VNIVERSITAT DE VALÈNCIA
OpenCourseWare

REDES NEURONALES ARTIFICIALES

ANTONIO J. SERRANO, EMILIO SORIA, JOSÉ D. MARTÍN

PROGRAMA 3ER CICLO (DOCTORADO)

Escola Tècnica Superior d'Enginyeria
Departament d´Enginyeria Electrònica

CURSO 2009-2010

Índice

1

Introducción a las Redes Neuronales

1.1. Introducción.

Si tuviéramos que definir la principal característica que nos separa del resto de animales seguramente, la gran mayoría de nosotros, responderíamos la capacidad de raciocinio. Esta capacidad nos ha permitido desarrollar una tecnología propia de tal manera que, en estos momentos, esta tecnología se orienta a descubrir su origen. ¿Cómo funciona el cerebro?, ¿se pueden construir modelos artificiales que lo emulen?, ¿se pueden desarrollar máquinas inteligentes?. Todas estas preguntas han conducido a un desarrollo exponencial de un campo multidisciplinar del conocimiento conocido como Inteligencia Artificial (I.A.). Este campo se podría dividir en dos clases que podríamos definir como "macroscópico" y microscópico".

En el primero de ellos se intenta modelizar el funcionamiento del cerebro basándose en reglas del tipo "si ocurre esto entonces...", el nombre de macroscópico se debe a que no se toma en cuenta en ningún momento la estructura interna del cerebro sino que modeliza su comportamiento en base a un funcionamiento que podríamos definir como global.

En la segunda aproximación se parte de la estructura que presenta el cerebro de tal forma que se construyen modelos que tienen en cuenta dicha estructura. De esta forma aparecen "neuronas artificiales" que se combinan entre sí para formar "estructuras multicapas" que, a su vez, pueden combinarse para formar "comités de expertos", etc. Esta forma de combinación recuerda la estructura en niveles del cerebro. Esta aproximación de la I.A. conocida como redes neuronales ha sufrido, en los últimos años, un incremento espectacular en publicaciones, aplicaciones comerciales, número de congresos celebrados, etc.

En este libro nos centraremos en ésta última aproximación; nuestro deseo es que teoría y práctica vayan unidos ya que pensamos que los textos existentes plantean ejemplos pero gran parte de ellos no los proporcionan. Así, en todos los temas, aparecen los fundamentos teóricos del modelo neuronal planteado así como una serie de ejemplos en los que se aplica dicho modelo. Entre estos ejemplos se encuentran los típicos problemas "juguete" así como una serie de aplicaciones reales para que se vea claramente el uso de estos sistemas.

El plan seguido en el desarrollo de este libro ha sido seguir una estructura ascendente en cuanto a las diferentes estructuras o niveles que nos encontramos en el estudio del cerebro. Se comenzará describiendo el elemento de cálculo más pequeño que nos encontramos en el cerebro, la neurona. Este apartado puede parecer de poca importancia, sin embargo, es básico para comprender el uso y la potencia de las redes neuronales. Posteriormente se explicarán estructuras que resultan de la combinación de éstas. Las estructuras con más de una neurona estudiadas serán:

- Perceptrón Multicapa. Esta estructura se aplica en problemas de modelización, clasificación y predicción.

- SOM. Estructura planteada en base a un modelo biológico. Las principales aplicaciones de esta red es el agrupamiento de datos de forma automática ("clustering") que permite la extracción de los parámetros más importantes en un problema.

- Red de Hopfield. Esta red tuvo gran importancia a nivel histórico pues supuso el resurgimiento del campo de las redes neuronales tras la dura crítica impuesta por Minsky y Papert [Minsky-69]. Los principales usos de esta red son como memoria asociativa y como herramienta para la resolución de problemas de optimización.

- Otras. Bajo este epígrafe englobaremos otras arquitecturas y algoritmos que han encontrado una amplia repercusión en los últimos tiempos como son las máquinas de vectores soporte (SVM), las estructuras OCON (One Class One Net), las redes neurodifusas, las estructuras conocidas como comités de expertos que combinan la salida de varias redes de diferente forma para obtener una salida final única,y por último las redes recurrentes.

1.2. ¿Qué son las redes neuronales?

El punto de partida en el estudio del cerebro lo podríamos fijar en pleno siglo XX con los trabajos de Santiago Ramón y Cajal uno de nuestros más grandes científicos. Fue él quien desarrolla la idea de neurona como el componente más pequeño en la estructura del cerebro. En casi todos los textos sobre redes neuronales se establece una analogía entre estos elementos y los componentes básicos de un ordenador: las puertas de silicio. En órdenes de velocidad las neuronas son de cinco a seis veces (en órdenes de magnitud) más lentas que las puertas lógicas de silicio. No obstante el cerebro suple esta menor velocidad con un mayor número de interconexiones. También hay que destacar la eficiencia del cerebro desde un punto de vista energético; a pesar del gran número de operaciones realizadas el cerebro no necesita de un ventilador como las modernas CPU's.

Si hubiera que destacar alguna característica del cerebro frente al ordenador se destacaría la alta interconexión de sus elementos constituyentes más pequeños: las neuronas. Esta capacidad de operar en paralelo le permite realizar tareas que necesitan una gran cantidad de cálculos y tiempo en potentes ordenadores. Un ejemplo cotidiano de esta característica es el reconocimiento de una cara en una fotografía; aunque se haya tomado mal y la persona esté un poco girada, la identificación de dicha persona no nos puede llevar mucho tiempo. Sin embargo, este giro puede poner en un serio aprieto a un ordenador. Otro ejemplo a destacar es el sistema de identificación de objetos de un murciélago. En su cerebro, del tamaño de un garbanzo, alberga un sistema que determina perfectamente la evolución de un obstáculo (velocidad relativa, posición, tamaño, etc) y ¡¡todo en cuestión de milisegundos!! [Suga-98]. Sobre este ejemplo sobran las analogías con los sistemas de radar y sonar creados por el hombre.

No existe una definición general de red neuronal artificial, existiendo diferentes según el texto o artículo consultado. Así nos encontramos con las siguientes definiciones:

- Una red neuronal es un modelo computacional, paralelo, compuesto de unidades procesadoras adaptativas con una alta interconexión entre ellas [Hassoun-95].

- Sistemas de procesado de la información que hacen uso de algunos de los principios que organizan la estructura del cerebro humano [Lin-96].

- Modelos matemáticos desarrollados para emular el cerebro humano [Chen-98].

- Sistema de procesado de la información que tiene características de funcionamiento comunes con las redes neuronales biológicas [Fausett-94].

- Sistema caracterizado por una red adaptativa combinada con técnicas de procesado paralelo de la información [Kung-93].

- Desde la perspectiva del reconocimiento de patrones las redes neuronales son una extensión de métodos clásicos estadísticos [Bishop-96].

Las definiciones expuestas son un botón de muestra pues cada autor las define de una manera. Parece ser que en todas ellas aparece el componente de simulación del comportamiento biológico; veremos más adelante (concretamente al tratar el

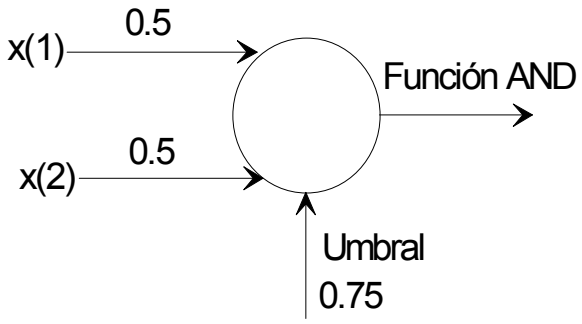
perceptrón multicapa) que no todas las redes emulan una determinada estructura neuronal. Lo que sí tienen en común estos elementos con el cerebro humano es la distribución de las operaciones a realizar en una serie de elementos básicos que, por analogía con los sistemas biológicos, se conocen como neuronas. Estos elementos están interconectados entre sí mediante una serie de conexiones que, siguiendo con la analogía biológica, se conocen como pesos sinápticos. Estos pesos varían con el tiempo mediante un proceso que se conoce como aprendizaje. Así pues podemos definir el aprendizaje de una red como el proceso por el cual modifica las conexiones entre neuronas, pesos sinápticos, para realizar la tarea deseada. Veremos más adelante los diferentes tipos de aprendizaje que existen.

1.3. Revisión histórica.

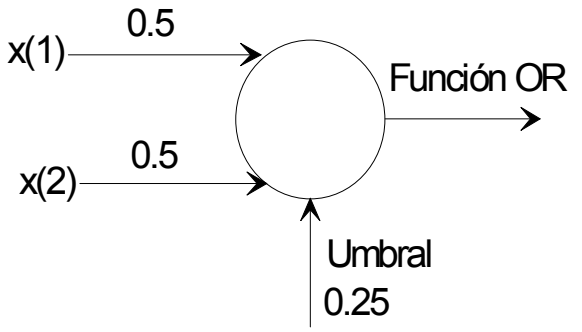
Cuando se narra la corta pero intensa historia de las redes neuronales también conocidas como modelos conexionistas se suele fijar el origen en los trabajos de McCulloch y Pitts. Sin embargo, existen trabajos anteriores que abrieron el camino a estos investigadores. Entre estos trabajos podemos destacar el realizado por Karl Lashley en los años 20. En su trabajo de 1950 se resume su investigación de 30 años; Lashley destaca que el proceso de aprendizaje es un proceso distribuido y no local a una determinada área del cerebro [Lashley-50]. Un estudiante de Lashley, D. Hebb recoge el testigo de su maestro y determina una de las reglas de aprendizaje más usadas en la regla del conexionismo y que, lógicamente, se conoce con el nombre de aprendizaje hebbiano. Las contribuciones de este investigador aparecen publicadas en su libro The Organization of the Behavior [Hebb-49]. En el capítulo 4 se da, por primera vez, una regla para la modificación de las sinapsis, es decir, una regla de aprendizaje fisiológica. Además propone que la conectividad del cerebro cambia continuamente conforme un organismo aprende cosas nuevas, creándose asociaciones neuronales con estos cambios. En su postulado de aprendizaje, Hebb sigue lo sugerido por Ramón y Cajal al afirmar que la efectividad de una sinapsis variable entre dos neuronas se incrementa por una repetida activación de una neurona sobre otra a través de esta sinapsis. Desde un punto de vista neurofisiológico la regla planteada por Hebb sería una regla variante-temporal, con un alto mecanismo interactivo que incrementa la eficacia sináptica como una función de la actividad pre y post sináptica. Desde un punto de vista conexionista la regla de Hebb es un tipo de aprendizaje no supervisado (no se necesita ningún "maestro") en el que las conexiones entre dos neuronas se incrementan si ambas se activan al mismo tiempo.

La siguiente gran contribución a considerar es el trabajo de McCulloch y Pitts. En este trabajo, se fijan las características de trabajo de lo que, posteriormente, se va a conocer como neurona de McCulloch-Pitts. Este tipo de neurona es un dispositivo binario (salida 0 ó 1), tiene un umbral de funcionamiento por debajo del cual está inactiva y puede recibir entradas excitadoras o inhibitorias cuya acción es absoluta: si existe alguna de estas entradas la neurona permanece inactiva. El modo de trabajo es simple, si no existe ninguna entrada inhibidora se determina la resultante de las entradas excitadoras y si ésta es mayor que el umbral, la salida es 1 y si no, la salida es 0 [McCulloch-43].

Se puede comprobar que este modelo puede sintetizar algunas de las funciones lógicas. En las siguientes figuras se demuestra este hecho (no se consideran entradas inhibitorias).



x(1)	x(2)	salida
0	0	0 (0<0.75)
0	1	0 (0.5<0.75)
1	0	0 (0.5<0.75)
1	1	1 (1>0.75)



x(1)	x(2)	salida
0	0	0 (0<0.25)
0	1	1 (0.5>0.25)
1	0	1 (0.5>0.25)
1	1	1 (1>0.25)

Se puede observar que, con un elemento tan simple como el que se acaba de definir, se pueden implementar un gran número de funciones lógicas mediante su combinación con elementos similares. Además, dado el estado de la neurofisiología en 1943, el modelo de McCulloch-Pitts se acercaba a lo conocido por esa época acerca de la actividad sináptica neuronal. Esta capacidad de modelizar funciones lógicas desató la euforia por estos elementos individuales; si se pueden modelizar funciones lógicas, ¿por qué no implementar un sistema de conocimiento mediante el uso de estas neuronas?. Veremos más adelante como acabó este sueño.

En 1956, Rochester, Holland, Haibt y Duda presentan un trabajo en el que, por primera vez, se verifica mediante simulaciones una teoría neuronal basada en el postulado de Hebb [Rochester-56]. Para realizar este trabajo eminentemente práctico, se tuvieron que hacer varias suposiciones que, inicialmente, no estaban en el trabajo de Hebb. Por ejemplo se acotó el valor de las sinapsis que, en principio, podía crecer sin límite.

Otro gran genio matemático, John Von Neumann, se planteó ideas conexionistas: en una recopilación de sus trabajos posterior a su muerte sugiere como posible camino para mejorar los ordenadores, de los cuales se puede considerar como uno de los padres, el estudio del sistema nervioso central. Existe un libro muy ameno en castellano que recopila estos trabajos [Von-Neumann-98].

En 1958 se producen las aportaciones de Selfridge y Rosenblatt. Estas contribuciones plantean implementaciones físicas de sistemas conexionistas. En su trabajo Selfridge plantea el sistema conocido como Pandemonium [Selfridge-58]. Este sistema consta de una serie de capas compuestas por lo que se conocen como

"demonios". Cada una de las diferentes capas de este sistema se reparten las diferentes tareas a realizar.

Por su parte, Rosenblatt, quince años después del estudio de McCulloch-Pitts, presenta una nueva aproximación al problema de reconocimiento de patrones mediante la introducción del perceptrón. Rosenblatt, planteó un dispositivo que realizara tareas que le interesaran a los psicólogos (él lo era). El hecho que fuera una máquina capaz de aprender la hacía irresistiblemente atractiva para los ingenieros [Rosenblatt-58].

En 1960 Widrow y Hoff presentan su ADALINE. Estas siglas tienen una historia curiosa: cuando las redes neuronales estaban en su máximo apogeo eran el acrónimo de Adaptive Linear Neuron; cuando las cosas empezaron a ir mal para las redes neuronales pero este sistema se seguía usando por los buenos resultados obtenidos con él se cambió a Adaptive Linear Element. El sistema planteado por Widrow estaba regido por un algoritmo de aprendizaje muy sencillo denominado LMS (Least Mean Square). Con este trabajo se propone un sistema adaptativo que puede aprender de forma más precisa y rápida que los perceptrones existentes [Widrow-60]. El trabajo de Widrow posibilitó el desarrollo de un área del procesado digital de señales (control de sistemas) que se conoce con el nombre de procesado (control) adaptativo [Haykin-96].

Block presenta en 1962 un trabajo que estudia los perceptrones más concretamente, presenta resultados sobre el perceptrón "MARK I" con 400 dispositivos receptores fotosensitivos dispuestos en una matriz 20 por 20 con un conjunto de 8 unidades de salida [Block-62]. Llegamos al trabajo de Minsky y Papert titulado Perceptrons que paralizó durante 10 años el avance de este campo de la inteligencia artificial. Este trabajo, que fue escrito y expuesto brillantemente, puso de manifiesto las limitaciones de los perceptrones. Estas limitaciones hacían referencia a la clase de problemas que se podían resolver usando estos elementos. Minsky y Papert demostraron que un perceptrón sólo podía resolver problemas linealmente separables que, para desgracia de los conexionistas, son los menos. Cuando se estudie el perceptrón se analizará con más detalle este trabajo. Además los autores expusieron, y por esto se les ha criticado, sus opiniones sobre las extensiones de los perceptrones (a sistemas multicapa); ellos plantearon su absoluta inutilidad práctica [Minsky-69]. También hay que tener en cuenta que, en el momento de la publicación de su trabajo, Minsky y Papert trabajaban en otro campo de la inteligencia artificial. Sin embargo, como se demostró más tarde, se equivocaron en sus conjeturas. El trabajo de Minsky y Papert supuso una paralización de los trabajos sobre temas conexionistas, sin embargo algunos investigadores continuaron trabajando.

Kohonen y Anderson proponen el mismo modelo de memoria asociativa de forma simultánea. A modo de demostración de los diferentes campos de conocimiento que engloban los sistemas conexionistasa estos autores tienen una formación diferente (Kohonen es ingeniero eléctrico y Anderson es neurofisiólogo). En el modelo artificial planteado la neurona es un sistema lineal que usa como regla de aprendizaje la regla de Hebb modificada[1]: estamos ante un asociador lineal [Anderson-72], [Kohonen-72].

1 El cambio en la sinapsis es proporcional al producto entre la entrada y la salida de la neurona.

En 1980, Stephen Grossberg, uno de los autores más prolíficos en el campo de las redes neuronales, establece un nuevo principio de auto-organización desarrollando las redes neuronales conocidas como ART (Adaptive Resonance Theory) [Grossberg-80]. Grossberg ha planteado diferentes modelos neuronales que han presentado una gran utilidad práctica (principalmente en el campo del reconocimiento de patrones). En 1982 J. Hopfield publica un trabajo clave para el resurgimiento de las redes neuronales. Gran parte del impacto de este trabajo se debió a la fama de Hopfield como distinguido físico teórico. En él, desarrolla la idea del uso de una función de energía para comprender la dinámica de una red neuronal recurrente con uniones sinápticas simétricas [Hopfield-82]. En este primer trabajo, Hopfield sólo permite salidas bipolares (0 ó 1, ±1). En un trabajo posterior amplía la función energía planteada para estos sistemas permitiendo la salida continua de las neuronas [Hopfield-84]. El principal uso de estas redes ha sido como memorias y como instrumento para resolver problemas de optimización como el problema del viajante [Adenso-96].

En el mismo año de 1982 Kohonen publica un importante artículo sobre mapas autoorganizativos que se ordenan de acuerdo a unas simples reglas. El aprendizaje que se da en el modelo planteado no necesita de un "maestro"; estamos ante un aprendizaje de tipo no supervisado [Kohonen-82]. Al año siguiente, en el número especial sobre modelos neuronales de la revista especializada IEEE Transactions on Systems, Man and Cybernetics, aparecen dos trabajos de gran importancia en el desarrollo de las redes neuronales. Fukushima, Miyake e Ito presentan una red neuronal, el Neocognitron, de tal forma que combinando ideas del campo de la fisiología, ingeniería y de la teoría neuronal crean un dispositivo que es capaz de ser aplicado con éxito en problemas de reconocimiento de patrones. Este trabajo, y de ahí lo de Neo, supone un perfeccionamiento de un modelo anterior presentado por los mismos autores y conocido como Cognitron. Este sistema fue probado con la tarea de identificar números escritos a mano [Fukushima-83]. El segundo trabajo, presentado por Barto, Sutton y Anderson estudia el aprendizaje reforzado y su aplicación en control. En este trabajo se plantea este nuevo tipo de aprendizaje en el que, a diferencia de trabajos anteriores sobre modelos supervisados, no es necesario un conocimiento total del error cometido por la red; lo único que se necesita es conocer el signo del error [Barto-83].

En 1983, Kirkpatrick, Gelatt y Vecchi describen un procedimiento de optimización que será posteriormente usado por Ackley, Hinton y Sejnowski en el desarrollo de un algoritmo de aprendizaje estocástico [Kirkpatrick-83]. Con este tipo de aprendizaje se busca evitar la obtención de un mínimo local en el proceso de aprendizaje de una red, problema que, como veremos más adelante, es el punto débil de la mayoría de sistemas conexionistas. Para ello se plantea un procedimiento de optimización derivado de la física estadística. El problema de este modelo es su baja velocidad de convergencia.

En 1986 aparece un trabajo que, junto al de Hopfield, resucitará el interés por las redes neuronales. En este trabajo Rumelhart, Hinton y Williams, desarrollan el algoritmo de aprendizaje de retropropagación (backpropagation) para redes neuronales multicapa dando una serie de ejemplos en los que se muestra la potencia del método desarrollado [Rumelhart-86]. Hay que destacar que, aunque este trabajo supone la publicidad mundial del procedimiento de retropropagación (lo estudiaremos al analizar el perceptrón multicapa) el trabajo de Paul Werbos lo planteaba doce años antes [Werbos-74]. A partir de ese año, el número de trabajos

sobre redes neuronales ha aumentado exponencialmente apareciendo un gran número de aportaciones tanto a los métodos de aprendizaje como a las arquitecturas y aplicaciones de las redes neuronales. Se podría destacar de entre todas estas aportaciones el trabajo de Broomhead y Lowe y el de Poggio y Girosi sobre el diseño de redes neuronales en capas usando RBF (Radial Basis Functions) [Broomhead-88], [Poggio-90], el trabajo intensivo desarrollado sobre las máquinas de vectores soporte [Vapnik-96], el desarrollo de la unión entre elementos neuronales y difusos y, por último, los trabajos sobre neuronas de pulsos (spike neurons), sobre este tema se recomienda visitar la página WEB http://diwww.epfl.ch/lami/team/gerstner/wg_pub.html. Finalmente hay que hacer mención a unos de los motores en el desarrollo de las redes neuronales en los últimos años: la predicción en series temporales. En este campo hay que destacar los trabajos de Elman [Elman-90], Jordan [Jordan-88], Robinson y Fallside [Robinson-91], Williams y Zisper [Williams-90] en la consecución de las redes recurrentes. Una generalización de las redes TDNN (orientadas especialmente a su utilización sobre series temporales) la realizó Eric Wan [Wan-93]. En su trabajo los pesos sinápticos eran filtros digitales de tipo FIR (Finite Impulse Response). Una extensión de esta idea sería considerar estos pesos como filtros de tipo IIR (Infinite Impulse Response) [Back-1995], o redes en celosía [Principe-97].

1.4. Ventajas de las redes neuronales.

Acabamos de ver el desarrollo histórico de los sistemas conexionistas; se ha comprobado que, es una ciencia multidisciplinar donde ingenieros, psicólogos, médicos, matemáticos y físicos teóricos han aportado algún elemento a estas teorías, pero, ¿por qué ese interés en estos sistemas?, ¿qué tienen en especial frente a otros que podríamos denominar clásicos?, en definitiva ¿qué cosas nuevas nos ofrecen?.

Al principio de este capítulo se ha comentado que la potencia computacional de una red neuronal deriva, principalmente, de su estructura de cálculo distribuido paralelo. Esta estructura le permite la resolución de problemas que necesitarían gran cantidad de tiempo en ordenadores "clásicos". Pero aparte de este hecho aparecen otras propiedades que las hacen especialmente atractivas para ser usadas en una gran cantidad de problemas prácticos [Haykin-96b], [Haykin-98]:

- Son sistemas distribuidos no lineales: Una neurona es un elemento no lineal por lo que una interconexión de ellas (red neuronal) también será un dispositivo no lineal. Esta propiedad permitirá la simulación de sistemas no lineales y caóticos, simulación que, con los sistemas clásicos lineales, no se puede realizar.

- Son sistemas tolerantes a fallos: Una red neuronal, al ser un sistema distribuido, permite el fallo de algunos elementos individuales (neuronas) sin alterar significativamente la respuesta total del sistema. Este hecho las hace especialmente atractivas frente a los computadoras actuales que, por lo general, son sistemas secuenciales de tal forma que un fallo en uno de sus componentes conlleva que el sistema total no funcione.

- Adaptabilidad: Una red neuronal tiene la capacidad de modificar los parámetros de los que depende su funcionamiento de acuerdo con los cambios que se produzcan en su entorno de trabajo (cambios en las entradas, presencia de ruido, etc...). Con respecto a la capacidad de adaptación hay que tener en cuenta que ésta no puede ser tampoco excesivamente grande ya que conduciría a tener un sistema inestable respondiendo a pequeñas perturbaciones. Este es el problema conocido como el dilema plasticidad-estabilidad.

- Establecen relaciones no lineales entre datos: Las redes neuronales son capaces de relacionar dos conjuntos de datos. Comparando con los métodos estadísticos clásicos que realizan la misma misión tienen como principal ventaja que los datos no tienen por qué cumplir las condiciones de linealidad, gausianidad y estacionariedad [Proakis-97]

- Posibilidad de implementación en VLSI: Esta posibilidad permite que estos sistemas puedan ser aplicados en sistemas de tiempo real, simulando sistemas biológicos mediante elementos de silicio. Uno de los científicos más prolíficos en este campo ha sido el profesor Calvin Mead [Mead-87a], [Mead-87b], [Mead-88].

Todas estas ventajas hacen el uso de las redes neuronales especialmente atractivo en un gran número de aplicaciones. Sin embargo antes de enunciar algunas (¡no todas!) de estas aplicaciones pasaremos a describir los diferentes modelos conexionistas que nos podemos encontrar.

En el campo de las redes neuronales se conoce con el nombre de arquitectura la forma en la que se unen los diferentes elementos, neuronas, mediante una serie de conexiones, pesos sinápticos. En principio podemos distinguir tres niveles, en cuanto a arquitectura se refiere, que los podemos definir como:

- Microestructura: Este nivel hace referencia al elemento más pequeño que nos podemos encontrar en un modelo conexionista: la neurona. Este es el nivel más pequeño pero no por ello es el menos importante; aquí se fijan características tan importantes como la función de activación que se explicará a continuación.

- Mesoestructura: Una vez sobrepasado el nivel neuronal llegamos a este nivel donde se fija la forma de conexión y la disposición de los elementos explicados anteriormente.

- Macroestructura: Las diferentes redes planteadas en el nivel anterior se pueden combinar entre sí para dar estructuras mayores alcanzándose mejores prestaciones.

Veamos más detenidamente todos estos niveles.

1.5. Modelos Neuronales.

En todo modelo artificial de neurona se tienen cuatro elementos básicos:

1. Un conjunto de conexiones, pesos o sinapsis que determinan el comportamiento de la neurona. Estas conexiones pueden ser excitadoras (presentan un signo positivo), o inhibidoras (conexiones negativas).

2. Un sumador que se encarga de sumar todas las entradas multiplicadas por las respectivas sinapsis.

3. Una función de activación no lineal para limitar la amplitud de la salida de la neurona.

4. Un umbral exterior que determina el umbral por encima del cual la neurona se activa.

Esquemáticamente, una neurona artificial quedaría representada por la Figura 1.1:

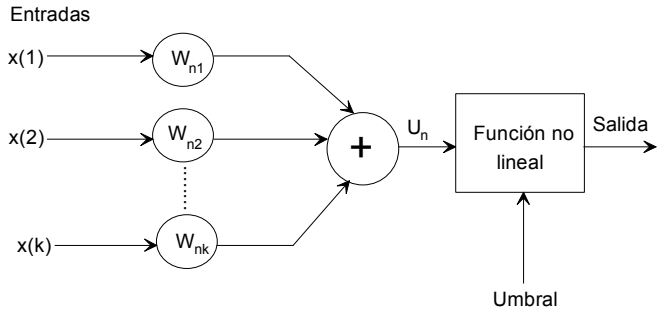


Figura 1.1. Esquema de un modelo neuronal.

Matemáticamente las operaciones a realizar serían:

$$U_n = \sum_{j=1}^{k} W_{nj}.x(j) \qquad \text{Ec. 1.1}$$

y

$$salida = \rho(U_n - umbral) \qquad \text{Ec. 1.2}$$

donde ρ es una función no lineal conocida como función de activación. Normalmente se asocia el umbral a la salida U_n mediante una entrada (que vale -1) y un peso adicional asociado. Es decir:

$$\text{umbral} = -W_{n0} \Rightarrow \begin{cases} U_n = \sum_{j=0}^{k} W_{nj}.x(j) \Rightarrow \text{salida} = \rho(U_n) \\ x(0) = 1 \end{cases} \qquad \text{Ec. 1.3}$$

El modelo descrito es el más usual, sin embargo, aparecen otros modelos que no realizan un promedio de las entradas directamente sino que, antes de multiplicar por los pesos se plantea una transformación de dichas entradas. Así, se tiene:

■ Transformación cuadrática:

$$U_n = \sum_{j=1}^{k} W_{nj} . x^2(j) \qquad \text{Ec. 1.4}$$

■ Transformación polinómica:

$$U_n = \sum_{j=1}^{k} \sum_{s=1}^{k} W_{njs} \cdot x(j) \cdot x(s) \qquad \text{Ec. 1.5}$$

■ Transformación esférica:

$$U_n = \frac{1}{\rho^2} \sum_{j=1}^{k} \left(x(j) - W_{nj} \right)^2 \qquad \text{Ec. 1.6}$$

El modelo planteado es el más común pero hay que destacar que es estático; uno más general consideraría salidas anteriores: tendríamos un modelo dinámico [Weigend-94]. La neurona definida de esta forma tendría memoria. Matemáticamente este hecho lo expresaríamos como:

$$\text{salida}_n = F(\text{salidas}_{n-k}, \text{entradas}) \quad \mathbf{k = 1,...n - 1} \qquad \text{Ec. 1.7}$$

Es decir, la salida en el instante n depende no sólo de las entradas como en el caso anterior sino que ahora aparece una dependencia con las salidas anteriores.

En cuanto a las funciones de activación existe un gran número inspiradas, todas ellas, en modelos biológicos. Algunas de estas funciones son:

■ Función signo o umbral.

$$\text{salida} = \begin{cases} \mathbf{1} & U_n \geq \mathbf{0} \\ \mathbf{0} & U_n < \mathbf{0} \end{cases} \qquad \text{Ec. 1.8}$$

Cuando una neurona usa esta función de activación se habla del modelo de McCulloch-Pitts.

■ Sigmoide.

$$salida = \frac{1}{1 + e^{(-a.U_n)}} \qquad \text{Ec. 1.9}$$

Donde a fija la pendiente de la función en el origen. Aumentando esta constante la sigmoide se asemeja a la función signo. Las funciones definidas varían entre 0 y

1; se pueden definir a partir de ellas otras funciones que varían entre -1 y 1 simplemente escalando las salidas entre éstos límites.

■ Función lineal a tramos.

$$salida = \begin{cases} 1 & U_n \geq \frac{1}{2} \\ U_n + \frac{1}{2} & -\frac{1}{2} > U_n > \frac{1}{2} \\ 0 & U_n < -\frac{1}{2} \end{cases} \qquad \text{Ec. 1.10}$$

■ Función Gaussiana.

$$salida = K_1 \cdot e^{\left(\frac{U_n - K_2}{K_3}\right)^2} \qquad \text{Ec. 1.11}$$

siendo K_i constante.

1.6. Arquitecturas neuronales.

Los elementos básicos comentados anteriormente se pueden conectar entre sí para dar lugar a las estructuras neuronales o modelos conexionistas que podríamos clasificar de diferentes formas según el criterio usado. Así se tendría:

1.6.1. Según el número de capas.

- Redes neuronales monocapas: Se corresponde con la red neuronal más sencilla ya que se tiene una capa de neuronas que proyectan las entradas a una capa de neuronas de salida donde se realizan diferentes cálculos. La capa de entrada, por no realizar ningún cálculo, no se cuenta de ahí el nombre de redes neuronales con una sola capa. Una aplicación típica de este tipo de redes es como memorias asociativas.

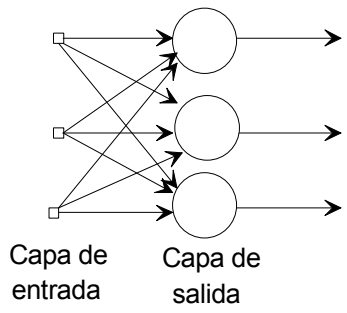


Figura 1.2. Red neuronal monocapa.

- Redes neuronales multicapa: Es una generalización de la anterior existiendo un conjunto de capas intermedias entre la entrada y la salida (capas ocultas). Este tipo de red puede estar total o parcialmente conectada.

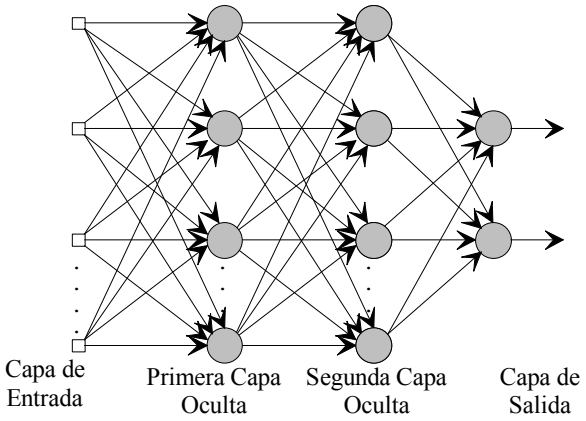


Figura 1.3. Esquema de una red neuronal multicapa.

1.6.2. Según el tipo de conexiones.

- Redes neuronales no recurrentes: En esta red la propagación de las señales se produce en un sentido solamente, no existiendo la posibilidad de realimentaciones. Lógicamente estas estructuras no tienen memoria.

- Redes neuronales recurrentes: Esta red viene caracterizada por la existencia de lazos de realimentación. Estos lazos pueden ser entre neuronas de diferentes capas, neuronas de la misma capa o, más sencillamente, entre una misma neurona. Esta estructura recurrente la hace especialmente adecuada para estudiar la dinámica de sistemas no lineales. La siguiente figura representa el esquema de una red recurrente.

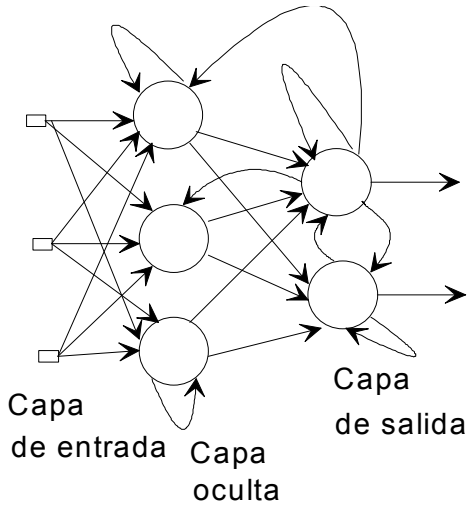


Figura 1.4. Red neuronal recurrente.

1.6.3. Según el grado de conexión.

- Redes neuronales totalmente conectadas. En este caso todas las neuronas de una capa se encuentran conectadas con las de la capa siguiente (redes no recurrentes) o con las de la anterior (redes recurrentes).

- Redes parcialmente conectadas. En este caso no se da la conexión total entre neuronas de diferentes capas.

Estas estructuras neuronales se podrían conectar entre sí para dar lugar a estructuras mayores: estamos en el nivel de la mesoestructura. Esta conexión se puede llevar a cabo de diferentes formas siendo las más usuales las estructuras en paralelo y jerárquicas. En la primera estructura se plantea un "consenso" entre las diferentes redes para obtener la salida mientras que en la estructura jerárquica existen redes subordinadas a otras que actúan como elementos centrales en la salida final de la red.

1.7. Métodos de aprendizaje.

En una red neuronal es necesario definir un procedimiento por el cual las conexiones del dispositivo varíen para proporcionar la salida deseada (algoritmo de aprendizaje). Los métodos de aprendizaje se pueden dividir en las siguientes categorías [Rojas-95]:

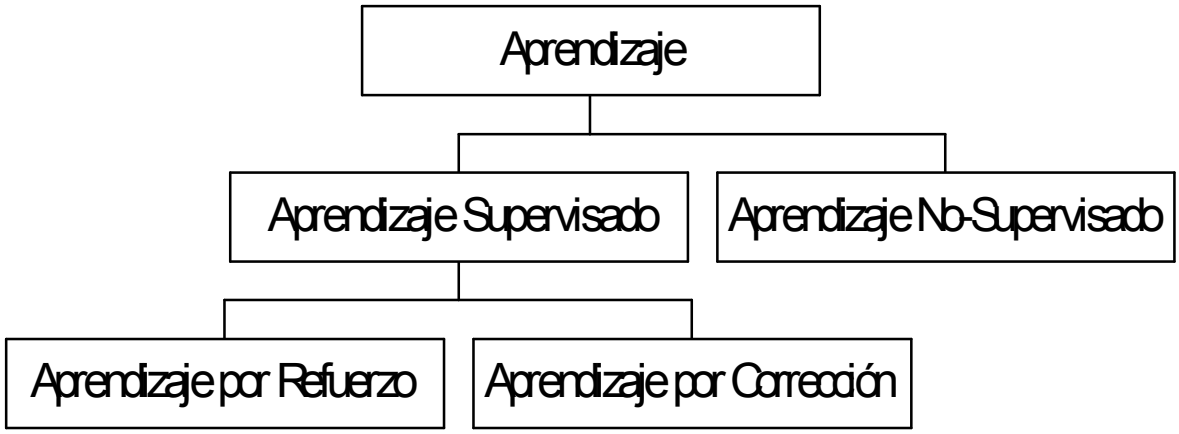


Figura 1.5.

La primera gran división en los métodos de aprendizaje es entre algoritmos supervisados y no supervisados. En los algoritmos no supervisados no se conoce la señal que debe dar la red neuronal (señal deseada). La red en este caso se organiza ella misma agrupando, según sus características, las diferentes señales de entrada. Estos sistemas proporcionan un método de clasificación de las diferentes entradas mediante técnicas de agrupamiento o clustering.

El aprendizaje supervisado presenta a la red las salidas que debe proporcionar ante las señales que se le presentan. Se observa la salida de la red y se determina la diferencia entre ésta y la señal deseada. Posteriormente, los pesos de la red son modificados de acuerdo con el error cometido. Este aprendizaje admite dos variantes: aprendizaje por refuerzo o aprendizaje por corrección. En el aprendizaje por refuerzo sólo conocemos si la salida de la red se corresponde o no con la señal deseada, es decir, nuestra información es de tipo booleana (verdadero o falso). En el aprendizaje por corrección conocemos la magnitud del error y ésta determina la magnitud en el cambio de los pesos.

1.8. Estructuras neuronales.

Las redes neuronales se pueden usar en una serie de estructuras según la aplicación a la que está destinado el sistema. Así pues, según la disposición de la red neuronal se tendrán las siguientes estructura:

1.8.1. Estructura directa.

Esta estructura presenta el siguiente esquema de bloques:

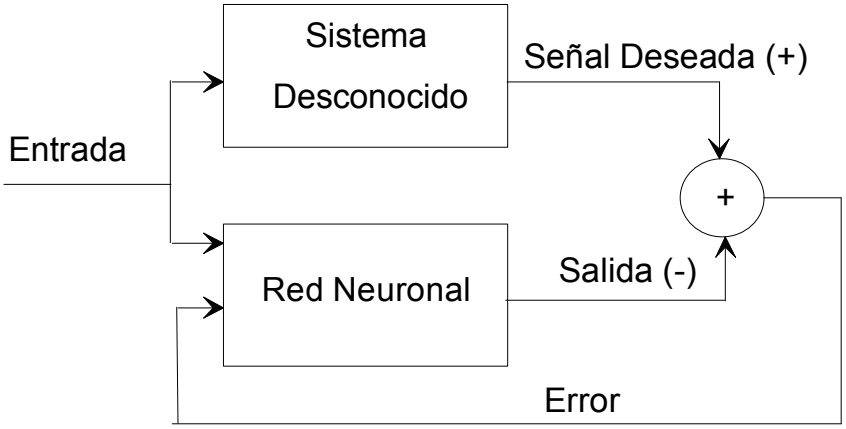


Figura 1.6.Esquema de una estructura directa.

Como se aprecia en la figura anterior, el sistema, en principio desconocido, y la red neuronal tienen las mismas entradas por lo que se conseguirá el mínimo error (objetivo de la red neuronal) cuando la salida de la red neuronal y la señal deseada sean iguales, o lo que es lo mismo, cuando la función de transferencia de la red neuronal sea igual a la del sistema desconocido. Así pues esta estructura tiene como finalidad la modelización de funciones de transferencia de sistemas de los que, en principio, no conocemos nada pero tenemos la posibilidad de excitarlos con una determinada entrada y así conocer su salida.

1.8.2. Estructura inversa.

En Figura 1.7 se da el diagrama de bloques de la estructura inversa:

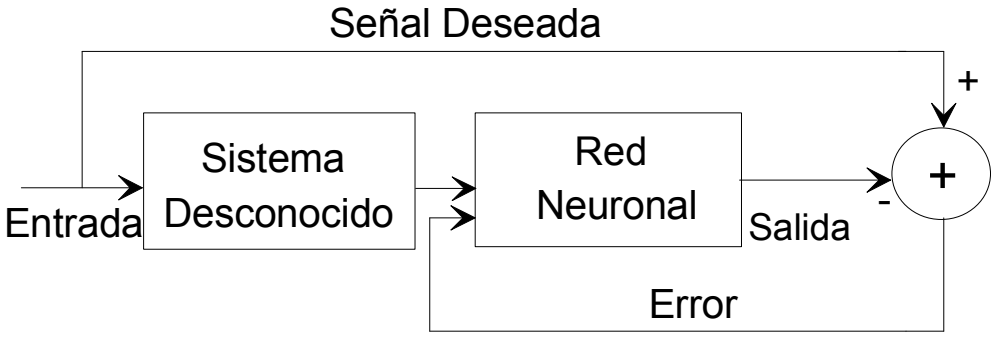


Figura 1.7. Esquema de bloques de la estructura inversa.

El mínimo error en esta estructura se obtendrá cuando la salida de la red neuronal sea la entrada al sistema desconocido lo que conlleva que la función de transferencia de la red neuronal sea la inversa del sistema desconocido. Hay que destacar que el perfecto funcionamiento de esta estructura depende de la estabilidad de la inversa de la función de transferencia del sistema desconocido. Aplicaciones típicas de esta estructura son la ecualización de canales de comunicación y la resolución de problemas de deconvolución.

1.8.3. Estructura con retardo.

El diagrama de bloques de dicha estructura viene dado por la Figura 1.8:

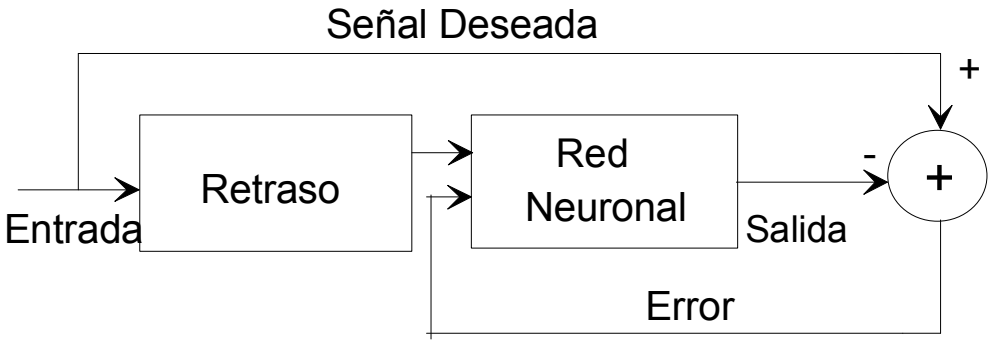


Figura 1.8. Esquema de bloques de la estructura inversa.

Según la definición de red neuronal, esta estructura tiende a minimizar la diferencia entre la señal deseada (señal de entrada en el instante n) y la salida de la red neuronal que será un determinado valor obtenido con valores de la señal. Se intenta, pues, modelizar la señal actual a partir de los valores anteriores de ésta. Este sistema se puede usar, pues, en problemas de predicción (a partir de las muestras pasadas se puede estimar la siguiente) y de control (si se conoce la evolución del sistema se puede alterar los parámetros de dicho sistema para cambiar dicha evolución).

