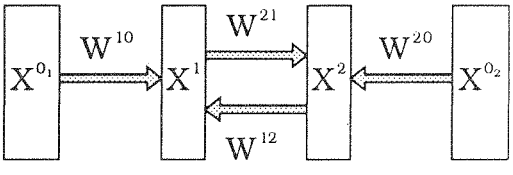

Figura 9. Memoria Asociativa Bidireccional

RPM recurrente tipo Elman (figura 10) [Elman, 1990]
- Topología:

$$\begin{pmatrix} Net^1 \\ Net^2 \\ Net^3 \end{pmatrix} = \begin{pmatrix} 0 & w^{11} & w^{12} & 0 \\ w^{20} & w^{21} & 0 & 0 \\ 0 & 0 & w^{32} & 0 \end{pmatrix} \begin{pmatrix} X^0 \\ X^1 \\ X^2 \\ X^3 \end{pmatrix} \qquad \begin{pmatrix} X^1 \\ X^2 \\ X^3 \end{pmatrix} = sigmoide \begin{pmatrix} Net^1 \\ Net^2 \\ Net^3 \end{pmatrix}$$

- Observación: $w_{ij}^{11} = w_{ij}^{12} = 0 \ \forall \ i \neq j$.

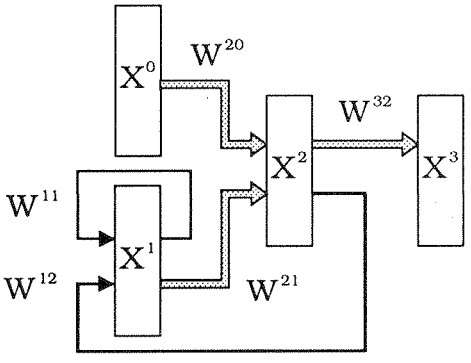

Figura 10. Red de Elman.

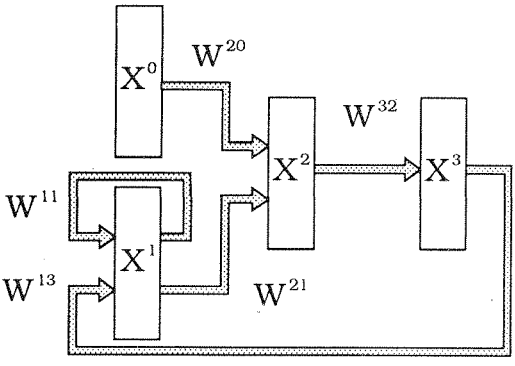

Figura 11. Red de Jordan.

RPM recurrente tipo Jordan (figura 11) [Jordan, 1986]
- Topología:

$$\begin{pmatrix} Net^1 \\ Net^2 \\ Net^3 \end{pmatrix} = \begin{pmatrix} 0 & W^{11} & 0 & W^{13} \\ W^{20} & W^{21} & 0 & 0 \\ 0 & 0 & W^{32} & 0 \end{pmatrix} \begin{pmatrix} X^0 \\ X^1 \\ X^2 \\ X^3 \end{pmatrix} \qquad \begin{pmatrix} X^1 \\ X^2 \\ X^3 \end{pmatrix} = \text{sigmoide} \begin{pmatrix} Net^1 \\ Net^2 \\ Net^3 \end{pmatrix}$$

4.2. Redes no recurrentes

Perceptron (figura 12) [Rosenblatt, 1958]
- Función de combinación: lineal
- Función de activación: paso (signo)
- Topología:

$$\left(Net^1\right) = \begin{pmatrix} W^{10} & 0 \end{pmatrix} \begin{pmatrix} X^0 \\ X^1 \end{pmatrix} \qquad \left(X^1\right) = \text{paso} \left(Net^1\right)$$

- Algoritmo de entrenamiento: delta (LMS)
- Aplicaciones: aprendizaje de funciones linealmente separables o de lógica de umbral.

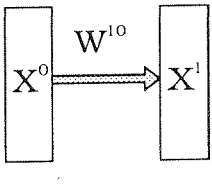

Figura 12. Red Perceptron.