1.8.4. Cancelador de ruido.

Cuando se utiliza una red neuronal, como un cancelador activo de ruido se tiene la siguiente estructura:

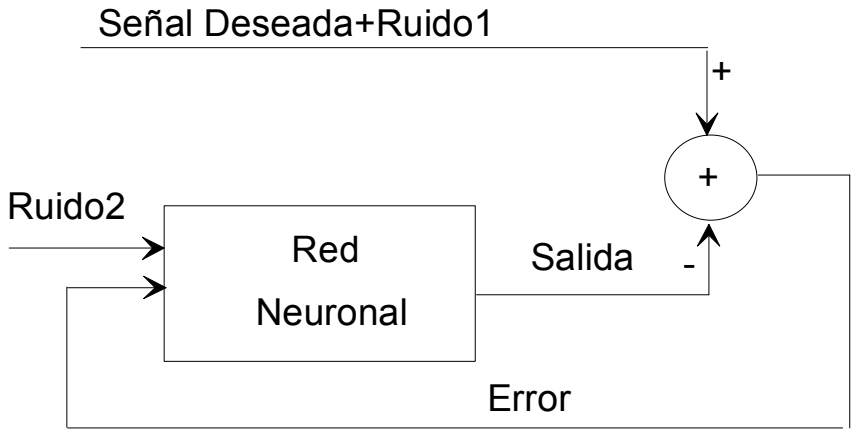


Figura 1.9. Esquema de un cancelador activo de ruido.

En el cancelador activo de ruido el papel central del dispositivo lo asume el algoritmo de la red neuronal. Este algoritmo tiene como objetivo minimizar el error cuadrático entre la salida de la red neuronal y la señal que se toma como referencia. Tomando en cuenta el esquema de la figura anterior se trata de minimizar la siguiente cantidad:

$$e^2(n) = \left(señal\ deseada + ruido1 - salida\right)^2 \qquad \text{Ec. 1.12}$$

La condición de funcionamiento es la correlación entre el ruido que entra a la red neuronal y el que hay en la señal de referencia (señal a limpiar). A nivel práctico este requerimiento es, la mayoría de las ocasiones relativamente sencillo de cumplir. Pensemos, por ejemplo, en la cancelación de ruido en un teléfono móvil. Aquí la señal de referencia sería nuestra voz contaminada con el ruido ambiental (señal de entrada al micrófono del teléfono) y la entrada a la red neuronal sería la señal que se tomaría en otro punto cercano al teléfono. Es necesario tener como precaución situar este último sensor de tal forma que nuestra voz (la señal de interés a recuperar) no aparezca en la entrada a la red neuronal.

De acuerdo con la expresión del error este será mínimo cuando la salida de la red neuronal sea igual al ruido que aparece en la señal de referencia ya que no existe ninguna relación entre la señal deseada y el ruido de la entrada a la red. En el momento de conseguir la minimización la salida de la red neuronal será el ruido1 y, por tanto, la señal de error será igual a la señal deseada, es decir obtendremos la señal limpia de ruido.

1.9. Aplicaciones de las Redes Neuronales.

Las aplicaciones de las redes neuronales las podríamos dividir según el tipo de problema a resolver. Dentro de cada subgrupo podemos establecer otra división según el campo del conocimiento donde se aplican.

1.9.1. Clasificación.

1.9.1.1.Medicina.

Las aplicaciones de clasificación en medicina encuentran su reflejo en problemas de diagnóstico médico. Es uno de los campos con más futuro y, hoy por hoy, uno de los menos desarrollados. Aplicaciones en este campo serían:

- Diagnóstico de cardiopatías. En este tipo de aplicaciones nos encontramos con el trabajo de Bortolan en el que usando este tipo de red clasifica una serie de ECG en diferentes tipos o clases [Bortolan-90]. El mismo autor en colaboración con otros presenta posteriormente una serie de modificaciones que mejoran sus resultados, descomponiendo el conjunto de entrenamiento o la estructura de la red a entrenar [Bortolan-91] o preprocesando los datos mediante una capa de RBF (Radial Basis Function) aumentando así la sensibilidad y especificidad del dispositivo [Silipo-96]. En este mismo sentido de mejorar el funcionamiento de la red, han aparecido una serie de trabajos en los que se usan transformadas del ECG como entradas a la red, DFT [Tsai-90], [Dokur-96], DCT [Hilera-95], [Kalita-93] y wavelets [Shukla-93]; otros combinan transformadas con valores temporales característicos del ECG [Thomson-93]. Otros caminos son reducir el número de muestras del ECG considerando sólo las más significativas [Har-93] o determinados parámetros de interés [Pretorius-92]. Una gran ventaja que ofrecen los perceptrones multicapa es la posibilidad de implementarlos en sistemas VLSI [Leong-91]. Esta posibilidad ha sido explotada para desarrollar desfibriladores usando una red de este tipo como dispositivo de decisión [Jabri-95] [Coggins-95].

Además del perceptrón multicapa con el backpropagation como algoritmo de aprendizaje se han usado otras redes como el mapa autoorganizativo (SOM) [Morabito-91] [Palreaddy-95] [Reinhardt-96], Fuzzy ARTMAP [Ham-91] [Ham-96], redes neuronales probabilísticas [Kramer-95] o combinaciones de perceptrones [Chi-91].

- Detección de tumores cancerígenos. Una red neuronal entrenada localiza y clasifica en imágenes médicas la posible existencia de tumores cancerígenos [Moallemi-91].

- Determinación de puntos característicos del ECG. En esta aplicación la red neuronal se utiliza para determinar puntos del ECG que serán usados

posteriormente por el cardiólogo para establecer las diferentes series temporales [Soria-97].

1.9.1.2.Farmacia.

Estrechamente unidas con las anteriores. En este campo se diagnostican posibles efectos adversos de la administración de un fármaco. Así se tienen aplicaciones del tipo:

- Predicción del riesgo de intoxicación por digoxina. En esta aplicación la tarea de la red neuronal es predecir el posible riesgo de intoxicación por digoxina que es un fármaco usado en problemas de corazón [Camps-98], [Soria-98], [Martin-99].

- Predicción de la respuesta emética. En esta aplicación la red neuronal determina como salida la respuesta emética. Esta respuesta está relacionada con el número de naúseas y vómitos que siente un paciente oncológico tras un tratamiento con quimioterapia [Serrano-98], [Soria-98].

1.9.1.3.Procesado de la señal.

En este campo las redes neuronales han encontrado un gran hueco de tal forma que ya existe una sociedad internacional sobre la aplicación de redes neuronales en problemas de procesado de la señal. Algunos problemas de clasificación donde se aplican las redes neuronales serían:

- Ecualización de canales de comunicación. Ecualizar un canal consiste en recuperar la señal que, al pasar a través de un canal de comunicaciones, sufre una distorsión. Cuando la señal de origen es binaria (0/1 ò ±1) este problema se puede considerar como un problema de clasificación. La aplicación de redes neuronales se ha mostrado más efectiva que el uso de otros sistemas [Gibson-91], [Chen-91], [Molina-99].

- Reconocimiento de patrones en imágenes. Esta aplicación evidencia la capacidad de las redes neuronales ya que se trata de una tarea relativamente sencilla para un ser humano pero tremendamente costosa de implementar en un sistema artificial [Vincent-95], [Golomb-95], [Roli-96].

- Reconocimiento de voz. Esta aplicación, de gran importancia de cara a la implementación de sistemas controlados por la voz, ha encontrado en las redes neuronales un camino para su desarrollo [Burr-88], [Waibel-89]..

- Sonar y Radar. La capacidad de las redes neuronales para clasificar determinados objetos (imágenes, sonidos, señáles unidimensionales, ...) les permite su aplicación en este campo como dispositivos para discernir los diferentes objetivos [Suga-90], [Ukrainec-96], [Gorman-96].

1.9.1.4. Economía.

En esta disciplina, donde hay que tomar decisiones entre un número de opciones, las redes neuronales son directamente aplicables frente a otros métodos

por sus características intrínsecamente no lineales. Así, algunas de estas aplicaciones serían:

- Concesión de créditos. En esta aplicación las redes neuronales en virtud de determinados marcadores económicos de la persona que pide el préstamo decide su viabilidad o no.

- Detección de posibles fraudes en tarjetas de crédito. Las redes neuronales pueden ser usadas como elementos discriminativos para conceder o no una determinada cantidad en un cajero automático

- Determinación de la posibilidad de quiebra de un banco. En esta aplicación la red neuronal determina el riesgo de quiebra de un banco en virtud de determinados parámetros económicos [Martin del Brio-96].

1.9.2. Modelización.

1.9.2.1.Medicina.

Las aplicaciones de modelización en medicina están relacionadas con el procesado de determinadas señales bioeléctricas tales como el electrocardiograma fetal (FECG), el electromiograma (EMG), electroencefalograma (EEG), etc. Algunas aplicaciones en este campo serían:

- Caracterización de la dinámica en la variabilidad cardíaca. La regulación del ritmo cardíaco se lleva a cabo por un sistema dinámico operando bajo un régimen caótico. Funciones de Base Radial (RBF, Radial Basis Function) han sido usadas para determinar esta dinámica [Bezerianos-99].

- Compresión de señales electrocardiográficas. Uno de los temas más activos actualmente en el campo de la ingeniería biomédica es la telemedicina. Esta disciplina consiste en el desarrollo de algoritmos que permitan el diagnóstico de una determinada enfermedad sin que el paciente se tenga que desplazar al centro médico. Las diferentes señales que necesita el médico se transmiten vía telefónica. Para aumentar la eficacia de esta transmisión se podría pensar en la compresión de la señal que consiste en aplicar diferentes algoritmos para reducir su tamaño. Uno de los métodos de compresión es con redes neuronales [Hilera-95].

- Predicción de enfermedades degenerativas cardíacas. . Pacientes que han sufrido un infarto recientemente presentan un cierto factor de riesgo de sufrir otro. Anderson plantea una red que intenta modelizar el comportamiento de las arterias coronarias [Anderson-96]. En este mismo sentido se orienta el trabajo de Azuaje que hace uso de técnicas no lineales como son los diagramas de Poincaré [Azauje-99].

1.9.2.2.Farmacia.

Aplicaciones de modelización en farmacia las podemos encontrar principalmente en el campo de la farmacocinética ya que esta disciplina intenta

determinar modelos que predigan la concentración de un determinado fármaco en sangre. Algunas de estas aplicaciones serían:

- Predicción del nivel de Tacrolimus en sangre. Este fármaco se utiliza en la terapia post-trasplante. Presenta un estrecho ámbito terapeútico (la concentración en sangre se debe mantener entre 5 y 15 ng/ml). El empleo de un perceptrón multicapa ha demostrado su utilidad en la predicción del nivel de este fármaco en sangre [Chen-99].

- Predicción del nivel de ciclosporina. La ciclosporina es un fármaco usado habitualmente para evitar la reacción de rechazo en transplantes de riñón, corazón, pulmón e hígado. Predecir la concentración de este fármaco a corto plazo ayudaría a la optimización de la dosis siguiente.

- Predicción de la concentración de Gentamicina. Corrigan, Mayo y Jamali demuestran en su trabajo la bondad de la predicción realizada por las redes neuronales en este problema comparando su predicción con herramientas que se pueden considerar clásicas en farmacocinética como es el paquete informático NONMEM [Corrigan-97].

1.9.2.3.Procesado de la señal.

En este campo algunas aplicaciones de modelización serían:

- Eliminación activa de ruido. Cuando el ruido y la señal de interés tienen los espectros frecuenciales solapados un filtrado selectivo en frecuencia no tiene sentido. En este caso hay que intentar otras aproximaciones. Una de estas es la cancelación activa de ruido aplicando sistemas adaptativos y redes neuronales [Tamura-89], [Tamura-90].

- Control. En este caso el sistema a controlar se modeliza para poder realizar predicciones de su comportamiento y, de esta forma, poder controlarlo más fácilmente. Este es el caso de, por ejemplo, un sistema de vertido de residuos y aceites [Tsao-93].

1.9.2.4.Economía.

El éxito o fracaso de la mayoría de las operaciones realizadas en economía dependen de la "visión de futuro" que tenga la empresa o el operador bursátil. Algunas de las aplicaciones de las redes en economía son:

- Predicción del gasto eléctrico de empresas y centrales. Mediante el uso de una red neuronal podemos estimar el consumo de una empresa y, por tanto, podemos administrar mejor los recursos eléctricos de dicha empresa [Sharkawi-96], [Kermanshashi-96].

- Cambio de moneda. Las redes neuronales se han usado para la predicción del cambio entre el dólar americano y el marco alemán [Gutjahr-97].

- Tendencias a corto y medio plazo en bolsas de valores. Si se buscan por Internet los productos derivados de las redes neuronales que se comercializan se encontrará rápidamente que la gran mayoría de ellos se orientan a aplicaciones de este tipo.

- Predicción de stocks. Uno de los mayores problemas que se puede encontrar una fábrica es la falta o un exceso de suministros. En el primer caso no puede producir y, en el segundo, si no dispone de un buen almacen, se puede producir el caos. Una buena previsión de la cantidad necesaria justa podría evitar muchos problemas [Weigend-97].

  Una aplicación habitual se encuentra en la predicción de intervalos de confianza en una operación bursátil.

1.9.2.5.Medio Ambiente.

Que vivimos en un ambiente dinámico y no lineal nadie lo puede negar; cualquier método aplicado a este campo necesariamente debe tener en cuenta estos hechos irrefutables. Tenemos, pues, otro campo importante de aplicación de las redes neuronales. Algunas aplicaciones de éstas serían:

- La predicción de irradiación solar junto con otros muchos trabajos encaminados a la predicción de la aparición de manchas solares han sido la base para la aplicación a otros problemas medioambientales. La actividad solar se encuentra relacionada con la existencia de campos magnéticos en la superficie del Sol y se cree que tiene alguna influencia en el tiempo sobre la Tierra. El origen de este tipo de series es un sistema dinámico ruidoso y caótico [MacPherson-95].

- Predicción de niveles tóxicos de ozono en zonas urbanas y rurales. Este gas nos protege de la radiación ultavioleta del sol, sin embargo, un exceso de puede conducir a problemas de salud. Una predicción de su concentración en la atmósfera a corto plazo (uno o dos días) podría conducir a la aplicación de medidas para evitar posibles incrementos indeseados en la concentración de este gas [Milton-98].

- Predicción de variaciones globales de temperatura. [Miyano-94]. Las variaciones de las temperaturas marinas nos alertan sobre la formación de huracanes y tormentas, por lo que se hace necesario su predicción.

A modo de resumen de este capítulo hemos visto como las redes neuronales se han convertido en un campo de investigación muy popular dentro de las ciencias computacionales, la neurobiología, el procesado de señales, óptica, física... por tanto, el campo requiere un estudio multidisciplinar. Las diferentes áreas del conocimiento donde podemos aplicar estos elementos quedarían reflejadas en la siguiente tabla:

Campos de Investigación.	Aplicaciones.
Ciencias de la Computación.	Aprendizaje de sistemas, procesado de la información no simbólica,...
Estadística	Modelos clasificadores, regresiones no lineales,...
Ingenierías	Control automático, procesado de señales,...
Ciencias del Conocimiento	Modelos de pensamiento y conciencia. (Función Cerebral de Alto Nivel)
Neurofisiología	Modelos de memoria, sistemas sensores y motores. (Función Cerebral de Nivel Medio)
Física	Modelos de fenómenos en mecánica estadística,..
Biología	Interpretación de secuencias de nucleótidos
Filosofía	Aprendizaje Inducido/Reforzado/Tutelado/...

Tabla 1.2. Uso práctico de las RNAs.

Las RNAs son en la práctica muy útiles en problemas en los que se asume una cierta tolerancia a la imprecisión y en los que tenemos un número elevado de datos de entrenamiento pero donde la aplicación de reglas rápidas y robustas no es posible. Las RNAs con una simple capa oculta son consistentes estimadores estadísticos para la regresión de funciones, y, por supuesto, excelentes clasificadores binarios. Actualmente son usadas en procesos de control, ayuda a la decisión clínica, ayuda a discapacidades físicas, modelización de mercados financieros, reconocimiento de patrones, etc.

Demos paso a continuación a este mundo de aplicaciones.....

2

Sistemas con una neurona

2.1. Descripción de una neurona.

El título del capítulo puede inducir a alguna sonrisa malvada para un lector malpensado; ¿qué podemos hacer con una neurona?. La pregunta puede llevar a un gran error ya que la respuesta más evidente es nada. Si pensáramos así habríamos eliminado de un plumazo métodos estadísticos clásicos como la regresión logística (sí, ¡es una neurona!) y áreas tan activas de investigación como es el procesado adaptativo de la señal.

Una neurona artificial no recurrente (la más común) viene definida por la siguiente figura:

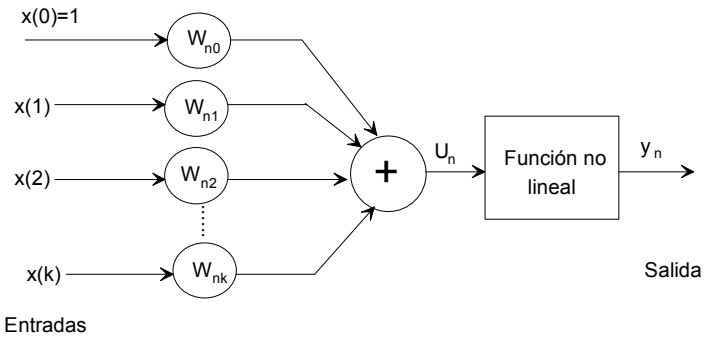


Figura 2.1. Esquema de una neurona sin memoria.

En este esquema se aprecia que la neurona no tiene realimentaciones; se podría plantear una estructura más general de neurona con una cierta memoria; esta estructura sería:

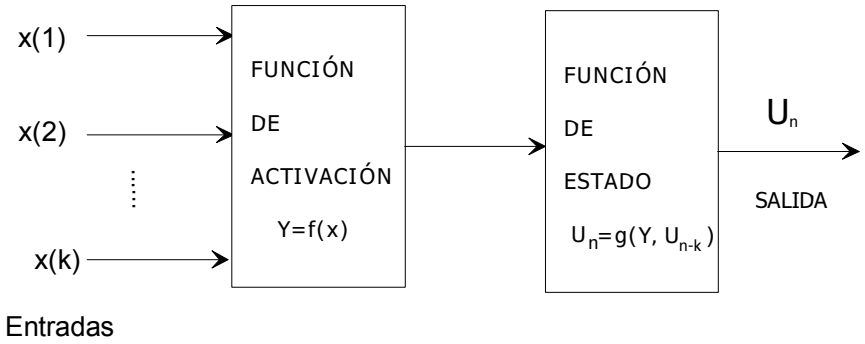


Figura 2.2. Esquema general de una neurona.

Este tipo de memoria puede ser finita o infinita según el tipo de función de estado. En este texto se utilizarán neuronas del tipo definido en la Figura 2.1; sin memoria.

En este tipo de memoria nos encontramos con los siguientes elementos:

- Pesos sinápticos. Son los valores W_{ki} que aparecen en la Figura 2.1. Estos pesos sinápticos son los parámetros del sistema, de tal forma que el proceso de aprendizaje se reduce a obtener los valores que mejor se comportan con nuestro problema.

- Sumador. Este elemento determina la suma promediada de las entradas. Es el reflejo de la acción integradora de las entradas que se produce en los axones de las células nerviosas biológicas.

- Función no lineal. Se aplica a la salida del sumador y, normalmente, es una función no lineal; se le conoce como función de activación. Entre todas las posibles variantes, destacamos las siguientes:

  - Función signo. Es la primera que se propuso dando lugar al modelo neuronal planteado por McCulloch-Pitts [McCulloch-43]. Esta función plantea una separación "dura" de los datos de entrada, esto es, son clasificados como ±1 ya que esta función queda definida como:

$$f(x) = \begin{cases} -1 \text{ si } x < 0 \\ 1 \text{ si } x \geq 0 \end{cases} \qquad \text{Ec. 2.1}$$

  - Sigmoide. Se deriva de la anterior. Como veremos más adelante, muchos algoritmos de aprendizaje exigen que la función de activación sea diferenciable. La función signo no lo es en el origen mientras que la sigmoide sí. Esta función viene definida según la siguiente expresión:

$$f(x) = \frac{a}{1 + e^{-b \cdot x}} \qquad \text{Ec. 2.2}$$

    donde a es la amplitud de la sigmoide y el factor b la pendiente en el origen; conforme ese factor b crece, la función se acerca a la función signo

(pero codificando las salidas como 0/a). La versión bipolar de esta función de activación (la tangente hiperbólica) tiene como expresión:

$$f(x) = a \cdot \frac{1 - e^{-b \cdot x}}{1 + e^{-b \cdot x}} \qquad \text{Ec. 2.3}$$

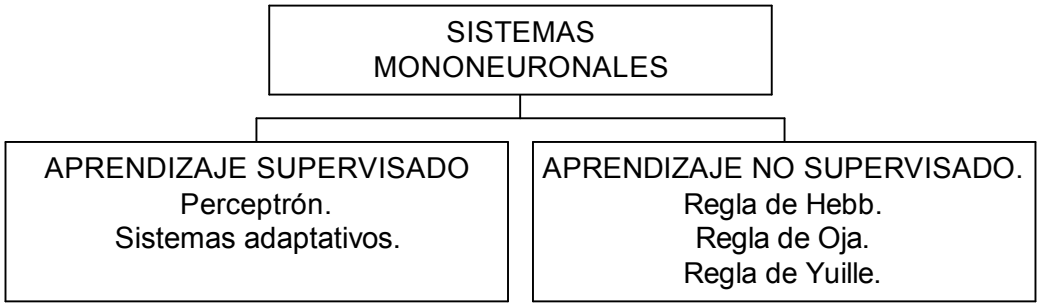
Ahora la salida está entre ±a.

- Función gaussiana. Esta función de activación es muy usada en un tipo de red neuronal conocida como función de base radial (RBF). Su expresión es:

$$f(x) = K_1 \cdot e^{-K_2 \cdot x^2} \qquad \text{Ec. 2.4}$$

Las principales características de esta función, a diferencia de las anteriores, es su localidad; la función tiende a cero a partir de un cierto x de tal forma que sólo para un determinado intervalo de entradas la salida de la función de activación es diferente de cero.

Ya conocemos algunos de los elementos básicos con los que vamos a trabajar este capítulo; el otro elemento es el algoritmo de aprendizaje a utilizar. Como se detalló en el capítulo anterior, éste puede ser de dos tipos: supervisado, existe un "maestro" o no supervisado; no existe tal maestro. Según el tipo de aprendizaje las estructuras mononeuronales a estudiar en este capítulo serán:



Veamos cada uno de estas estructuras por separado.

2.2. Perceptrón.

La primera estructura neuronal a estudiar es la más sencilla y la que, desde un punto de vista histórico, apareció la primera. Esta estructura se usó en problemas de clasificación y tiene el siguiente esquema:

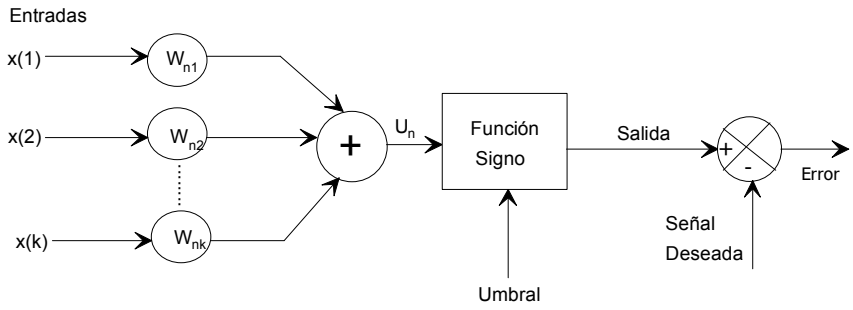


Figura 2.3. Esquema del perceptrón.

El funcionamiento del perceptrón es muy sencillo; se basa en comparar la salida del sistema con una señal deseada (que es la que debería dar el sistema). De la estructura, más concretamente de la función de activación, se deduce que este elemento es un clasificador binario ya que puede determinar la pertenencia por parte del vector de entrada a dos clases diferentes.

El algoritmo que sigue este sistema es de tipo supervisado ya que necesitamos un elemento exterior que plantee la clase de pertenencia del elemento de entrada. El procedimiento de aprendizaje comienza por la inicialización aleatoria de los pesos, para, posteriormente, irlos ajustando conforme la red se equivoca en la asignación de clase al vector de entrada presente en ese momento. Veamos todos estos pasos con más detalle:

Proceso de aprendizaje del perceptrón.

1. Inicialización de los pesos.

2. Determinación de la salida:

$$y = \sum_{k=1}^{N} w_k^n \cdot x_k^n \qquad \text{Ec. 2.5}$$

3. Comparación con el umbral:

$$u = y - \text{umbral} \qquad \text{Ec. 2.6}$$

4. Aplicación de la función signo

$$o = signo(u) \qquad \text{Ec. 2.7}$$

5. Comparación con la señal deseada.

6. Actualización de los coeficientes si existe error.

$$w_k = w_k + \alpha \cdot (d - o) \cdot x_k \qquad \text{Ec. 2.8}$$

Este elemento neuronal, a pesar de su extremada sencillez, ha sido estudiado con profundidad y de él se pueden extraer una serie de interesantes conclusiones.

Antes de seguir con su análisis, se van a simplificar algunas de las operaciones definidas anteriormente. En primer lugar, se suele englobar el umbral dentro de la

salida del sistema mediante un peso adicional conectado a una entrada de valor uno, es decir:

$$o = \sum_{k=1}^{N} w_k \cdot x_k - umbral \Rightarrow \sum_{k=0}^{N} w_k \cdot x_k \text{ con } x_0 \text{ igual a 1 y } w_0 \text{ igual a - umbral} \quad \text{Ec. 2.9}$$

El funcionamiento del perceptrón como clasificador se basa en la función signo, es decir:

$$\text{sign}(o) = \begin{cases} +1 \text{ si } o \geq 0 \\ 0 \text{ si } o < 0 \end{cases} \quad \text{Ec. 2.10}$$

Modificamos el rango de la función signo (±1 en el caso más común) porque resulta más conveniente para los ejemplos que vienen a continuación.

De la anterior expresión, se deduce que la frontera de decisión se toma para o=0. Tomemos un caso simple en el que tendremos dos entradas, x_1 y x_2, para comprender qué significa esto. Si la salida, tras aplicar la función signo, vale cero significa que:

$$o = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 \quad \text{Ec. 2.11}$$

Despejando x_2 en la anterior ecuación:

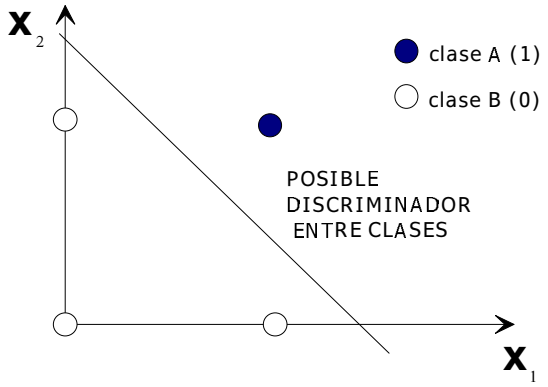
$$x_2 = -\frac{w_o}{w_2} - \frac{w_1}{w_2} \cdot x_1 \quad \text{Ec. 2.12}$$

Si nos fijamos, la anterior expresión es la ecuación de una recta en el espacio definido por los patrones de entrada; estos es, la superficie de separación entre clases diferentes es una recta. De aquí se deduce que se tendrá una clasificación perfecta si los patrones son linealmente separables.

A modo de ejemplo sencillo si se quiere diseñar una puerta AND de dos entradas con un perceptrón se tiene que clasificar el siguiente conjunto de patrones:

X_1				
X_2				
Deseada				

Si se representa gráficamente el conjunto de entrada se tiene:



Como se aprecia de la figura, la implementación de una puerta AND es simplemente un problema de clasificación. Una posible solución a nivel neuronal sería la siguiente red de una neurona:

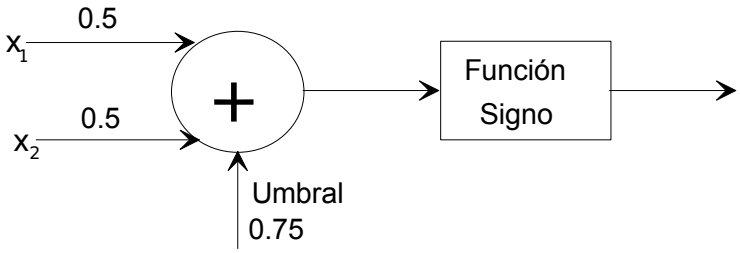


Figura 2.4. Implementación neuronal de una puerta AND

La implementación de una puerta OR es análoga al caso de la puerta AND. Así, a nivel neuronal se tendría:

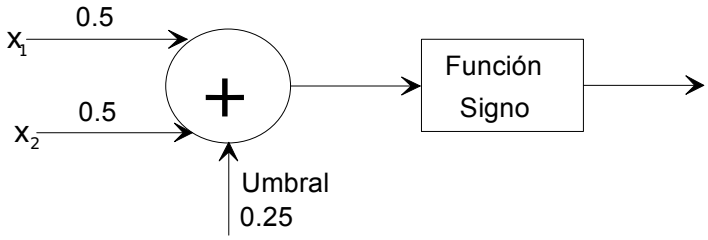


Figura 2.5. Implementación neuronal de una puerta OR.

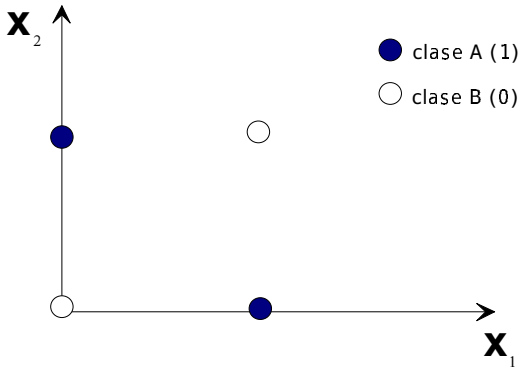
Hemos comprobado que las puertas lógicas AND y OR son implementables a nivel neuronal. Este hecho animó a los primeros investigadores sobre redes neuronales, ya que les posibilitaba construir elementos lógicos tomado como base estos elementos básicos, las neuronas artificiales.

Hasta ahora hemos visto patrones linealmente separables, pero, ¿qué ocurre cuando no tenemos este tipo de patrones?. Pensemos, a modo de ejemplo, en la puerta XOR de dos entradas. Esta operación lógica viene definida por la siguiente tabla de valores.

X_1	0	1	0	1
X_2	0	0	1	1

Deseada	0	1	1	0

Que representados gráficamente, quedan como:



De la figura se ve claramente que los patrones no son linealmente separables: no existe ninguna recta que deje a cada lado los elementos de diferentes clases. En este punto se tienen dos posibles soluciones:

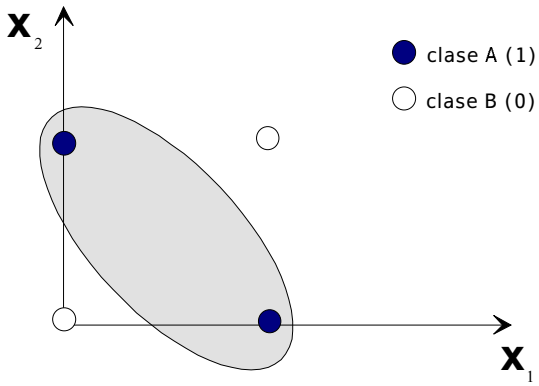
- Definir otras superficies de separación; aquí, por ejemplo, se podrían definir elipses como separadores, es decir, habría que definir una serie de pesos de tal forma que las clases quedaran clasificadas de acuerdo a la siguiente expresión:

$$signo(w_1 \cdot x_1^2 + w_2 \cdot x_2^2 + w_3 \cdot x_1 \cdot x_2 + w_4 \cdot x_1 + w_5 \cdot x_2 + w_6) \qquad \text{Ec. 2.13}$$

  Ahora la superficie de separación queda definida por:

$$(w_1 \cdot x_1^2 + w_2 \cdot x_2^2 + w_3 \cdot x_1 \cdot x_2 + w_4 \cdot x_1 + w_5 \cdot x_2 + w_6) = 0 \qquad \text{Ec. 2.14}$$

  Así, a nivel gráfico una posible separación sería:



  Una generalización de este concepto dará lugar a las máquinas de vectores soporte (SVM) como se verá en un capítulo posterior [Vapnik-96].

- La segunda solución surge de la combinación de diferentes perceptrones para dar lugar a esa puerta lógica ya que cualquier puerta lógica se puede sintetizar a partir de puertas AND y OR.

El problema de la puerta XOR, y en definitiva los problemas no linealmente separables, fueron el punto central de la discusión de Minsky y Papert en su libro Perceptrons sobre las limitaciones de estos elementos [Minsky-69]. Su crítica provocó el abandono de la investigación sobre redes neuronales, y solamente algunos investigadores aislados (Anderson, Kohonen, Amari, Hopfield, ...) continuaron trabajando sobre estos temas. Sin embargo esta situación cambió radicalmente con la llegada del algoritmo Backpropagation propuesto originalmente por Paul Werbos [Werbos-73] en su tesis doctoral y, difundido años más tarde gracias al trabajo de Rumelhart, Mclelland y Hinton [Rumelhart-86].

Se puede demostrar que siempre que tengamos patrones linealmente separables, el perceptrón llega a una solución del problema [Kung-93], [Hassoun-95]. Sin embargo, cuando los patrones no son linealmente separables, el algoritmo de aprendizaje puede dar lugar a oscilaciones en los valores de los pesos. Para solucionar estas oscilaciones se plantean diferentes soluciones entre las que cabría destacar el algoritmo de bolsillo (pocket algorithm) [Gallant-93]. Este algoritmo consiste en la aplicación del algoritmo del perceptrón pero llevando en paralelo dos vectores de pesos de tal forma que uno de ellos es el que mejor resultado presenta. Si el otro vector, que sigue el algoritmo del perceptrón clásico, obtiene mejores resultados que los obtenidos con el que tenemos almacenado (guardado en un "bolsillo"), se reemplaza el vector guardado por el vector que supera sus resultados. De esta forma, siempre no aseguramos una solución aunque no sea la óptima, evitando las oscilaciones que provoca el algoritmo del perceptrón.

2.3. Sistemas adaptativos.

Llegamos a uno de los elementos centrales en el procesado digital de señales, por el número de aplicaciones que tiene: la adalina. Este elemento sigue teniendo la estructura del perceptrón cambiando la ecuación de actualización de los pesos. Este cambio se refleja en la Figura 2.6.

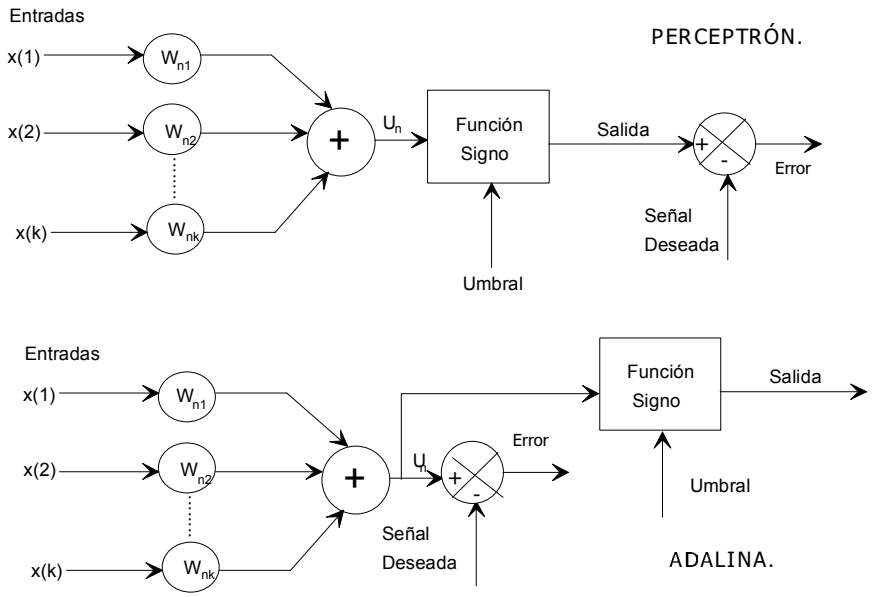


Figura 2.6. Esquema comparativo de un perceptrón y una adalina.

Si nos fijamos en la Figura 2.6, vemos que la estructura permanece igual pero con el detalle, gran detalle como veremos posteriormente, de que la señal de error se calcula antes de aplicar la función signo. Este hecho permite enfocar el aprendizaje desde un punto de vista funcional: mi sistema óptimo será aquél que me asegure un error mínimo y, por tanto, lo que busco es el mínimo de una función creciente del error. Como veremos más adelante existen diferentes posibilidades a la hora de escoger esta función; la que se escogió a nivel histórico en primer lugar, y la que se usa en la gran mayoría de aplicaciones, es la función cuadrática, definida por:

$$J(n) = error^2(n) \qquad \text{Ec. 2.15}$$

Veamos qué significa esta función mediante un ejemplo sencillo; consideremos una neurona con dos entradas, la función de error sería:

$$J(n) = \left(d(n) - w_0 - w_1 \cdot x_1 - w_2 \cdot x_2\right)^2 \qquad \text{Ec. 2.16}$$

En sistemas con una sola neurona es usual trabajar con el vector de pesos y con el vector de entrada; estos vectores quedan definidos de la siguiente forma:

$$\begin{aligned} W_n &= [w_0\ w_1\ w_2]^t \\ X_n &= [1\ x_1\ x_2]^t \end{aligned} \qquad \text{Ec. 2.17}$$

donde el superíndice t indica trasposición. Con esta notación se llega a:

$$J(n) = \left(d(n) - W_n^t \cdot X_n\right)^2 \qquad \text{Ec. 2.18}$$

que, desarrollando conduce a:

$$J(n) = d^2(n) - 2 \cdot d(n) \cdot W_n^t \cdot X_n + W_n^t \cdot X_n \cdot X_n^t \cdot W_n \qquad \text{Ec. 2.19}$$

Esta ecuación es la de un hiperparaboloide, es decir, una generalización a más de dos dimensiones de una parábola. Como su análogo bidimensional, este sistema sólo presenta un mínimo, y por tanto, éste es global. El objetivo es pues, localizar un mínimo en una función. Los procedimientos usados para esta localización son

conocidos en teoría del procesado de la señal como algoritmos adaptativos. La siguiente figura muestra los algoritmos adaptativos más extendidos:

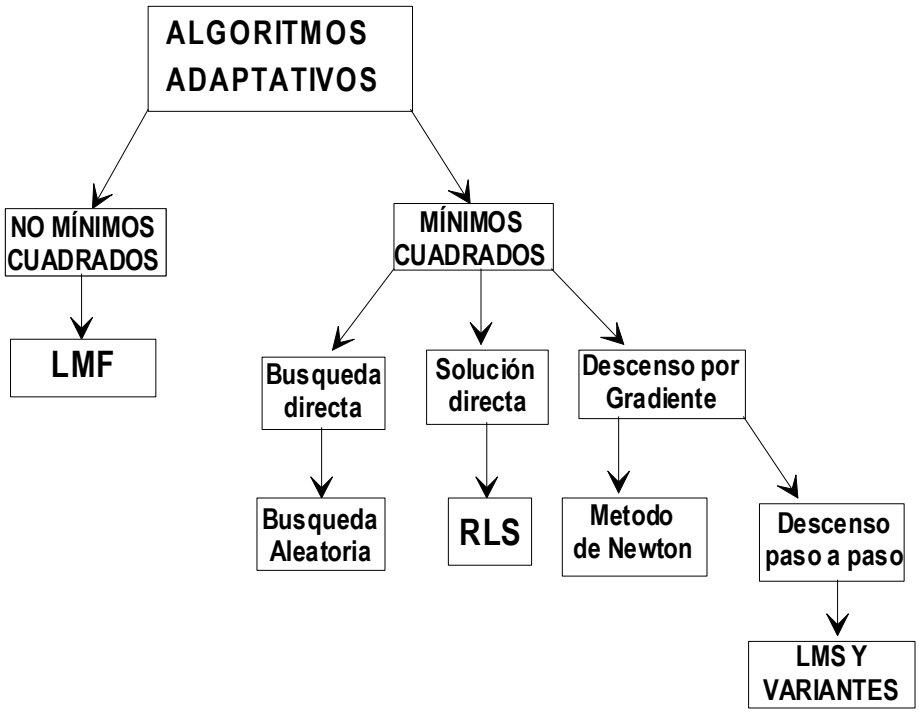


Figura 2.7. Esquema de los diferentes algoritmos adaptativos.

En la figura aparece una primera gran división entre los algoritmos adaptativos según la función de error a minimizar; si es el error cuadrático tendremos los métodos de mínimos cuadrados; si la función a minimizar es la potencia a la cuarta del error tendremos los procedimientos conocidos como LMF (Least Mean Four). En cuanto a los métodos de mínimos cuadrados podemos tener tres posibilidades:

- Búsqueda directa. Son todos aquellos métodos que buscan directamente el mínimo; entre estos métodos destacamos el de búsqueda aleatoria (la filosofía de este método es: ¡¡intentándolo muchas veces al final se consigue!!).

- Solución directa. Estos métodos plantean el problema de minimización de una función cuadrática pero, en vez de minimizar el error actual, minimizan un promedio de éstos. La familia más extendida de estos métodos son los conocidos como RLS (Recursive Least Squares). Estos procedimientos son muy rápidos, en cuanto a velocidad de convergencia al mínimo se refiere, pero tienen una alta carga computacional.

- Descenso por gradiente:

  - Método de Newton. En este método de búsqueda se utiliza información sobre el crecimiento y curvatura de la función de error (primera y segunda derivada respectivamente). Son rápidos, comparados con los siguientes (descenso paso a paso), pero presentan problemas cuando la función no es cuadrática o, en su defecto, no se puede aproximar con la precisión necesaria como una función de este tipo cerca del mínimo.

- Descenso paso a paso. Estos métodos son iterativos y sólo utilizan información sobre la primera derivada, por tanto son más sencillos que los anteriores. Son los más usuales en sistemas neuronales por su sencillez, facilidad de implementación y baja carga computacional frente a otros métodos. Por su importancia nos centraremos en el estudio del LMS y sus variantes.

2.4. Obtención del algoritmo de aprendizaje LMS.

Como ya se ha comentado el aprendizaje de un sistema consiste en determinar los valores de los parámetros w que minimizan la expresión 4. Matemáticamente habría que derivar parcialmente la función J con respecto a los parámetros w para, posteriormente igualar esta derivada a cero:

$$\left(d(n) - W_n^t \cdot X_n\right) X_n = 0 \Leftrightarrow X_n \cdot d(n) = X_n \cdot X_n^t \cdot W_n \qquad \text{Ec. 2.20}$$

El producto de los vectores de entrada, que da lugar a una matriz, se conoce como matriz de autocorrelación R. El producto de la matriz de entrada por el vector se conoce como correlación cruzada g.

Si examinamos detenidamente la ecuación 6 podemos apreciar que para cada n tenemos un sistema de ecuaciones lineales con K incógnitas. El coste computacional de la resolución de este sistema de ecuaciones es alto por lo que hay que intentar otra aproximación a este problema.