Red perceptron multicapa (RPM) (figura 6b) [Rumelhart y col., 1986; Werbos, 1974; Parker, 1985]
- Función de combinación: lineal
- Función de activación: sigmoide y variantes
- Topología de una red con sólo una capa oculta (la generalización a un mayor número de
capas resulta trivial):

$$\begin{pmatrix} Net^1 \\ Net^2 \end{pmatrix} = \begin{pmatrix} W^{10} & 0 & 0 \\ W^{20} & W^{21} & 0 \end{pmatrix} \begin{pmatrix} X^0 \\ X^1 \\ X^2 \end{pmatrix} \qquad \begin{pmatrix} X^1 \\ X^2 \end{pmatrix} = \text{sigmoide} \begin{pmatrix} Net^1 \\ Net^2 \end{pmatrix}$$

- Algoritmo de entrenamiento: retropropagación, probabilístico de Boltzmann o de Cauchy.
- Aplicaciones: aproximador universal de funciones, control no lineal, codificación, reconocimiento de patrones, clasificación, etc.

RBF (Radial Basis Function) (figura 13) [Moody y Darken, 1989; Poggio y Girosi, 1990b]
- Función de combinación: *1ª capa* media-varianza (con $V_{ij}^{10} = V$, $\forall i \in X^1$ y $\forall j \in X^0$), *2ª capa* lineal
- Función de activación: *1ª capa* gaussiana, *2ª capa* identidad
- Topología:

$$\left(Net^1\right) = \left(\frac{W^{10} - X^0}{V^{10}}\right)^T \qquad \left(\frac{W^{10} - X^0}{V^{10}}\right) \qquad X^1 = \exp\left(-\frac{1}{2}\left(Net^1\right)^2\right)$$

$$\left(Net^2\right) = \begin{pmatrix} 0 & W^{21} & 0 \end{pmatrix} \begin{pmatrix} X^0 \\ X^1 \\ X^2 \end{pmatrix} \qquad \left(X^2\right) = \left(Net^2\right)$$

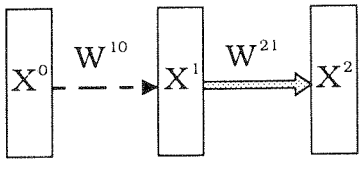

Figura 13. Red de funciones de base radial (RBF).

- Observación: la definición que usamos de división vectorial A = B/C resulta de $a_i = b_i/c_i$.
- Algoritmo de entrenamiento: *1ª capa* aprendizaje competitivo (Kohonen) o cualquiera no
supervisado, *2ª capa* LMS
- Aplicaciones: mapeo de funciones no lineales, clasificación, aproximador universal de funciones, etc.

5. CONCLUSIONES

En este trabajo hemos realizado un pequeño repaso a algunos de los modelos del nuevamente emergente campo de las Redes Neuronales Artificiales (RNA). Nuestro deseo de brevedad nos ha impedido profundizar más en algunos aspectos, y se han dejado aparte algunas RNA que nos han parecido menos importantes o demasiado específicas de ciertas aplicaciones.

A pesar de que las RNA han creado grandes expectativas, antes de poder abordar eficientemente muchas de sus potenciales aplicaciones hay que resolver algunos problemas apremiantes. Uno de ellos es el de su implementación. Lo ideal sería desarrollar auténticos «chips neuronales» y en esta línea se están dedicando numerosos esfuerzos. Pero mientras no se consiguen arquitecturas realmente eficientes, que adecuen las características de las RNA a la implementación microelectrónica, óptica, o cualquiera otra, si realmente es posible, es conveniente utilizar los actuales avances de la tecnología de los computadores para simularlas. Entre las técnicas más prometedoras están el uso de computadores paralelos, ya que las RNA son intrínsecamente paralelas, las arquitecturas sistólicas [Yáñez y col., 1991; Barro y col., 1991] y/o el uso de computadores vectoriales [Sánchez y col., 1994, 1995], debido a que la mayoría de operaciones que se realizan implican manipular vectores, tal como hemos tratado de poner de manifiesto en este trabajo.