El algoritmo LMS se basa en la minimización de una función de error J definida como el cuadrado del error, es decir:

$$J = e^2(n) \qquad \text{Ec. 2.21}$$

Para mayor sencillez se definen los coeficientes de la neurona y las señales de entrada de forma vectorial:

$$X(n) = [x_1 \ldots\ldots\ldots\ldots x_n]^t$$
$$F(n) = [f_1(n) \ldots\ldots\ldots\ldots f_L(n)]^t \qquad \text{Ec. 2.22}$$

indicando t trasposición. Destacar que en los coeficientes correspondientes a los pesos sinápticos y a la entrada aparece un índice n que indica su dependencia temporal.

Con esta notación la salida de la neurona queda simplemente como un producto:

$$\text{salida}(n) = F_n^t \cdot X_n \qquad \text{Ec. 2.23}$$

por lo que la función de error queda:

$$J(n) = \left[\text{deseada}(n) - F_n^t \cdot X_n\right] \qquad \text{Ec. 2.24}$$

Si se desarrolla la expresión 10 (llamando a la señal deseada d(n) por simplicidad) se llega a:

$$J(n) = F_n^t \cdot X_n \cdot X_n^t \cdot F_n - 2 \cdot d(n) \cdot F_n^t \cdot X_n + d^2(n) \qquad \text{Ec. 2.25}$$

Como se puede apreciar esta ecuación se correspondería con un hiperparaboloide con un único mínimo global. De forma gráfica la función de error para un filtro de longitud dos tendría un aspecto parecido al de la Figura 2.8:

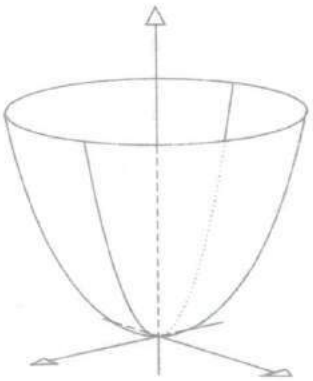


Figura 2.8. . Representación de una superficie de error típica.

Un posible procedimiento iterativo para determinar el mínimo sería situarse en cualquier punto de esa hipersuperficie, es decir considerar unos ciertos valores iniciales para los coeficientes del filtro, y desplazarnos a lo largo de ella siempre apuntando en la dirección del mínimo. Esta dirección viene dada por la opuesta al gradiente ya que éste siempre da la dirección de máximo crecimiento de la función [Luenberger-73]. De forma matemática, el procedimiento tendría la siguiente expresión:

$$F_{n+1} = F_n - a \cdot D_n \cdot \nabla_F J_n \qquad \text{Ec. 2.26}$$

En la anterior ecuación, D_n es una matriz que penaliza o no ciertas direcciones, existiendo diferentes algoritmos adaptativos que determinan de forma automática y en función de las características de las señales de entrada y deseada los términos de dicha matriz. En casi todas las variantes del LMS esta matriz se toma como identidad. En cuanto al parámetro a, éste recibe el nombre de constante de adaptación y es el responsable de las características de funcionamiento de un sistema adaptativo basado en la familia LMS.

El siguiente paso consiste en la determinación del gradiente de J. Para ello, se derivará parcialmente la expresión 10 con respecto de los coeficientes del filtro, es decir:

$$\frac{\partial J}{\partial f_l(n)} = 2 \cdot error(n) \cdot \frac{\partial error(n)}{\partial f_l(n)} \qquad \text{Ec. 2.27}$$

obteniéndose:

$$\frac{\partial J}{\partial f_1(n)} = -2 \cdot error(n) \cdot x_{n-l+1}) \qquad \text{Ec. 2.28}$$

ecuación que, expresada en forma vectorial, conduce a la expresión para la adaptación de los coeficientes de la adalina:

$$F_{n+1} = F_n + a \cdot error(n) \cdot X_n \qquad \text{Ec. 2.29}$$

Un punto importante en este procedimiento iterativo es su perfecto funcionamiento en sistemas donde la función de error presenta un único mínimo global, lo que permite cualquier inicialización de los coeficientes del filtro. Este hecho ocurre siempre en sistemas sin realimentación salida-entrada en la neurona, es decir, en los sistemas estáticos [Haykin-96].

Así pues, este sistema tendría la siguientes etapas en su funcionamiento:

a) Inicialización de los pesos sinápticos.

b) Determinación de la salida de la neurona:

$$salida(n) = F_n^t \cdot X_n \qquad \text{Ec. 2.30}$$

c) Determinación del error del sistema

$$error(n) = deseada(n) - salida(n) \qquad \text{Ec. 2.31}$$

d) Actualización de los coeficientes del sistema:

$$F_{n+1} = F_n + a \cdot error(n) \cdot X_n \qquad \text{Ec. 2.32}$$

2.4.1. Características de funcionamiento.

Como anteriormente se ha mencionado, la constante de adaptación determina las características de funcionamiento:

- Velocidad de convergencia.
- Desajuste en el estado estacionario con el sistema óptimo.
- Estabilidad.

De forma intuitiva, se pueden estudiar todas estas características sin recurrir en principio, a desarrollos matemáticos. El parámetro a que aparece en la ecuación de actualización de los pesos sinápticos controla el tamaño de paso en el movimiento de los coeficientes del filtro hacia los valores óptimos. Si este tamaño de paso es pequeño se necesitaran muchas iteraciones para llegar a esos valores óptimos. Por el contrario, si el parámetro es grande, se llegará en pocas iteraciones, es decir, la velocidad de convergencia del sistema aumentará. Así pues, aumentando la constante de adaptación aumenta la velocidad de convergencia del sistema por lo que interesa tener una constante de adaptación grande. Por otra parte si este tamaño de paso es muy grande podemos "pasarnos" del mínimo y oscilar en torno a él no llegando nunca a ese valor. Es decir, si la constante de adaptación aumenta, el error en estado estacionario aumenta por lo que interesa tener una constante de adaptación pequeña. Si se incrementa demasiado la constante el sistema puede pasar de oscilar a ser inestable. Por tanto, es necesario llegar a una solución de compromiso en la elección de esta constante. A continuación se dan una serie de expresiones que definen el funcionamiento del algoritmo.

2.4.2. Estabilidad del algoritmo

Se estudiará la acotación de la constante de adaptación. Existen gran cantidad de estudios teóricos sobre las características de funcionamiento del LMS para diferentes valores de esa constante [Widrow-76], [Eweda-87], [Foley-87], [Gholkar-90], [Clarkson-93], [Cowan-87], [Macchi-83], [Gardner-87], [Kwong-92]. Sin embargo, todos ellos llevan asociados un conjunto de aproximaciones para simplificar el análisis matemático resultante.

En un artículo que se puede considerar clásico [Widrow-75], Widrow sentó las bases del algoritmo LMS dando los límites para la constante de adaptación que aseguraban la convergencia del valor medio del filtro dado por el algoritmo LMS hacia el valor óptimo. En otro artículo posterior el mismo autor amplía el análisis dando como límite para la constante [Widrow-76]:

$$0 < a < \frac{2}{\lambda_{\max}} \qquad \text{Ec. 2.33}$$

Siendo $\lambda_{máx}$ el mayor de los valores propios de la matriz de autocorrelación de la entrada [Haykin-96]. Esta igualdad lleva a un conocimiento previo de la matriz de autocorrelación, lo cual no es deseable por las razones anteriormente expuestas. Se puede simplificar esta relación aplicando la siguiente desigualdad:

$$\sum_{r=1}^{L} \lambda_r \geq \lambda_{\max} \qquad \text{Ec. 2.34}$$

La matriz de autocorrelación es una matriz de Toeplitz; esto es R(i,j)=R(j,i), por lo que:

$$\sum_{i=1}^{L} \lambda_i = traza(R) = \sum_{r=1}^{L} R(r,r) = L \cdot R(1,1) = L \cdot E[x^2(n)] \qquad \text{Ec. 2.35}$$

donde se puede observar que el último término es la energía de la señal de entrada.

Este valor es más fácil de conocer que los valores propios de la matriz de autocorrelación. Juntando las tres últimas expresiones se llega a los siguientes límites para la constante de adaptación [Clarkson-93]:

$$0 < a < \frac{2}{L \cdot E[x^2(n)]} \qquad \text{Ec. 2.36}$$

Numerosos autores trabajando bajo supuestos diferentes (unos considerando procesos de Markov para simular el ruido, otros tomándolo como ruido blanco...), han encontrado diferentes valores que acotan el valor máximo de la constante de adaptación del LMS para este tipo de convergencia.

2.4.3. Desajuste con el sistema óptimo.

En la obtención del algoritmo LMS sustituimos el valor de J por una estimación. Ésta se calcula para cada instante y tiene como valor:

$$\nabla J_n^{est} = -2 \cdot e(n) \cdot \mathrm{X}_n \qquad \text{Ec. 2.37}$$

La estimación puede considerarse formada por dos términos: el gradiente y un factor de ruido. Este último término es el responsable de que el valor esperado del error cuadrático medio no sea el mínimo sino que aparezcan fluctuaciones sobre este valor. Estas variaciones son debidas a cambios aleatorios de e(n) debidos al ruido (influencia de otras señales, ruido de redondeo, ...). Esta circunstancia provoca que el valor esperado de J no sea el mínimo, obteniéndose [Clarkson-93]:

$$J_\infty \cong J_{\min} \cdot (1 + \frac{a}{2} \cdot L \cdot pot.entrada) \qquad \text{Ec. 2.38}$$

En la anterior expresión, J_{min} es el valor esperado del error cuadrático medio para el filtro de Wiener o filtro óptimo. En [Clarkson-93] y [Widrow-75] se puede encontrar su expresión:

$$J_{\min} = E[d^2] - F_{opt}^{t} \cdot g \qquad \text{Ec. 2.39}$$

donde $E[d^2]$ es la energía de la señal deseada.

Por tanto, un aumento en la constante de adaptación supone un incremento en el valor esperado del error cuadrático al final de la adaptación por lo que el funcionamiento de nuestro filtro se degrada al aumentar esta constante. Además, como se explicó en el apartado anterior, un aumento de la constante puede hacer nuestro algoritmo inestable. Estos dos factores nos llevarían a disminuir la constante;

sin embargo hay un tercer elemento en el funcionamiento del filtro que compensa estos dos problemas: la velocidad de convergencia al valor óptimo del algoritmo.

2.4.4. Velocidad de convergencia del algoritmo LMS

Otro factor de interés en el funcionamiento del algoritmo LMS es su rapidez de convergencia. Suponiendo la misma hipótesis que para el desarrollo de la estabilidad del LMS, se cumple que el valor esperado del filtro implementado a través del LMS converge al valor óptimo según la relación exponencial [Clarkson-93], [Widrow-75]:

$$U_n(j) = (1 - a \cdot \lambda_j)^n \cdot U_0(j) \qquad \text{Ec. 2.40}$$

Aquí $U_n(j)$ es un factor proporcional a la diferencia entre las componentes j-ésimas del filtro de Wiener, o filtro óptimo, y las del valor esperado del filtro generado por el LMS.

Se deduce de la expresión que cada componente converge a su correspondiente valor óptimo con una velocidad diferente. Este problema se le conoce como el de la disparidad en los valores propios. Una secuencia cuya matriz de autocorrelación tenga una gran diferencia entre sus valores propios máximo y mínimo presentará dificultades para ser tratada con el algoritmo LMS.

Para que se produzca la convergencia de todas las componentes se debe cumplir:

$$|1 - a \cdot \lambda_{\max}| < 1 \qquad \text{Ec. 2.41}$$

Se pueden definir unas constantes de tiempo para la convergencia de cada componente dadas por:

$$t_j = \frac{1}{a \cdot \lambda_j} \qquad \text{Ec. 2.42}$$

La velocidad en la convergencia del algoritmo vendrá limitada por el menor valor propio de la matriz de autocorrelación, que dará la mayor constante de tiempo, esto es:

$$t = \frac{1}{a \cdot \lambda_{\min}} \qquad \text{.Ec. 2.43}$$

De aquí se deduce que una disminución en la constante de adaptación conlleva una ralentización de la convergencia del LMS.

La constante de adaptación también influye en la implementación práctica de este algoritmo con un microprocesador (o un DSP). Caraiscos y Liu, hacen un análisis de los errores de cuantificación en el LMS llegando a la conclusión que este error es inversamente proporcional a la constante de adaptación [Caraiscos-84]. En este mismo artículo se da una solución para esta dependencia: deben asignarse más bits para los coeficientes sinápticos que para los datos. Al dotar a los pesos del sistema de mayor precisión se permite conseguir una adaptación perfecta.

2.5. Variantes del LMS.

2.5.1. Variantes con signo.

2.5.1.1.Variante con signo en el error.

Existen aplicaciones del LMS donde el factor crítico de funcionamiento es la rapidez de cálculo del algoritmo. En estos casos se busca aumentar la velocidad de computación y reducir el hardware necesario. Para estos problemas se idearon las variantes con signo. La ventaja de usar estos algoritmos estriba en la reducción de multiplicaciones a operaciones sobre bits. En esta variante, la ecuación de los coeficientes queda como:

$$F_{n+1} = F_n + a \cdot sign[e(n)] \cdot X_n \qquad \text{Ec. 2.44}$$

Si la constante a se escoge como una potencia de dos, esta ecuación se reduce, a nivel hardware, a desplazamientos de bits y sumas por lo que la velocidad de cálculo aumenta.

Al calcular el signo del error cuantificamos el gradiente de J por lo que no calculamos su verdadero valor. Por esto, el funcionamiento del algoritmo se degrada dando un mayor tiempo en la convergencia o una mayor oscilación en el estado estable del algoritmo. El gran problema de este algoritmo radica en su estabilidad, difícil de evaluar a priori aunque existe numerosa bibliografía sobre el tema considerando diferentes tipos de entrada.

2.5.1.2.Signo entrada

Esta variante guarda mucha similitud con la anterior. Ahora no se considera el signo del error sino que se calcula el signo de la entrada por lo que la ecuación que fija los coeficientes del filtro adaptativo queda como:

$$F_{n+1} = F_n + a \cdot e(n) \cdot sign[X_n] \qquad \text{Ec. 2.45}$$

Aunque los dos algoritmos se parezcan, presentan diferentes características de funcionamiento. En el signo error, cambiamos el tamaño de paso para la búsqueda del mínimo pero no la dirección del gradiente, hecho que sí ocurre en éste. Además, se ha demostrado que existen señales para las que el LMS es estable y este algoritmo es inestable aún variando la constante de adaptación [Clarkson-93], [Sethares-88].

2.5.1.3.Algoritmo signo-signo

Este algoritmo es el que menos requerimientos de hardware necesita de las variantes con signo del LMS. Aquí se calcula el signo del error y la entrada antes de hacer su producto. Si se toma como constante de adaptación una potencia de dos el algoritmo es muy sencillo de implementar a nivel hardware.

La ecuación de los coeficientes del sistema queda:

$$F_{n+1} = F_n + a \cdot sign[e(n)] \cdot sign[X_n] \qquad \text{Ec. 2.46}$$

Este algoritmo es el más sencillo de los expuestos en este apartado por lo que es el de uso más extendido de entre las variantes con signo. Sin embargo, es el menos robusto numéricamente, apareciendo inestabilidades incluso con secuencias periódicas de longitud tres [Clarkson-93], [Dasgupta-90].

2.5.2. Variantes con el estimador del gradiente filtrado.

2.5.2.1.Filtrado lineal (ALMS).

Uno de los inconvenientes del LMS es su sensibilidad al ruido que conlleva un valor esperado del error cuadrático medio superior al mínimo, ya que este ruido produce un desajuste en los coeficientes del filtro adaptativo frente a los del óptimo. Estas variantes tratan de reducir la incidencia del nivel de ruido en la determinación de los coeficientes.

En la obtención del LMS el gradiente de J se calcula mediante una estimación para cada instante:

$$\nabla_{est} J = -2 \cdot e(n) \cdot X_n \qquad \text{Ec. 2.47}$$

Si en un determinado momento aparece ruido no deseado que perturbe las entradas, éste se reflejará en el error e(n) produciendo la inestabilidad del algoritmo. Para eliminar este problema hay que dotarle a la estimación del gradiente de mayor robustez numérica. La manera de conseguirlo es filtrar esa componente de ruido antes de evaluar la estimación [Clarkson-93], [Roy-90], [Williamson-93], [Clarkson-89].

Uno de los métodos consiste en promediar las estimaciones más recientes. Esto es equivalente a aplicarles un filtro pasa-baja. Al promediar se reduce la posible disparidad que puede aparecer entre las estimaciones a causa del ruido. Se podrían usar otros tipos de filtros más sofisticados, sin embargo esto aumentaría el tiempo de computación del algoritmo, característica muy importante en los sistemas en tiempo real.

Las ecuaciones del LMS promediado (averaged LMS, ALMS) son las mismas que para el LMS cambiando solamente la correspondiente al cálculo de los coeficientes del filtro, que queda:

$$F_{n+1} = F_n + \frac{a}{N} \cdot \sum_{j=n-N+1}^{n} e(j) \cdot X_j \qquad \text{Ec. 2.48}$$

Este algoritmo presenta una menor rapidez en la convergencia y menor capacidad para adaptarse a cambios en las entradas que el LMS. Un estudio más profundo de este algoritmo lo realizan Haweel y Clarkson en [Clarkson-89].

2.5.2.2.*Momentum* LMS (MLMS)

La expresión que da los coeficientes del filtro adaptativo en el LMS es:

$$F_{n+1} = F_n + a \cdot e(n) \cdot X_n \qquad \text{Ec. 2.49}$$

a esta expresión se añade un término más que tiene como valor:

$$\mu \cdot [F_n - F_{n-1}] \qquad \text{Ec. 2.50}$$

por lo que se obtiene:

$$F_{n+1} = F_n + a \cdot e(n) \cdot X_n + \mu \cdot [F_n - F_{n-1}] \qquad \text{Ec. 2.51}$$

El significado de añadir este término se puede ver de una forma intuitiva. Este factor nos da información sobre el incremento en los coeficientes del filtro entre dos instantes sucesivos. Si el incremento es grande nos indica que estamos lejos del mínimo. Así pues para acelerar la convergencia se puede sumar a los coeficientes del filtro una fracción de este incremento que viene fijada por el parámetro μ.

No obstante, cerca del mínimo esta suma es perjudicial pues produce una mayor lentitud en la convergencia. Una solución intermedia es considerar μ igual a cero para un cierto valor, pequeño, del incremento en los coeficientes del filtro. Con esta acción volvemos a tener el algoritmo LMS.

Otra interpretación menos intuitiva del MLMS se obtiene de [Clarkson-93] [Roy-90]. El MLMS se puede considerar como un sistema en cascada, formado por dos subsistemas: un filtro que actúa sobre las estimaciones del gradiente y el LMS. El filtro es pasa-baja si μ tiende a 1 y pasa-alta si μ tiende a -1. Roy y Shynk realizan un estudio sobre las características de funcionamiento de esta variante del LMS: convergencia, estabilidad y desajuste de los coeficientes cerca del mínimo.

2.5.2.3.Filtrado no lineal: LMS con cálculo de la mediana.

Los dos algoritmos anteriores realizaban un filtrado lineal de la estimación del gradiente de J. Este tipo de filtrado presenta problemas de convergencia y estabilidad en presencia de ruido de tipo impulsional [Clarkson-89], [Clarkson-93]. Esta nueva variante resuelve el problema calculando la mediana de las N estimaciones más recientes del gradiente de J.

La ecuación de los coeficientes del filtro es:

$$F_{n+1} = F_n + a \cdot mediana\,[e(n) \cdot X_n, e(n-1) \cdot X_{n-1} \ldots . e(n-N+1) \cdot X_{n-N+1}] \qquad \text{Ec. 2.52}$$

Donde, dada una secuencia C_n cuyos elementos están ordenados:

$$c(1) < c(2) < \ldots . < c(2 \cdot N - 1) \qquad \text{Ec. 2.53}$$

la mediana de esta secuencia es:

$$mediana\, C_n = c(N) \qquad \text{Ec. 2.54}$$

De la definición de mediana y de la ecuación de actualización de los pesos se deducen importantes características del funcionamiento de este algoritmo.

Si en un instante determinado, un ruido de tipo impulsional interfiere nuestro sistema, siempre que el valor de N (número de elementos considerados para calcular la mediana) sea lo suficientemente grande, este ruido no se verá reflejado en los

coeficientes del filtro adaptativo. Si la duración del ruido es mayor que N/2 este algoritmo no lo elimina, apareciendo un mal funcionamiento.

Otro problema añadido es la rapidez de computación del algoritmo. Para aumentar la velocidad de cálculo conviene considerar N pequeño sin embargo, esta elección limita la eficacia del método a la cancelación de perturbaciones muy breves.

2.5.3. **Variantes rápidas.**

2.5.3.1.LMS con constante de adaptación variable

Se trata de un tipo de algoritmo LMS con una constante que cambia para cada instante. En [Clarkson-93] y como ejemplo de este tipo de algoritmos, la constante de adaptación vale:

$$a_n = \frac{1}{n+c} \qquad \text{Ec. 2.55}$$

donde c es una constante de valor pequeño.

Esta elección de la constante permite tener una gran velocidad de convergencia pues pequeños valores de n dan una constante de adaptación alta. Ocurre lo contrario para n grande lo cual es conveniente pues estamos cerca del sistema óptimo y una constante alta produce mayor desajuste en los coeficientes del filtro. El problema de esta variante viene con los cambios en la señal de entrada en un instante n (con n grande) pues el sistema no podrá adaptarse a ellos porque la velocidad de adaptación es menor al haberse reducido la constante.

Otras variantes de este tipo se encuentran analizadas e implementadas en [Evans-93]. La constante de adaptación se decrementa si ocurre un número determinado de cambios (m_o) de signo en la estimación de J, o se aumenta si no se producen cambios de signo en otro intervalo de tiempo fijado a priori (m_1). Se definen dos formas para aumentar o disminuir la constante:

$$a_n = a_{n-1} + \alpha$$
$$a_n = \frac{a_n}{\alpha} \qquad \text{Ec. 2.56}$$

donde α es mayor que uno, escogiéndose como potencia de dos para eliminar la carga computacional del algoritmo, así las multiplicaciones quedan reducidas a desplazamientos de bits. La otra forma que da el artículo para este algoritmo es:

$$a_n = a_{n-1} + \alpha$$
$$a_n = a_{n-1} - \alpha \qquad \text{Ec. 2.57}$$

Para evitar problemas de inestabilidad se definen a_{min} y a_{max}, de tal manera que si la constante excede estos límites, toma estos valores previamente definidos.

Estos algoritmos presentan una gran sensibilidad ante la elección de sus parámetros (a_{min}, a_{max}, m_0, m_1, y α). Si la elección es buena, estas variantes presentan un mejor funcionamiento que el LMS con un pequeño incremento en la complejidad

del hardware implicado en su desarrollo. Sin embargo, cuando la relación señal-ruido es baja presenta peor funcionamiento que el LMS. Se puede ver un análisis completo de estas variantes y una implementación práctica en el trabajo desarrollado por Evans.

2.5.3.2.LMS normalizado (NLMS).

En el LMS la constante de adaptación y la energía de la señal de entrada están relacionadas. Se comprobó en el capítulo anterior que cambios en la señal podrían conducir a la inestabilidad del algoritmo. Esto resulta especialmente crítico cuando no se conocen las características de la señal de entrada, así como si se trabaja en ambientes ruidosos.

Para evitar esta dependencia, la constante de adaptación toma el valor:

$$a_n = \frac{\mu}{\sum_{r=1}^{L} x_{n-r+1}^2} \qquad \text{Ec. 2.58}$$

donde el denominador es la energía de la señal de entrada y μ un parámetro. El algoritmo cuya constante de adaptación toma la expresión anterior recibe el nombre de normalizado.

Al normalizar desaparece la dependencia de la constante con la energía de entrada por lo que los límites para la estabilidad del algoritmo quedan [Slock-93]:

$$0 < \mu < 2 \qquad \text{Ec. 2.59}$$

Este algoritmo presenta varias mejoras frente al LMS:

- Mayor rapidez de convergencia.
- No se necesita un conocimiento previo de la energía de la señal de entrada.
- Mejor ajuste de los coeficientes del filtro adaptativo a los del óptimo.

La principal desventaja del NLMS radica en su mayor tiempo de cálculo debido a la necesidad de obtener la energía de la señal de entrada en cada instante para el cálculo de la constante de adaptación. En este cálculo hay que asegurarse de que la energía de la señal es distinta de cero ya que si esto ocurre no se puede obtener la constante, porque se tendría una división por cero. Para salvar este problema se le añade una constante de pequeño valor al denominador de la expresión de la constante de adaptación.

Se puede demostrar que la mejor constante de adaptación es la correspondiente al algoritmo normalizado con μ igual a uno [Soria-98]. Para ello se desarrolla en serie de Taylor el error en el instante n+1 en función del error en el instante anterior, es decir:

$$e(n+1) = e(n) + \sum_{i=1}^{L} \frac{\partial e(n)}{\partial f(i)} \Delta f(i) \qquad \text{Ec. 2.60}$$

Los términos de orden dos y superiores no aparecen pues el error tiene una dependencia lineal con los coeficientes del filtro:

$$e(n) = d(n) - \sum_{r=1}^{L} f(r) \cdot x_{n-r+1} \qquad \text{Ec. 2.61}$$

así pues:

$$\frac{\partial e(n)}{\partial f(i)} = - x_{n-i+1} \qquad \text{Ec. 2.62}$$

y, considerando la expresión del LMS:

$$f_{n+1}(i) = f_n(i) + a \cdot e(n) \cdot x_{n-i+1} \qquad \text{Ec. 2.63}$$

entonces:

$$\Delta f(i) = a \cdot e(n) \cdot x_{n-i+1} \qquad \text{Ec. 2.64}$$

por lo que finalmente se obtiene:

$$e(n+1) = e(n) \cdot [1 - a \cdot \sum_{i=1}^{L} x_{n-i+1}^2] \qquad \text{Ec. 2.65}$$

Si la expresión anterior se eleva al cuadrado, se deriva y se iguala a cero obtenemos que la constante que minimiza $e^2(n+1)$ es:

$$a_n = \frac{1}{\sum_{i=1}^{L} x_{n-i+1}^2} \qquad \text{Ec. 2.66}$$

Esta constante es la misma que la correspondiente al algoritmo normalizado (NLMS) con μ igual a 1. Slock realiza un análisis más profundo de este algoritmo, tanto de una forma teórica como experimental [Slock-93].

2.5.4. Extensión no lineal de la Adalina.

En el sistema planteado la salida es lineal con respecto de los coeficientes del filtro. Este tipo de salida es útil en problemas de clasificación en los que se tiene un determinado rango de variación en la señal deseada. Sin embargo, en problemas de clasificación binaria, la señal deseada toma un par de valores concretos (por ejemplo ±1 o 0 y 1). En este caso se podría plantear una función como la utilizada por el perceptrón, la función signo. El problema de usar esta función es que no es derivable por lo que no se puede aplicar la regla delta que actualiza los pesos sinápticos de acuerdo a la siguiente relación:

$$\Delta w_n = -\alpha \cdot \nabla_{W_n} J \qquad \text{Ec. 2.67}$$

Lógicamente para aplicar esta regla necesitamos una función derivable. La función derivable que más se parece a la función signo es la sigmoide (rango de salida de 0 a 1) y la tangente hiperbólica (rango de salida entre ±1). En este caso, nuestra estructura queda definida por la siguiente figura:

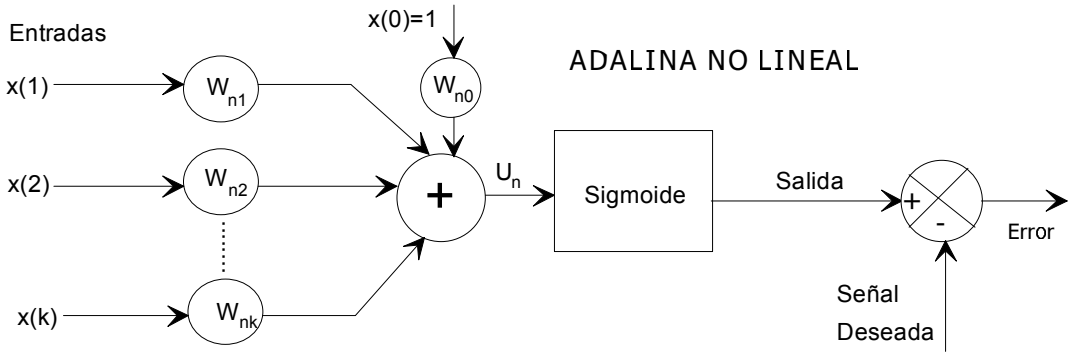


Figura 2.9. Esquema de una adalina no lineal.

Tendremos las siguientes ecuaciones:

$$\text{salida}(n) = f(\sum_{s=0}^{k} W_{ns} \cdot x(s)) \qquad \text{Ec. 2.68}$$

$$\text{error}(n)=\text{deseada}(n)-\text{salida}(n) \qquad \text{Ec. 2.69}$$

Recordando la definición de J (error(n) al cuadrado), la ecuación de actualización de los pesos queda como:

$$\Delta W_{ns} = 2 \cdot \alpha \cdot x(s) \cdot f'(\text{salida}(n)) \qquad \text{Ec. 2.70}$$

si comparamos la ecuación 17 con la ecuación 15 (que da la actualización de los pesos de una adalina) vemos que la diferencia que aparece es el término de la derivada de la función no lineal aplicada. Las funciones no lineales utilizadas habitualmente, tangente hiperbólica y sigmoide, presentan buenas propiedades en cuanto a la derivada se refiere ya que se cumple:

$$\begin{aligned} &\text{SIGMOIDE} \Rightarrow f' = f \cdot (1-f) \\ &\text{TANG.HIPERB.} \Rightarrow f' = \frac{1}{2} \cdot \left(1-f^2\right) \end{aligned} \qquad \text{Ec. 2.71}$$

por lo que la ecuación de actualización de los pesos queda como:

$$\begin{aligned} \Delta W_{ns} &= 2 \cdot \alpha \cdot x(s) \cdot \text{salida}(n) \cdot [1-\text{salida}(n)] \\ \Delta W_{ns} &= \alpha \cdot x(s) \cdot [1-\text{salida}^2(n)] \end{aligned} \qquad \text{Ec. 2.72}$$

observando la ecuación 18 aparece claramente expuesto un fenómeno que volveremos a tener al estudiar el perceptrón multicapa: la saturación neuronal. Si nos fijamos en dicha ecuación, cuando la salida tome los valores extremos de la función no lineal (0/1 para la sigmoide, ±1 para la tangente hiperbólica) el incremento de los pesos será cero (no se actualizan) independientemente del error cometido. Para evitar este problema existen dos sencillas soluciones:

a) Inicializar los pesos con valores pequeños, positivos y negativos, de tal forma que, al principio, la salida del sistema quede lejos de los valores extremos de la función.

b) Añadir una constante de pequeño valor a la derivada de la función. El problema de esta solución es su mal funcionamiento cuando los pesos sinápticos se corresponden con los del sistema óptimo.

Hasta ahora hemos estudiado el aprendizaje supervisado en una neurona; pasaremos a continuación al otro gran tipo de aprendizaje: el aprendizaje no supervisado.

2.5.5. Aprendizaje Hebbiano.

En 1949 Hebb postuló un mecanismo de aprendizaje basado en el siguiente criterio fisiológico: "Cuando un axón A está suficientemente cerca como para conseguir excitar a una célula B y repetida o persistentemente toma parte en su actuación algún proceso de crecimiento o cambio metabólico tiene lugar en una o ambas células, de tal forma que la eficiencia de A, cuando la célula a activar es B aumenta" [Hebb-49]. El aprendizaje hebbiano consiste básicamente en el ajuste de los pesos de las conexiones de acuerdo con la correlación entre los valores de entrada y salida de cada neurona. Particularicemos esta regla para la neurona más sencilla, sin función de activación. De acuerdo a la regla expuesta la regla de actualización de los pesos de la neurona (se ha escogido una por sencillez) sería:

$$\Delta w_i = \alpha \cdot x_i \cdot y \qquad \text{Ec. 2.73}$$

con:

$$y = \sum_{k=0}^{L} w_k \cdot x_k \qquad \text{Ec. 2.74}$$

A partir de una regla tan simple se pueden extraer una serie de conclusiones. Si hacemos corresponder la ecuación 19 con la actualización de los pesos de una neurona se tiene, de acuerdo a la regla delta:

$$\Delta w_{ij} = -\alpha \cdot \nabla J \qquad \text{Ec. 2.75}$$

Se llega entonces a que la función a minimizar en este aprendizaje, y con este tipo de neurona (lineal, sin función de activación) es:

$$J = -\frac{1}{2} \cdot y^2 \qquad \text{Ec. 2.76}$$

Si tenemos en cuenta que las entradas pueden ser señales aleatorias a las que se les ha eliminado el valor medio lo que se busca mediante la minimización de la ecuación 22 es la maximización de la varianza de la salida; en definitiva, con este aprendizaje y con este tipo de neurona lo que se calculará será la dirección (no olvidemos que estamos calculando un vector de pesos) para la cual la varianza de la salida es máxima. Otra conclusión que se extrae de la función a minimizar es que este algoritmo con este tipo de neurona hace crecer indefinidamente los pesos ya que la expresión 22 es menor conforme aumenta la salida lo que produce la inestabilidad del algoritmo. ¿Qué variaciones deberíamos introducir en esta actualización para que los pesos no crecieran de forma desproporcionada? La primera variación evidente que aparece es dividir por la norma de los pesos en cada iteración, de tal forma que la nueva norma de estos pesos sea uno, evitándose su crecimiento sin control:

$$\mathbf{w_{n+1}} = \frac{\mathbf{w_n} + \alpha \cdot \mathbf{x_n} \cdot \mathbf{y_n}}{\left\| \mathbf{w_n} + \alpha \cdot \mathbf{x_n} \cdot \mathbf{y_n} \right\|} \qquad \text{Ec. 2.77}$$

donde w_n, x_n e y_n son el vector de pesos, vector de entrada y la salida de la neurona respectivamente definidos como:

$$\begin{aligned} w_n &= [w_{n1}\ w_{n2}\ .....w_{nk}] \\ x_n &= [x_{n1}\ x_{n2}\ .....x_{nk}] \\ y_n &= w_n^t \cdot x_n \end{aligned} \qquad \text{Ec. 2.78}$$

De la expresión 23 se deduce que el coste computacional de esta variación es alto ya que hay que calcular la norma de un vector y luego dividir por ella. Es necesario pues simplificar esta expresión. En primer lugar se tendrá en cuenta que:

$$\left\| \mathbf{w_n} + \alpha \cdot \mathbf{x_n} \cdot \mathbf{y_n} \right\|^2 = \left(\mathbf{w_n} + \alpha \cdot \mathbf{x_n} \cdot \mathbf{y_n} \right) \cdot \left(\mathbf{w_n} + \alpha \cdot \mathbf{x_n} \cdot \mathbf{y_n} \right) \qquad \text{Ec. 2.79}$$

donde el producto de vectores es un producto escalar. Si desarrollamos el segundo término se llega a:

$$\left\| \mathbf{w_n} + \alpha \cdot \mathbf{x_n} \cdot \mathbf{y_n} \right\|^2 = \left\| \mathbf{w_n} \right\|^2 + 2 \cdot \alpha \cdot \mathbf{w_n^t} \cdot \mathbf{x_n} \cdot \mathbf{y_n} + \alpha^2 \cdot \left\| \mathbf{x_n} \right\|^2 \cdot \mathbf{y_n^2} \qquad \text{Ec. 2.80}$$

Si suponemos que la constante es pequeña podemos despreciar el último término, además en el segundo aparece la salida y_n. Por otro lado, la norma del vector en el instante n vale 1 (se normaliza en cada instante). Según todo lo dicho la expresión 25 queda como:

$$\left\| \mathbf{w_n} + \alpha \cdot \mathbf{x_n} \cdot \mathbf{y_n} \right\|^2 \approx \mathbf{1} + \mathbf{2} \cdot \alpha \cdot \mathbf{y_n^2} \qquad \text{Ec. 2.81}$$

Si se combinan las ecuaciones 25 y 26 se llega a:

$$\mathbf{w_{n+1}} = \left(\mathbf{w_n} + \alpha \cdot \mathbf{x_n} \cdot \mathbf{y_n} \right) \cdot \left(\mathbf{1} + \mathbf{2} \cdot \alpha \cdot \mathbf{y_n^2} \right)^{\frac{1}{2}} \qquad \text{Ec. 2.82}$$

Como la constante es pequeña, en el último término entre paréntesis se puede usar la siguiente aproximación:

$$\left(\mathbf{1} + \mathbf{x} \right)^{\beta} \cong \mathbf{1} + \beta \cdot \mathbf{x} \qquad \text{Ec. 2.83}$$

sustituyendo en la ecuación 27 queda:

$$\mathbf{w_{n+1}} = \left(\mathbf{w_n} + \alpha \cdot \mathbf{x_n} \cdot \mathbf{y_n} \right) \cdot \left(\mathbf{1} - \alpha \cdot \mathbf{y_n^2} \right) \qquad \text{Ec. 2.84}$$

Si realizamos el producto y no tenemos en cuenta los términos cuadráticos de la constante se llega a:

$$w_{n+1} = w_n + \alpha \cdot x_n \cdot y_n - \alpha \cdot w_n \cdot y_n^2 \qquad \text{Ec. 2.85}$$

Esta regla de aprendizaje es conocida como Regla de Oja, en honor a su descubridor [Hassoun-95]. Veamos a continuación una aplicación de esta regla. En la Figura 2.10 aparece un conjunto de vectores de entrenamiento a los que se le va aplicar esta regla de aprendizaje. Las dos líneas muestran la dirección proporcionada por el vector asociado al mayor valor propio de la matriz de autocorrelación de la señal y la dirección proporcionada por los pesos tras el periodo de adaptación según la regla definida por la regla de Oja.

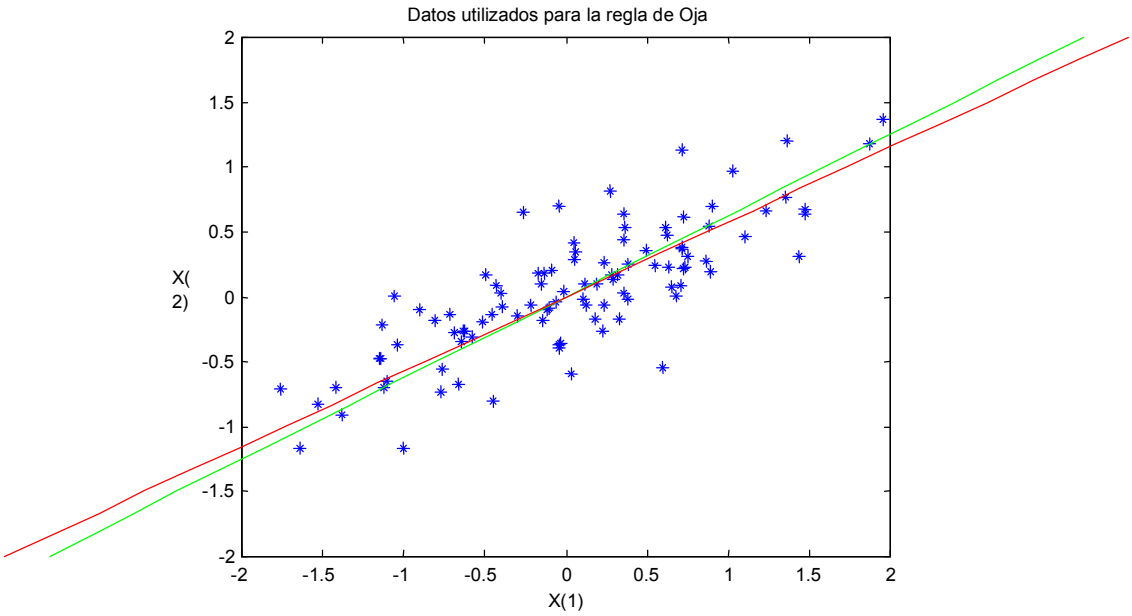


Figura 2.10. Resultados obtenidos con la regla de Oja.

Si se estudia detenidamente la expresión de actualización de los pesos según la regla de Oja, encontramos la razón por la que el sistema no es inestable, tenemos una realimentación negativa de los pesos; dicha realimentación controla el crecimiento de los pesos. Oja, posteriormente amplía su regla usando neuronas con una función de activación no lineal de tal forma que la expresión de actualización de los pesos queda:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \alpha \cdot \mathbf{x}_n \cdot \mathbf{v}_n - \alpha \cdot \cdot \mathbf{w}_n \cdot \mathbf{v}_n^2 \qquad \text{Ec. 2.86}$$

Donde v_n viene definido por:

$$\begin{aligned} \mathbf{v}_n &= g(y_n) \\ \mathbf{y}_n &= \mathbf{w}_n^t \cdot \mathbf{x}_n \end{aligned} \qquad \text{Ec. 2.87}$$

siendo g una función no lineal.

Esta regla es conocida como regla de Yuille [Hassoun-95]. De nuevo nos aparece una realimentación negativa en los pesos que controla el crecimiento de éstos.

3

El Perceptón Multicapa

3.1. Introducción.

El perceptrón multicapa es la red neuronal artificial más conocida y con un mayor número de aplicaciones. Su historia comienza en 1958 cuando Rosenblatt publica los primeros trabajos sobre un modelo neuronal y su algoritmo de aprendizaje que llama perceptrón [Rosenblatt-58].

El perceptrón (explicado en el anterior capítulo) está formado por una única neurona, por lo que su utilización está limitada a la clasificación de patrones en dos clases. Si se expande esta capa de salida con más de una neurona se podrán clasificar más de dos clases aunque con la limitación demostrada por Minsky y Papert en 1969 de que estas clases deben ser linealmente separables [Minsky-69]. Esta limitación es bastante problemática porque imposibilita al perceptrón para resolver un problema tan sencillo como el de la función lógica XOR de 2 entradas, cuya tabla de verdad es la mostrada en la tabla1:

x	y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

La Figura 3.1 muestra que las dos clases son no linealmente separables representándose los ceros por círculos y los unos por cuadrados.

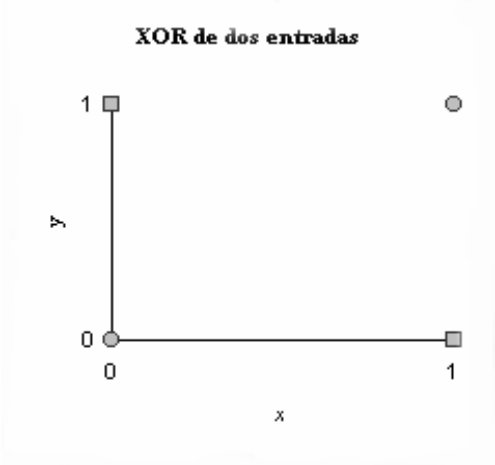


Figura 3.1. Función lógica XOR de dos entradas.

La separación de clases no linealmente separables se consigue introduciendo una capa de neuronas entre la salida y la entrada. Aparece entonces el problema de asignar un "error" durante el proceso de aprendizaje a estas neuronas; este problema es conocido en la teoría de los sistemas conexionistas como el problema de la asignación de crédito (assignment credit). Veremos como fue brillantemente resuelto por una serie de investigadores de forma independiente pero que alcanzó su difusión en el trabajo de Rumelhart, Hinton y Williams [Rumelhart-86].

3.2. Arquitectura del perceptrón multicapa.

El perceptrón multicapa (MultiLayer Perceptron, MLP) es una red formada por una capa de entrada, al menos una capa oculta y una capa de salida; su estructura se muestra en la Figura 3.2.

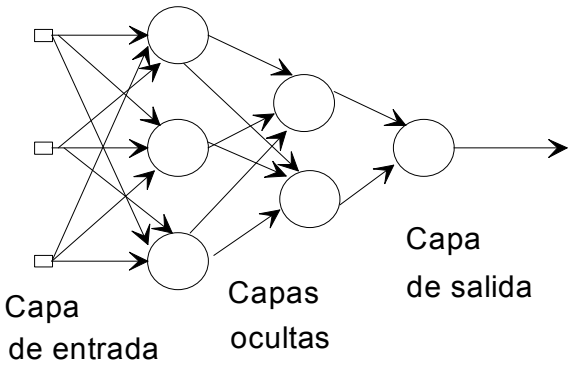


Figura 3.2. Estructura de un perceptrón multicapa.

Las cuatro características fundamentales de esta arquitectura son las siguientes:

1. Se trata de una estructura altamente no lineal.
2. Presenta tolerancia a fallos.
3. El sistema es capaz de establecer una relación entre dos conjuntos de datos.
4. Existe la posibilidad de realizar una implementación hardware.

De la Figura 3.2 se desprende que es una estructura formada por nodos o neuronas que propagan la señal hacia la salida. Las conexiones entre las neuronas se denominan pesos sinápticos, que son optimizados por el algoritmo de aprendizaje.

La propagación se realiza de manera que cada neurona hace una combinación lineal de las señales procedentes de las neuronas de la capa anterior siendo los coeficientes de esta combinación los pesos sinápticos. A continuación, aplica una función de activación no lineal. Estas neuronas se conocen como neuronas no lineales y aseguran la alta no linealidad en el MLP. En la Figura 3.3 se muestra el esquema de una neurona no lineal:

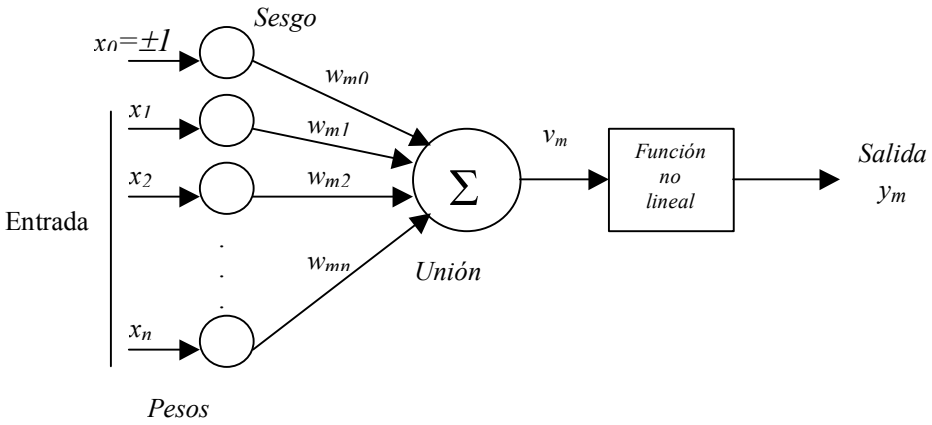


Figura 3.3. Esquema de una neurona no lineal.

La función no lineal que se aplica a la salida de la neurona se conoce como función de activación y ha de cumplir únicamente dos condiciones, ser continua y diferenciable. Las tres funciones más utilizadas son:

- Signo. La primera utilizada, con ella se planteó el modelo de neurona de McCulloch-Pitts y el perceptrón de Rosenblatt [McCulloch-43], [Rosenblatt-58]. Esta función queda definida por:

$$\text{sign}(x) = \begin{cases} 1 \text{ si } x > 0 \\ -1 \text{ si } x \leq 0 \end{cases} \qquad \text{Ec. 3.1}$$

  Sus malas propiedades matemáticas (no es diferenciable en 0) obligó a su abandono. Las dos siguientes derivan de esta función signo.

- Sigmoide: Su expresión se muestra en la ecuación 2 y toma valores entre 0 y 1 para una variación de la variable independiente x entre $+\infty$ y $-\infty$.

$$f_1(x) = \frac{1}{1+e^{-x}} \qquad \text{Ec. 3.2}$$

- Tangente hiperbólica: de manera similar a lo que ocurre con la función sigmoide, la tangente hiperbólica cuya expresión se muestra en la ecuación 3 toma valores entre –1 y +1 para una variación de la variable independiente entre +∞ y -∞. En este caso, con –1 se codificaría la mínima actividad de la neurona y con +1 la máxima.

$$f_2(x) = \frac{1-e^{-x}}{1+e^{-x}} \qquad \text{Ec. 3.3}$$

- Función lineal a tramos: en este caso se utiliza una función más sencilla que las anteriores formada por tres tramos lineales unidos para formar una función no lineal. Lo más habitual es que esta función pueda tener valores comprendidos entre –1 y +1 como se muestra en la ecuación 4, ya que la expresión resulta más sencilla, pero también puede considerarse una función que varíe entre 0 y 1 que tendría una expresión como la que se muestra en la ecuación 5.

$$f_3(x) = \begin{cases} -1 & si \quad x < -1 \\ x & si \quad -1 < x < 1 \\ 1 & si \quad x > 1 \end{cases} \qquad \text{Ec. 3.4}$$

$$f_4(x) = \begin{cases} 0 & si \quad x < -1 \\ 0.5 + 0.5x & si \quad -1 < x < 1 \\ 1 & si \quad x > 1 \end{cases} \qquad \text{Ec. 3.5}$$

En cuanto a la forma de disponer estos elementos básicos, las neuronas tenemos un gran número de posibilidades. El número de neuronas que forman las capas de entrada y salida está determinado por el problema, mientras que el número de capas ocultas y de neuronas en cada una de ellas no está fijado ni por el problema ni por ninguna regla teórica por lo que el diseñador es quien decide esta arquitectura en función de la aplicación que se va a hacer de la red y tras realizar un barrido en los valores de estos dos factores. Únicamente está demostrado que dado un conjunto de datos conexo, con una sola capa oculta es posible establecer una relación entre el conjunto de datos, aunque no se especifica el número de neuronas necesarias. Si el conjunto no es conexo hacen falta al menos dos capas ocultas [Kolmogorov-57].

Existen algunos métodos empíricos para la caracterización de las capas ocultas pero cada uno de ellos funciona bien únicamente en un tipo determinado de problemas. Intuitivamente, es lógico pensar que ante estos inconvenientes, la solución idónea es implementar una red con muchas capas ocultas y una gran cantidad de neuronas en cada una de ellas; sin embargo, esto tiene una serie de inconvenientes:

Aumento de la carga computacional, lo que implica una mayor dificultad de implementación en tiempo real y un crecimiento en el tiempo de aprendizaje por parte de la red.

Pérdida en la capacidad de generalización: Al aumentar el número de neuronas en la capa oculta, aumenta el número de pesos sinápticos, por lo que la red está caracterizada por más parámetros; esto permite una mejor modelización de los patrones utilizados pero se pierde capacidad de generalización ya que un patrón no usado en la modelización tendrá más dificultades a la hora de ajustarse a un modelo con un gran número de parámetros.

3.3. **Algoritmo de aprendizaje Backpropagation.**

Antes de explicar ningún algoritmo de aprendizaje en particular, hay que reseñar que cada uno de los algoritmos que se van a ver tiene unas especiales características que lo hacen adecuado a unas determinadas circunstancias. El diseñador en función de las circunstancias particulares en las que se encuentre (problema a resolver, tipo de plataforma disponible, etc.) debe escoger la más adecuada. De todos modos, las características que debería cumplir un algoritmo óptimo son:

- Eficacia.
- Robustez, para adaptarse a diferentes problemas.
- Independencia respecto a las condiciones iniciales.
- Alta capacidad de generalización.
- Coste computacional bajo.
- Sencillez en los razonamientos empleados.

Existen diferentes algoritmos de aprendizaje que optimizan las conexiones entre las neuronas según el error que esté cometiendo la red, entendiendo por error la diferencia que existe entre la salida ofrecida por la red y la salida deseada. Los más utilizados son los algoritmos por descenso de gradiente que se basan en la minimización o maximización de una determinada función. Lo más habitual es minimizar alguna función monótona creciente del error, como por ejemplo el valor absoluto del error o el error cuadrático medio; esta función a minimizar se denomina función de coste.

La función de coste más usada es la cuadrática [Bishop-96]:

$$J = \frac{1}{2M} \sum_{i=1}^{M} \sum_{j=1}^{N} e_j^{\,2}(i) \qquad \text{Ec. 3.6}$$

En esta expresión M es el número de patrones utilizados para entrenar la red y N el número de neuronas en la capa de salida. Esta función de coste supone una distribución de errores de tipo normal; situación que, normalmente se da en problemas de modelización.

También es muy utilizada la función de coste entrópica [Bishop-96]:

$$J = \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N}\left((1+d_j(i))\ln\left[\frac{1+d_j(i)}{1+o_j(i)}\right] + (1-d_j(i))\ln\left[\frac{1-d_j(i)}{1-o_j(i)}\right]\right) \qquad \text{Ec. 3.7}$$

Esta función de coste supone una distribución de errores de tipo binomial, que es la distribución que se da en un problema de clasificación.

Otra función de coste habitual es la que usa la norma de Minkowski, cuya expresión se muestra en la ecuación 8, y que dependiendo del valor de R, permite minimizar la función del error que más interese. En particular, podría pensarse en minimizar el error cuadrático al principio del entrenamiento (R=2) y el valor absoluto del error al final del mismo (R=1) [Bishop-96]:

$$J = \frac{1}{R\cdot N}\sum_{i=1}^{L}\sum_{j=1}^{N}\left|d_j(i)-o_j(i)\right|^R \qquad \text{Ec. 3.8}$$

Una vez definida la función de coste a utilizar hay que aplicar un procedimiento de minimización de dicha función; este proceso recibe el nombre de aprendizaje de la red; existen dos tipos:

a) On-Line: el aprendizaje se realiza patrón a patrón. Durante todo el entrenamiento se le pasa a la red cada entrada junto con su salida deseada. Se mide el error y en función de éste se adaptan los pesos sinápticos mediante el algoritmo de aprendizaje escogido.

b) Batch: el aprendizaje se realiza época a época. Una época supone el paso de todos los patrones de entrenamiento por la red. En este tipo de aprendizaje, se le pasa a la red todos los patrones de entrenamiento se mira el error total cometido y se adapta los pesos en función de este valor del error promediado según el número de patrones [Haykin-98].

El algoritmo de aprendizaje backpropagation es un algoritmo de descenso por gradiente que retropropaga las señales desde la capa de salida hasta la capa de entrada optimizando los valores de los pesos sinápticos mediante un proceso iterativo que se basa en la minimización de la función de coste. Por ello, puede dividirse el algoritmo en dos fases:

1. Propagación hacia delante: se propagan las señales desde la capa de entrada hasta la de salida, determinándose la salida de la red y el error cometido al comparar ésta con el valor de la salida deseada que se le facilita a la red durante la etapa de aprendizaje.

2. Propagación hacia atrás: En función de los errores cometidos en la capa de salida, el algoritmo se encarga de optimizar los valores de los pesos sinápticos que determinan las conexiones entre las neuronas mediante la retropropagación del error desde la capa de salida a la de entrada a través de las sucesivas capas ocultas.

Para la descripción de este algoritmo se tiene en cuenta el esquema de una neurona y el error como la diferencia existente entre la salida deseada y la salida observada. Se obtendrá el algoritmo para el caso más sencillo de una red formada por una capa de entrada, una capa oculta y una de salida; la conexión entre una

neurona oculta y una de salida puede observarse en la Figura 3.4. La extensión para el caso de más de una capa oculta es inmediata, se tratará de realizar una retropropagación del error durante un número mayor de capas.

En lo sucesivo $\{x_1, ..., x_n\}$ serán las entradas a la red, es decir, que formarán la capa de entrada y constituirán las entradas a las neuronas de la capa oculta mientras que x_0 es la entrada que, dependiendo de que su valor sea +1 o –1, se denomina bias o umbral. Por otra parte $\{w_{m1}, ..., w_{mn}\}$ serán los pesos sinápticos que conectan las entradas con la neurona oculta m y w_{m0} es el sesgo (peso sináptico correspondiente a x_0):

$$v_m(t) = \sum_{i=0}^{n} w_{mi} \cdot x_i(t) \qquad \text{Ec. 3.9}$$

En la expresión anterior el índice m denota a la neurona y el índice t indica el número de iteración (en el aprendizaje on-line se corresponde con el número de patrón). Si se denota por φ la función de activación, la salida de la neurona viene dada por la siguiente expresión:

$$y_m(t) = \varphi_m(v_m(t)) \qquad \text{Ec. 3.10}$$

El conjunto $\{y_m\}$ que tiene tantos elementos como neuronas ocultas, definirá las salidas de las neuronas de la capa oculta y por tanto, se corresponderá con las entradas a las neuronas de la capa de salida. Los pesos sinápticos que conectan las entradas $\{y_m\}$ con la neurona p los denotaremos por $\{h_{p1}, ..., h_{pr}\}$ donde p denota a la neurona y r el número de neuronas ocultas, ahora el sesgo viene definido por h_{p0}:

$$z_p(t) = \sum_{j=0}^{r} h_{pj} \cdot y_j \qquad \text{Ec. 3.11}$$

Si ϕ es la función de activación, la salida de la neurona de salida será:

$$o_p(t) = \phi_p(z_p(t)) \qquad \text{Ec. 3.12}$$

Denotando por d_p el valor de la salida deseada, el error a la salida de la red queda definido por:

$$e_p = d_p - o_p \qquad \text{Ec. 3.13}$$

El proceso iterativo que actualiza los pesos sinápticos para llevarlos a su valor óptimo es el mismo que se vio en el tema anterior: la regla delta:

$$\Delta\xi(t) = -\alpha \frac{\partial J}{\partial \xi(t)} \qquad \text{Ec. 3.14}$$

siendo ξ el peso sináptico que desea actualizarse y α la constante de adaptación que mide la velocidad de aprendizaje de la red.

En el proceso de retropropagación, los primeros pesos en actualizarse son los de la capa de salida. La derivada parcial de la función de coste respecto a estos pesos sinápticos puede expresarse teniendo en cuenta la regla de la cadena del siguiente modo:

$$\frac{\partial J}{\partial h_{pj}(t)} = \frac{\partial J}{\partial e_p(t)} \frac{\partial e_p(t)}{\partial o_p(t)} \frac{\partial o_p(t)}{\partial z_p(t)} \frac{\partial z_p(t)}{\partial h_{pj}(t)} \qquad \text{Ec. 3.15}$$

La primera derivada depende de la función de coste que se haya escogido, por ejemplo para la función cuadrática que es la más habitual, el resultado será igual a $2 \cdot e_p(t)$. Por otra parte, la segunda de las derivadas es trivial a partir de la ecuación 13 y vale –1. La tercera derivada depende de la función de activación que se utilice ya que derivando la ecuación 12 se tiene que:

$$\frac{\partial o_p(t)}{\partial z_p(t)} = \phi_p'(z_p(t)) \qquad \text{Ec. 3.16}$$

Por último, de la ecuación 11 se deduce:

$$\frac{\partial z_p(t)}{\partial h_{pj}(t)} = y_j(t) \qquad \text{Ec. 3.17}$$

Es decir, que la actualización iterativa de los pesos de la capa de salida viene dada por:

$$\Delta h_{pj}(t) = 2 \cdot \alpha \cdot e_p(t) \cdot \phi_p'(z_p(t)) \cdot y_j(t) \qquad \text{Ec. 3.18}$$

que para el caso del sesgo se particulariza del siguiente modo en el caso de que se utilice como entrada y_0=+1:

$$\Delta h_{p0} = 2 \cdot \alpha \cdot e_p(t) \cdot \phi_p'(z_p(t)) \cdot 1 \qquad \text{Ec. 3.19}$$

Siguiendo con la retropropagación, se actualizan los pesos sinápticos de las neuronas de la capa oculta. La derivada parcial de la función de coste respecto a estos pesos se puede expresar mediante la regla de la cadena por:

$$\frac{\partial \mathrm{J}}{\partial \mathrm{w}_{mi}} = \sum_{\mathrm{p}} \frac{\partial \mathrm{J}}{\partial \mathrm{e}_{\mathrm{p}}(\mathrm{t})} \frac{\partial \mathrm{e}_{\mathrm{p}}(\mathrm{t})}{\partial \mathrm{o}_{\mathrm{p}}(\mathrm{t})} \frac{\partial \mathrm{o}_{\mathrm{p}}(\mathrm{t})}{\partial \mathrm{z}_{\mathrm{p}}(\mathrm{t})} \frac{\partial \mathrm{z}_{\mathrm{p}}(\mathrm{t})}{\partial \mathrm{y}_{\mathrm{m}}(\mathrm{t})} \frac{\partial \mathrm{y}_{\mathrm{m}}(\mathrm{t})}{\partial \mathrm{v}_{\mathrm{m}}(\mathrm{t})} \frac{\partial \mathrm{v}_{\mathrm{m}}(\mathrm{t})}{\partial \mathrm{w}_{\mathrm{mi}}(\mathrm{t})} \qquad \text{Ec. 3.20}$$

Realizando las correspondientes derivadas se tiene que la actualización de los pesos descrita en la ecuación 14 se puede poner como:

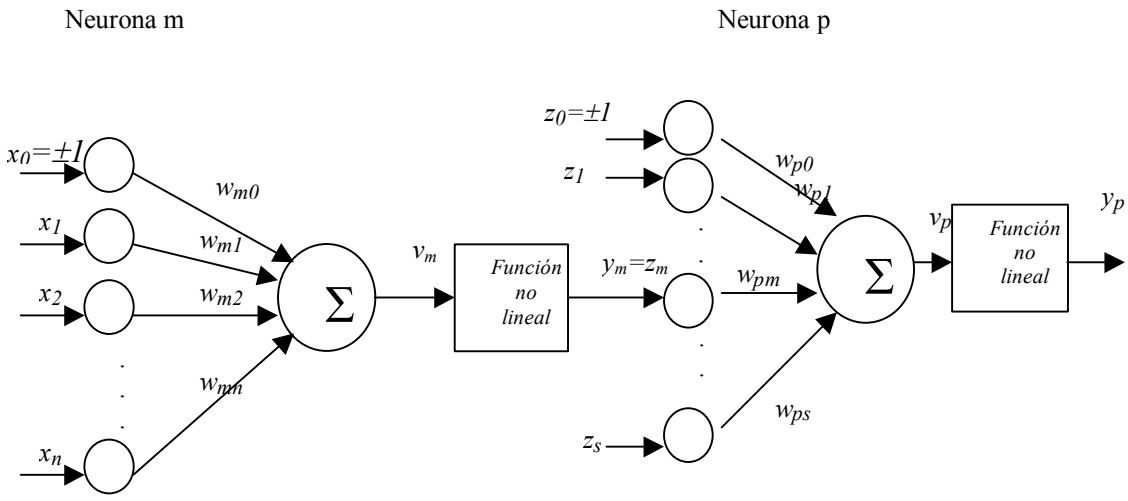
$$\Delta w_{mi}(t) = 2 \cdot \alpha \cdot \sum_p e_p(t) \cdot \phi_p'(z_p(t)) \cdot h_{pm}(t) \cdot \varphi_m'(v_m(t)) \cdot x_i(t) \qquad \text{Ec. 3.21}$$

Que se particulariza para el caso del sesgo de modo análogo a lo que sucedía con las neuronas de la capa de salida.

En el caso de que se tenga más de una capa oculta el proceso es exactamente el mismo, se ha de seguir la retropropagación atravesando todas las capas hasta llegar a la de entrada, de manera que cada una de estas capas ocultas añadirá el correspondiente factor al algoritmo de actualización de los pesos.

De las ecuaciones anteriores, se tiene que la actualización de los pesos depende de la señal de error $e_p(t)$. Esta señal de error tiene un tratamiento distinto dependiendo de la capa a la que pertenezcan las neuronas, es decir, que estén situadas en la capa de salida o en la capa oculta ya que mientras para las neuronas de la capa de salida esta función de error se puede obtener directamente: hay que hallar la diferencia entre la salida deseada y la observada per la red. Para las neuronas de la capa oculta esta función de error se obtiene mediante un método

indirecto que se basa en la conexión entre neuronas de distintas capas y la aplicación de la regla de la cadena como se ha visto. Utilizando la misma nomenclatura con la que se ha derivado al algoritmo backpropagation, como la neurona p pertenece a la capa de salida su función de error, que vendrá dada por d_p-y_p siendo d_p el valor de la salida deseada correspondiente, puede determinarse directamente.



Además del error, los otros dos factores que intervienen en la actualización de los pesos son la constante de adaptación cuyo significado y tratamiento es el mismo que en los sistemas adaptativos y las entradas a las neuronas.

De todas las expresiones anteriores se tiene que la actualización depende de la función de activación utilizada ya que la derivada, que aparece en la expresión de actualización de los pesos cambia. Para el caso de que la función de activación utilizada sea la sigmoide cuya expresión se muestra en la ecuación 1, se tiene que la derivada es igual a:

$$\varphi'(v(t)) = \varphi \cdot (1-\varphi) \qquad \text{Ec. 3.22}$$

Si la función de activación es la tangente hiperbólica, cuya expresión se muestra en la ecuación 2, la derivada pasa a ser la siguiente:

$$\varphi'(v(t)) = \frac{1}{2} \cdot (1-\varphi^2) \qquad \text{Ec. 3.23}$$

Por otro lado, si las funciones de activación utilizadas son lineales a tramos como las de las ecuaciones 3 y 4, la expresión de la correspondiente derivada es mucho más sencilla; para el caso de tener una función lineal a tramos entre –1 y 1 que simplifica la tangente hiperbólica, como la mostrada en la ecuación 3, se tiene que:

$$\varphi'(v(t)) = \begin{cases} 1 & si \quad -1 < x < 1 \\ 0 & en\ otro\ caso \end{cases} \qquad \text{Ec. 3.24}$$

Si se considera una función lineal a tramos ente 0 y 1 que simplifica la sigmoide, ecuación 4 se tiene que:

$$\varphi'(v(t)) = \begin{cases} 0.5 & si \quad -1 < x < 1 \\ 0 & en\ otro\ caso \end{cases} \qquad \text{Ec. (3.25)}$$

Pueden plantearse otras funciones de activación, fundamentalmente pequeñas variaciones a alguna de las cuatro anteriores; el algoritmo puede particularizarse para cada función de activación realizando la correspondiente derivada.

3.4. Inconvenientes del algoritmo *Backpropagation*.

A pesar de ser el algoritmo más utilizado en la práctica, el backpropagation tiene una serie de problemas:

3.4.1. Saturación de las neuronas.

Este problema se vio ya en el capítulo anterior en el caso de tener una sola neurona. En la actualización de los pesos aparece la derivada de la función de activación, que como puede observarse en las ecuaciones 30, 31, 32 y 33, vale cero en los extremos de la función, es decir, si este término es próximo a cero los pesos no se actualizarán aunque se esté cometiendo un error a la salida. Existen algunos métodos para solventar el problema. Una primera posibilidad es utilizar otro tipo de funciones de activación, por ejemplo funciones gaussianas. También puede plantearse una función de activación modificada de manera que $\varphi_\beta(x)=\beta \cdot x+(1-\beta)\varphi(x)$ donde $1 \geq \beta \geq 0$. Inicialmente, $\beta=1$ por lo que la derivada de la función de activación es igual a la unidad, y va descendiendo a medida que la red aprende hasta llegar a $\beta=0$ situación en la que la derivada de la función modificada será igual a la de la función original. Lo que se evita con este método es que las neuronas se saturen al principio del aprendizaje de la red, lo que provocaría que la red no aprendiese.

Otra posibilidad consiste en plantear que la amplitud y la pendiente en el origen de la sigmoide (si se ha escogido ésta como función de activación) vaya cambiando de manera adaptativa conforme la red aprende [Chen-96]. La función de activación sería:

$$\varphi(x) = \frac{a}{1+e^{-b \cdot x}} \qquad \text{Ec. (3.26)}$$

La actualización de los parámetros a y b puede plantearse por descenso de gradiente:

$$\begin{aligned} a(t+1) &= a(t) - \alpha \nabla_a J \\ b(t+1) &= b(t) - \alpha \nabla_b J \end{aligned} \qquad \text{Ec. (3.27)}$$

Puede utilizarse una posibilidad análoga en el caso de que la función de activación sea la tangente hiperbólica.

3.4.2. Inicialización de los pesos.

La inicialización de los pesos sinápticos es fundamental en el funcionamiento de una red, ya que el algoritmo de aprendizaje se basa en partir de un punto determinado de la función de error y moverse por ella hasta llegar al mínimo más cercano, que no tiene por qué ser el mínimo global de la función sino que puede tratarse de un mínimo local; por tanto, es fundamental el punto de partida que viene determinado por la inicialización de los pesos para que el mínimo alcanzado sea un mínimo global. Existen algunos algoritmos para evitar el problema de los mínimos locales, destacando sobre todos el algoritmo ERA (Expanded Range Approximation), que modifica el clásico backpropagation comprimiendo las salidas deseadas hacia su valor medio al principio del entrenamiento y expandiéndolas a continuación hasta llegar a sus valores originales al final del entrenamiento ya que se ha demostrado por sus autores que se garantiza que la función de error alcanza su mínimo global para el problema juguete de la función lógica XOR [Gorse-97]. Además ha sido aplicada a otros problemas mejorando los resultados obtenidos con el backpropagation clásico [Martín-99]. En definitiva, el algoritmo plantea una modificación de la señal deseada del siguiente modo:

$$d_{ent} = \langle d \rangle + \lambda [d - \langle d \rangle] \qquad \text{Ec. 3.28}$$

donde d representa el valor original de la señal deseada, <d> su valor medio y d_{ent} el valor de la señal deseada que se utiliza para entrenar a la red.

Por otro lado, la inicialización de los pesos afectará directamente al comportamiento del algoritmo en tres factores:

- Tiempo de convergencia de la red; ya que éste dependerá no solamente de la constante de adaptación, sino también de la distancia inicial al mínimo.
- Mínimo alcanzado. Este algoritmo converge al mínimo más cercano; éste puede ser un mínimo local.
- Saturación de las neuronas, Valores altos de los pesos pueden provocar una saturación en las neuronas. Solución inmediata; considerar unos valores iniciales de los pesos pequeños.

3.4.3. Zonas planas.

La modificación de los pesos es proporcional a la derivada de la función de error escogida, En las zonas planas este valor es cercano a cero; los pesos no se actualizan o si lo hacen es en muy pequeña medida. Una posible solución es añadir un término a la derivada pero tiene el problema de que en las proximidades del mínimo, como la derivada vale cero, pueden aparecer inestabilidades.

3.4.4. Elección de la constante de adaptación.

Como ya se explicó en el capítulo anterior, se ha de llegar a un compromiso en la elección de la constante de adaptación, ya que un valor excesivo de ésta puede dar lugar a inestabilidades mientras que un valor demasiado pequeño puede implicar un tiempo de convergencia muy elevado.

3.4.5. Parada del aprendizaje.

El problema consiste en elegir el momento idóneo para detener el aprendizaje de la red. Aunque existen otros criterios, lo más habitual es dividir los datos en dos conjuntos: entrenamiento y generalización; con el primero de ellos se entrena la red y con el segundo se comprueba el funcionamiento de la red entrenada con patrones no vistos anteriormente. El objetivo es la mejor generalización posible, ya que llega un momento en el que el error de generalización empieza a aumentar porque la red está "sobreaprendiendo" los patrones de entrenamiento lo que conduce a una peor capacidad de generalización. Existen también varios criterios sobre el porcentaje de patrones que deben destinarse a cada etapa, suele entrenarse con el 66 % de los patrones dejando el resto para validar la red, aunque también es habitual destinar el 50 % de los patrones a cada etapa. Normalmente esta elección depende del problema a tratar.

3.4.6. Elección de la arquitectura.

Como ya se ha explicado, la elección del número de neuronas ocultas es bastante problemática porque es básicamente un proceso de prueba y error. No obstante, existen métodos de poda y crecimiento que permiten determinar la estructura más óptima.

3.4.7. Elección de los patrones de entrenamiento.

El conjunto de patrones de entrenamiento ha de ser representativo del problema para que la red tenga un funcionamiento correcto sobre todo el conjunto de datos. Como ejemplo supongamos que se quieren clasificar los patrones que se muestran en la Figura 3.5, donde se representan las dos clases por círculos y cruces:

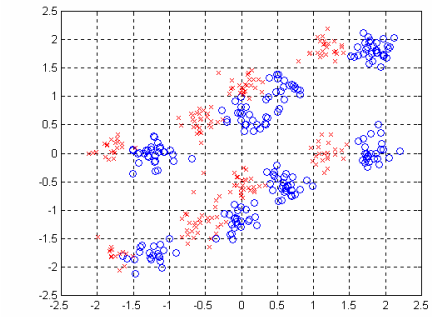


Figura 3.4. Ejemplo patrones de entrenamiento.

Si en lugar de escoger un conjunto de patrones representativo para entrenar la red, se toma por ejemplo el conjunto situado más hacia la izquierda, una posible frontera de decisión sería la que se muestra en la siguiente figura:

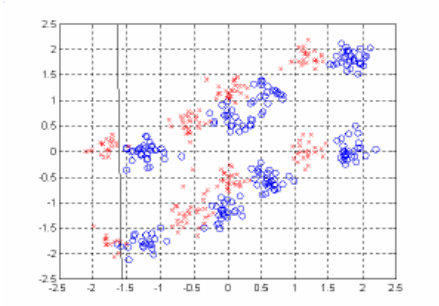


Figura 3.5. Ejemplo frontera de decisión.

Se observa que la frontera definida no es adecuada para el problema. Una posibilidad para evitar unos patrones de entrenamiento inadecuados consiste en agrupar los vectores de entrada según algoritmos de agrupamiento o clustering para, posteriormente tomar elementos de varios clusters.

3.5. Variantes del algoritmo *Backpropagation.*

Se han planteado una serie de variantes al clásico algoritmo backpropagation. A continuación se citarán las más importantes:

3.5.1. Momento.

Esta variante es muy similar al backpropagation clásico ya que el incremento de pesos es igual al gradiente de la función de error con signo negativo, pero además se le añade un término que es el incremento de pesos anterior, es decir, que vendrá dado por la siguiente expresión:

$$\Delta w_{mi}(n) = -\alpha \cdot \nabla J + \mu \cdot \Delta w_{mi}(n-1) \qquad \text{Ec. 3.29}$$

El nuevo término controla la velocidad de acercamiento al mínimo acelerándola cuando se está lejos de éste y ralentizándola cuando se está cerca. La constante de momento μ da una mayor o menor importancia al término de corrección. Este algoritmo presenta problemas en las proximidades del mínimo.

3.5.2. Silva-Almeida.

La modificación que introduce este método es hacer variar la constante de adaptación en función de la distancia al mínimo. Para conocer esta distancia se evalúan dos signos del gradiente de la función de error consecutivos. Lejos del mínimo estos gradientes tendrán el mismo signo (avanzamos al mínimo en la misma dirección) mientras que, en las proximidades de éste, los gradientes consecutivos tendrán signos distintos porque se estará oscilando alrededor del mínimo. Esta situación queda reflejada en el ejemplo de la figura donde se representa la función de error y los pasos que se dan desde la situación inicial, alejada del mínimo, hasta llegar a las proximidades de éste. Se observa que, al principio, los saltos realizados en la función de error son siempre en la misma dirección, mientras que cuando se está cerca del mínimo se oscila alrededor de él.

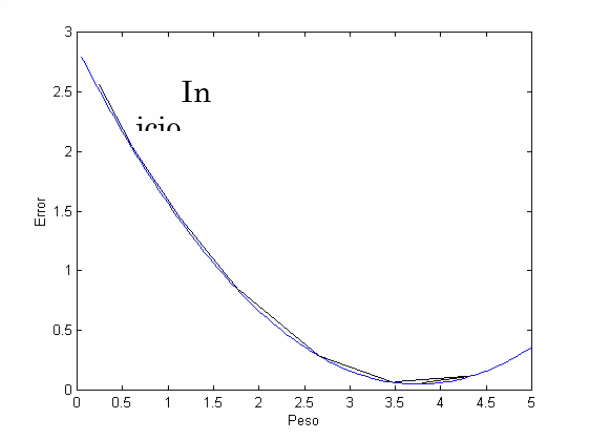


Figura 3.6. Función de error: pasos para llegar al mínimo.

La actualización de pesos que realiza el algoritmo es la misma que para el backpropagation pero teniendo en cuenta la variación de la constante de adaptación definida por:

$$\alpha(t) = \begin{cases} \alpha(t-1)\cdot u \Longleftrightarrow \left(\nabla_{w_{ij}(t)} J\right)\left(\nabla_{w_{ij}(t-1)} J\right) > 0 \\ \alpha(t-1)\cdot d \Longleftrightarrow \left(\nabla_{w_{ij}(t)} J\right)\left(\nabla_{w_{ij}(t-1)} J\right) < 0 \end{cases} \qquad \text{Ec. 3.30}$$

siendo d<1 y u>1.

3.5.3. DELTA-BAR-DELTA

Este método plantea una modificación al algoritmo backpropagation similar a la planteada en el Silva-Almeida pero mejorada para evitar inestabilidades durante la variación de la constante de adaptación. Ahora la variación de la constante de adaptación que se plantea es la siguiente:

$$\alpha(t) = \begin{cases} \alpha(t-1)+u \Longleftrightarrow \left(\nabla_{w_{ij}(t)} J\right)\left(\delta_{ij}(t-1)\right) > 0 \\ \alpha(t-1)\cdot d \Longleftrightarrow \left(\nabla_{w_{ij}(t)} J\right)\left(\delta_{ij}(t-1)\right) < 0 \end{cases} \qquad \text{Ec. 3.31}$$

siendo:

$$\delta_{ij}(t-1) = (1-\theta)\nabla_{w_{ij}(t-1)} J + \theta\cdot\delta_{ij}(t-2) \qquad \text{Ec. 3.32}$$

con la condición $0<\theta<1$.

En las expresiones anteriores se pueden observar principalmente dos diferencias respecto a las del método Silva-Almeida:

1. El aumento del valor de la constante de adaptación para acelerar la convergencia ya no es exponencial sino lineal por lo que disminuyen las posibilidades de inestabilidades en el algoritmo por una constante excesivamente grande.

2. La decisión sobre si se aumenta o disminuye la constante ya no depende exclusivamente de dos gradientes consecutivos sino que se realiza comparando el gradiente y el promediado exponencial de los anteriores (δ).

3.5.4. RPROP.

Se trata de un algoritmo similar a los anteriores ya que también plantea una variación de la constante de adaptación, pero en este caso la actualización de los pesos viene dada por la siguiente expresión:

$$\Delta w_{ij}(t) = -\alpha(t) \cdot \text{signo}\left(\nabla_{w_{ij}(t)} J\right) \qquad \text{Ec. 3.33}$$

La utilización del signo del gradiente en la actualización de los pesos supone un ahorro en la carga computacional. Por otro lado, la constante de adaptación viene dada en este caso por:

$$\alpha(t) = \begin{cases} min(\alpha(t) \cdot u, \alpha_{max}) \Longleftrightarrow \left(\nabla_{w_{ij}(t)} J\right)\left(\nabla_{w_{ij}(t-1)} J\right) > 0 \\ max(\alpha(t) \cdot d, \alpha_{min}) \Longleftrightarrow \left(\nabla_{w_{ij}(t)} J\right)\left(\nabla_{w_{ij}(t-1)} J\right) < 0 \end{cases} \qquad \text{Ec. 3.34}$$

siendo u>1 y d<1.

Como puede observarse, la constante puede tomar dos valores distintos dependiendo del signo de los dos últimos gradientes. De nuevo se intenta que tenga un valor bajo en las proximidades del mínimo y mayor lejos de éste, controlando así la velocidad de convergencia, pero en este caso los valores están limitados para evitar que sea excesivamente grande para evitar inestabilidades (igual que Delta-Bar-Delta) y demasiado pequeño para evitar que la velocidad de convergencia sea excesivamente pequeña.

3.6. Otros algoritmos de aprendizaje.

Además del clásico algoritmo backpropagation y sus variantes, existen otros algoritmos que, no siendo tan habituales, deben tenerse en cuenta.

3.6.1. Algoritmos de segundo orden.

Los algoritmos anteriores son de primer orden ya que solamente se considera el gradiente de la función de coste. Sin embargo, considerando su desarrollo de Taylor, aparecen más términos:

$$J(t+r) \cong J(t) + (\nabla J)^T \cdot r + \frac{1}{2} \cdot r^T \cdot (\nabla^2 J) \cdot r + \ldots \qquad \text{Ec. 3.35}$$

donde $\nabla^2 J$ es la matriz hessiana:

$$\nabla^2 J = \begin{bmatrix} \frac{\partial^2 J}{\partial t_1^{\,2}} & \cdots & \frac{\partial^2 J}{\partial t_1 \partial t_n} \\ \cdots & \cdots & \cdots \\ \frac{\partial^2 J}{\partial t_n \partial t_1} & \cdots & \frac{\partial^2 J}{\partial t_n^{\,2}} \end{bmatrix} \qquad \text{Ec. (3.36)}$$

Hasta ahora, al aplicar el método por descenso de gradiente, se suponía una aproximación de la función de coste hasta el segundo término del desarrollo de Taylor; al considerar más términos la aproximación será cada vez mejor aunque la expresión resultante será más complicada.

Se puede plantear el considerar también el tercer término del desarrollo, que aporta información sobre la curvatura de la función de coste, lo que puede ayudar a la convergencia hacia el sistema óptimo (primera derivada igualada a cero). El sistema óptimo viene dado, por tanto, por:

$$\mathrm{r} = -\left(\nabla^2 \mathrm{J}(\mathrm{n})\right)^{1} \cdot \nabla \mathrm{J}(\mathrm{n}) \qquad \text{Ec. (3.37)}$$

Si la función de coste es la cuadrática, esta actualización de los pesos conduce a un mínimo en un solo paso. Las dos principales desventajas del método son la elevada carga computacional que conlleva el cálculo de la inversa de $\nabla^2 \mathrm{J}(\mathrm{n})$ en cada iteración y los problemas numéricos de redondeo que tiene.

3.6.2. Alopex.

Este algoritmo tiene como ventaja principal la sencillez de implementación con tecnología VLSI. En la actualización de los pesos no se tiene en cuenta ninguna derivada. Por tanto, la función de error ya no ha de ser continua y diferenciable y se reduce la influencia de la inicialización de los pesos en el mínimo alcanzado. La actualización de los pesos viene dada por la siguiente expresión:

$$\mathrm{w}_{\mathrm{ij}}(\mathrm{n}) = \mathrm{w}_{\mathrm{ij}}(\mathrm{n}-1) + \lambda_{\mathrm{ij}} \qquad \text{Ec. (3.38)}$$

siendo el parámetro λ_{ij} que determina la actualización de los pesos:

$$\lambda_{ij} = \begin{cases} -\delta & \textit{con probabilidad } P_{ij} \\ +\delta & \textit{con probabilidad } 1 - P_{ij} \end{cases} \qquad \text{Ec. (3.39)}$$

donde δ es un valor constante y la probabilidad es:

$$P_{ij}(t) = \frac{1}{1 + e^{\frac{\theta_{ij}(t)}{T}}} \qquad \text{Ec. (3.40)}$$

El parámetro T controla la aleatoriedad de los incrementos de los pesos mientras que θ_{ij} es igual a:

$$\theta_{ij}(t) = \Delta w_{ij}(t) \cdot \Delta E(t) \qquad \text{Ec. (3.41)}$$

siendo:

$$\Delta w_{ij}(t) = w_{ij}(t-1) - w_{ij}(t-2) \qquad \text{Ec. 3.42}$$

$$\Delta E(t) = E(t-1) - E(t-2) \qquad \text{Ec. 3.43}$$

donde E representa la suma de los errores cometidos a la salida de la red neuronal.

Dependiendo de los parámetros anteriores, los pesos se verán incrementados o decrementados. A pesar de su sencillez no ha sido un algoritmo excesivamente estudiado ni aplicado debido a los problemas que se tienen para la determinación de los parámetros adecuados para el aprendizaje

3.6.3. Algoritmos genéticos.

Los algoritmos genéticos son algoritmos de optimización de funciones, pero a diferencia de los anteriores realizan una optimización global, es decir, buscan el mínimo global de la función de error por lo que se evitan los problemas asociados con los mínimos locales y las precauciones que eran necesarias para no caer en ellos. El mayor inconveniente que tienen es su baja velocidad de convergencia y su alta carga computacional. Se basan en la maximización de una función F(s) siendo s vectores con componentes de valores 0 y 1. Cada uno de estos vectores s recibe el nombre de cromosoma. El primer paso es codificar las variables del problema de forma binaria, es decir, en forma de 0's y 1's. A continuación, se define una población, que es un conjunto de cromosomas, que se puede escoger de manera aleatoria o usando algún conocimiento heurístico del problema, por ejemplo la posición aproximada del mínimo global de la función de error. Después se mira la calidad de cada cromosoma de acuerdo con su valor en la función F(s). La definición más aceptada para la calidad es la que viene dada por el valor normalizado de la función para ese cromosoma:

$$\text{calidad}_i = \frac{F(s_i)}{\sum_j F(s_j)} \qquad \text{Ec. 3.44}$$

El valor de calidad sirve para definir la probabilidad de escoger ese cromosoma con el fin de determinar la próxima población. El procedimiento más usado es usar una circunferencia, asignando a cada cromosoma un ángulo proporcional a ese valor de calidad. A continuación, se fija un punto y se gira un ángulo, determinado de forma aleatoria, de tal manera que se escoge el cromosoma situado en ese punto predeterminado. Con este método se escoge una población de las mismas dimensiones que la que había originalmente. Con los cromosomas que se han escogido se realizan operaciones de cruzamiento y mutación. De la población de cromosomas escogida, se toman pares de cromosomas que se cruzarán con una probabilidad P_c. Aleatoriamente se escoge un número (de 1 a k-1, siendo k la dimensión de los cromosomas) de manera que éstos intercambian los valores de los bits a partir de dicho número. Este proceso se repite para toda la población. Tras el proceso de cruzamiento, tiene lugar el proceso de mutación que consiste en complementar los bits que forman el cromosoma con una probabilidad P_m. Después del proceso de mutación se vuelve a evaluar la calidad de cada cromosoma y se repite el ciclo, que se representa en modo de diagrama de bloques definido en la siguiente figura:

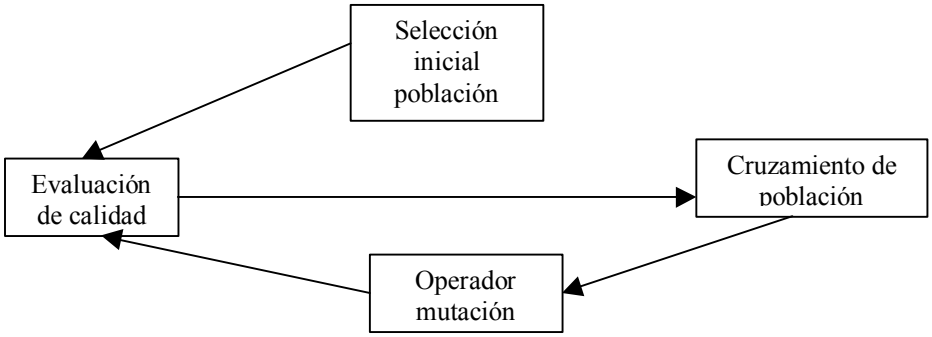


Figura 3.7. Diagrama de bloques de las operaciones que realiza un algoritmo genético.