REFERENCIAS

Ackley, D.H., Hinton, G.E. y Sejnowski, T.J., «A learning algorithm for Boltzmann machines», *Cognitive Science*, Vol. 9, N. 1, (1985), 147-169.

Almeida, L.B., «A learning rule for asynchronous perceptrons with feedback in combinatorial environment», en Caudill, M. y Butler, C. (Eds.), *IEEE First International Conference on Neural Networks*, 1987, 609-618.

Alon, N., Dewdney, A.K. y Ott, T.J., «Efficient simulation of finite automata by neural nets», *Journal of the Association of Computing Machinery*, Vol. 38, N. 2, (1991), 495-514.

Barro, S., Bugarín, A. y Yáñez, A, «Systolic implementation of Hopfield networks of arbitrary size», en Prieto, A. (Ed.), *Lecture Notes in Computer Science -540; IWANN'91*, Springer-Verlag, 1991, 268-276.

Barto A.G., Sutton, R.S. y Anderson, Ch.W., «Neuron-like adaptive elements that can solve difficult learning control problems», *IEEE Trans. on Syst., Man and Cybern.*, Vol. 13, N. 5, (1983), 834-846.

Barto, A.G. y Anandan, P., «Pattern-recognizing stochastic learning automata», *IEEE Trans. on Syst., Man and Cybern.*, Vol. 15, N. 3, (1985), 360-375.

Cabello, D., Barro, S., Mosquera, A., Penedo, M.G., Pardo, J.M. y Carreira, M.J., «Aplicaciones de redes neuronales artificiales en procesado de imágenes», *en este libro*, 1995.

Cabestany, J., Moreno, J.M. y Castillo, F., «Realización física de redes neuronales artificiales (Neurocomputadores)», *en este libro*, 1995.

Carpenter, G.A. y Grossberg, S., «The ART of adaptive pattern recognition by a self-organizing neural network», *Computer*, marzo, (1988), 77-88.

Díaz, A., García, A., Jurado, A. y Sandoval, F., «Aplicación de las redes neuronales artificiales a las telecomunicaciones», *en este libro*, 1995.

Elman, J.L., «Finding structure in time», *Computer Science*, Vol. 14, (1990), 179-211.

Fukushima, K., Miyake, S. y Ito, T., «Neocognitron: a neural network model for a mechanism of visual pattern recognition», *IEEE Trans. on Systems, Man and Cybernetics*, Vol. 13, (1983), 826-834.

Giles, C.L. y Maxell, T., «Learning, invariance, and generalization in high-order neural networks», *Applied Optics*, Vol. 26, N. 23, (1987), 4972-4978.

Haykin, S., *Neural Networks. A comprehensive foundation*, Macmillan College Publishing Company, 1995.

Hebb, D.O., *The organization of behavior*, Wiley, New York, 1949.

Hecht-Nielsen, R., «Counterpropagation networks», *Applied Optics*, Vol. 26, N. 23, (1987), 4979-4984.

Hecht-Nielsen, R., «Applications of counterpropagation networks», *Neural Networks*, Vol. 1, (1988), 131-139.

Hecht-Nielsen, R., *Neurocomputing*, Addison-Wesley Publishing Company, New York, 1990.

Hertz, J., Krough, A. y Palmer, R.G., *Introduction to the theory of neural computation*, Addison-Wesley Publishing Company, 1991.

Hinton, G.E. y Sejnowski, T.J., «Optimal Perceptual Inference», *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (1983), 448-453.

Hinton, G.E. y Sejnowski, T.J., «Learning and Relearning in Boltzmann Machines», en Rumelhart, D.E. y McClelland, J.L. (Eds.), *Parallel Distributed Processing*, Vol. 1, 1986.

Hopfield, J.J., «Neurons networks and physical systems with emergent collective computational abilities», *Proc. Natl. Acad. Sci. USA*, Vol. 79, (1982), 2554-2558.

Hopfield, J.J., «Neurons with graded response have collective computational properties like those of two-state neurons», *Proc. Natl. Acad. Sci. USA*, Vol. 81, (1984), 3088-3092.