Los algoritmos genéticos pueden aplicarse a las redes neuronales en dos aspectos fundamentalmente:

- Dada una arquitectura, se pueden optimizar los valores de los pesos sinápticos, lo cual es una ventaja respecto a los anteriores algoritmos al tratarse de métodos de búsqueda de mínimo global. El inconveniente es la lentitud en la convergencia [Hassoun-95].

- También pueden utilizarse para optimizar la propia arquitectura de la red. Es decir, el algoritmo genético se encargaría de escoger el número de neuronas ocultas óptimo y también la constante de adaptación, lo que nos ahorraría los habituales barridos en estos valores permitiendo encontrar directamente los óptimos. El inconveniente es la elevada carga computacional que ello implica [Vonk-97].

3.7. Optimización de la arquitectura de la red.

En este apartado se estudiarán las formas en las que se puede optimizar la arquitectura de un perceptrón multicapa. Existen dos aproximaciones: métodos de crecimiento y su complementaria: los métodos de poda.

Estos últimos son unos procedimientos en los que se produce la eliminación de pesos o de neuronas innecesarias mientras la red aprende, de modo que ésta se simplifica con el aprendizaje. El algoritmo consiste en considerar una red neuronal con un número de neuronas mayor del requerido, en principio, de manera que el algoritmo se encargue de "podar" las conexiones que no sean necesarias. Sus dos ventajas principales son:

- Se simplifica la estructura de la red, lo que es adecuado en vistas a una posible implementación hardware.

- Observando las conexiones entre la capa de entrada y la primera capa oculta después de aplicar métodos de poda, se puede determinar la relevancia de las distintas entradas a la red, ya que si existe alguna entrada que tiene las

conexiones "podadas", esto querrá decir que dicha entrada no sirve para discriminar ninguna característica por lo que debe ser eliminada.

Los métodos se pueden agrupar en una serie de clases [Reed-93]:

3.7.1. Términos de penalización (penalty methods).

Los métodos basados en términos de penalización son los más sencillos y utilizados en la práctica. Se basan en añadir un término de regularización para eliminar los pesos innecesarios, ya que este término provoca un decaimiento en su valor de manera que se puede plantear eliminar aquellos pesos que estén por debajo de un determinado umbral.

Los más habitualmente usados son:

$$J_{tot} = J + \lambda \cdot \sum_{i,j} \left| w_{ij} \right| \qquad \text{Ec. 3.45}$$

$$J_{tot} = J + \lambda \cdot \sum_{i,j} w_{ij}^{\,2} \qquad \text{Ec. 3.46}$$

$$J_{tot} = J + \lambda \cdot \sum_{i,j} \frac{w_{ij}^{\,2} / w_0^{\,2}}{1 + w_{ij}^{\,2} / w_0^{\,2}} \qquad \text{Ec. 3.47}$$

El parámetro λ determina la importancia de los términos de regularización y debe escogerse con un valor pequeño ya que la función de coste a minimizar debe ser fundamentalmente la original. Por otro lado, el sumatorio se extiende sobre todos los pesos de la red, es decir tanto los de la capa oculta como los de salida.

3.7.2. Métodos de 2º orden.

Estos métodos determinan la importancia de una neurona dentro de la red, los más habituales son el OBD (Optimal Brain Damage) y el OBS (Optimal Brain Surgeon) que son métodos de segundo orden que requieren el cálculo del hessiano de la función de coste ya que este término da información sobre cada elemento de la red neuronal.

3.7.3. Poda interactiva.

Este método consiste en inspeccionar una red entrenada y decidir qué nodos se eliminan mediante dos reglas [Reed-93]:

- Si una entrada tiene una salida constante durante todo el conjunto de entrenamiento puede ser eliminada. Posteriormente ha de ajustarse el peso de sesgo de las siguientes capas.
- Si existe un conjunto de neuronas que tienen una salida altamente correlacionada para todos los patrones de entrenamiento, el conjunto de

neuronas pueden agruparse en una única unidad, es decir, que se pasaría de un conjunto de n neuronas a una sola.

3.7.4. Otros métodos

Aunque los métodos más citados en la literatura de redes neuronales y utilizados en la práctica son los anteriores, existen también otros métodos. Uno de ellos consiste en plantear la poda de los pesos innecesarios mediante algoritmos genéticos; aunque se trata de un método muy eficiente tiene el inconveniente de su elevada carga computacional. Kruschke plantea una poda basada en que las neuronas compiten para sobrevivir; a partir del vector de pesos se obtiene una medida de la importancia de cada neurona [Reed-93]. Otro método de poda consiste en una estimación de las neuronas ocultas más representativas mediante una aproximación de subespacios [Kung-93].

El segundo tipo de procedimientos sigue una filosofía opuesta: se parte de una red de pequeño tamaño y se añaden neuronas conforma la red aprende. Están menos extendidos en la práctica que los métodos de poda ya que la mayoría presentan una carga computacional mayor. Aunque existen muchos algoritmos de crecimiento el más conocido es el conocido como Cascade-Correlation (C-C) planteado por Falhman y Lebière [Falhman-90]. El algoritmo C-C parte de una estructura sin capas ocultas, de modo que las capas de entrada y salida están conectadas entre sí. El procedimiento consiste en entrenar estas conexiones mediante alguno de los algoritmos estudiados, por ejemplo, el backpropagation y observar si se produce un cambio en el error. Si no se produce un cambio en el error después de un número determinado de épocas se añaden una o varias neuronas. Estas neuronas tienen pesos de entrada que las conectan a las entradas existentes o a las neuronas que en pasos anteriores del algoritmo se han introducido mientras que la salida no se conecta en un primer momento a ninguna neurona de la red. Las conexiones de entrada a la neurona se adaptan de tal forma que intentan maximizar la correlación entre la salida de la neurona introducida y el error cometido por la red. Para realizar esta maximización respecto a los pesos de la neurona introducida, se puede plantear un método por ascenso de gradiente. Primeramente se entrenan los pesos de entrada a la red, y a continuación los de salida. La estrategia consiste en mantener los pesos de entrada de la neurona introducida y entrenar el resto de pesos de la red de acuerdo al criterio de minimización de una función de coste. Si no se produce ningún cambio en el error tras un número determinado de épocas y el valor de éste es alto, o no se ajusta a nuestras especificaciones, se introduce una nueva neurona oculta y se repite el procedimiento.

3.8. Tratamiento de los datos.

El tratamiento de los datos es básico para el adecuado funcionamiento de una red ya que la forma de presentar los datos a la red influirá en la respuesta de ésta. Existen varios procedimientos para tratar los datos de la red, entre los que cabe destacar los siguientes:

3.8.1. Normalización de las entradas.

En primer lugar, hay que tener en cuenta que las variables de entrada pueden tener diferencias de valores de varios órdenes de magnitud de forma que el aprendizaje de la red se verá influenciado por estas diferencias ya que el incremento de pesos de una neurona es proporcional a su entrada. Una manera de evitar el problema es asignar a todas las entradas valores parecidos normalizando las entradas. Existen varias posibilidades:

$$x_k^* = \frac{x_k - \bar{x}_k}{\sigma_{x_k}} \quad \text{siendo} \begin{cases} x_k \text{ una variable de entrada} \\ x_k^* \text{ el valor normalizado de la entrada } x_k \\ \bar{x}_k \text{ el valor medio de } x_k \\ \sigma_{x_k} \text{ la desviación estándar de } x_k \end{cases} \qquad \text{Ec. (3.48)}$$

Suele utilizarse esta normalización porque presenta valor medio cero y varianza unidad, con lo que evita tendencias de las entradas fuera de los rangos considerados como normales. Otra posible normalización de los datos de entrada es la siguiente [Kennedy-98]:

$$x_k^* = \left[\frac{x_k - \min 1}{\max 1 - \min 1} \right] [\max 2 - \min 2] + \min 2 \qquad \text{Ec. (3.49)}$$

donde entran en juego los valores máximos y mínimos antes y después de la transformación (que se denotan respectivamente por 1 y 2). La ventaja principal de este método es que al tratarse de una transformación lineal, se mantienen las relaciones existentes entre los patrones de entrenamiento originales. También se utiliza una normalización no lineal que hace uso de las funciones sigmoide o tangente hiperbólica, dependiendo de que se codifiquen los datos en el rango entre 0 y 1 o en el rango entre –1 y +1 respectivamente. La normalización vendría dada por la siguiente expresión:

$$x_k^* = \frac{1 - e^{-t}}{1 + e^{-t}} \quad siendo \quad t = \frac{x_k - \bar{x}_k}{\sigma_{x_k}} \qquad \text{Ec. (3.50)}$$

Con esta normalización, se tendrá que los valores cercanos al valor medio se transformarán en la zona lineal mientras que los alejados al valor medio lo harán en la zona plana por lo que es especialmente útil cuando alguna de las entradas presenta un rango de variación muy grande.

3.8.2. Codificación de los datos.

Hay que ser extremadamente cuidadoso a la hora de codificar los datos ya que una codificación no adecuada puede repercutir en un mal funcionamiento de la red. En cuanto a los datos de entrada, la codificación de datos numéricos (longitud, peso, ...) no tiene excesivos problemas ya que en todo caso se debería realizar una normalización de estos datos como se vio anteriormente. Sin embargo, cuando se trata con variables discretas como por ejemplo el sexo, es peligroso codificar hombre/mujer de la forma 0/1 ya que como la actualización de los pesos depende del

valor de las entradas no se produciría actualización aunque se cometiese error cuando se tuviese el sexo codificado con un 0. Por ello, es preferible utilizar otro tipo de codificaciones como por ejemplo +1/-1.

También hay que codificar las salidas de algún modo. Si por ejemplo, se quieren clasificar patrones en dos clases se puede optar por situar una neurona a la salida y codificar como perteneciente a una clase u otra dependiendo de que a la salida de ésta tenga un valor cercano a 1 ó 0 (si se utiliza la sigmoide como función de activación) o bien un valor cercano a +1 ó –1 (si se utliza la tangente hiperbólica como función de activación). Otra posibilidad sería situar dos neuronas a la salida, cada una de ellas representando a una clase, de manera que se considerase que un patrón pertenece a una clase u otra dependiendo de la neurona que presenta mayor actividad. Es decir, que una clase se codificaría como 1/0 y la otra como 0/1 (análogamente +1/-1 y –1/+1).

3.8.3. Información de los patrones.

En ocasiones, puede surgir el problema de una falta de información de los patrones debido a que falten componentes de los datos de entrada a la red. Algunas soluciones que se pueden aplicar cuando ocurre esto son las siguientes:

1. Eliminar los datos no completos.
2. Dar un valor a las componentes que faltan similar a la del resto de los patrones.
3. Utilizar el valor medio de esa componente sobre todos los vectores de entrada.
4. Codificar de alguna manera que ese valor es desconocido o que falta información.

3.8.4. Extracción de características.

La extracción de características está motivada porque existen muchos problemas que tienen un gran número de entradas lo que hace que la carga computacional aumente. Una extracción de características que seleccione las entradas realmente necesarias mejorará la eficacia de la red. Para ello, las dos tareas que habría que realizar serían [Kung-93]:

1. Extracción de las características más relevantes.
2. Eliminación de redundancias entre datos.

3.8.5. Consistencia de los datos.

Puede ocurrir que fallos humanos o imprecisiones en los instrumentos de medida provoquen una inconsistencia en los datos. Evidentemente estos datos no deben utilizarse para entrenar a la red. Existen básicamente tres posibilidades para eliminar las inconsistencias que aparezcan:

1. Mediante inspección visual si el problema no es excesivamente complejo.
2. Mediante histogramas.
3. Mediante la colocación de unos umbrales máximo y mínimo que invaliden cualquier dato fuera de ellos.

4

Mapas autoorganizativos

4.1. Introducción al aprendizaje supervisado.

En el capítulo anterior estudiábamos una estructura basada en aprendizaje supervisado: a la red se le da a conocer en todo momento qué señal debería asignar a la salida sin funcionara correctamente. Existen sin embargo aplicaciones donde esta asignación no es posible; una de estas aplicaciones típicas son las de agrupamiento o "clustering" de los datos. Aquí el problema consiste en agrupar los datos de alguna forma de tal manera que se puedan establecer relaciones entre ellos. Este tipo de aprendizaje donde no existe ningún factor externo a la red neuronal que determine el buen/mal funcionamiento se conoce como aprendizaje no supervisado. La red no recibe ninguna información por parte del entorno que le indique si la salida generada en respuesta a una determinada entrada es o no correcta, por ello suele decirse que estas redes tienen la capacidad de autoorganizarse.

En cuanto a las aplicaciones, éstas dependen de qué tipo de información pueden ofrecer a la salida este tipo de redes agrupándose en seis grandes bloques:

1) Análisis de similitud entre patrones. Aparece cuando existe una única neurona cuya salida es continua indicando el grado de similitud o parecido entre la entrada actual y el promedio de los patrones representados en los pesos sinápticos.

2) Análisis de componentes principales. Extensión del caso anterior a varias neuronas. Consistente en encontrar una base del espacio de entrada (pesos) que se corresponda con los rasgos más destacables del espacio sensorial.
3) Agrupamiento. La red está compuesta con neuronas de salida discreta y cada una de ellas representa una categoría.
4) Memoria asociativa. Generalización del anterior a salidas continuas y donde el vector salida es el vector propio de la clase.
5) Codificación. Análogo al anterior pero en este caso la red no proporciona como salida el vector propio de la clase sino una versión codificada.
6) Mapas de rasgos. Las neuronas se ordenan geométricamente llevando a cabo una proyección de un espacio en otro.

4.2. Tipos de aprendizaje supervisado.

4.2.1. Aprendizaje Hebbiano.

En las redes no supervisadas no existe una información externa que nos indique la forma en la que debemos modificar las conexiones sinápticas en función de los resultados o respuestas ofrecidas por la red ante los datos de entrada, es por tanto necesario definir una regla que dirija el aprendizaje de este tipo de redes sin hacer uso de información externa alguna.

En el capítulo anterior hemos visto este tipo de aprendizaje aplicado al caso de una neurona, ahora extenderemos este tipo de aprendizaje a estructuras multicapa.

Una de las reglas más importantes en el aprendizaje no supervisado de una neurona es la regla de Oja; una extensión de esta regla a varias neuronas sería la definida por Sanger [Sanger-89]. Aquí se tiene una red definida por la siguiente figura:

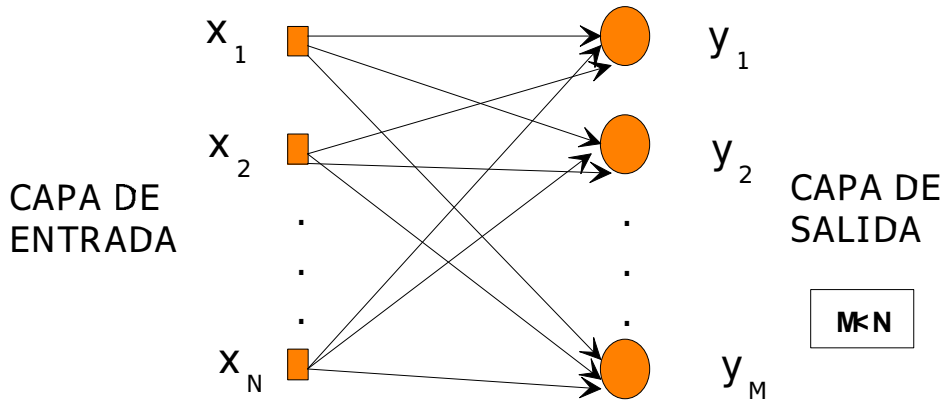


Figura 4.1. Esquema de la red para aplicar la regla de Sanger.

Todas las neuronas de la salida de la red presentan una función de activación lineal, se cumple entonces:

$$y_k(n) = \sum_{s=1}^{N} w_{ks} \cdot x_s(n) \qquad \text{Ec. 4.1}$$

La regla de aprendizaje planteada queda definida por:

$$\Delta w_{ks}(n) = \alpha \cdot \left(y_k(n) \cdot x_s(n) - y_k(n) \cdot \sum_{r=1}^{k} w_{rs}(n) \cdot y_r \right) \qquad \text{Ec. 4.2}$$

El aprendizaje hebbiano aparece en otras estructuras neuronales que tienen un uso bastante extendido: las memorias asociativas. Esta memorias presentan una estructura con una capa de entrada y una de salida, Figura 4.2.

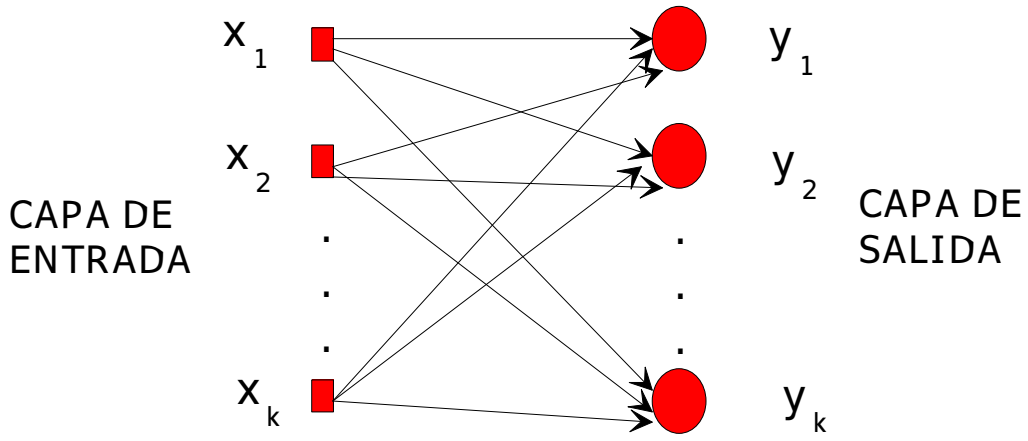


Figura 4.2. Esquema de una memoria asociativa.

Como se aprecia de la figura los patrones de entrada y de salida van a ser vectores con k-componentes:

$$\mathbf{x_p} = [x_{p1}\ x_{p2}\ \ldots\ldots x_{pk}]$$
$$\mathbf{y_p} = [y_{p1}\ y_{p2}\ \ldots\ldots y_{pk}] \qquad \text{Ec. 4.3}$$

donde el superíndice t indica trasposición.

Las neuronas que forman esta memoria asociativa son lineales por lo que la salida viene definida por la siguiente relación:

$$\mathbf{y_p} = \mathbf{M} \cdot \mathbf{x_p} \qquad \text{Ec. 4.4}$$

donde M es una matriz de pesos determinada por la salida y entrada de los patrones a memorizar. Si se tienen N-patrones la matriz de memoria del sistema vendrá definida por:

$$\mathbf{M} = \sum_{s=1}^{N} \mathbf{W(s)} \qquad \text{Ec. 4.5}$$

donde

$$\mathbf{W(s) = y_s \cdot x_s^t} \qquad \text{Ec. } \text{¶}4.6$$

de acuerdo con esta expresión y con la ecuación 4, la matriz M queda definida por la siguiente ecuación iterativa:

$$\mathbf{M_k = M_{k-1} + y_k \cdot x_k^t} \qquad \text{Ec. } \text{¶}4.7$$

Si se compara esta expresión con la de actualización de los pesos e un aprendizaje de tipo hebbiano encontramos una correspondencia total.

Veamos a continuación como funcionan estas memorias. Supongamos que una vez entrenada la red (se conoce la matriz) se tiene como entrada el patrón x_j. De acuerdo con la ecuación 3 la salida vendrá dada por:

$$\mathbf{y = M \cdot x_j} \qquad \text{Ec. } \text{¶}4.8$$

que, aplicando las ecuaciones 5 y 6 se llega a:

$$\mathbf{y} = \left(\sum_{s=1}^{N} \mathbf{y_s \cdot x_s^t} \right) \cdot \mathbf{x_j} \qquad \text{Ec. } \text{¶}4.9$$

El sumatorio de esta expresión lo podemos dividir en dos :

$$\mathbf{y} = \sum_{\substack{s=1 \\ s \neq j}}^{N} \mathbf{y_s} \cdot \left(\mathbf{x_s^t \cdot x_j} \right) + \mathbf{y_j} \cdot \left(\mathbf{x_j^t \cdot x_j} \right) \qquad \text{Ec. } \text{¶}4.10$$

Los términos que aparecen entre paréntesis son productos escalares de vectores. Si se impone que los vectores de entrada sean ortonormales:

$$\mathbf{x_j^t \cdot x_k} = \begin{cases} \mathbf{1 \text{ si } j = k} \\ \mathbf{0 \text{ si } j \neq k} \end{cases} \qquad \text{Ec. } \text{¶}4.11$$

entonces, la expresión 8 da lugar al vector de salida y_j. Si no se da esta condición el patrón que se obtiene a la salida no es el deseado (j) sino que se obtiene el vector y_j más un término de ruido (el primer sumatorio).

4.2.2. Aprendizaje competitivo

El aprendizaje competitivo es un tipo de aprendizaje no supervisado que sirve de base para varios modelos de redes neuronales artificiales. El objetivo de las redes basadas en este tipo de aprendizaje es llevar a cabo una categorización entre los datos de entrada. Se trata de que estímulos parecidos sean clasificados como pertenecientes a la misma categoría, mediante un proceso de búsqueda de esas categorías que la red debe hacer de forma no supervisada. La arquitectura básica de este tipo de redes consiste en dos capas distintas. La capa F1 recibe los estímulos o entradas procedentes del entorno, la capa F2 es la denominada capa de competición y es la que produce la salida de la red. Cada neurona de la capa F1 está conectada con todas y cada una de las neuronas de la capa F2 a través de pesos sinápticos adaptativos (i.e que pueden modificar su valor en base a un a regla de aprendizaje determinada). Las neuronas de la capa F2 además de recibir las entradas de la capa F1 ponderadas por los pesos sinápticos, tienen conexiones

laterales inhibitorias con el resto de neuronas de la capa y una conexión excitatoria consigo misma como puede verse en la Figura 4.3 para una de las neuronas de esta capa F2.

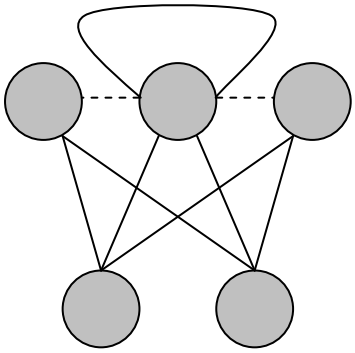


Figura 4.3. Esquema de una estructura competitiva.

Las conexiones existentes entre las neuronas de la capa F2 son fijas y permiten que la neurona con mayor valor de excitación se refuerze más a sí misma vía autoconexión excitatoria y a su vez inhiba con más fuerza al resto de neuronas de la capa. Esta dinámica conduce a un proceso competitivo en el que todas las neuronas de la capa intentan aumentar su activación a la vez que tratan de reducir la activación de las restantes, el proceso dinámico continúa hasta que la red se estabiliza. En ese momento existirá una vencedora de la competición que será la salida que consideraremos. El algoritmo de aprendizaje queda como:

- Se recibe el estímulo en la capa F1
- Propagación de la señal hasta F2 a través de las conexiones adaptativas entre F1 y F2. Se calcula el valor de las excitaciones a cada una de las neuronas de F2.
- Proceso de competición mediante inhibición lateral y autorefuerzo
- Finalizada la competición se produce la modificación de los pesos adaptativos asociados a la neurona ganadora.

Tras la competición, la neurona vencedora es la que mejor se corresponde con el estímulo de entrada. Lo que se pretende con el aprendizaje es reforzar esa correspondencia modificando las conexiones entre F1 (que recibe el estímulo) y la neurona ganadora, de manera que, para dicha neurona, sea más fácil "reconocer" el mismo estímulo o estímulos parecidos en una presentación posterior.

Las carencias respecto a la estabilidad de las categorías establecidas mediante este tipo de aprendizaje condujeron a Grossberg a la introducción de estructuras capaces de estabilizar un proceso de establecimiento de categorías en el marco de la denominada Teoría de la Resonancia Adaptativa que será comentada posteriormente.

4.3. Mapas Auto-Organizativos.

4.3.1. Descripción general del SOM.

El modelo de red neuronal auto-organizativa más popular y mejor estudiado es el SOM ("Self Organizing Map") propuesto por el finlandés Teuvo Kohonen en 1984. El modelo toma como base el agrupamiento de tareas afines que tiene lugar en las diferentes zonas del cerebro. En este modelo las neuronas se organizan en una arquitectura en dos capas. La primera es la capa de entrada o sensorial, que consiste en m neuronas, una por cada variable de entrada. El procesamiento se realiza en la segunda capa o capa de competición, que forma el mapa de rasgos:

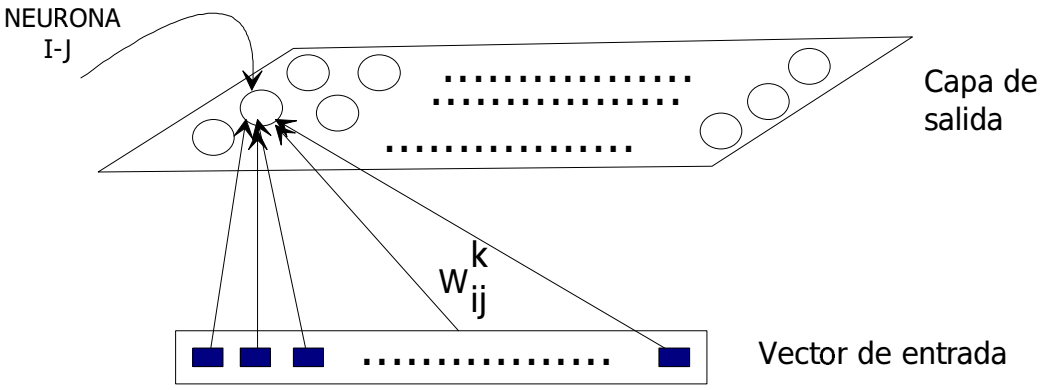


Figura 4.4. Arquitectura del SOM.

Cada una de las neuronas de entrada está conectada con todas las neuronas de la segunda capa mediante pesos sinápticos. La capa de entrada tiene la misma dimensión que el dato o patrón de entrada, y la actividad de las neuronas de esta capa es proporcional a dicho patrón. A cada una de las neuronas de la segunda capa se le asigna un vector de pesos que tiene la misma dimensión que los vectores de entrada, es decir, para la neurona i,j (primer índice fila, segundo columna) se tendrá:

$$w_{ij} = [w_{ij}^{1}\ w_{ij}^{2} \ldots\ldots\ldots w_{ij}^{k}] \qquad \text{Ec. 4.12}$$

siendo k la longitud del vector de entrada.

Veamos de una forma intuitiva el funcionamiento de esta red. Lo que se persigue en definitiva, es que patrones semejantes de entrada aumenten la actividad de neuronas próximas en la capa de salida de dicho mapa auto-organizativo. Una forma de realizar esto es establecer una función de semejanza entre el vector de entrada y los vectores de pesos asociados a cada una de las neuronas de salida. Nos encontramos aquí con el primer problema, ¿qué entendemos por parecido?. Para que nuestro sistema funcione debemos encontrar medidas de similitud válidas. Existen diferentes formas de medir estas diferencias siendo las más extendidas las definidas en la tabla 4.1.

FUNCIÓN DISTANCIA	EXPRESIÓN MATEMÁTICA
CUADRÁTICA	$\left[\sum_{s=1}^{k}\left(w_{ij}^{s}-x^{s}\right)^{2}\right]^{\frac{1}{2}}$
MANHATTAN	$\left[\sum_{s=1}^{k}\left\|w_{ij}^{s}-x^{s}\right\|\right]$
MINKOWSKI	$\left[\sum_{s=1}^{k}\left(w_{ij}^{s}-x^{s}\right)^{\lambda}\right]^{\frac{1}{\lambda}}$

Tabla 4.1. Diferentes métricas para comparar dos vectores.

La función que más se utiliza es la norma cuadrática. Si los vectores tienen la misma norma entonces podemos deducir otra función similitud a partir de la cuadrática. El hecho de tener la misma norma puede parecer difícil de darse en la realidad pero no es así, existen muchos sistemas que usan vectores normalizados (norma 1). Si dos vectores tienen la misma norma es lógico pensar que el parecido entre ellos vendrá definido por el ángulo que forman entre sí. En efecto, si desarrollamos la expresión de la distancia cuadrática como producto escalar de un vector consigo mismo:

$$\left[\sum_{s=1}^{k}\left(w_{ij}^{s}-x^{s}\right)^{2}\right]^{\frac{1}{2}}=\left[\left(\vec{w}_{ij}-\vec{x}\right)\left(\vec{w}_{ij}-\vec{x}\right)\right]^{\frac{1}{2}}=\left[\left\|\vec{w}_{ij}\right\|^{2}+\left\|\vec{x}\right\|^{2}-2\cdot\vec{w}_{ij}\cdot\vec{x}\right]^{\frac{1}{2}} \qquad \text{Ec. 4.13}$$

que, aplicando la normalización de los vectores y la definición de producto escalar se llega a:

$$\left[\sum_{s=1}^{k}\left(w_{ij}^{s}-x^{s}\right)^{2}\right]^{\frac{1}{2}}=\left[2-2\cdot\cos(\theta)\right]^{\frac{1}{2}} \qquad \text{Ec. 4.14}$$

donde cos(θ) es el ángulo que forman los dos vectores. Así pues minimizar la distancia entre dos vectores con la misma norma es lo mismo que maximizar el coseno del ángulo entre los dos vectores.

Una vez determinado el criterio de funcionamiento de nuestro sistema el punto importante a tratar es el aprendizaje de la red o, lo que es equivalente, la actualización de los pesos de la red. Como en el aprendizaje de un perceptrón multicapa al principio se supone un desconocimiento total del problema por lo que estos pesos se inicializarán de forma aleatoria. Seguidamente se le pasa a la red un vector de entrada y se aplica el criterio de similitud escogido a este vector y a los vectores de las neuronas de la capa de salida. En esta capa es cuando se lleva a cabo un proceso de competición entre las distintas neuronas, el vencedor de esta competición es aquella célula cuyo vector de pesos posee un mayor grado de similitud con el vector de entrada. Como buscamos que este vector sea el vencedor si el patrón de entrada vuelve a aparecer hay que hacer que el vector de pesos de la neurona ganadora se parezca más al vector de entrada. La forma de efectuar esto cuando se usa como función similitud la distancia cuadrática es aplicar la siguiente actualización de pesos sinápticos:

$$w_{ks}''' = w_{ks}''' + \alpha \cdot (x''' - w_{ks}''') \qquad \text{Ec. 4.15}$$

donde la neurona ks es la vencedora. El incremento en la similitud entre el patrón de entrada y los pesos sinápticos de la neurona ganadora queda expuesto claramente en la Figura 4.5:

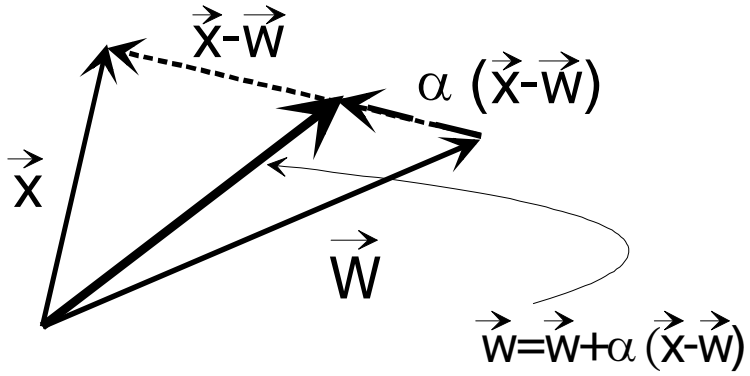


Figura 4.5. Esquema de la actualización de los pesos en el SOM.

Como vemos de la Figura 4.5 lo que hacemos al actualizar según la ecuación 8 es acercar el vector de pesos al vector de entrada.

Si este proceso se repite, al final la red autoorganizativa especializa cada neurona de la capa de salida en la representación de una clase a la que pertenece dicha información de entrada. Sin embargo, esto no nos garantiza que los representantes de clases parecidas se dispongan cerca en la capa de salida. Para ello, el mapa de Kohonen incorpora una interacción lateral entre las neuronas adyacentes con el fin de conseguir que neuronas próximas en la capa de salida sintonicen o representen a patrones similares en la de entrada. El alcance de esta interaccción viene fijada por una función definida como función de vecindad. La función de vecindad determina el grupo de neuronas más o menos próximas a la neurona ganadora en el proceso de competición y la intensidad con que éstas deben modificar sus pesos sinápticos, es decir, ahora la modificación de los pesos no sólo se aplica a la neurona ganadora sino que también a aquellas que comprenda el alcance de las interacciones laterales definidas por la función vecindad. De este modo se consigue que el mapeado generado cumpla el objetivo de que patrones similares en la entrada se correspondan con la activación de neuronas próximas en la capa de salida.

4.3.2. Algoritmo de aprendizaje.

A continuación se describe el algoritmo habitual para el entrenamiento del mapa de Kohonen.

1. Inicialización de los pesos que se puede realizar de diferentes formas: asignando como valores iniciales de los pesos unas determinadas entradas o, simplemente, determinándolos de forma aleatoria.

2. Presentación en cada iteración de un patrón de entrada x(t).

3. Determinación de la similitud entre los pesos de cada neurona y las entradas. Si consideramos la distancia euclidea como medida de comparación tenemos

$$d(w_{ij}, x_R) = \sum_{k=1}^{M} (w_{ij}^k - x_R^k)^2 \qquad \text{Ec. } 4.16$$

4. Determinación de la neurona ganadora (NG). Será aquella en la que obtengamos menor medida de comparación obtengamos.

5. Actualización de los pesos sinápticos; si estamos usando como función similitud la función cuadrática se tiene:

$$w_{ij}(n+1) = w_{ij}(n) + \alpha(n)h(n)(x_j - w_{ij}) \qquad \text{Ec. } 4.17$$

donde $\propto$(n) es la velocidad de aprendizaje, equivalente a la constante de adaptación del perceptrón multicapa y h(n) es la función de vecindad (con pico de amplitud en la neurona vencedora).

6. Si se ha alcanzado el máximo número de iteraciones acabar, si no volver al paso 2.

En el algoritmo planteado han aparecido varias variables que hay que tener en cuenta y que estudiaremos por separado. Estas son la función vecindad, el ritmo de aprendizaje y la actualización de los pesos de acuerdo con la función de similitud utilizada.

La función de vecindad tiene una forma definida pero su radio suele variar con el tiempo de tal forma que se parte de un radio grande con el fin de obtener una ordenación global del mapa y se reduce hasta finalmente actualizar solamente los pesos de la neurona vencedora y aquellas muy próximas. Dicho de otro modo, primero tenemos un aprendizaje a grandes rasgos que conforme avanza el número de iteraciones se especializa. Las dos funciones de vecindad más extendidas quedan expuestas en la Figura 4.6:

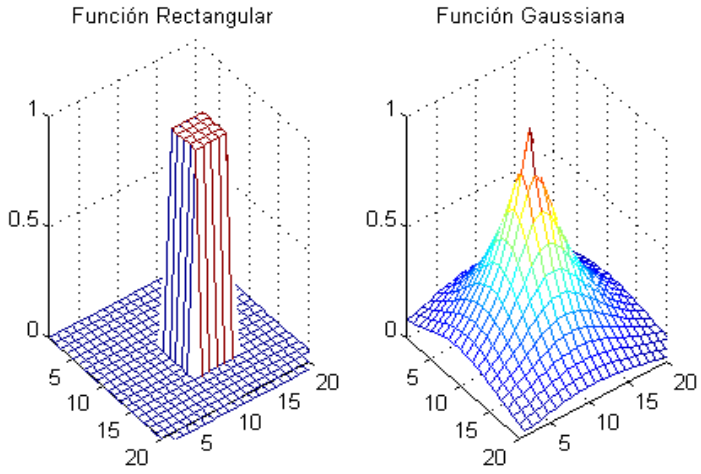


Figura 4.6. Funciones de vecindad más extendidas.

Como se ha dicho anteriormente el número de vecinos que se actualizan se reduce conforme el aprendizaje avanza, a modo de ejemplo, para la función gaussiana se tiene como función vecindad:

$$h_{ws} = e^{-\frac{\|\vec{w}-\vec{s}\|^2}{2\cdot\sigma^2}} \qquad \text{Ec. 4.18}$$

Aquí la anchura de la función viene fijada por el parámetro σ^2. Este parámetro normalmente sigue una variación fijada por la siguiente expresión:

$$\sigma(t) = \sigma_i \cdot \left(\frac{\sigma_f}{\sigma_i}\right)^{\left(\frac{t}{t_{MAX}}\right)} \qquad \text{Ec. 4.19}$$

Donde los subíndices i y f hacen referencia a los valores iniciales y finales respectivamente siendo t el índice temporal.

El ritmo de aprendizaje fija la velocidad de cambio de los pesos. Este se puede tomar constante o en función del número de iteraciones. La elección más usual es considerar la misma variación exponencial que la definida para el radio de vecindad.

Por último la actualización de los pesos tiene en cuenta el criterio de similitud usado para determinar la neurona ganadora. Se tienen diferentes actualizaciones en función de la medida usada. A continuación presentamos algunos criterios con sus correspondientes reglas de aprendizaje.

Función Distancia	Regla de Aprendizaje
$d(\vec{w}_{ij}, \vec{x}_k) = \sum_{R=1}^{M} \left\| w_{ij}^R - x_k^R \right\|$	$\vec{w}_{ij}(n+1) = \vec{w}_{ij}(n) + \alpha(n) \cdot signo(\vec{x}_k - \vec{w}_{ij})$
$d(\vec{w}_{ij}, \vec{x}_k) = \sum_{R=1}^{M} (w_{ij}^R - x_k^R)^2$	$\vec{w}_{ij}(n+1) = \vec{w}_{ij}(n) + \alpha(n)(\vec{x}_k - \vec{w}_{ij})$
$d(\vec{w}_{ij}, \vec{x}_k) = \sum_{R=1}^{M} (w_{ij}^R \cdot x_k^R) = y_{ij}$	$\vec{w}_{ij}(n+1) = \vec{w}_{ij}(n) + \alpha(n)(\vec{x}_k - y_{ij}(n) \cdot \vec{w}_{ij}(n))$
$d(\vec{w}_{ij}, \vec{x}_k) = \sum_{R=1}^{M} (w_{ij}^R - x_k^R)$	$w_{ijk}(n+1) = \frac{w_{ijk}(n) + \alpha(n) x_k(n)}{\left\| w_{ijk}(n) + \alpha(n) x_k(n) \right\|}$

Como aplicación del SOM se va a considerar la modelización de funciones de distribución. Se toman dos distribuciones de datos diferentes; estas distribuciones van a ser una distribución uniforme y una corona circular.

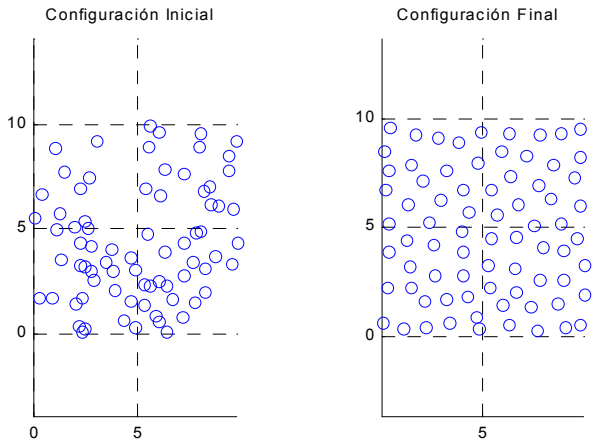


Figura 4.7. Pesos obtenidos cuando se entrena con una distribución uniforme de datos.

Como se observa de la figura los pesos representan la distribución usada (se reparten de forma uniforme en el rango de variación de los datos de entrada. Para entrenar este sistema se ha escogido una inicialización aleatoria de los pesos, una función de vecindad gaussiana (σ_i y σ_j igual a 0.1 y 0.05 respectivamente) con una relación de aprendizaje inicial de 0'75 y final de 0'1. Como variación de estos parámetros se ha considerado la definida por las ecuaciones 5'28 y 5'29. Otro ejemplo de distribución de los datos sería una distribución circular. Los resultados obtenidos usando los mismos parámetros de aprendizaje se muestran en la figura.

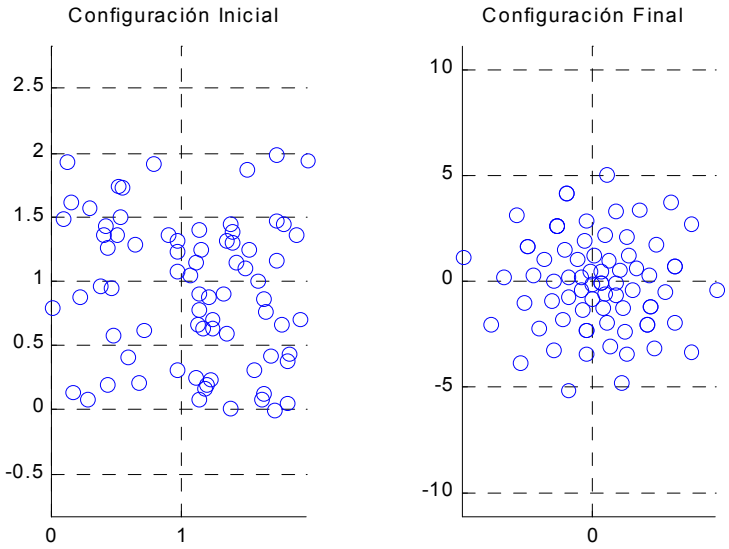


Figura 4.8. Esquema de los resultados aplicando un

SOM con una distribución de datos circular.

4.3.3. Variaciones del SOM.

Existen variaciones del SOM que proporcionan un mejor resultado para algunas aplicaciones específicas. En este apartado se verán algunas de estas variaciones.

Neural Gas [Fritzke-97]. En este algoritmo los pesos son ordenados de acuerdo con la función similitud utilizada de tal forma que son actualizados de forma proporcional al lugar que ocupan en esta lista ordenada. Este algoritmo tendría los siguientes pasos:

1. Determinación de la función similitud del vector de entrada a cada uno de los pesos.

2. Ordenación de los diferentes pesos según el valor de esta función. Así denotamos por K_{ij} el lugar ocupado por el peso ij.

3. Actualización de los pesos de acuerdo a la siguiente expresión:

$$W_{ij}''' = W_{ij}''' + h_\lambda(K_{ij}) \cdot \left(X - W_{ij}'''\right) \qquad \text{Ec. 4.20}$$

donde se tiene:

$$h_\lambda(K) = e^{-\left(\frac{k}{\lambda(t)}\right)} \qquad \text{Ec. 4.21}$$

y donde el parámetro λ sigue la variación:

$$\lambda(t) = \lambda_i \cdot \left(\frac{\lambda_f}{\lambda_i}\right)^{\frac{t}{t_{MAX}}} \qquad \text{Ec. 4.22}$$

donde λ_i y λ_j son los valores inicial y final de dicho parámetro.

4. Volver al paso 1.

En este algoritmo se comprueba que la función vecindad se ha sustituido por otra semejante, la función h(K). Semejante porque la actualización de un peso sigue dependiendo de su distancia al vector ganador. Sin embargo, ya no se usa el valor de esa distancia directamente sino otra cantidad relacionada con ella.

De la expresión del algoritmo podemos deducir que su principal problema es la carga computacional frente al SOM original. Este incremento se debe principalmente a la tarea de ordenar los pesos según su distancia al vector de entrada. Ventaja frente al SOM es que la actualización de los pesos alejados del vencedor no se penaliza tan fuertemente y, por tanto, se consiguen resultados más parecidos a la distribución que se quiere modelizar.

Máxima Entropía [Rose-90] . La diferencia de este algoritmo con el SOM descrito anteriormente no es muy grande. En este algoritmo la actualización de los pesos viene definida por la siguiente expresión:

$$\Delta w_k = \alpha \cdot (\mathrm{x} - \mathrm{w_k}) \frac{e^{\frac{-\|\mathrm{x}-\mathrm{w_i}\|^2}{T}}}{\sum_{\mathrm{i}=1}^{N} e^{\frac{-\|\mathrm{x}-\mathrm{w_i}\|^2}{T}}} \qquad \text{Ec. } \natural 4.23$$

El parámetro T recibe el nombre de temperatura y disminuye con el número de iteraciones. Se le conoce con este nombre porque las bases de este algoritmo se encuentran en la mecánica estadística.

Algoritmo de consciencia. En este algoritmo se plantea penalizar la actualización de los pesos en relación al número de veces que se han actualizado. Lo que se pretende evitar con esta modificación es el hecho que una neurona acapare todas las actualizaciones. Este algoritmo vendría definido por los siguiente pasos:

1. Inicialización de las variables p_k, contador del número de actualización del peso k, a 0 y los pesos del sistema.

2. Determinación de la función similitud del vector de entrada a cada uno de los pesos. Obtención del peso más cercano, w_i.

3. Determinación, para cada peso, de la variable que determina el número de actualizaciones:

$$p_k = p_k + \alpha \cdot (\beta - p_k) \qquad \text{Ec. } \natural 4.24$$

donde β vale 1 si el peso k es el vector más cercano al vector de entrada y 0 en otro caso. El parámetro α toma un valor cercano a 0.

4. Determinación del peso definido por:

$$\mathrm{w_s} = \mathrm{mínimo}(\mathrm{distancia}(\mathrm{x}, \mathrm{w_k}) - \mathrm{b_k}) \qquad \text{Ec. } \natural 4.25$$

con el parámetro b_k definido como:

$$b_k = C \cdot \left(\frac{1}{N} - p_k \right) \qquad \text{Ec. } \natural 4.26$$

siendo N el número de neuronas y C un parámetro que, normalmente, se toma igual a 1.

5. Seguidamente se actualiza solamente el peso ganador; no se aplica ninguna función vecindad.

$$w_s = w_s + \mu \cdot (x - w_s) \qquad \text{Ec. } \natural 4.27$$

Siendo μ la constante de aprendizaje del mapa autoorganizativo.

4.4. LVQ.

Este es el acrónimo de Learning Vector Quantization. Su arquitectura y forma de funcionar es similar a los mapas autoorganizados, sin embargo, el tipo de aprendizaje en estos sistemas es de tipo supervisado. Existen diferentes variantes que pasaremos a describir a continuación.

4.4.1. LVQ1

El algoritmo de aprendizaje de esta red consiste en determinar el patrón más cercano, según alguna medida de similitud, al patrón de entrada. Si las clases del vector de entrada y el vencedor son iguales se "acerca" el vector ganador al vector de entrada en el sentido expuesto cuando se explicó la regla de aprendizaje en redes de tipo SOM. Si no ocurre esto se "aleja". Si usamos como función similitud la distancia euclídea este algoritmo lo podríamos expresar como:

$$\begin{aligned} w^* &= w^* + \alpha \cdot (x - w^*) \text{ si clase}(x) = \text{clase}(w^*) \\ w^* &= w^* - \alpha \cdot (x - w^*) \text{ si clase}(x) \neq \text{clase}(w^*) \end{aligned} \qquad \text{Ec. } 4.28$$

siendo w el vector de pesos de la neurona vencedora y x el vector de entrada.

Con esta actualización acercamos el vector de pesos al vector de entrada de tal forma que si, en el futuro se tiene un patrón parecido al vector de entrada se seguirá asignando al mismo vector de pesos. Normalmente este algoritmo toma una constante de aprendizaje que es inversamente proporcional al número de iteraciones. Respecto de este punto hay que ser cautelosos, una constante demasiado pequeña puede conducir a cambios demasiado pequeños en los pesos como para ser representativos. Por contra una constante muy grande puede producir oscilaciones en el valor de éstos sin que se llegue a una convergencia de los pesos de la red que permitan modelizar la distribución de los datos de entrada.

Como ejemplo de aplicación de este algoritmo se considera un problema de clasificación de vectores de dos dimensiones. En este problema se tienen cuatro clases correspondientes a diferentes zonas del plano. En primer lugar se considerarán que tenemos cuatro vectores de pesos, uno para cada clase.

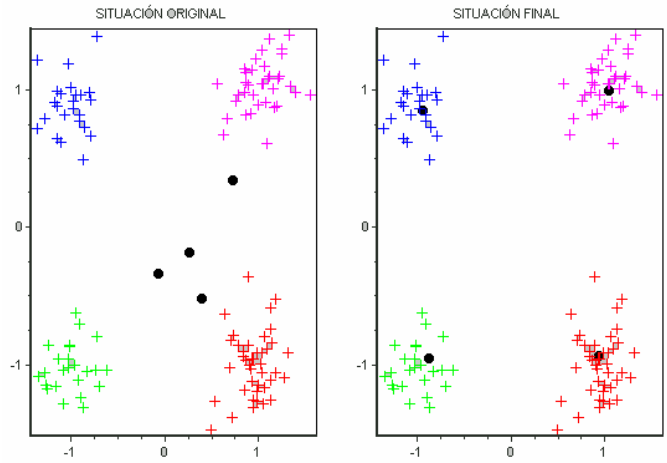


Figura 4.9. Resultados obtenidos con el algoritmo LVQ1.

En la gráfica se aprecia que los vectores de los pesos tienden hacia los centroides de las diferentes agrupamientos ("clusters") definidos en este ejemplo. Una vez que el sistema ha sido entrenado puede funcionar como un clasificador;

asignando al vector de entrada la clase correspondiente al vector de pesos más cercano a dicho vector. A cada uno de los pesos que forman la red se le conoce con el sobrenombre de libro de código ("codebook" fue el término anglosajón usado para definirlos). Si nos fijamos en la gráfica podemos ver de forma intuitiva las razones de este nombre: los vectores de entrada pueden ser codificados considerando como referencia los pesos de la red. Esto permite usar un menor número de bits pues la distancia a estos vectores es menor que al origen y, por tanto, reduciremos la codificación de los datos (de ahí el nombre de este algoritmo, Learning Vector Quantization.

Existe una variación de este algoritmo conocida como OLVQ1 (Optimum LVQ1) en el que la constante de aprendizaje vienen definida por la siguiente expresión:

$$\alpha_l(n) = \frac{\alpha_l(n-1)}{1+(-1)^{\beta}\cdot\alpha_l(n-1)} \qquad \text{Ec. } \natural 4.29$$

Donde el parámetro β vale 0 si se clasifica correctamente el patrón y 1 en caso contrario y el subíndice l hace referencia al peso que se actualiza. Esta variación es fácil de analizar, si nos equivocamos aumentamos la constante de aprendizaje por lo que nos alejamos más rápido y si acertamos la constante disminuye produciéndose entonces un ajuste fino en el aprendizaje de los pesos.

4.4.2. LVQ2.1.

Este algoritmo considera, aparte del vector de pesos vencedor, el vector de pesos siguiente en cuanto a cercanía al vector de entrada. La adaptación de estos pesos dependen de varios factores:

- Los dos vectores anteriormente mencionados representan a clases diferentes.
- El vector de entrada pertenece a la misma clase que uno de ellos.
- El vector de entrada está en una ventana definida sobre el punto medio de estos dos vectores, estos es, si definimos la anchura de la ventana como w se cumple:

$$\text{mín imo}\left(\frac{d_1}{d_2},\frac{d_2}{d_1}\right) > \frac{1-w}{1+w} \qquad \text{Ec. } \natural 4.30$$

siendo d_1 y d_2 las distancias de los dos vectores más próximos al vector de entrada.

Si se dan estos factores se actualizan los pesos de acuerdo a la siguiente regla:

$$\begin{aligned} w_k &= w_k + \alpha\cdot(x - w_k) \\ w_j &= w_j - \alpha\cdot(x - w_j) \end{aligned} \qquad \text{Ec. } \natural 4.31$$

donde los pesos k y j son los más cercanos, perteneciendo x a la clase definida por el peso k. De la ecuación de actualización de los pesos lo que se hace en este algoritmo es intentar corregir el posible error de asignación de clases cuando tenemos dos vectores de pesos muy próximos entre sí. Si nos equivocamos entonces alejamos el vector que no representa el patrón de entrada y acercamos el otro

vector. De esta forma se pone especial cuidado en la definición de las fronteras entre clases.

4.4.3. LVQ3.

En el anterior algoritmo se pone especial cuidado en las fronteras entre clases, sin embargo no se preocupa de dónde acaban situándose los pesos al final del proceso de aprendizaje. Para solucionar este problema aparece esta variante. Se sigue manteniendo la actualización de los pesos anterior (sujeta a las mismas condiciones). Si los dos pesos más cercanos representan la misma clase se actualizan de acuerdo con la siguiente expresión:

$$w_k = w_k + \beta \cdot \alpha \cdot (x - w_k) \qquad \text{Ec. 4.32}$$

donde β es un parámetro menor que 1.

Con esta actualización aseguramos que los pesos no sólo respeten las fronteras de decisión sino que, además, representen la distribución de los datos de entrada.

El procedimiento aconsejado es usar el OLVQ1 hasta la convergencia (normalmente rápida) para, posteriormente, usar el LVQ1 o/y el LVQ3; por otra parte no se aconseja utilizar en muchas iteraciones la variante 2.1 [Ripley-96].

Todas estas variantes junto con el SOM se encuentran disponibles en la dirección Web http://www.cis.hut.fi/nnrc/. Aparte del paquete informático se dispone de abundante documentación sobre el tema por lo que se recomienda la visita a esta página Web.

4.5. ART (Adaptive Resonance Theory).

La Teoría de la Resonancia Adaptativa, nace de la necesidad de superar lo que Grossberg ha definido como dilema estabilidad – plasticidad; trata de diseñar sistemas capaces de aprender nuevas categorías (plasticidad de la red) pero de manera estable, es decir, sin que el aprendizaje de "cosas nuevas" conduzca irremediablemente al "olvido catastrófico" de categorías previamente aprendidas o establecidas por la red. Es también necesario poseer mecanismos que indiquen en qué estado debe mantenerse la red (plástico – estable) dependiendo de las entradas actuales al sistema. El modelo ART propone añadir a las redes competitivas un mecanismo de realimentación entre las capas F1 y F2. Este mecanismo asegura el aprendizaje de nuevas categorías sin destruir o degradar la información sobre categorías previamente establecidas, realizándose todo el proceso de forma no supervisada.

La teoría de la resonancia adaptativa se basa en la idea de que el aprendizaje en este tipo de redes, se dá cuando existe un fenómeno de resonancia entre el estímulo de entrada y algún prototipo de las categorías establecidas por la red. Si el estímulo de entrada entra en resonancia con alguno de estos prototipos, es decir, si el estimulo de entrada es lo suficientemente parecido a un prototipo establecido por la red, entonces se considera que el estímulo pertenece a dicha categoría.

Posteriormente se lleva a cabo una leve modificación del prototipo representante de la categoría para que éste incorpore algunos de los rasgos o "características" del estímulo presentado. Si el estímulo de entrada no resuena con ningún prototipo, es decir, no se parece lo suficiente a ningún prototipo de categoría preexistente, la red se encarga de crear una nueva categoría asociada a dicho estímulo. El grado de similitud entre un estímulo de entrada y el prototipo de categoría seleccionado en el proceso competitivo lo controla un parámetro, denominado parámetro de vigilancia. Los valores la vigilancia están entre cero y uno. Valores de la vigilancia próximos a uno permiten que la red establezca categorías de modo fino o dicho en otras palabras, la red establece categorías cuyas diferencias son pequeñas. Vigilancias con valores más próximos al valor cero llevan a categorizaciones donde las diferencias entre los prototipos que a éstas representan son más gruesas o menos sutiles.

Dentro de este tipo de redes podemos distinguir dos tipos especialmente adecuados para las tareas de clasificación de patrones:

- ART1. Usada para patrones de entrada binarios (0 y 1).
- ART2. Extensión de la anterior y que puede ser aplicada con entradas continuas.

A continuación pasamos a describir cada una de estas redes.

4.5.1. ART1.

El modo básico de funcionamiento de esta red ha sido descrito anteriormente y consiste en la creación de una nueva clase si el patrón de entrada no se parece a las ya definidas o, en caso contrario, actualizar el vector prototipo de la que más se parezca. En la descripción que acabamos de realizar hay que llevar a cabo dos tareas:

- Determinación del vector prototipo más parecido al vector de entrada.
- Comprobación que el nivel de semejanza es el adecuado.

Estos dos procesos se pueden implementar a nivel neuronal. Aquí, por sencillez y para una mayor claridad de ideas, no se describirá la arquitectura completa que supondría la implementación neuronal de los dos puntos anteriormente mencionados. El modo de funcionamiento de la red ART1 sería el siguiente [Kung-93]:

1. Inicialización de los pesos de la red que conectan la capa de entrada con la de salida. Aquí nos encontramos con dos tipos de pesos los s_{ij} de tipo binario (0 y 1) y los w_{ij} de tipo real. Inicialmente se consideran los s_{ij} igual a 1 y los w_{ij} igual a 1/(N+1); siendo N el número de clases que consideramos al principio.

2. Dado un vector de entrada x (los vectores los consideramos vectores fila), se determina para la neurona j de la capa de salida el siguiente producto escalar:

$$y_k = w_k^t \cdot x \qquad \text{Ec. 4.33}$$

donde el superíndice t indica trasposición.

3. Determinación de la neurona que presenta la mayor salida. Esta neurona la denotaremos por neurona v.

4. Comprobar si la neurona cumple:

$$y_v = \frac{s_v^t \cdot x}{\|x\|^2} \geq \rho \qquad \text{Ec. 4.34}$$

con $0<\rho<1$.

5. Si se cumple 4 entonces el vector de entrada pertenece a la clase definida por la neurona v; actualizando los vectores prototipos de esta clase según el siguiente procedimiento:

$$w_{vj} = \frac{\mathbf{s}_v^t \cdot \mathbf{x}}{\beta + \mathbf{s}_v^t \cdot \mathbf{x}} \qquad \text{Ec. 4.35}$$

$$s_{vj} = s_{vj} \cdot x_j \ \forall \ j \qquad \text{Ec. 4.36}$$

donde β es un parámetro que normalmente se toma como 0.5 (para evitar divisiones por cero).

6. Si se cumple 4, sin tener en cuenta el vector v, se tomará la neurona que presente el mayor valor de salida. Se repiten el test y, si se sigue en la misma situación de no cumplirse la condición definida en el punto 4, se crea una nueva neurona de salida, un nuevo prototipo, cuyo vector de pesos se corresponde con el vector de entrada.

El primer punto que llama la atención es la normalización de los pesos w_{ij} (pasos 1 y 5). Con esta normalización le damos más importancia a los vectores de pesos con menor número de unos (recordemos que las entradas en esta red son binarias 0 y 1). En efecto, supongamos que se tienen dos vectores de pesos donde uno de ellos tiene unos en las mismas posiciones que el otro vector; a modo de ejemplo s_1 =[1 1 0 0 0 0 1] y s_2 =[1 1 0 0 1 1 1] teniendo como vector de entrada X=[1 1 0 0 0 0 0]; aquí el producto escalar con el vector de entrada es el mismo para los dos vectores de pesos, dos, pero, después de normalizar, el vector ganador resulta s_1 (tiene menor norma, 3 frente a 5 de s_2). Este sesgo hacia los vectores con menor número de unos se mantiene en la ley de actualización de los pesos.

En cuanto al factor ρ, denominado parámetro de vigilancia, controla el grado de similitud del vector de entrada con el vector prototipo; un valor próximo a cero supone una baja semejanza mientras que un valor próximo a uno indica una alta semejanza con el prototipo. Esto se traduce en la creación de más prototipos conforme aumenta el valor del parámetro.

Como ejemplo de aplicación de las ART1 se plantea la clasificación de los siguientes patrones binarios (las cuadrículas en negro se corresponden con unos y los blancos con ceros). En las simulaciones el parámetro de tolerancia se cambiará comprobándose la variación del número de prototipos con él. Los patrones considerados han sido los siguientes:

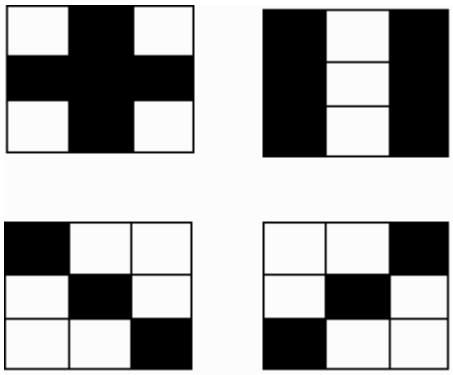


Figura 4.10. Esquema de los patrones de entrada usados.

Si se plantea una red ART1 con estos patrones y un parámetros de vigilancia de 0.8 los vectores prototipo de las clases se corresponden con los vectores de entrada, estableciéndose al final cuatro categorías. Sin embargo, si se baja el parámetro de vigilancia a 0.3 el número de categorías que se establece es de tres con los siguientes vectores prototipo:

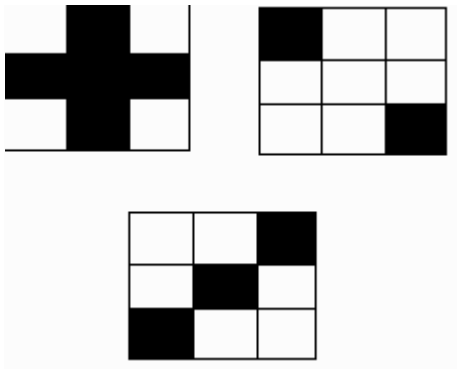


Figura 4.11. Prototipos de salida con un parámetro de 0.3.

Se aprecia que al bajar el parámetro de vigilancia aparecen menos agrupamientos; el sistema se vuelve menos restrictivo a la hora de incluir los vectores de entrada en las clases ya existentes.

4.5.2. ART2.

En esta red se admiten entradas analógicas, ya no tienen que ser ceros o unos. La filosofía de aprendizaje de la red sigue manteniéndose:

- La estructura de la red no cambia si los patrones de entrada son semejantes a los vectores prototipos que definen las diferentes clases.
- Si el vector de entrada es muy diferente a las clases existentes se crea una nueva.

Las variaciones que se plantean en esta red en relación a su predecesora se basan en los criterios para comparar la semejanza entre dos vectores; ahora se usa la distancia euclídea como método de comparación. El otro cambio estriba en la actualización de los pesos w_{ij} (aquí no aparecen los s_{ij}). Esta actualización sigue la siguiente expresión:

$$w_k = \frac{x + w_k \cdot N_k}{N_k + 1} \qquad \text{Ec. } 4.37$$

donde N_k es el número de vectores de entrada que tiene el agrupamiento k.

5

Hopfield, Comités de expertos, OCON, Recurrentes, Neurodifusas y SVM

RED DE HOPFIELD. ESTRUCTURA, ALGORITMO DE APRENDIZAJE Y REDES DERIVADAS.

5.1. Red de Hopfield.

5.1.1. Introducción.

Los diferentes tipos de sistemas neuronales explicados hasta el momento (sistemas adaptativos, perceptrones multicapa y mapas autoorganizativos) tienen en común que las conexiones entre los elementos constituyentes de estas redes son siempre en un sentido, es decir, no existen realimentaciones entre neuronas. En este capítulo, estudiaremos un tipo de arquitectura donde sí existen estas realimentaciones. La red en particular que se va a estudiar es la llamada red de Hopfield, cuyo principal campo de aplicación es la optimización de procesos.

La red fue propuesta en 1982 por el afamado físico teórico John Hopfield. Su gran reputación provocó una importante difusión de esta red, lo que a su vez supuso un crecimiento del campo de las redes neuronales que se encontraba en un estado de abandono tras las críticas que había sufrido por Minsky y Papert en 1969, como ya se explicó en el Capítulo 2. Aunque en su trabajo Hopfield recoge ideas de otros investigadores (Amari y Grossberg principalmente), el enfoque con que plantea la red es totalmente novedoso, y tiene tres puntos destacables fundamentalmente [Hopfield-82]:

1. Se basa en el planteamiento de una memoria asociativa; se hace necesario entonces definir una función energía.

2. Pone de manifiesto la analogía existente entre su modelo y la física estadística clásica, lo que permite usar las herramientas matemáticas que se utilizaban desde hacía mucho tiempo en ese campo y que, por tanto, eran bien conocidas.

3. Se destaca en su trabajo la facilidad de implementación de su modelo aprovechando la tecnología VLSI ("Very Large Scale Integration").

La red recurrente que plantea el modelo propuesto por Hopfield se basa en almacenar información en un sistema que presenta una configuración dinámica estable, es decir, la red de Hopfield se plantea como una memoria asociativa o memoria direccionable por contenido. Intuitivamente, la idea de Hopfield es localizar cada patrón que se quiere almacenar en la red en el fondo de un "valle" de nuestra función energía. El modo de funcionamiento de esta memoria dinámica será partir de un determinado estado inicial (información de partida) tras lo cual se dejará evolucionar al sistema hasta llegar a un estado estable. Este estado estable será el patrón que se corresponde con nuestro estado inicial. Hay que destacar que el patrón inicial puede ser una versión deteriorada o incompleta del patrón que se

quiere obtener, de manera que la red encontrará el patrón de los almacenados inicialmente que más se parece a la entrada planteada. En la siguiente figura (Fig. 1) se muestra la gráfica de una función energía donde aparecen montañas y valles.

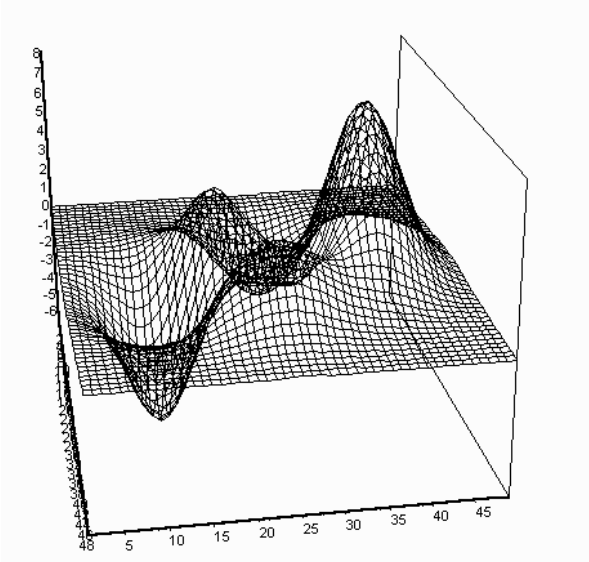


Figura 5.1. Esquema de una función energía.

5.1.2. Arquitectura.

El modelo original de la red de Hopfield es una red monocapa con N neuronas cuyas salidas toman el valor 0/1 [Hopfield-82]. En lo que sigue se tomarán estas salidas como ±1, convenio seguido por la mayoría de textos sobre redes neuronales. Cada neurona se conecta a las demás, apareciendo conexiones laterales, pero no se conectan consigo mismas por lo que no existen conexiones auto-recurrentes. Esta es la diferencia básica desde el punto de vista de la arquitectura con las redes recurrentes que se verán en un capítulo posterior. Además los pesos asociados a pares de neuronas son simétricos, es decir se da la igualdad $w_{ij}=w_{ji}$:

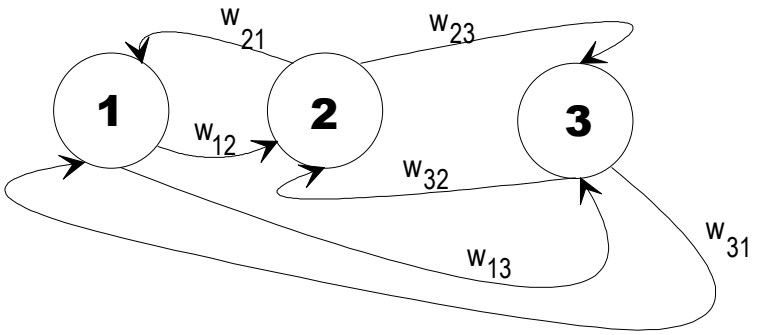


Figura 5.2. Esquema de la red de Hopfield original.

El tipo de neurona usado en el trabajo original de Hopfield es la neurona de McCulloch-Pitts que, recordemos, responde al siguiente esquema:

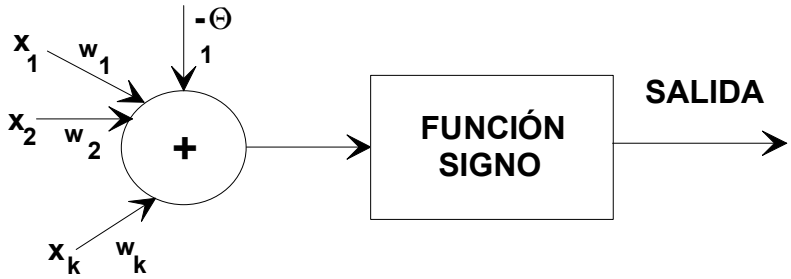


Figura 5.3. Esquema de la neurona de McCulloch-Pitts.

Es decir, para la neurona j, hay que realizar las siguientes operaciones matemáticas:

$$v_j = signo(h_j) = signo(\sum_{s=1}^{K} w_{js} \cdot x_s - \theta_j) \qquad \text{Ec. 5.1}$$

Donde θ_j se conoce como el umbral de la neurona y h_j recibe el nombre de potencial postsináptico. Lógicamente, de acuerdo con la última expresión, las salidas de las neuronas son binarias.

Las tres diferencias del modelo respecto a la red neuronal más conocida en esos momentos, que era el perceptrón multicapa, y que el propio Hopfield destaca en su trabajo, son las siguientes [Hopfield-82]:

- Su modelo incluye realimentaciones, que son básicas en su modo de funcionamiento.
- La elección de la arquitectura del perceptrón multicapa se realiza de forma arbitraria.
- El perceptrón multicapa funciona de manera síncrona, es decir, todas las neuronas cambian al mismo tiempo. La red de Hopfield permite un funcionamiento tanto síncrono como asíncrono, pero es de destacar que el funcionamiento síncrono no es el habitual en las neuronas biológicas.

Posteriormente, Hopfield amplió el rango de salida de la neurona permitiendo una salida continua [Hopfield-84]. Para ello cambió la función signo a la salida de la neurona por una sigmoide (rango de salida 0-1). En lo que sigue denominaremos el primer caso como caso discreto y el segundo como caso continuo. Aparte de la diferencia sobre el rango de variación de la salida aparece otra en cuanto a los pesos de la red; en el caso continuo se permite la auto-realimentación en una neurona (es decir, se permite que $w_{ii} \neq 0$).

5.1.3. Funcionamiento de la red de Hopfield.

El primer paso a la hora de trabajar con una red de Hopfield es codificar y representar la información en forma de vector. Esta codificación será binaria si se usa la neurona de McCulloch-Pitts o continua si se trata de una neurona no lineal que utiliza como función de activación la sigmoide (rango de variación entre 0 y 1) o la tangente hiperbólica (rango de variación entre –1 y +1). El vector de entrada,

que denominaremos x, debe tener tantas componentes como neuronas tenga la red. La entrada es aplicada en t = 0 a la única capa que tiene la red, determinándose así las salidas $v_j(0)$. Debido a las realimentaciones, estas salidas se convierten en las nuevas entradas a la red.

Tenemos pues una relación dinámica que responde a la siguiente ley:

$$x_j(t+1) = f\left(\sum_{s=1}^{N} w_{js} \cdot x_s(t) - \theta_j\right) \qquad \text{Ec. 5.2}$$

En la anterior ecuación, f es la función de activación considerada. Si se usa la función signo existe una ambigüedad en forma de discontinuidad de la función cuando la entrada es 0. En este caso, se puede o bien considerar ese valor como ±1 o considerar que la salida de la red es el mismo valor que el de la entrada.

Recordemos que la red de Hopfield funciona como una memoria dinámica que asocia la entrada a la red con unos determinados patrones, y que como se trata de una "red recurrente" las salidas de la red se convierten en las nuevas entradas; por tanto, la condición de parada, analizando la ecuación de la dinámica de la red, será aquélla que lleve a una situación de equilibrio, que se producirá cuando se tenga la siguiente igualdad:

$$x(t+1)=x(t) \qquad \text{Ec. 5.3}$$

El siguiente punto a tratar es la forma de actualizar las neuronas: el algoritmo de aprendizaje. Existen dos posibilidades de funcionamiento, que suponen dos formas distintas de realizar esta actualización:

1. Dinámica asíncrona o modo serie. En cada instante de tiempo solamente se actualiza una neurona. Esta neurona puede ser seleccionada aleatoriamente o bien se puede seguir un orden preestablecido, pero deben actualizarse todas. Se debe intentar distribuir de forma aleatoria pero uniforme esa actualización; una posibilidad en este sentido es la actualización de modo cíclico, ya que se puede demostrar (lo haremos en el apartado de la función energía) que se llega a un estado estable.

2. Dinámica síncrona o modo paralelo. En un instante t de tiempo, todas o varias neuronas, actualizan su estado a la vez. En el caso de que sean todas las neuronas de la red hablamos de un modo completamente paralelo, que es el caso más común. Diferentes dinámicas de actualización aplicadas sobre la misma arquitectura conducen a resultados distintos.

Veamos la diferencia de funcionamiento entre estas dos formas de actualizar los pesos con un simple ejemplo; supongamos que se tiene la siguiente red discreta de Hopfield en la que el valor de los pesos queda representado en la figura y se han tomado los sesgos todos iguales a uno.

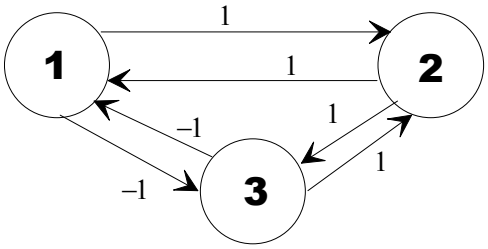


Figura 5.4. Esquema de la red discreta usada.

Como vector inicial de entrada de esta red se ha tomado el vector [1 -1 1]. En un modo de actualización síncrona oscilamos entre los estados [-1 1 -1] y [1 -1 1]. A modo de ejemplo, se verá la primera actualización, que se rige por la relación dinámica representada en la ecuación 2:

$$v_1 = \text{signo}[w_{12}x_2+w_{13}x_3-\theta_1] = \text{signo}[(1\cdot(-1))+((-1)\cdot 1)-1] = \text{signo}[-3]=-1 \qquad \text{Ec. 5.4}$$

$$v_2 = \text{signo}[w_{21}x_1+w_{23}x_3-\theta_2] = \text{signo}[(1\cdot 1)+(1\cdot 1)-1] = \text{signo}[1]=1 \qquad \text{Ec. 5.5}$$

$$v_3 = \text{signo}[w_{31}x_1+w_{32}x_2-\theta_3] = \text{signo}[((-1)\cdot(-1))+(1\cdot(-1))-1] = \text{signo}[-3]=-1 \qquad \text{Ec. 5.6}$$

En las anteriores ecuaciones, representamos por v_j la salida correspondiente a la neurona j, y se ha utilizado la función signo como función de activación, ya que se trata de una red de Hopfield discreta. Como se puede observar, en efecto se producirá una oscilación entre los estados [1 -1 1] y [-1 1 -1] si la actualización es síncrona.

Si el modo de actualización es asíncrono, actualizándose primero la neurona 1, a continuación la 2 y por último la 3, se tiene que siguiendo la misma notación:

$$x_1 = \text{signo}[w_{12}x_2+w_{13}x_3-\theta_1] = \text{signo}[(1\cdot(-1))+((-1)\cdot 1)-1] = \text{signo}[-3] = -1 \qquad \text{Ec. 5.7}$$

$$x_2 = \text{signo}[w_{21}x_1+w_{23}x_3-\theta_2] = \text{signo}[(1\cdot(-1))+(1\cdot 1)-1] = \text{signo}[-1] = -1 \qquad \text{Ec. 5.8}$$

$$x_3 = \text{signo}[w_{31}x_1+w_{32}x_2-\theta_3] = \text{signo}[((-1)\cdot(-1))+(1\cdot(-1))-1] = \text{signo}[-1] = -1 \qquad \text{Ec. 5.9}$$

Se observa como el estado final para el caso de actualización asíncrona alcanzado por la red es [-1 -1 -1].

5.1.4. Función energía.

Para a obtención de la función energía para la red discreta se va a partir de la ecuación de actualización de los pesos. Es un enfoque diferente al planteado por Hopfield en su trabajo original pero tiene como ventaja su sencillez. Supondremos que el modo de actualización es asíncrono (una neurona cada vez) y que la salida varía entre ±1. Así, de acuerdo con la ecuación que rige la dinámica de la red discreta de Hopfield, se tendrá:

$$x_j(t+1) = signo(h_j(t)) = \begin{cases} h_j(t) > 0 \Leftrightarrow x_j(t+1) = 1 \Rightarrow \Delta x_j(t) = x_j(t+1) - x_j(t) \geq 0 \\ h_j(t) < 0 \Leftrightarrow x_j(t+1) = -1 \Rightarrow \Delta x_j(t) = x_j(t+1) - x_j(t) \leq 0 \end{cases} \qquad \text{Ec. 5.10}$$

Las dos expresiones anteriores se pueden unir en:

$$h_j(t)\cdot \Delta x(t) \geq \mathbf{0} \Leftrightarrow \left(\sum_{s=1}^{N} w_{js}\cdot x_s(t) - \theta_j\right)\cdot \Delta x_j(t) \geq \mathbf{0} \qquad \text{Ec. 5.11}$$

Se tiene entonces que:

$$-\left(\sum_{s=1}^{N} w_{js}\cdot x_s(t) - \theta_j\right)\cdot \Delta x_j(t) \leq \mathbf{0} \qquad \text{Ec. 5.12}$$

Si se define para la energía de la neurona j como:

$$E_j = -\sum_{s} w_{js}\cdot x_s\cdot x_j + \theta_j\cdot x_j \qquad \text{Ec. 5.13}$$

Teniendo en cuenta que en cada iteración sólo se actualiza una neurona, la variación de energía al actualizar la neurona j (las x_s con $s\neq j$ son fijas) la podemos calcular determinando incrementos en la expresión anterior:

$$\Delta E_j = -\left(\sum_{s=1}^{N} w_{js}\cdot x_s(t)\cdot \Delta x_j(t)\right) + \theta_j\cdot \Delta x_j(t) \qquad \text{Ec. 5.14}$$

Esta es precisamente la cantidad definida en el primer miembro de la desigualdad que aparece en la ecuación 6.

Si se tiene en cuenta que no existen conexiones auto-recurrentes (es decir, $w_{ii}=0$ para todo i) y que las conexiones son simétricas ($w_{ij}= w_{ji}$) se tendrá que la suma de las energías de las diferentes neuronas, energía de la red, tomará la siguiente expresión:

$$E = -\frac{1}{2}\cdot \sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}\cdot x_i\cdot x_j + \sum_{i=1}^{N}\theta_i\cdot x_i \qquad \text{Ec. 5.15}$$

De la forma en que la hemos construido, en cada paso se produce un decremento de la función energía y esta variación es siempre menor o igual a cero, o dicho de una forma simple, la energía siempre decrece. Como la energía es una cantidad finita, la red siempre llegará a un estado de mínima energía que se corresponderá con un mínimo local, que puede ser el mínimo global de dicha función.

5.1.5. Regla de aprendizaje de los pesos.

De acuerdo con lo dicho anteriormente, es necesario que los patrones que la red debe almacenar se correspondan con mínimos locales o globales de la función energía. El aprendizaje consistirá entonces en conseguir que la red almacene esos patrones como estados estables del sistema. La primera regla planteada para la determinación de los pesos fue la regla de Hebb. Si se trabajara con neuronas con salidas ±1, los pesos de se actualizarían de acuerdo a:

$$w_{ij} = \begin{cases} \frac{1}{S}\cdot \sum_{p=1}^{S} \varepsilon_i^p\cdot \varepsilon_j^p & \text{si } i \neq j \\ 0 & \text{si } i = j \end{cases} \qquad \text{Ec. 5.16}$$

En la anterior ecuación, ε_i^k es la componente i-ésima del patrón k-ésimo que se quiere almacenar en nuestra memoria asociativa mientras que S hace referencia al número de patrones. Sin embargo, si se trabaja con neuronas cuyas salidas toman el valor 0 o 1, los pesos se determinan según la siguiente fórmula:

$$w_{ij} = \begin{cases} \frac{1}{S} \cdot \sum_{p=1}^{S} (2 \cdot \varepsilon_i^p - 1) \cdot (2 \cdot \varepsilon_j^p - 1) & \text{si i} \neq \text{j} \\ 0 & \text{si i} = \text{j} \end{cases} \qquad \text{Ec. 5.17}$$

En cuanto a los umbrales de la función de activación θ_i, toman el valor cero cuando se trabaja con ±1 como valores de salida y si son 0 y 1 se considera el siguiente valor:

$$\theta_i = \frac{1}{2} \cdot \sum_{j=1}^{N} w_{ji} \qquad \text{Ec. 5.18}$$

El significado de los pesos es claro si se tiene en cuenta que el uso dado a esta red es como memoria. Es necesario tener unos pesos que definan los patrones que se quiere almacenar, donde debe quedar reflejado el parecido entre los diferentes patrones. Esto es lo que se hace en las ecuaciones 10 y 11, que determinan la correlación (esto es, el parecido) entre el patrón i y el patrón j.

Las ecuaciones anteriores se pueden expresar matricialmente de una forma más compacta. En efecto, si se considera la matriz de pesos W de dimensiones N×N (siendo N el número de componentes de los patrones):

$$W = \begin{bmatrix} w_{11} & \dots & w_{1N} \\ \dots & \dots & \dots \\ w_{N1} & \dots & w_{NN} \end{bmatrix} \qquad \text{Ec. 5.19}$$

y se representan los patrones que se quieren almacenar en la red en forma de vectores según la siguiente notación:

$$\begin{aligned} E_1 &= [\varepsilon_1^1 \dots\dots\dots \varepsilon_N^1] \\ E_2 &= [\varepsilon_1^2 \dots\dots\dots \varepsilon_N^2] \\ &\dots\dots\dots \\ E_S &= [\varepsilon_1^S \dots\dots\dots \varepsilon_N^S] \end{aligned} \qquad \text{Ec. 5.20}$$

Entonces la ley de aprendizaje de los pesos (las neuronas tienen como salidas ±1) toma la siguiente expresión:

$$W = \frac{1}{S} \cdot \left(\sum_{k=1}^{S} E_k \cdot E_k^t - I \right) \qquad \text{Ec. 5.21}$$

Donde I es la matriz identidad, lo que permite asegurar que todos los elementos de la diagonal principal de W son iguales a cero.

5.1.6. Aplicaciones de la red de Hopfield.

Aunque debido a su funcionamiento como memoria asociativa también ha sido utilizada en el terreno del reconocimiento de patrones, el campo de aplicación

fundamental de la red de Hopfield es el de la optimización de procesos. De hecho existen muchos problemas de este tipo prácticamente irresolubles desde un punto de vista convencional y que, sin embargo, son abordables con esta red. Por ejemplo, imaginemos un problema en el que se deben colocar N reinas que no se den jaque en un tablero de ajedrez de N×N cuadros; el número de posibles combinaciones es igual a $N^2!/(N^2-N)!N!$, por lo que en el caso N=8, que es el caso de los tableros de ajedrez normales, se tendría un total de 4.426.165.368 posibles soluciones a explorar.

Cuando intentamos resolver un problema de optimización mediante la red de Hopfield hay que intentar fijar el objetivo del problema mediante una función objetivo o función coste que hay que minimizar. A continuación esta función se compara con la función energía de la red de Hopfield determinándose los valores de los pesos (w_{ij})y los umbrales en términos de la función coste planteada para resolver el problema de optimización planteado. Veamos algunas aplicaciones típicas de esta red:

5.1.6.1.Problema del multi-flop.

Éste es un problema muy sencillo que da idea de la forma de trabajar con problemas de optimización cuando se usa la red de Hopfield. Es la generalización de un flip-flop, buscamos que el estado final de la red tenga sólo una neurona activada (sólo un 1 en el vector de salida y el resto 0). La función energía a definir (si estamos usando la función escalón con salidas 0 o 1) sería:

$$E = (\sum_{k=1}^{N} x_k - 1)^2 \qquad \text{Ec. 5.22}$$

Esta expresión será mínima cuando sólo una de las neuronas esté activa. Si desarrollamos esta expresión llegamos a:

$$E = \sum_{k=1}^{N} x_k^2 + \sum_{i \neq j}^{N} x_i \cdot x_j - 2 \cdot \sum_{k=1}^{N} x_k + 1 \qquad \text{Ec. 5.23}$$

Como las salidas son 0 ó 1 se cumple que $x=x^2$, por lo que la última expresión queda:

$$E = \sum_{i \neq j}^{N} x_i \cdot x_j - \sum_{k=1}^{N} x_k + 1 \qquad \text{Ec. 5.24}$$

Esta expresión es equivalente a:

$$E = -\frac{1}{2} \cdot \sum_{i \neq j}^{N} (-2) \cdot x_i \cdot x_j + \sum_{i=1}^{N} (-1) \cdot x_i + 1 \qquad \text{Ec. 5.25}$$

Como el uno es una constante y no interviene en la optimización de la anterior cantidad, podemos comparar esta expresión con la de la función energía para llegar a:

$$\begin{aligned} w_{ij} &= -2 \\ \theta_i &= -1 \end{aligned} \qquad \text{Ec. 5.26}$$

Así pues, con estos pesos al final sólo quedaría una neurona activada.

5.1.6.2.Problema de las 8 torres.

Este problema es un poco más complicado que el anterior. Se trata de colocar 8 torres en un tablero de ajedrez sin que se amenacen : hay que situar cada torre en una fila y columna diferente a las demás. Este problema puede ser visto como una generalización del problema del multi-flop. Denotamos por x_{ij} el estado de la unidad correspondiente al cuadrado ij (primer índice fila, segundo columna) del tablero. Se debe cumplir que en cada fila sólo tiene que estar activada una neurona, es decir, que en cada fila haya una sola torre. Si suponemos que la salida de las neuronas está entre 0 y 1 se debe cumplir que la suma de las x en una fila debe ser 1, es decir, debemos minimizar:

$$E_1 = \sum_{j=1}^{n} (\sum_{i=1}^{n} x_{ij} - 1)^2 \qquad \text{Ec. 5.27}$$

Como ocurre lo mismo para cada columna deberemos minimizar:

$$E_2 = \sum_{i=1}^{n} (\sum_{j=1}^{n} x_{ij} - 1)^2 \qquad \text{Ec. 5.28}$$

Lógicamente la función a minimizar será la suma de las dos energías anteriores realizando la correspondencia con una red de Hopfield bidimensional.