Hopfield, J.J. y Tank, D.W., «Neural computation of decisions in optimization problems», *Biological Cybernetics*, Vol. 52, (1985), 141-152.

Hush, D.R. y Horne, B.G., «An Overview of neural networks. Part I: static networks», *Informática y Automática*, Vol. 25, N. 1, (1992a), 19-36.

Hush, D.R. y Horne, B.G., «An Overview of neural networks. Part II: dynamic networks», *Informática y Automática*, Vol. 25, N. 2, (1992b), 17-32.

Hush, D.R. y Horne, B.G., «Progress in Supervised Neural Networks», *IEEE Signal Processing Magazine*, enero, (1993), 8-39.

Jordan, M.I., «Attractor dynamics and parallelism in a connectionist sequential machine», *Cognitive Science*, (1986), 112-127.

Kohonen, T., «Self-organized formation of topologically correct feature maps», *Biological Cybernetics*, Vol. 43, (1982), 59-69.

Kohonen, T., *Self-organization and associative memory*, Springer Verlag, New York, 1988.

Kohonen, T., «The Self-Organizing Map», *Proceedings of the IEEE*, Vol. 78, N. 9, (1990), 1464-1480.

Kolomogorov, A.N., «On the representation of continuous functions of several variables by superposition of continuous functions», *Dokl. Akad. Nauk SSSR*, Vol. 114, (1957), 953-956.

Kosko, B., «Bidirectional Asociative Memories», *IEEE Trans. on Systems, Man and Cybernetics*, Vol. 16, N. 1, (1987), 49-60.

Kung, S.Y. y Hwang, J.N., «A Unified Systolic Architecture for Artificial Neural Networks», *Journal of Parallel and Distributed Computing*, Vol. 6, (1989), 358-387.

Kůrková, V., «Kolmogorov's Theorem and Multilayer Neural Networks», *Neural Networks*, Vol. 5, N. 3, (1992), 501-506.

Lippmann, R.P., «An Introduction to Computing with Neural Nets», *IEEE ASSP Magazine*, abril, (1987), 4-22.

López, F.J., Acevedo, Mª.I. y Jaramillo, M.A., «Redes neuronales. Evolución histórica», *Mundo Electrónico*, N. 197, julio, (1989), 57-71.

Minsky, M., *Computation: Finite and infinite machines*, Prentice-Hall, Englewood Cliffs, 1967.

Minsky, M. y Papert, S., *Perceptrons*, MIT Press, Cambridge, MA, 1969.

Mira, J. y Delgado, A.E., «Computación neuronal avanzada: fundamentos biológicos y aspectos metodológicos», *en este libro*, 1995.

Moody, J. y Darken, C., «Fast learning in networks of locally-tuned processing units», *Neural Computation*, Vol. 1, (1989), 281-294.

Pao, Y.H., *Adaptive Pattern Recognition and Neural networks*, Addison-Wesley Publishing Company, 1989.

Parker, D.B., «Learning logic», *Technical Report* TR-47, Center for Computational Research in Economics and Management Sciencie, MIT, (1985).

Pearlmutter, B.A., «Learning state space trajectories in recurrent neural networks», *International Joint Conference on Neural Networks*, (1989a), 365-372.

Pearlmutter, B.A., «Learning state space trajectories in recurrent neural networks», *Neural Computation*, Vol. 1, (1989b), 263-269.

Pineda, F.J., «Dynamics and architecture in neural computation», *Journal of Complexity*, Vol. 4, (1988a), 216-245.

Pineda, F.J., «Generalization of backpropagation to recurrent and higher order neural networks», en Anderson, D.Z. (Ed.), *Neural Information Processing Systems*, 1988b, 602-611.

Poggio, T. y Girosi, F., «Networks for approximation and learning», *Proceedings of the IEEE*, septiembre, (1990a), 1481-1497.

Poggio, T. y Girosi, F., «Regularization algorithms for learning that are equivalent to multilayer networks», *Science*, Vol. 247, (1990b), 978-982.

Powel, M.J.D., «Radial basis functions for multivariate interpolation: A review», *Technical Report* DAMPT 1985/NA 12, Dept. of App. Math. and Theor. Physics, Cambridge University, Cambridge, England, (1985).