5.1.6.3.Problema del viajante.

Este problema consiste en recorrer N ciudades por el camino más corto, pasando sólo una vez por cada una de ellas, acabando en el punto de partida del camino. La complejidad radica en el número de posibles caminos que se pueden plantear (N!/2N). Este problema se puede resolver planteando una red de Hopfield con N^2 neuronas. Cada fila se asocia con la ciudad y la columna con las paradas. Así, si en la situación final tenemos en la posición 2-3 un 1 significa que la ciudad 2 será visitada en tercer lugar.

La red usada será de tipo continuo con valores en el margen [0 1]. La función objetivo de este problema habría que plantearla de acuerdo a las siguientes condiciones:

- En cada fila y columna sólo puede estar activada una neurona. Si no se cumple esta condición tendríamos que una ciudad se recorre dos veces.

- En cada fila y columna debe estar activada una neurona. De esta forma nos aseguramos que todas las ciudades son visitadas.

De acuerdo con esto, nuestra función objetivo será:

$$E = \frac{A}{2} \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{\substack{l=1 \\ l \neq j}}^{N} s_{ij} \cdot s_{il} + \frac{B}{2} \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{\substack{k=1 \\ k \neq i}}^{N} s_{ij} \cdot s_{kj} + \frac{C}{2} \cdot \left(\sum_{i=1}^{N} \sum_{j=1}^{N} s_{ij} - N \right)^2 + \frac{D}{2} \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{\substack{k=1 \\ k \neq j}}^{N} d_{ij} \cdot (s_{ij} \cdot s_{kj+1} + s_{ij} \cdot s_{kj-1})$$

Ec. 5.29

donde A, B, C y D son constantes que dan la importancia de cada término.

Veamos qué significado tienen los términos de la función objetivo. El primer término es el encargado de evitar que dos neuronas de la misma fila se activen (tengan las dos al mismo tiempo un valor de uno). El siguiente término se encarga de que no se activen dos neuronas de la misma columna. El tercero obliga a que haya N ciudades en la ruta y el último toma en consideración que la distancia total sea mínima. Una vez que tenemos la función objetivo la compararemos con la función energía de una red de Hopfield bicapa (generalización de la monocapa).

Dicha función es:

$$E = -\frac{1}{2} \cdot \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N} w_{ij,kl} \cdot s_{ij} \cdot s_{kl} + \sum_{i=1}^{N}\sum_{j=1}^{N} \theta_{ij} \cdot s_{ij} \qquad \text{Ec. 5.30}$$

Si comparamos ambas expresiones, para que sean equivalentes, los valores de los pesos de las conexiones entre dos neuronas, situadas una en la fila i y columna j, y otra en fila k y columna l, han de ser:

$$w_{ij,kl} = -A \cdot \delta_{ik} \cdot (1-\delta_{jl}) - B \cdot \delta_{jl} \cdot (1-\delta_{ik}) - C - D \cdot d_{ik} \cdot (\delta_{j,l+1} + \delta_{j,l-1}) \qquad \text{Ec. 5.31}$$

Donde δ es la delta de Kronecker cumpliéndose:

$$\delta_{sk} = \begin{cases} 1 & si \;\; s = k \\ 0 & si \;\; s \neq k \end{cases}$$

La expresión de los umbrales quedaría como:

$$\theta_{ij} = -C \cdot N$$

Resumiendo, si se implementa una red de Hopfield continua de N×N neuronas con los pesos y umbrales anteriormente mencionados y se deja un cierto número de iteraciones, se llegará a una situación de estabilidad en la que sólo habrá N neuronas activas, una por cada fila y columna, que nos dará el camino óptimo a seguir. El principal problema radica en la elección de las constantes que aparecen en la función a minimizar.

5.1.6.4.El problema de las 8 reinas.

Este problema es muy parecido al de las 8 torres pero más complicado: debemos colocar 8 reinas. Las energías que definíamos para el problema de las torres se mantienen, pero hay que añadir el hecho de que la reina puede amenazar en diagonal. Para ello se define una función energía dada por:

$$E_3 = \sum_{j=1}^{7}\left((\sum_{i=1}^{8-j} x_{i,i+j-1}) - 0.5 \right)^2 + \sum_{j=2}^{7}\left((\sum_{i=j}^{8} x_{i,i-j+1}) - 0.5 \right)^2 \qquad \text{Ec. 5.32}$$

La última expresión se ha calculado para diagonales “positivas”; si nos fijamos en las diagonales “negativas” llegamos a:

$$E_3 = \sum_{j=2}^{8}\left((\sum_{i=1}^{j} x_{i,j-i+1}) - 0.5 \right)^2 + \sum_{j=2}^{7}\left((\sum_{i=j}^{8} x_{i,8+j-i}) - 0.5 \right)^2 \qquad \text{Ec. 5.33}$$

El factor 0.5 aparece como consecuencia de que tan buena es una solución en la que aparezca una reina en una diagonal como que no aparezca. Para reflejar este

hecho se toma un factor cercano a los dos puntos extremos de la función de activación (0, 1).

El paso final es determinar la energía total como la suma de los diferentes términos definidos anteriormente, operar y hacer una equivalencia con una red de Hopfield bidimensional de la misma manera que en el problema del viajante.

OCON.

5.2. Redes basadas en la decisión y estructuras OCON.

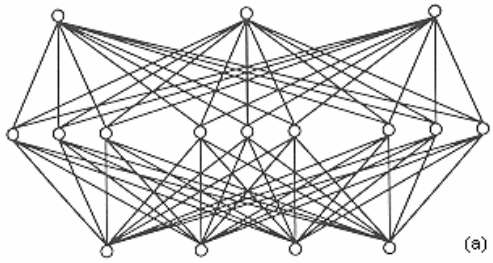
El perceptrón sólo puede ser aplicado con éxito cuando sea factible realizar una separación de las clases de los patrones mediante límites de decisión lineales. Por contra, el perceptrón multicapa nos ofrece un dominio mucho mayor a la hora de decidir la clasificación de los patrones.

La filosofía de las Redes Neuronales Basadas en la Decisión ("Decision-Based Neural Networks", DBNN) se enfoca en las redes supervisadas de aprendizaje reforzado, aportando dos características fundamentales en el tratamiento eficiente de los problemas: una estructura óptima y un algoritmo específico [Kung-95].

5.2.1. Factores estructurales de las RNAs: OCON y ACON.

Ante problemas de clasificación es conveniente hacer una división del trabajo, por lo que se pueden implementar modelos caracterizados por tener una dedicación especial a cada categoría a clasificar.

Para entender el tipo de estructuras neuronales que estamos analizando necesitamos diferenciar las estructuras OCON frente a las ACON. Las redes OCON ("One Class in One Net", Figura 5.5) lo consiguen estando compuestas por varias subredes o redes locales trabajando en paralelo y estando cada una de ellas encargada de modelizar una clase o categoría lo que lleva a una simplicidad y rapidez de convergencia. Se puede optar también por una estructura ACON ("All Classes in One Net", Figura 5.5a) en las que todas las clases se modelizan en una misma red. Tienen una mayor complejidad estructural lo cual implica un mayor coste computacional.



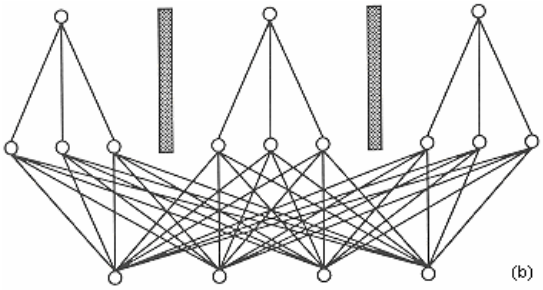


Figura 5.5. Estructuras de modelización. (a) Estructura ACON. (b) Estructura OCON.

En definitiva, la elección de una u otra arquitectura depende si la tarea a realizar queremos llevarla a cabo de manera modular (OCON) o de manera conjunta (ACON). La elección de una estructura u otra dependerá de la aplicación: número de clases, posibilidad de simplificación o fragmentación del problema, tamaño de las redes, etc.

5.2.2. Formulación de las DBNNs.

La idea para explotar esa especificidad de la estructura OCON da como resultado los siguientes algoritmos de aprendizaje: el basado en la decisión (DBNNs) y el de decisión suave, difusa o "fuzzy" ("Fuzzy Decision Neural Networks", FDNNs). Este segundo método en realidad es una modificación del algoritmo de aprendizaje del primero ya que considera la clasificación en términos de la distancia máxima posible entre dos regiones de clasificación vecinas. Hay que hacer énfasis en que el poder de este tipo de técnicas se halla en la acción conjunta de la estructura y la regla de aprendizaje.

El camino a seguir tiene como objetivo una buena clasificación y se planteará con los siguientes argumentos:

1. ¿Cuándo actualizamos los pesos?
Sólo cuando no obtengamos una correcta clasificación.

2. ¿Qué subredes actualizamos?
Aplicaremos un refuerzo del aprendizaje a la subred correspondiente a la clase ganadora y un anti-refuerzo a la perdedora. De esta manera conseguimos incentivar una competición entre las subredes, otorgando un "premio" en forma de gradiente negativo a la subred correspondiente a la clase ganadora y un "castigo" en forma de gradiente positivo a la subred correspondiente a la clase perdedora.

3. ¿Cómo actualizamos?
Trazaremos el límite de la clasificación ajustando los pesos únicamente cuando no obtengamos una correcta clasificación en la dirección del gradiente a la subred ganadora (aprendizaje reforzado) o en la dirección contraria a la perdedora (aprendizaje anti-reforzado).

$$\Delta w = \pm \eta \nabla \phi (x, w) \qquad \text{Ec. 5.34}$$

El signo + corresponderá al aprendizaje reforzado y el signo - al aprendizaje anti-reforzado. Hemos representado con ϕ la función discriminante, es decir, la función matemática que caracteriza a una red neuronal. Como se puede

observar, esta función depende de los pesos ω de la red y de las entradas x a la red.

Definimos el gradiente de esa función así:

$$\nabla\phi(x,w) = \frac{\partial\phi(x,w)}{\partial w} = \left[\frac{\partial\phi}{\partial w_1}, \frac{\partial\phi}{\partial w_2}, \ldots, \frac{\partial\phi}{\partial w_N}\right]^T \qquad \text{Ec. 5.35}$$

Analicemos en primer lugar el algoritmo de aprendizaje.

5.2.2.1. Algoritmo DBNN.

Supongamos que $S = \{x^{(1)}, \ldots, x^{(M)}\}$ es un conjunto de patrones dados, cada uno de ellos correspondientes a una clase de las L clases $\{\Omega_i, i = 1, \ldots, L\}$. Cada clase está modelada por una subred cuya funciones discriminantes son $\phi(x,w_i)$, donde i = 1,..., L. Supongamos que el patrón m-ésimo $x^{(m)}$ pertenece a la clase Ω_i, y que

$$\phi(x^{(m)}, w_j^{(m)}) > \phi(x^{(m)}, w_l^{(m)}), \forall l \neq j \qquad \text{Ec. 5.36}$$

Esto quiere decir que la clase ganadora para el patrón $x^{(m)}$ es la clase j-ésima, correspondiente a la subred j-ésima.

- Cuando, el patrón $x^{(m)}$ está bien clasificado y por tanto no necesita ninguna actualización de pesos.
- Cuando, el patrón $x^{(m)}$ no está bien clasificado y por tanto necesita actualización de pesos, que se realiza de la siguiente manera:

Aprendizaje Reforzado: $w_i^{(m+1)} = w_i^{(m)} + \eta\nabla\phi(x,w_i)$ Ec. 5.37

Aprendizaje Anti-Reforzado: $w_j^{(m+1)} = w_j^{(m)} - \eta\nabla\phi(x,w_j)$ Ec. 5.38

Si analizamos el algoritmo de aprendizaje lo que hacemos es "acercar" el vector de pesos de la clase que debería ganar al patrón presentado y alejamos el vector de pesos de la clase ganadora pero incorrecta. Este algoritmo nos permite una rápida convergencia y un excelente comportamiento en muchas aplicaciones prácticas, pudiendo ser implementado en distintos tipos de estructuras.

5.2.2.2. Estructuras DBNN.

Las estructuras DBNN o jerárquicas se caracterizan por dos factores: su estructura y su función base. Las funciones base típicas son: las LBF ("Linear Basis Function"), RBF ("Radial Basis Function") y la EBF ("Elliptic Basis Function"). Dependiendo de la aplicación utilizaremos una función base, que en nuestro caso será la sigmoide o LBF al igual que en el perceptrón multicapa. Cabe decir que las estructuras DBNN combinan la Regla de Aprendizaje del Perceptrón con la Estructura Jerárquica, llamándose por ello "Hierarchical Perceptron" (HiPer). Tenemos dos estructuras básicas: la de nodo oculto y la de subcluster (Figura 5.6).

1. DBNN de Nodo Oculto (Hidden-Node).

En la Figura 5.6a vemos como cada subred está formada por varios subnodos ocultos, cada uno representado por $\psi_l(x, w_{k_l})$. La función discriminante de la subred que representa la clase l es una combinación lineal de los valores de los subnodos:

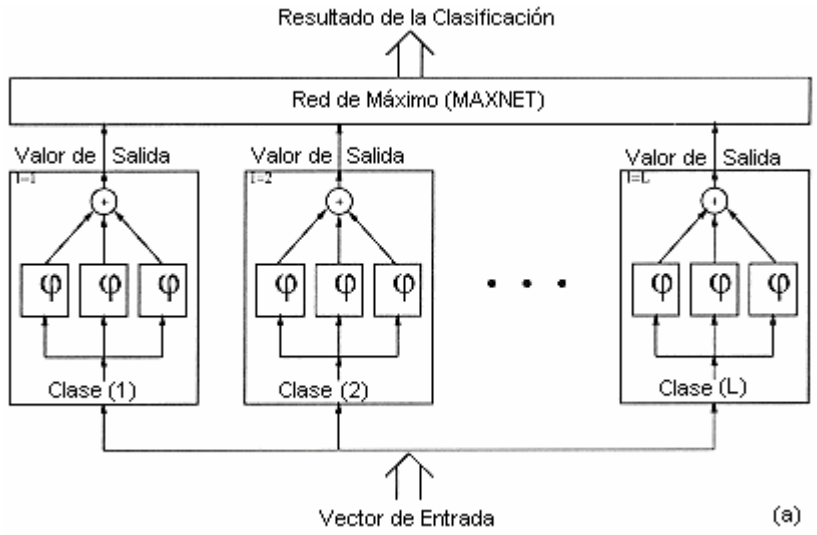
$$\phi(x, w_l) = \sum_{k_l=1}^{K_l} c_{k_l} \psi_l(x, w_{k_l}) \qquad \text{Ec. 5.39}$$

donde el conjunto c_{k_l} hace referencia a los coeficientes de la capa superior y w_l al vector de pesos. Usamos el mismo Algoritmo de Aprendizaje, Ec. (1), para cada subred.

2. DBNN de Subcluster.

En esta estructura usamos una aproximación llamada "el ganador se lo lleva todo", que podemos observar en la (Figura 5.6b). Es por tanto necesario introducir los conceptos de ganador local y de ganador global. El ganador local de la l-ésima subred será indexada con s_l y es:

$$s_l = Arg\left[\max_{s_l} \left\{ \psi_l(x, w_{s_l}) \right\} \right] \qquad \text{Ec. 5.40}$$



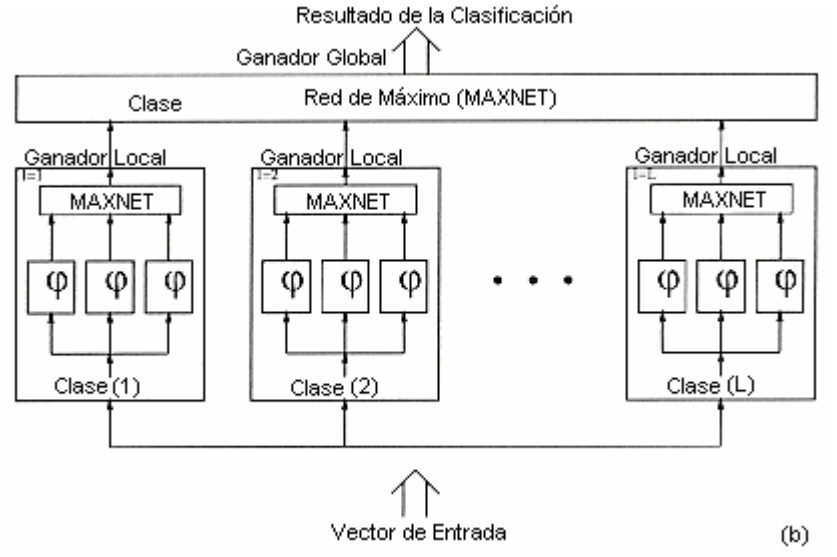


Figura 5.6. Estructuras DBNN. (a) Hidden-Node. (b) Subcluster.

El ganador global es el ganador sobre todas las subredes: el ganador de los ganadores locales. Marcaremos con subred-j a la ganadora global si su ganador local gana a todos los ganadores locales (uno por subred), es decir:

$$\psi_j(x, w_{s_j}) > \psi_l(x, w_{s_l}), \forall l \neq j \qquad \text{Ec. 5.41}$$

Un patrón será clasificado en la clase j-ésima si la subred-j es la ganadora global. Esta estructura sigue el Algoritmo DBNN sustituyendo la función discriminante de las subredes por aquellas de las ganadoras locales:

$$\phi(x, w_i) \Leftrightarrow \psi_i(x, w_{s_i}) \qquad \text{Ec. 5.42}$$

y

$$\phi(x, w_j) \Leftrightarrow \psi_j(x, w_{s_j}) \qquad \text{Ec. 5.43}$$

5.2.2.3. Algoritmo Subcluster DBNN.

Supongamos que empleamos un número de subclusters para representar una clase (recordemos que cada subred se dedica a modelizar una clase o categoría y está formada por subnodos o subclusters), y que además, representamos con s_i, s_j, los ganadores locales. Si el patrón $x^{(m)}$ pertenece a la clase i pero es la subred-j la elegida como ganadora global ($j \neq i$), es decir, $x^{(m)}$ no está bien clasificado, hay que hacer la siguiente actualización de los pesos:

Aprendizaje Reforzado: $$w_{s_i}^{(m+1)} = w_{s_i}^{(m)} + \eta \nabla \psi(x, w_{s_i}) \qquad \text{Ec. 5.44}$$

Aprendizaje Anti-Reforzado: $$w_{s_j}^{(m+1)} = w_{s_j}^{(m)} - \eta \nabla \psi(x, w_{s_j}) \qquad \text{Ec. 5.45}$$

En otras palabras, el aprendizaje no reforzado (-) es aplicado al subcluster ganador local de la subred ganadora global, y el reforzado (+) es aplicado al ganador local dentro de la clase correcta, es decir, la que debería ser la ganadora.

5.2.3. Formulación de las FDNNs.

Cuando las clases están claramente separadas existen varias soluciones en el ajuste de los pesos, pudiendo ser explotado ese hecho de forma que se intente conseguir el margen de separación ε más amplio entre dos regiones vecinas. Esto nos conduce a realizar algunas modificaciones en el Algoritmo de Aprendizaje. Así, la Ec. 5.35 referente al límite de decisión quedará como sigue:

$$\phi(x^{(m)}, w_j^{(m)}) > \phi(x^{(m)}, w_l^{(m)}) + \varepsilon \qquad \text{Ec. 5.46}$$

Sin embargo, cuando las clases no son tan fácilmente separables, necesitaremos una técnica de aproximación totalmente diferente al trabajar en la frontera de la decisión. Necesitamos, por tanto, para alcanzar una buena generalización, un factor de tolerancia a la desclasificación [Kung-93a], [Kung-93b], [Taur-93]. Una manera de dar esa tolerancia para un caso no separable consiste en ignorar los patrones que son indecisos continuamente; aquellos en los que la red varía su clasificación para cada época.Otra forma -más elegante- es la que nos ocupa, que está basada en una decisión difusa ("fuzzy"). Se trata de imponer una función penalizadora para las malas decisiones y otra función penalizadora para las marginalmente buenas. Esto nos proporciona una decisión suave por contraposición a la decisión fuerte del Aprendizaje DBNN.

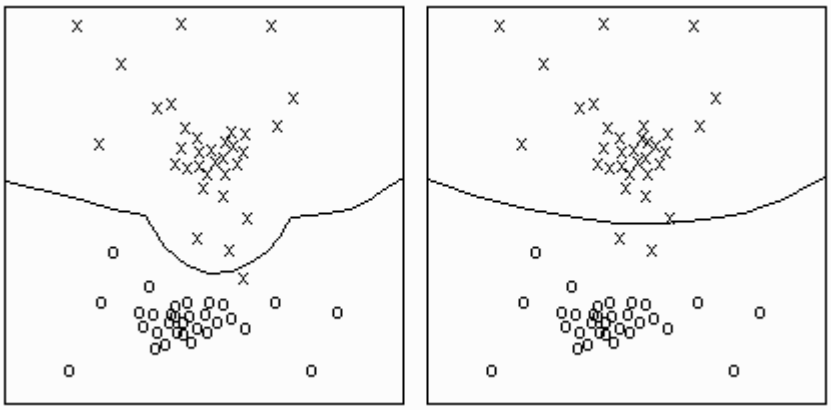


Figura 5.7. Ejemplo de separación de clases. (a) DBNNs. (b) FDNNs.

Veamos el Algoritmo de Aprendizaje.

5.2.3.1. Algoritmo FDNN [1], [2].

Supongamos que $S = \{x^{(1)},..., x^{(M)}\}$ es un conjunto de patrones dados, cada uno de ellos correspondientes a una de las L clases $\{\Omega_i, i = 1,..., L\}$. Cada clase está modelada por una subred con función discriminante $\phi(x,w_i)$, donde i = 1,..., L. Definimos j como la clase rival a la ganadora, esto es, la ganadora entre las perdedoras:

$$j = Arg\left[\max_{j \neq i} \left\{ \phi(x^{(m)}, w_j) \right\} \right] \qquad \text{Ec. 5.47}$$

donde hemos excluido el índice i de la clase correcta. Para un patrón de entrenamiento, una medida de la desclasificación podría ser:

$$d = d^{(m)}(x^{(m)}, w) = -\phi_i(x^{(m)}, w_i^{(m)}) + \phi_j(x^{(m)}, w_j^{(m)}) \qquad \text{Ec. 5.48}$$

donde w representa a todos los pesos involucrados. Una forma más general (para aplicaciones con más de dos clases) de esta medida podría ser:

$$d = -\phi_i(x^{(m)}, w_i^{(m)}) + \left\{ \frac{1}{L-1} \sum_{j \neq i} \phi_j(x^{(m)}, w_j^{(m)})^{\gamma} \right\}^{1/\gamma} \qquad \text{Ec. 5.49}$$

donde $\gamma > 0$ y ϕ se supone positivo. Si $\gamma \to \infty$, la Ec. 5.4 tiende a transformarse en la Ec. 5.48. Como vemos en la Figura 5.8a, tenemos dos situaciones que analizar: d positiva (el patrón asociado no estará bien clasificado), y d negativa (el patrón sí está bien clasificado en la clase i).

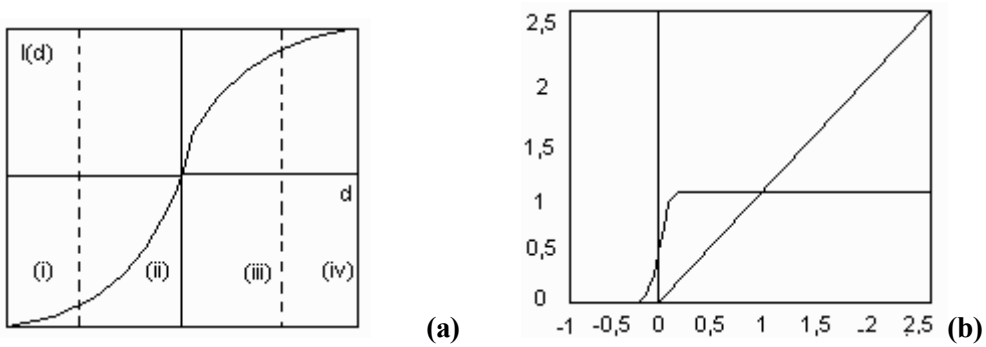


Figura 5.8. (a) Regiones de Interés en la clasificación. En DBNNs sólo importa clasificación correcta o incorrecta. En FDNNs distinguimos cuatro zonas de Clasificación: (i) Correcta pero con vigilancia, (ii) Correcta pero con mejora, (iii) Incorrecta con posibilidades de mejora, y (iv) Incorrecta irremisiblemente. (b) Diferencia entre las funciones de penalización: en DBNNs es lineal mientras que en las FDNNs se muestran funciones suaves.

Podemos profundizar más en la actuación dependiendo de la magnitud de d:

1. Si d es positivo el patrón asociado estará mal clasificado respecto a la clase retante j. De hecho, cuanto mayor sea d mayor error tendremos.
2. Cuando d sea negativo el patrón estará bien clasificado en la clase j.
3. Cuando d tome valores negativos pequeños la clase correcta ganará a la clase retante por un estrecho margen. El patrón estará bien clasificado pero con la necesidad de ser "vigilado".
4. Si d toma valores positivos pequeños no tenemos una buena clasificación.

Algunas funciones de penalización que conducen a un error mínimo en la clasificación pueden describirse así:

$$l(d) = \frac{1}{1 + e^{-d/\xi}} \qquad \text{Ec. 5.50}$$

$$l(d) = \begin{cases} 0 & d \leq -\xi/2 \\ (d + \xi/2)/\xi & -\xi/2 < d < \xi/2 \\ 1 & \xi/2 \geq d \end{cases} \qquad \text{Ec. 5.51}$$

Notar que cuando ξ se aproxima a cero, las funciones de penalización aproximan su forma a la de la función escalón dada por:

$$l(d) = \begin{cases} 0 & d < 0 \\ 1 & d \geq 0 \end{cases} \qquad \text{Ec. 5.52}$$

Tendremos pues:

- Aprendizaje Reforzado:

$$w_j^{(m+1)} = w_j^{(m)} + \eta l'(d) \nabla \phi(x, w_j) \qquad \text{Ec. 5.53}$$

- Aprendizaje Anti-Reforzado:

$$w_j^{(m+1)} = w_j^{(m)} - \eta l'(d) \nabla \phi(x, w_j) \qquad \text{Ec. 5.54}$$

donde l'(d) es la derivada de la función penalizadora evaluada en d.

De las Ecs. 5.50, 5.53 y 5.54, se puede obtener fácilmente la expresión final de la corrección de los pesos.

COMITÉS DE EXPERTOS.

5.3. Comités de expertos.

5.3.1. Introducción.

Existen diversas aplicaciones en las que las técnicas de computación neuronal ofrecen alternativas muy atractivas sobre otras técnicas más clásicas, particularmente cuando los datos contienen ruido o cuando no existe un conocimiento explícito a priori sobre dichos datos.

En la mayoría de aplicaciones prácticas, el criterio más importante para evaluar el funcionamiento de una RNA entrenada es su capacidad para generalizar conocimientos adquiridos. El problema es que, aunque las RNAs correctamente entrenadas ofrecen buenos resultados, inevitablemente, cometen errores al generalizar sobre datos noveles. Además, cuando entrenamos RNAs, el conjunto de entrenamiento es finito y en ocasiones está severamente limitado en tamaño, lo que agrava los errores en la generalización.

Por estas razones debemos desarrollar y explorar técnicas que nos permitan mejorar las capacidades de generalización para RNAs. Cuando empleamos Redes Neuronales multicapa con propagación hacia delante (MLFNN), existen diversos parámetros ajustables (e.g inicializaciones de los pesos sinápticos, número de neuronas ocultas, funciones de activación, conjuntos de entrenamiento) que pueden afectar a la capacidad de una RNA. Una aproximación muy extendida a la hora de abordar aplicaciones con redes neuronales es la de entrenar varias redes (variando topologías, inicializaciones de pesos sinápticos etc) y, posteriormente, escoger aquella que ofrece la mayor capacidad de generalización. Podemos inferir rápidamente que, bajo esta aproximación, estamos perdiendo información acerca del conocimiento sobre el problema que las redes con peores capacidades o habilidades en generalización han adquirido, información que, a priori, no deberíamos despreciar. El usar un Comité de Expertos mejora la capacidad de generalización, resultado que se obtiene combinando las distintas redes.

Una vez introducida la filosofía básica de los comités de RNAs o más genéricamente Comités de Expertos, trataremos de dar una visión preliminar sobre dos cuestiones: ¿Cuándo y por qué funcionan los comités?

5.3.2. Tipos de Comités. ¿Cuándo y por qué funcionan los Comités de RNAs?

Existe cantidad de evidencias y pruebas que el uso de comités de RNAs consigue una mejora en la capacidad de generalización [Perrone-93], [Hashem-97]. La idea de un comité es la siguiente: Las RNAs que conforman un comité son entrenadas individualmente y, posteriormente, en la etapa de validación sus predicciones son combinadas [Krogh-95]. Las máquinas en comité son aproximadores universales y pueden clasificarse en dos grandes categorías [Haykin-98]:

- Estructuras estáticas: Las respuestas de los distintos expertos son combinadas por medio de un mecanismo que no involucra a la señal de entrada. En esta categoría pueden destacarse los siguientes métodos:
  - Promediado: Las salidas de diferentes redes son linealmente combinadas para producir la salida total del Comité.
  - Boosting: Existe un algoritmo de aprendizaje "débil", es decir, un algoritmo que no involucra directamente modificaciones de pesos sinápticos, que permite mejorar la capacidad de generalización de un conjunto de redes previamente entrenadas.
- Estructuras Dinámicas: La señal de entrada actúa sobre el mecanismo que integra la salida de los distintos expertos en una salida única de todo el comité. Destacan:
  - Mezcla de Expertos: Las respuestas de los distintos expertos son combinadas de forma lineal a través de una red de combinación.
  - Mezcla de Expertos Jerarquizada: Existen diversos módulos controlados por redes de combinación jerarquizadas de acuerdo a un determinado orden.

La idea que subyace al hecho de mejorar la capacidad de generalización empleando RNAs en comités, que los errores de generalización generados por las distintas redes (expertos) que conforman el comité no están correlacionados. Así, si creamos un comité en base a diferentes redes, debemos esperar que los errores que se cometen en la generalización estén poco o nada correlacionados. De esta forma cuando una red falle ante una determinada entrada de validación el resto de redes del comité ofrecerá una salida correcta y, por promedio, la salida del comité será la correcta.

5.3.3. Método básico de formación de Comités.

5.3.3.1. BEM.

En esta sección presentamos el método básico de formación de comités (Basic Ensemble Method, BEM), el cual combina una población de RNAs previamente entrenadas para estimar una función f(x)

Supongamos que tenemos dos conjuntos de datos finitos cuyos elementos son todos independientes y son variables con distribución uniforme: un conjunto de entrenamiento A y un conjunto de validación CV.

$$A = \{(x_m, y_m)\} \quad \text{Ec. 5.55}$$

$$CV = \{(x_l, y_l)\} \quad \text{Ec. 5.56}$$

Además supondremos que hemos usado A para generar un conjunto de funciones $f_i(x)$ (i = 1.....N) de manera que cada uno de los elementos de ese conjunto aproxima la función f(x). Nuestro objetivo es encontrar la mejor aproximación a f(x) usando el conjunto de funciones obtenido (F).

Una aproximación ya clásica es utilizar o escoger la red del conjunto F que ofrezca un error cuadrático medio de generalización mínimo respecto al conjunto CV.

$$MSE[f_i] = E_{CV}[(y_m - f_i(x_m))^2] \quad \text{Ec. 5.57}$$

Por tanto, el estimador asociado a este tipo de elección y que llamaremos f_N, puede escribirse como:

$$f_N = \arg\min_i \{MSE[f_i]\} \quad \text{Ec. 5.58}$$

Este tipo de aproximación resulta insatisfactoria por dos razones:

- Al seleccionar una única red de la población de redes representada por F, estamos descartando información de utilidad almacenada en las redes descartadas;

- Ya que el conjunto CV es aleatorio, existe la probabilidad de descartar una red que ofrezca un comportamiento mejor que la red seleccionada como f_N sobre un conjunto de validación distinto tomado de la misma distribución de datos.

Definimos las funciones:

$$m_i(x) \equiv f(x) - f_i(x) \quad \text{Ec. 5.59}$$

y de esa manera podemos definir el error cuadrático medio asociado a una red de la población F como:

$$MSE[f_i] = E[m_i^2] \quad \text{Ec. 5.60}$$

Así pues el error cuadrático medio promedio (i=1.....N) será:

$$\overline{MSE} = \frac{1}{N}\sum_{i=1}^{i=N} E[m_i^2] \quad \text{Ec. 5.61}$$

Definimos f_{BEM} (x) como:

$$f_{BEM}(x) \equiv \frac{1}{N}\sum_{i=1}^{i=N} f_i(x\ ) = f(x) - \frac{1}{N}\sum_{i=1}^{i=N} m_i(x) \quad \text{Ec. 5.62}$$

Asumiendo que las funciones $m_i(x)$ son mutuamente independientes con media cero podemos calcular el error cuadrático medio asociado al estimador f_{BEM} (x) como:

$$MSE(f_{BEM}) = E\left[\left(\frac{1}{N}\sum_{i=1}^{i=N} m_i\right)^2\right] \quad \text{Ec. 5.63}$$

$$= \frac{1}{N^2} E\left[\sum_{i=1}^{i=N} m_i^2\right] + \frac{1}{N^2} E\left[\sum_{i\neq j} m_i m_j\right] \quad \text{Ec. 5.64}$$

Aplicando que las variables m_i no están correlacionadas obtenemos:

$$= \frac{1}{N^2} E\left[\sum_{i=1}^{i=N} m_i^2\right] + \frac{1}{N^2}\sum_{i\neq j} E[m_i]E[m_j] = \frac{1}{N^2} E\left[\sum_{i=1}^{i=N} m_i^2\right] \quad \text{Ec. 5.65}$$

es decir:

$$MSE(f_{BEM}) = \frac{1}{N}\overline{MSE} \quad \text{Ec. 5.66}$$

Este resultado nos dice que, promediando la salida de nuestras redes podemos reducir el error cuadrático medio en el conjunto de validación en un factor N respecto a la media de error cuadrático ofrecida por la población de redes si se cumplen las premisas de partida.

Consideremos a los elementos individuales de la población F. Si pensamos en las funciones $m_i(x)$ como ruido aleatorio añadido a la función f(x) con esas funciones de media cero y no correlacionadas, entonces, el promediado simple de las salidas de las RNA que conforman el comité es realiza el papel de un filtro suavizador o eliminador de ruido [Proakis-97]:

$$f_{BEM}(x) \equiv \frac{1}{N}\sum_{i=1}^{i=N} f_i(x) = f(x) - \frac{1}{N}\sum_{i=1}^{i=N} m_i(x) \quad \text{Ec. 5.67}$$

Un beneficio adicional del método BEM la diferente naturaleza que pueden tener los integrantes del comité.

5.3.3.2. Mezcla de expertos.

Consideremos la estructura que aparece en la siguiente figura conocida como mezcla de expertos (ME)

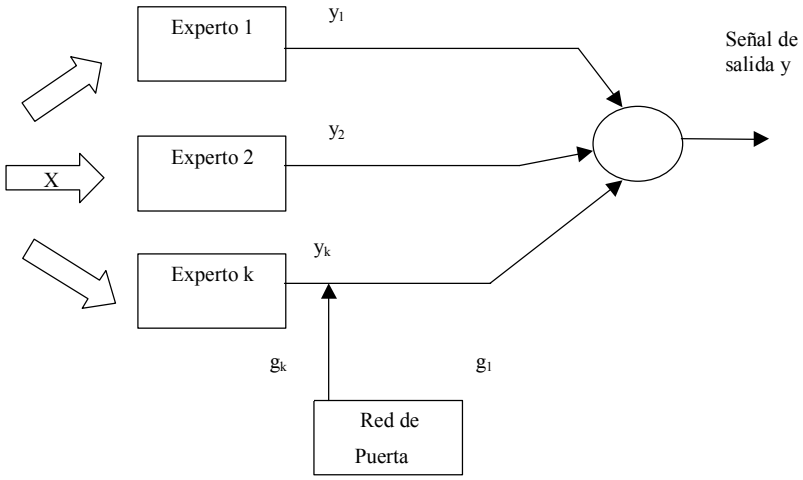


Figura 5.9. Diagrama de bloques del modelo de mezcla de expertos. La salida escalar está controlada mediante la red de puerta.

Este modelo consiste en K módulos supervisados llamados redes expertas y una unidad integradora denominada red de puerta. Asumimos que en éste modelo los distintos expertos están "especializados" en regiones distintas del espacio de entrada y de ahí la necesidad de contar con una unidad como la red de puerta.

En éste modelo cada experto consiste en una única neurona o filtro lineal cuya salida puede expresarse como:

$$y_k = w_k^T x \qquad \text{Ec. 5.68}$$

donde k=1, 2, . . ., K.

La red de puerta consiste en una red monocapa de K neuronas, cada una de ellas asignada a un experto en concreto. Las salidas de las neuronas de la red de puerta son no lineales y están definidas por la función:

$$g_k = \frac{\exp(u_k)}{\sum_{j=1}^{K} \exp(u_j)} \qquad \text{Ec. 5.69}$$

donde u_k es el producto escalar del vector x y del vector de pesos sinápticos a_k.

La expresión anterior puede ser considerada como una generalización multientrada de la función logística. Es una versión diferenciable de la operación " el ganador se lo lleva todo ", razón por la cual es conocida en la literatura como softmax. Además la suma de las g_k es siempre la unidad lo que confiere a estas funciones un significado de probabilidad: probabilidad de que la salida del experto k sea la correcta ante la entrada actual.

Los perceptrones lineales simples de una estructura en mezcla de expertos aprenden rápidamente, son muy simples y además son muy efectivos en muchas aplicaciones de clasificación. En la aproximación de mezclas de expertos los distintos perceptrones son entrenados de manera que converjan a distintas partes del espacio de entrada y, por lo tanto, aprenden fronteras de separación locales y

distintas en cada caso. La red de puerta es la unidad integradora que decide tras el aprendizaje correspondiente qué perceptrón es el adecuado.

5.3.4. Creación efectiva de comités.

Hasta ahora se ha hecho hincapié en la importancia que tiene el tener errores no correlacionados en sus predicciones. Sin embargo, la decisión sobre los miembros que conforman un comité es de tanta importancia para el correcto funcionamiento de dicho comité como la forma en la que las salidas de los distintos miembros del comité son combinadas para dar la salida total. Hasta el momento hemos descrito técnicas para diseño de la salida conjunta del comité.

En este apartado nos centraremos en la discusión de métodos para crear a los integrantes de un comité. En este sentido existen dos aproximaciones. La primera y más simple consiste en generar una serie de redes neuronales, ensamblarlas en un comité y testear la efectividad del clasificador así obtenido. Esta es la aproximación que adoptan una gran cantidad de investigadores. Una segunda aproximación que según algunos autores es una aproximación más efectiva, es la creación de un conjunto de redes a través de un proceso e incluye la idea de selección: en lugar de incluir en el comité todas las redes que han sido generadas a lo largo de ese proceso, se establece un criterio de selección mediante el cual se mejora la capacidad y efectividad del comité así formado. En esta dirección la investigación actual se encamina hacia métodos de creación de comités mediante algoritmos genéticos o programación evolutiva. Mediante estos algoritmos se obtienen poblaciones de redes con arquitecturas y conjuntos de entrenamiento distintos [Opitz-96].

Algunos de los parámetros que pueden ser ajustados para la creación de un conjunto de RNA que conforme un comité son los siguientes:

- Distinta inicialización del conjunto de pesos sinápticos.
- Variación de las topologías de las distintas redes.
- Variación del algoritmo de entrenamiento.
- Variación de los datos de entrenamiento.

Una vez mencionadas las técnicas para crear el conjunto de redes necesario para formar el comité, pasamos a comentar posibles criterios de selección. Perrone y Cooper sugieren una selección heurística en la que la población de RNA es ordenada en virtud del error cuadrático medio, el comité lo conforman aquellas redes con menor error cuadrático medio [Perrone-93]. El proceso puede ser más elaborado construyendo inicialmente un comité de pequeño tamaño al que se le van añadiendo redes siguiendo el orden de error cuadrático establecido, siempre y cuando el error cuadrático medio de comité decrezca por esa adición.

Trabajos más recientes utilizan un método de generación de miembros y selección mediante algoritmos genéticos hasta que cierto criterio se cumple. En resumen, hay tres argumentos importantes a la hora de formar un comité de expertos:

1. Una metodología que permita crear a los miembros del comité.

2. Un criterio de selección a la hora de incluir aquellos elementos de la población inicial de RNA que conformen un comité con efectividad óptima.
3. Diseño de la salida conjunta del comité.

5.4. Redes Recurrentes.

5.4.1. Introducción.

Las arquitecturas estáticas, como el perceptrón multicapa entrenado con el algoritmo de retropropagación, son estructuras sin la habilidad para capturar características temporales de los patrones. Esto no quiere decir que el problema resida en el algoritmo sino más bien en la estructura empleada. Aún a pesar de que podemos emplear un estructura estática, como el perceptrón multicapa, en la predicción de series temporales si lo alimentamos con entradas sucesivamente retardadas (Figura 5.10b), este preproceso ya constituye, en cierto modo, una variación de su estructura, convirtiendo al típico MLP en una estructura dinámica.

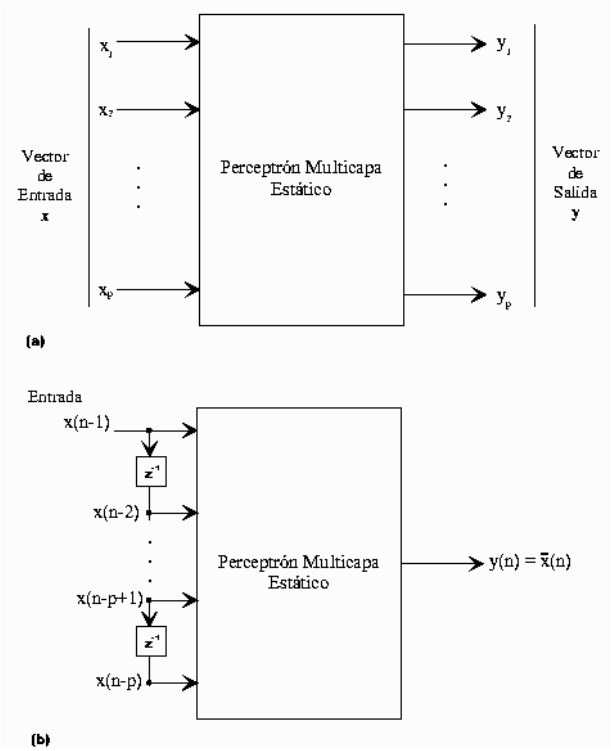


Figura 5.10. Perceptrón Multicapa usado como (a) clasificador de patrones y como (b) predictor no lineal.