Prieto, A. y Fernández, F.J., «Entornos de programación para redes neuronales artificiales», *en este libro*, 1995.

Rosenblatt, F., «The perceptron: a probabilistic model for information and organization in the brain», *Psychological Review*, Vol. 65, (1958), 386-408.

Rumelhart, D.E., Hinton, G.E. y Williams, R.J., «Learning internal representations by error propagation», en Rumelhart, D.E. y McClelland, J.L. (Eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Vol. 1, MIT Press, Cambridge, MA, 1986.

Sánchez, E., Barro, S. y Regueiro, C.V., «Artificial neural networks implementation on vectorial supercomputers», *IEEE International Conference on Neural Networks*, Orlando, FL, (1994), 3938-3943.

Sánchez, E., Barro, S., Regueiro, C.V. y Lama, M., «Supercomputación vectorial y redes neuronales artificiales», *en este libro*, 1995.

Siegelman, H. y Sontag, E.D., «Neural networks are universal computing devices», *Technical Report* SYCON-91-08, Rutgers Center for Systems and Control, (1991).

Sutton, R.S., «Learning to predict by the methods of temporal differences», *Machine Learning*, Vol. 3, N. 9, (1988), 44.

Szu, H., «Fast Simulated Annealing», en Denker, J.S. (Ed.), *Neural Networks for Computing*, 1986, 420-425.

Szu, H. y Hartley, R., «Fast simulated annealing», *Physics Letters*, Vol. 1222, N. 3/4, (1987), 157-162.

Torras, C., «Neural learning algorithms and their applications in robotics», en Babloyantz, A. (Ed.), *Self-organization, emerging properties, and learning*, Plenum Press, New York, 1991, 161-176.

Torras, C., «Aplicaciones de las redes neuronales artificiales en robótica», *en este libro*, 1995.

Valderrama, E. y Carrabina, J., «Chips neuronales», *en este libro*, 1995.

Wan, E.A., «Temporal backpropagation for FIR neural networks», *IEEE International Joint Conference on Neural Networks*, Vol. 1, (1990a), 575-580.

Wan, E.A., «Temporal backpropagation: An efficient algorithm for finite impulse response neural networks», en Touretzky, D.S., Elman, J.L., Sejnowski, T.J. y Hinton, G.E. (Eds.), *Proceedings of the 1990 Connectionist Models Summer School*, 1990b, 131-140.

Wassermann, P.D., *Neural Computing: Theory and practice*, Van Nostrand Reinhold, New York, 1989.

Werbos, P.J., *Beyond regression: new tools for prediction and analysis in the behavioral sciences*, Tesis Doctoral, Universidad de Harvard, 1974.

Widrow, B., «Generalization and information storage in networks of adaline 'neurons'», en Yovitz, M.C., Jacobi, G.T. y Goldstein, G.D. (Eds.), *Self-Organizing Systems*, Sparta, 1962, 435-461.

Widrow, B. y Hoff, M.E. Jr., «Adaptive switching circuits», *IRE WESCON Convention Record*, (1960), 96-104.

Widrow, B. y Lehr, M.A., «30 years of adaptive neural networks: Perceptron, madaline, and backpropagation», *Proceedings of the IEEE*, Vol. 78, (1990), 1415-1442.

Williams, R.J. y Zipser D., «A learning algorithm for continually running fully recurrent neural networks», *Neural Computation*, Vol. 1, (1989a), 270-280.

Williams, R.J. y Zipser D., «Experimental analysis of the Real-Time Recurrent Learning Algorithm», *Connection Science*, Vol. 1, (1989b), 87-111.

Yáñez, A., Barro, S. y Bugarín, A., «Backpropagation multilayer perceptron: a modular implementation», en Prieto, A. (Ed.), *Lecture Notes in Computer Science -540; IWANN'91*, Springer-Verlag, 1991, 285-295.

Zornetzer, S.F., Davis, J.L. y Lau, C., *An Introduction to Neural and Electronic Networks*, Academic Press, 1990.