Este tipo de relación estática es idóneo en aplicaciones de reconocimiento y clasificación de patrones donde la entrada y la salida representan patrones independientes del tiempo, o bien, en la predicción no lineal de series temporales estacionarias, entendiendo por estacionarias, aquéllas cuya media y varianza se mantienen constantes a lo largo del tiempo [Makridakis-98]. En estos casos podemos emplear un perceptrón como el representado en la Figura 5.10b donde los elementos z^{-1} representan unidades de retardo y el valor de entrada x se representa

en términos de las muestras pasadas {x(n-1), x(n-2), ...,x(n-p)} siendo p el orden de la predicción.

Se plantea la posibilidad de introducir la variable temporal en un perceptrón multicapa. La manera de hacerlo se basa en dotar a la red de memoria [Elman-90]. Existen dos formas de conseguir esto:

- Introduciendo retardos temporales en la estructura sináptica de la red y ajustar sus valores en la fase de entrenamiento o aprendizaje;
- Permitiendo realimentaciones entre las neuronas que forman la red.

A. Las Redes de Retardo Temporal (TDNN).

Una de las técnicas más populares la constituyen las redes neuronales "de retardo temporal" (Time-Delay Neural Network, TDNN), introducidas por Lang y Hinton [Lang-88] y Waibel et al. [Waibel-89]. Se trata de redes multicapa realimentadas cuyas neuronas ocultas y de salida son "replicadas" a lo largo del tiempo, esto es, las salidas de una capa se almacenan en varios instantes temporales tras lo cual alimentan a la siguiente capa. La topología de este tipo de redes se incluye dentro de un perceptrón multicapa donde cada conexión sináptica está representada por un filtro FIR [Wan-93]. Esta red, junto al algoritmo de aprendizaje denominado Temporal backpropagation, es conocida como red FIR o perceptrón multicapa FIR. Las Redes TDNN y las FIR son conceptualmente equivalentes, y, únicamente, varían en cuanto a su representación gráfica y la descripción en la notación de sus ecuaciones.

B. Redes Recurrentes.

Las redes recurrentes (RR), también conocidas como redes realimentadas ("feedback networks") son redes en las que se permiten auto-conexiones o lazos de realimentación en las neuronas, y conexiones hacia atrás en las capas. Una de las consecuencias fundamentales derivadas del uso de estas conexiones es la posibilidad de modelizar comportamientos dinámicos, tarea imposible de realizar con las redes de propagación hacia delante ("feed-forward networks"). Esta capacidad de las RRs va desde el reconocimiento de patrones espacio-temporales, reconocimiento de señales o la predicción en series temporales. Otro posible beneficio que podemos obtener de su empleo es la obtención de resultados comparables con los obtenidos con redes de mayor tamaño sin realimentación. Por todo ello las RRs han sido ampliamente aceptadas como herramientas flexibles en el procesado de series temporales, identificación de sistemas y problemas de control.

Para entrenar este tipo de redes se siguen, principalmente, dos aproximaciones:

- "Backpropagation Through Time". Una idea intuitiva para entrenar redes recurrentes es desdoblarla en una red multicapa que crezca una capa cada instante. Rumelhart en 1986 denominó a esta técnica "unfolding in time" y constituyó para Eric A. Wan la técnica que demostraba la equivalencia entre una red FIR y una TDNN.

- "Real-Time Recurrent Learning". Introducidas por Williams y Zisper en 1989, son capaces de realizar un entrenamiento de forma continua y

difieren de la Red de Hopfield en dos aspectos importantes: tienen neuronas ocultas y poseen una dinámica arbitraria [Williams-89].

A continuación se discutirán las estructuras dinámicas más empleadas y sencillas así como los algoritmos de aprendizaje asociados. Las estructuras recurrentes más complejas conocidas como redes totalmente recurrentes ("Real-Time Recurrent Learning" y "Time-Dependent Recurrent Backpropagation") no serán estudiadas aquí por su alto coste computacional.

5.4.2. Tipos de Redes Recurrentes.

5.4.2.1. Redes Retropropagadas Recurrentes.

Tanto el algoritmo de retropropagación como su estructura constituyen un caso especial de las redes retropropagadas recurrentes. Podemos generalizar ese comportamiento añadiendo conexiones realimentadas de tal manera que se convergerá a un estado estable [Pineda-88], [Almeida-87], [Rohwer-87]. El algoritmo para entrenar la red recurrente final se le conoce como "Recurrent Backpropagation" o RBP. En la Figura 5.11 mostramos un ejemplo de este tipo de redes donde los nodos 1 y 2 son nodos de salida con las salidas deseadas d_1 y d_2; los nodos 1, 5, 6 y 7 son nodos de entrada; y los nodos 3 y 4 son nodos ocultos. El nodo 1 es tanto de entrada como de salida (Fig. 2).

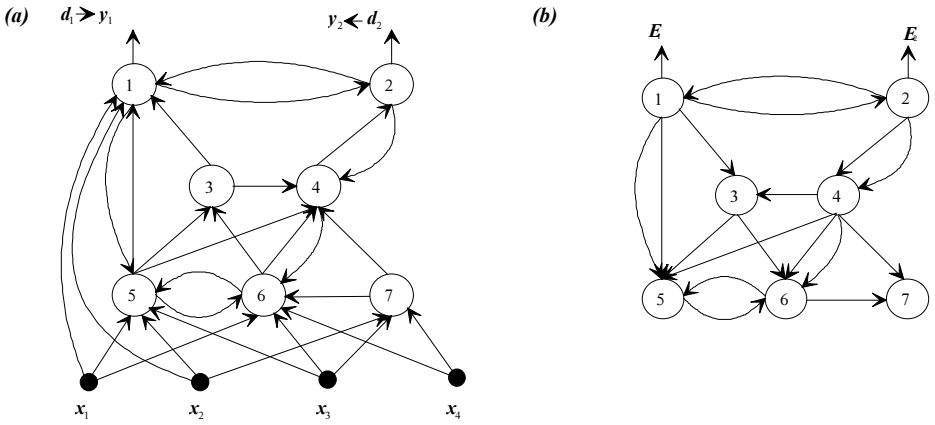


Figura 5.11. Ejemplo de una Red Retropropagada Recurrente (RBP). (a) Una red recurrente de siete nodos donde las salidas deseadas son d_1 y d_2. (b) La correspondiente red de propagación del error donde los errores de entrada son E_1 y E_2.

5.4.2.2.Redes Recurrentes Parciales.

Estas redes han sido empleadas con éxito en tareas de reconocimiento y reproducción de secuencias de estados ya que constan de conexiones asímetricas. Estas estructuras son redes multicapa parcial o totalmente conectadas donde las conexiones de realimentación existentes deben ser cuidadosamente elegidas. Esta recurrencia limitada permite a la red recordar secuencias temporales de valores sin que ello suponga un coste computacional excesivamente grande (como ocurre con las redes totalmente recurrentes). En la mayoría de los casos, los pesos de realimentación se fijan existiendo la posibilidad de emplear el algoritmo estándar

de retropropagación. A estas redes se las conoce también como redes secuenciales y a los nodos que reciben la realimentación como nodos contextuales.

En esta red la propagación "hacia delante" se supone muy rápida, mientras que la señal de realimentación sí se temporiza. Por lo tanto, en el instante t las unidades contextuales tienen señales procedentes del estado de la red en t-1. En este sentido, las unidades contextuales actúan como elementos de memoria y esto les confiere una enorme utilidad ante problemas de predicción o de reconocimiento de patrones al definir un estado en función de las entradas más el conjunto de variables que definieron el estado anterior.

Existe gran variedad de RRs parciales ya que las unidades contextuales, reciben la señal de realimentación, pueden ser los nodos de entrada o los de la capa oculta, mientras que las que proporcionan esa señal pueden ser bien nodos de salida u ocultos. A continuación discutiremos los modelos más extendidos.

Red Secuencial de Jordan.

La primera red parcialmente recurrente fue propuesta por Jordan denominándose Red Secuencial de Jordan [Lin-95]. Esta estructura se forma en dos fases: en la primera se añaden conexiones recurrentes desde las salidas de la red hacia las unidades contextuales C_i que forman una capa contextual, y en la segunda se realizan bucles sobre éstas.

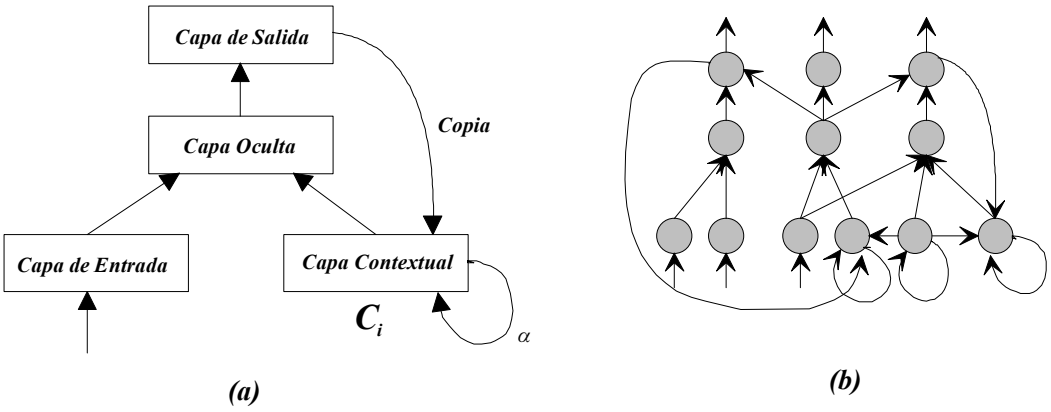


Figura 5.12. Red Secuencial de Jordan. (a) Modelo General. (b) Ejemplo de red de Jordan.

En la Fig. 3b se muestra un ejemplo de la red de Jordan, las salidas, asociadas con cada estado, se realimentan sobre las unidades contextuales que representan el siguiente estado del sistema y trabajan en paralelo con las entradas. Los bucles sobre las unidades contextuales, C_i, les proporcionan cierta memoria. La activación de estas unidades contextuales viene gobernada por la siguiente ecuación:

$$\dot{C}_i(t) = -\alpha C_i(t) + y_i(t) \qquad \text{Ec. 5.70}$$

Donde y_i son las activaciones de los nodos de salida y α es el factor que mide el valor de la realimentación ($0 < \alpha < 1$). La solución a la ecuación anterior es:

$$C_i(t) = C_i(0)e^{-\alpha t} + \int_0^t e^{-\alpha(t-s)} y_i(s)ds \qquad \text{Ec. 5.71}$$

De esta ecuación se deduce que, si las activaciones y_i fueran fijas, entonces C_i decaería exponencialmente hacia y_i/α, "olvidando" gradualmente sus valores previos. Es más, la ecuación anterior, nos indica que en general las unidades contextuales acumulan una suma pesada y desplazada. Cuando elegimos $\alpha = 0$ la respuesta del sistema se extiende indefinidamente hacia el pasado pero, por el contrario, se vuelve menos sensible ante los detalles. Por tanto, deberemos escoger α de acuerdo a la escala temporal de la secuencia. En tiempos discretos la regla de las unidades contextuales viene dada por:

$$C_i(t+1) = (1-\alpha)C_i(t) + y_i(t) \qquad \text{Ec. 5.72}$$

Como todas las conexiones modificables de un modelo de Jordan son de realimentación, se puede entrenar mediante el algoritmo típico de retropropagación tratando las unidades contextuales como entradas sin introducir términos como $\partial C_k/\partial w_{ij}$ a la hora de derivar.

Red recurrente simple de Elman. La red secuencial de Jordan se puede entrenar para reconocer y distinguir diferentes secuencias de entrada. Sin embargo, con secuencias de longitud creciente, la red encuentra muchas dificultades en su tarea. Para solucionar estos inconvenientes, Elman introdujo una arquitectura llamada SRN o "Simple Recurrent Network" [Elman-90]. En una red de este estilo, las conexiones de realimentación van desde la capa oculta a la capa contextual tal [Lin-95]:

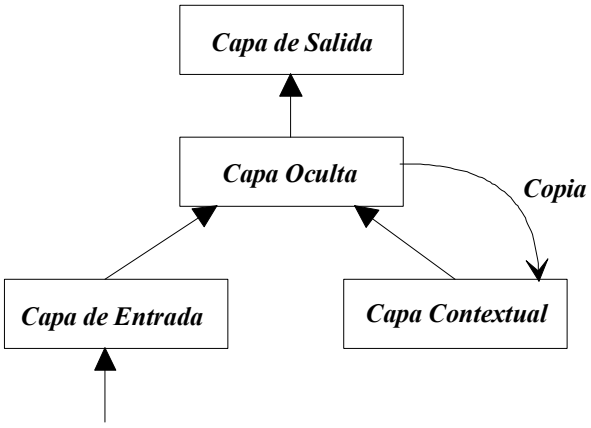


Figura 5.13. Estructura de una Red Recurrente Simple de Elman.

La capa de entrada la podemos considerar dividida en dos partes: unidades de entrada y unidades contextuales. Las unidades contextuales simplemente cogen una copia de las salidas (activaciones) de los nodos ocultos del instante previo, lo que proporciona, en cierto modo, una memoria al sistema. Al igual que con la red de Jordan, aquí se puede realizar una transformación de tal forma que se pueda aplicar el algoritmo de retropropagación. Como las salidas de la capa oculta son función tanto del estado actual como del anterior, la recurrencia en una SRN nos representa

tanto una dependencia temporal como una dependencia de la tarea. Estas redes han mostrado un excelente comportamiento ante tareas de predicción de series temporales.

Red Recurrente Parcial Simplificada. Este tipo de RR fue originariamente propuesta por Robinson y Fallside [Robinson-91] y su estructura se puede ver como una versión simplificada de la RR de Elman sin neuronas ocultas.

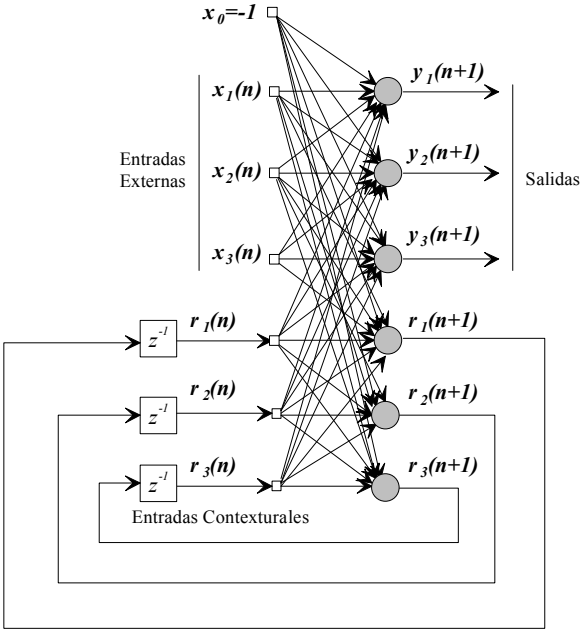


Figura 5.14. Red recurrente parcial simplificada con K = 3 neuronas de salida y L = 3 neuronas contextuales.

La red consiste en una capa de entrada (entradas externas más entradas contextuales) y una de salida. Las neuronas de salida producen el vector de salida total y(n+1) mientras que las neuronas contextuales producen el vector de realimentación r(n) tras un retardo aplicado a cada una de sus componentes. La red opera concatenando el vector de entrada externo x(n) y el vector realimentado r(n) de la siguiente manera:

$$\mathrm{u}(n) = [-1, \mathrm{x}(n), \mathrm{r}(n)] \qquad \text{Ec. 5.73}$$

La entrada fija -1 representa el umbral aplicado a cada neurona. El vector de entrada se multiplica por una matriz de pesos W(n) proporcionando un vector de actividad interno en el instante n:

$$\mathrm{v}(n) = [\mathrm{v}_0(n), \mathrm{v}_c(n)]^T = \mathrm{W}(n)\mathrm{u}(n) \qquad \text{Ec. 5.74}$$

Si la función de activación de las neuronas la definimos como $\phi(\cdot)$, el vector de salida calculado para el instante n+1 vendrá dado por:

$$\mathrm{y}(n+1) = \phi(\mathrm{v}_0(n)) \qquad \text{Ec. 5.75}$$

De manera similar, el vector de realimentación (antes del retardo) se define como:

$$r(n+1) = \phi(v_c(n)) \quad \text{Ec. 5.76}$$

Al final, el vector de salida y(n+1) se compara con el vector de salida deseada d(n+1) y se acumula su error de acuerdo a la función de coste elegida. En el algoritmo presentado por Robinson y Fallside, las salidas de la red se trataban como probabilidades empleándose la función de coste entrópica en el entrenamiento (salidas codificadas como 0 y 1):

$$H_{d\|y}(n+1) = \sum_{k=1}^{K}[d_k(n+1)\log y_k(n+1) - (1-d_k(n+1))\log(1-y_k(n+1))] \quad \text{Ec. 5.77}$$

Donde $y_k(n+1)$ es el k-ésimo elemento del vector de salida y(n+1) y $d_k(n+1)$ es el correspondiente valor de la respuesta deseada.

Resumiendo, las redes de Jordan y Elman extienden el MLP-BP con unidades contextuales, cuya función es recordar la actividad del pasado. Estas unidades contextuales son necesarias cuando se intenta aprender patrones dependientes con el tiempo, esto es, cuando los valores pasados influyen en el estado actual del sistema. Podemos tratar las unidades contextuales como entradas, procedentes de una fuente externa, y como las conexiones recurrentes son fijas, podemos emplear el BP estático en el entrenamiento. Si las conexiones recurrentes fueran adaptativas, deberíamos emplear el BP temporal, tal y como ocurre con las redes FIR.

Entre las ventajas evidentes del empleo de estas redes destaca su capacidad de resolver problemas en los que existe una cierta dependencia temporal. La red de Jordan es ligeramente más versátil que la de Elman ya que puede retener valores pasados de la información aunque su mayor dificultad está en la elección de la constante de tiempo de memoria. El otro inconveniente radica en el hecho que el pasado está atenuado de forma exponencial, lo que no siempre es conveniente. Estos problemas hacen que la red recurrente de Elman sea la más empleada.

MÁQUINAS DE VECTORES SOPORTE (SVM).

5.5. Máquinas de vectores soporte (SVM).

Hasta ahora el aprendizaje supervisado se ha visto como la minimización de una función de coste que mide la calidad del funcionamiento del sistema. En virtud de esta medida se escogen los diferentes parámetros que definen el comportamiento del sistema [Haykin-98].

Otro punto de vista, tan válido como el anterior, procede de un punto de vista geométrico. Veamos este punto de vista con un ejemplo sencillo: un problema de clasificación en dos clases que, en principio, supondremos que es linealmente separable. En este caso se busca determinar el hiperplano que separa los conjuntos de puntos. Un ejemplo sencillo de este tipo de problemas sería el representado en la siguiente figura

Una representación de los datos junto con los posibles planos que separan las dos clases queda representado en la figura anterior. Podemos observar que, en principio, existen infinitas soluciones de este problema. De todas las soluciones nos interesa aquella que generalice mejor. Ésta es la mayor virtud y principal ventaja de las redes neuronales frente a otros métodos de clasificación (métodos bayesianos, basados en reglas, etc). Podemos expresar esta condición de una manera más formal. La solución que mejor generaliza es aquella que maximiza la distancia entre el plano de separación y el patrón más cercano de cada clase. Por esta razón a los patrones más cercanos se les conoce como vectores soporte: la solución del problema depende de ellos. Así pues buscamos un sistema lineal que:

- Nos resuelva el problema de clasificación.
- Nos proporcione la mayor capacidad de generalización.

Los puntos definidos en un hiperplano quedan definidos por la siguiente ecuación:

$$H \equiv W \cdot X + b = 0 \qquad \text{Ec. 5.78}$$

Donde $W=[w_1\ w_2\ \ldots\ldots w_N]$ es el vector característico perpendicular al plano de separación y $X=[x_1\ x_2\ \ldots\ldots x_N]$ el vector de entrada. Así pues, la clasificación se basará en las siguientes condiciones:

$$W \cdot X + b \geq 0 \quad \text{si } y_i = +1 \qquad \text{Ec. 5.79}$$

$$W \cdot X + b \leq 0 \quad \text{si } y_i = -1 \qquad \text{Ec. 5.80}$$

Escalando adecuadamente w y b el problema puede ser replanteado de la siguiente forma:

$$W \cdot X + b \geq 1 \quad \text{si } y_i = 1 \qquad \text{Ec. (5.81)}$$

$$W \cdot X + b \leq -1 \quad \text{si } y_i = -1 \qquad \text{Ec. (5.82)}$$

Las condiciones (4) y (5) se pueden combinar en una sola:

$$y_i[W \cdot X + b - 1] \geq 0, \quad i = 1 \ldots L \qquad \text{Ec. (5.83)}$$

Los puntos de entrenamiento que cumplen la igualdad de la condición (4) están sobre el hiperplano $H_1 \equiv w \,.\, x_i + b = 1$ (la existencia de estos puntos dependerá del escalado realizado sobre w y b), y los análogos de la condición (5) estarán sobre el hiperplano $H_2 \equiv w \,.\, x_i + b = -1$ (figura 2). Estos dos hiperplanos son paralelos a H, tienen el mismo vector director, y están a una distancia $1/\|w\|$ de él. Además, estos puntos que están contenidos en H_1 y H_2 son los puntos más cercanos a H, por lo que $d_+ = d_- = 1/\|w\|$ y el margen del hiperplano H (distancia a los vectores más cercanos pertenecientes a diferentes clases) a maximizar es $2/\|w\|$. Es inmediato comprobar que maximizar esta cantidad es equivalente a minimizar la cantidad definida por la norma del vector al cuadrado, es decir $\|w\|^2$. El cuadrado se toma por sencillez en el manejo de las expresiones que se obtendrán a continuación.

La solución del problema viene dada al minimizar la función objetivo $0.5*|w||^2$ con la condición que el sistema resuelva el problema de clasificación. Estas condiciones se establecerán mediante el método de los multiplicadores de Lagrange, De acuerdo con lo expuesto para este problema de optimización se plantea el siguiente Lagrangiano:

$$L_P = \frac{\|w\|^2}{2} - \sum_{i=1}^{L} \alpha_i [y_i (W \cdot X_i + b) - 1] \qquad \text{Ec. (5.84)}$$

Debemos minimizar L_P con respecto w y b, y además se debe cumplir que las derivadas respecto de α_i se anulen con $\alpha_i \geq 0$ (para todo i). Al ser éste un problema de programación cuadrática convexa podemos hallar, equivalentemente, la solución planteando el "problema dual": maximizar L_P sujeto a las condiciones de que las derivadas respecto de w y b se anulen con la condición adicional que $\alpha_i \geq 0$. Este planteamiento equivalente es más sencillo de resolver [Stitson-96], [Vapnik-96].

Imponiendo que las derivadas de L_P respecto de w y b sean nulas se obtienen las condiciones:

$$\frac{dL_P}{dw} = 0 \Rightarrow W = \sum_{i=1}^{L} \alpha_i . y_i . X_i \qquad \text{Ec. (5.85)}$$

$$\frac{dL_P}{db} = 0 \Rightarrow \sum_{i=1}^{L} \alpha_i . y_i = 0 \qquad \text{Ec. (5.86)}$$

Sustituyendo (8) y (9) en (7) se obtiene la expresión necesaria para plantear el problema dual:

$$L_D = \sum_{i=1}^{L} \alpha_i - \sum_{i=1}^{L} \alpha_i . H_{ij} . \alpha_j \qquad \text{Ec. 5.87}$$

donde se ha introducido el Hessiano : $H_{ij} = y_i y_j X_i X_j$

Como en el problema dual se ha de maximizar respecto de α_i , se ha de cumplir:

$$\frac{dL_D}{d\alpha_i} = 0 \Rightarrow (H\alpha)_i = 1, \forall\, i \qquad \text{Ec. 5.88}$$

donde se observa que para hallar los multiplicadores de Lagrange y resolver así el problema, es necesario calcular la inversa del Hessiano.

Hay que destacar que el hiperplano de separación será una combinación lineal de los puntos de entrenamiento, i.e., dependerá únicamente de estos como se aprecia en la expresión 8. En concreto, y como habíamos planteado anteriormente, esta dependencia será sólo respecto de los vectores soporte (los puntos que caen sobre los planos H_1 o H_2) por lo que el multiplicador de Lagrange (α_i) correspondiente a dichos puntos será no nulo, no ocurriendo este hecho para el resto de puntos de entrenamiento.

Si los patrones no son linealmente separables se utiliza una función de transformación (kernel) que transforma los patrones a un espacio de dimensión superior donde sí son linealmente separables. Además, para el caso en que el problema tampoco sea linealmente separable en el espacio imagen mapeado, se añade a la función objetivo un término que representa los errores de clasificación durante el entrenamiento, ponderado con un factor fijado a priori [Vapnik-96]

Bibliografía.

[Adenso-96] Adenso, A. et alt. "Optimización Heurística y Redes Neuronales". Ed. Paraninfo, 1996.

[Almeida-87] Almeida, L.B. "A learning rule for asynchronous perceptrons with feedback in a combinatorial enviroment". Proc. IEEE Int. Conf. Neural Networks, vol. II, 609-618, San Diego, 1987.

[Anderson-72] Anderson, J.A. "A Simple Neural Network Generating an Interactive Memory". Recopilado en Neurocomputing: Foundations of Research, pp 181-192. MIT Press, 1989.

[Anderson-96] Anderson, G.T.et alt. . "A Neural Network Model for Prediccion of Progression of Left Anterior Descending Coronary Artery Stenosis in Hyperlipidemic Patients after a First Myocardial Infarction". IEEE Technology Update Series, Neural Networks Applications. Patrick K. Simpson Ed.-96.

[Azuaje-99] Azuaje, F. et alt. "Predicting Coronary Disease Risk Based on Short-Term RR Interval Measurements: A Neural Network Approach". Artificial Intelligence, vol 15, nº 3, pp 275-297, Marzo 1999.

[Back-95] A.D. Back, E. Wan, Steve Lawrence, and A.C. Tsoi, "A unifying view of some training algorithms for multilayer perceptrons with FIR filter synapses". IEEE Press. Pp.146-154, 1995.

[Barto-83] Barto, A.G., Sutton, R.S., Anderson, C. A. "Neuronlike Adaptive Elements that can Solve Difficult Learning Control Problems". Recopilado en Neurocomputing: Foundations of Research, pp 535-549. MIT Press, 1989.

[Bezerianos-99] Bezerianos, A., Papadmitriou, S., Alexopoulos, D. "Radial Basis Function Neural Networks for the Characterization of Heart Rate Variability Dinamics". Artificial Intelligence in Medicine, vol 15, nº 3, pp 215-234, 1999.

[Bishop-96] Bishop, C.M. "Neural Networks for Pattern Recognition". Clarendon Press, Oxford, 1996.

[Block-62] Block, H.D. "The Perceptron: A Model for Brain Functioning". Neurocomputing: Foundations of Research, pp 139-150. MIT Press, 1989.

[Bortolan-90] Bortolan, G., Degani, R., Willems, J.L. "Neural Networks for ECG Classification". Computers in Cardiology, pp 269-272, 1990.

[Bortolan-91] Bortolan, G., Degani, R., Willems, J.L. "ECG Classification with Neural Networks and Cluster Analysis". Computers in Cardiology, pp 177-180, 1991.

[Broomhead-88] Broomhead, D.S, Lowe, D. "Multivariable Functional Interpolation and Adaptive Networks". Complex Systems, vol 2, 321-355.

[Burr-88] Burr, D.J. "Experiments on Neural Net Recognition of Spoken and Written Text". IEEE Transactions on Acoustics, Speech, and Signal Processing, vol 36, nº 7, pp 1162-1168, Julio 1988.

[Camps-98] Camps, G. "Redes Neuronales aplicadas en el problema de la intoxicación por digoxina". Proyecto final de carrera Ingeniería Electrónica, Universitat de València, 1998.

[Caraiscos-84] Caraiscos, C., Liu, B. "A Roundoff Error Analysis of the LMS Adaptive Algorithm". IEEE Transc. Acoust, Speech, Signal Processing, vol ASSP-32, número 1, pp 34-41, Febrero 1984.

[Clarkson-89] Clarkson, P.M., Haweel, T.H. "A Median LMS Algorithm" Proc. IEE Electronics Letters, pp 520-522, año 1989.

[Clarkson-93] Clarkson, P.M. "Optimal and Adaptive Signal Processing". CRC Press, 1993.

[Coggins-95] Coggins, R., et alt. "A Low-Power Network for On-Line Diagnosis of Heart Patients". IEEE Micro, pp 18-25, Junio 1995.

[Corrigan-97] Corrigan, B.W., Mayo, P.R., Jamali, F. "Application of a Neural Network for Gentamicin Concentration in a General Hospital Population". Therapeutic Drug Monitoring, vol 19, nº 1, pp 25-28, 1997.

[Cowan-86] Cowan, CF.N., Grant, P.M. "Adaptive Filters". Prentice-Hall, 1986.

[Chen-91] Chen, S., et alt. "Reconstruction of Binary Signals Using an Adaptive Radial Basis-Function Equalizer". Signal Processing, vol 22, pp 77-93, 1991.

[Chen-96] Chen, C.T., Chang, W.D. "A Feedforward Neural Network with Function Shape Autotuning". Neural Networks, vol 9, nº 4, pp 627-641, 1996.

[Chen-98] Chen, G., Dong, X. "From Chaos to Order: Methodologies, Perspectives, and Applications". World Scientific Series, 1998.

[Chen-99] Chen, H.Y., et alt. "Prediction of Tacrolimus Blood Levels by Using the Neural Network with Genetic Algorithm in Liver Transplantion Patients". Therapeutic Drug monitoring, vol 21, nº 1, pp 50-56, 1999.

[Chi-91] Chi, Z., Jabri, M.A. "Identification of Supraventricular and Ventricular Arrhythmias Using a Combination of Three Neural Networks". Computers in Cardiology, pp 169-172, 1991.

[Dasgupta-90] S.Dasgupta, C.R.Johnson, JR, A.M Barksho. "Sign-sign LMS Convergence with Independence Stochastics Inputs". IEEE Transc. on Information Theory, vol 36, número 1, pp 197-201, 1990.

[Dokur-96] Dokur, Z. Et alt. "Detection of ECG Waveforms by Using Artificial Neural Networks". IEEE Engineering in Medicine and Biology,

paper 1038, 1996.

[Elman-90] Elman, J.L., 1990. "Finding Structure in Time". Cognitive Science 14, 179-211.

[Evans-93] Evans, J.B., Xue, P., Liu, B. "Analysis and Implementation of Variable Step Size Adaptive Algorithms". IEEE Transactions on Signal Processing, vol 41, nº 8, pp 2517-2534, Agosto 1993.

[Eweda-87] Eweda, E. Macchi, O. "Convergence of the RLS and LMS Adaptive Filter". IEEE Transactions on Circuits and Systems, vol CAS-34, nº 7, pp 799-803, 1987.

[Falhman-90] Fahlman, S.E., Lebière, C. "The Cascade-Correlation Learning Algorithm". Technical report CMU-CS_90-100, disponible en

[Fausett-94] Fausett, L. "Fundamentals of Neural Networks: Architectures, Algorithms and Applications". Prentice-Hall, 1994.

[Fogelman-98] Fogelman-soulié, F. "Application of Neural Networks". Recopilado en The Handbook of Brain Theory & Neural Networks, ed. MIT Press, 1998.

[Foley-87] Foley, J.B., Boland, F.M. "Comparison between Steepest Descent and LMS Algorithms in Adaptive Filters". IEE Proc. vol 134, Pt F, número 3, pp 283-289, año 1987.

[Fukushima-83] Fukushima, K., Miyake, S., Ito, T. "Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition". Recopilado en Neurocomputing: Foundations of Research, pp 526-534, MIT Press, 1989.

[Gallant-93] Gallant, S. "Neural Networks Learning and Expert Systems". MIT Press, 1993

[Gardner-87] W.A.Gardner. "Nonstationary Learning Characteristics of the LMS Algorithm".IEEE Transc. on Circuits and Systems, vol CAS-34, número 10, pp 1199-1207, año 1987.

[Gholkar-90] Gholkar, V.A. "Mean Square Convergence Analysis of LMS Algorithm". Electronic Letters, pp 1705-1706, 1990.

[Gibson-91] Gibson, G.J., Siu, S., Cowan, C.F.N. "The Application of Nonlinear Structures to the Reconstruction of Binary Signals". IEEE Transactions on Signal Processing, vol 39, pp 1877-1884, 1991.

[Golomb-95] Golomb, B., Sejnowski, T. "Sex Recognition fron Faces Using Neural Networks". Applications of Neural Networks, Kluwer Academic Publishers, 1995.

[Gorse-97] Gorse, D., Shepeherd, A.J., Taylor, J.G. "The New ERA in Supervised Learning". Neural Networks, vol 10, nº 2, pp 343-352, 1197.

[Grossberg-80] Grossberg, S. "How Does a Brain Build a Cognitive Code". Recopilado en Neurocomputing: Foundations of Research, pp 349-

399, MIT Press, 1989.

[Gutjahr-97] Steffen Gutjahr, Martin Riedmiller and Justus Klingermann, "Daily Prediction of the Foreign Exchange Rate Between the US Dollar and the German Mark Using Neural Networks", 1997.

[Hair-98] Hair, J.F.H., et alt. "Multivariate Data Analysis". Prentice Hall 1998.

[Ham-93] Ham, F.M., Han, S. "Cardiac Arrhythmia Classification using Fuzzy ARTMAP". IEEE Engineering in Medicine and Biology, pp 288-289, 1993.

[Ham-96] Ham, F.M., Han, S. "Classification of Cardiac Arrhythmias Using Fuzzy ARTMAP". IEEE Transactions on Biomedical Engineering vol 43, nº4,pp 425-429,1996.

[Har-93] Har, J. et alt. "Neural Network Electrocardiogram Classification Through Key Feature Extraction". IEEE Engineering in Medicine and Biology, pp 727-728, 1993.

[Hashem-97] Hashem, S. " Optimal Linear Combinations of Neural Networks". Neural Networks, vol 10, pp 599-614, 1997.

[Hassoun-95] Hassoun, M.H. "Fundamentals of Artificial Neural Networks". MIT Press, 1995.

[Hayashi-92] Hayashi, Y., Czogala, E., Buckley, J. "Fuzzy Neural Controller". Proc. IEEE Int. Conf. Fuzzy Syst., 197-202, San Diego, 1992.

[Haykin-96] Haykin, S. "Adaptive Filter Theory". Prentice-Hall, 1996.

[Haykin-96b] Haykin, S. "Neural Networks Expand SP's Horizons". IEEE Signal Processing Magazine, vol 13, nº 2, pp 24-49, Marzo 1996.

[Haykin-98] Haykin, S. "Neural Networks: A Comprehensive Foundation". Prentice-Hall, 1998.

[Hebb-49] Hebb, D.O. "Introduction and Chapter 4, The First Stage of Perception: Growth and Assembly". Aparece recopilado en Neurocomputing: Foundations of Research, pp 45-56, MIT Press, 1989.

[Hilera-95] Hilera, J.R., Martínez, V.J. "Redes Neuronales Artificiales: Fundamentos, Modelos y Aplicaciones". Editorial Ra-Ma, 1995.

[Hopfield-82] Hopfield, J.J. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities". Proc. of the National Academy of Sciences, vol 79, pp 2254-2558, recopilado en Neurocomputing: Foundations of Research, pp 460-464, MIT Press, 1989.

[Hopfield-84] Hopfield, J.J. "Neurnos with Graded Response Have Collective Computational Properties like Those of Two-State Neurons". Proc. of the National Academy of sciences, vol 81, pp 3088-3092, recopilado en Neurocomputing: Foundations of Research, pp 579-583, MIT Press, 1989

[Jabri-95] Jabri, M. et alt. "ANN Based Classification of Arrhythmias". Applications of Neural Networks, Kluwer Academic Publishers, 1995.

[Jang-97] Jang, J.S.R., Sun, C.T., Mizutani, E. "Neuro-Fuzzy & Soft Computing". Prentice-Hall, 1997.

[Kalita-93] Kalita, S. et alt. "Effective ECG Classification Using Single Layer Neural Network with Data Pre-processing". IEEE Engineering in Medicine and Biology, pp 734-735, 1993.

[Kennedy-98] Kennedy, R.L. et alt. "Solving Data Mining Problems Trough Pattern Recognition". Prentice-Hall. 1998.

[Kermanshashi-96] Kermanshashi, et al. "Artificial Neural Network for Forecasting Daily Loads of a Canadian Electric Utility" IEEE Technology Update Series, Neural Networks Applications. Patrick K. Simpson Ed.-96.

[Kirkpatrick-83] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P. "Optimization by Simulated Annealing". Recopilado en Neurocomputing: Foundations of Research, pp 554-567, 1989.

[Kohonen-72] Kohonen, T. "Correlation Matrix Memories". Recopilado en Neurocomputing: Foundations of Research, pp 174.180, MIT Press, 1989.

[Kohonen-82] Kohonen, T. "Self-Organized Formation of Topologically correct Feature Maps". Recopilado en Neurocomputing: Foundations of Research, pp 511-521, MIT Press, 1989.

[Kolmogorov-57] Kolmogorov, A. N. "On the Representation of Continuous Functions of Several Variables by Superposition of Continuous Functions of One Variable and Addition". Doklady Akademiia Nauk SSRR 114 (5), 953-956.

[Kosko] Kosko, B. "Fuzzy Engineering". Prentice-Hall, 1997.

[Kramer-95] Kramer, C., McKay, B., belina, J. "Probabilistic Neural Network Array Architecture for ECG Classification". IEEE Engineering in Medicine and Biology, paper 448, 1995.

[Krogh-97] Krogh, A. " Statistical Mechanics of Ensemble Learning". Physical Review, E55, 811, 1997.

[Kung-93] Kung, S.Y. "Digital Neural Networks". Prentice-Hall, 1993.

[Kung-95] S. Y. Kung, J. S. Taur, "Decision-Based Neural Networks with Signal/Image Classification Applications", IEEE Transactions on Neural Networks, vol. 6, nº. 1, pp. 170-181, 1995.

[Kwong-92] Kwong, R.H., Johnston, E.W. "A Variable Step Size Algorithm". IEEE Transactions on Signal Processing, vol 40, nº 7, pp 1633-1642, Julio 1992

[Lang-88] Lang, K. J., Hinton, E.."The Development of the Time-Delay Neural

Network architecture for Speech Recognition", Technical Report CMU-CS-88-152. Carnegie-Mellon University, Pittsburgh, PA. 1988.

[Lashley-50] Lashley, K.S. "In Search of the Engram". Society of Experimental Biology Symposium, nº 4: Physiological Mechanisms in Animal Behaviour". Ambridge University Press, recopilado en Neurocomputing: Foundations of Research, pp 59-63, MIT Press, 1989.

[Leong-91] Leong, P.H.W., Jabri, M.A. "Arrhythmia Classification Using Two Intracardiac Leads". Computers in Cardiology, pp 189-192, 1991.

[Lin-96] Lin, Ch.T., Lee, G. "Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems". Prentic-Hall, 1996.

[Luenberger-73] Luenberger, D.G. "Introduction to Linear and Nonlinear Programming". Adisson-Wesley, 1973.

[MacPherson-95] MacPherson, K.P., Conway, A.J. Brown, J. "Prediction of Solar and Geomagnetic Activity Data Using Neural Networks", Journal of Geophysical Research, 1995.

[Macchi-83] O.Macchi, E.Eweda. "Second-order Convergence Analysis of Stochastic Adaptive Linear Filtering". IEEE Transc. Automat. Contr. vol AC-28 pp 76-85, Enero 1983.

[Makridakis-98] Makridakis, S., Wheelwright, S., Hyndman, R. "Forecasting. Methods and Applications"., Ed. John Wiley and Sons, 1998,

[Martín-99] Martín J.D. "Aplicación de redes neuro-difusas en problemas de modelización y clasificación". Proyecto final de carrera, Ingeniería Eelctrónica, Universitat de València, 1999.

[May-96] May, G.S. "Applications of Neural Networks in Semiconductor Manufacturing Processes". Handbook of Fuzzy Logic and Neural Network. IEEE Press, 1996.

[McCulloch-43] McCulloch, W.S., Pitts, W. "A Logical Calculus of the Ideas Immanent in Nervous System". Recopilado en Neurocomputing: Foundations of Research, pp 18-27, MIT Press 1987.

[Mead-87a] Mead, C.A. "Silicon Models of Neural Computation". 1 st IEEE International Conference on Neural Networks, vol I, pp 93-106, San Diego, CA. 1987.

[Mead-87b] Mead, C.A. "Neural Hardware for Vision". Enginnering and Science 2-7, 1987.

[Mead-88] Mead, C.A., Mahowald, M.A. "A Silicon Model of Early Visual Processing". Neural Networks 1, pp 91-97, 1988.

[Mendel-95] Mendel, J. M. "Fuzzy Logic for Engineering: A Tutorial". Proc. IEEE, Vol. 83, No. 3, March 1995.

[Minsky-69] Minsky, M., Papert, S. "Perceptrons". Recopilado en Neurocomputing: Foundations of Research, pp 161-173. MIT Press. 1989.

[Miyano-94] Takaya Miyano and Federico Girosi, "Forecasting Global Temperature Variations by Neural Networks",A.I. Memo No. 1447, CBCL Memo No. 101, Aug-1994.

[Moamelli-91] Moamelli, C. "Classifying Cells for Cancer Diagnosis Using Neural Networks". IEEE Expert, 1991.

[Molina-99] Molina, J. "Uso de comités de expertos en problemas de clasificación". Proyecto final de carrera, Ingeniería Electrónica, Universitat de València, 1999.

[Morabito-91] Morabito, M. et alt. "QRS Morphological Classification Using Artificial Neural Networks". Computers in Cardiology, pp 181-184, 1991.

[Opitz-97] Opitz, D.W., Shavlik, J.W. " Actively Searching for an Effective Neural Network Ensemble". Connection Science, 8 (3&4), pp 337-354, 1996.

[Palreddy-95] Palreddy, S., Tompkins, W.J., Hu, Y.H. "Customization of ECG Beat classifiers Developed Using SOM and LVQ". IEEE Engineering in Medicine and Biology, paper 778, 1995.

[Pao-96] Pao, Y. H., Yip, P. P. C. "Neural Net Process Monitoring and Optimal Control". Fuzzy Logic and Neural Network Handbook. IEEE Press, 1996.

[Perrone-93] Perrone, M., Cooper, L. "When Networks Disagree: Ensemble Methods for Hybrid Neural Networks". Neural Networks for Speech and Signal Processing, Chapman Hall, 1993.

[Pineda-88] Pineda, F.J. "Dynamics and architecture for neural computation". J. Complexity 4, pp. 216-245.

[Poggio-90] Poggio, T., Girosi, F. "Networks for Approximation and Learning". Proceedings IEEE, vol 78, nº 9, pp 1481-1497, Septiembre 1990.

[Principe-97] J. Principe, L. Giles, N. Morgan, E. Wilson. Steve Lawrence Andrew D. Back Ah Chung Tsoi C. Lee Giles, "The Gamma MLP Using Multiple Temporal Resolutions for Improved Classification. IEEE Workshop on Neural Networks for Signal Processing VII, pp. 62-367,

[Pretorius-92] Pretorius, L.C., Nel, C. "Feature Extraction from ECG for Classification by Artificial Neural Networks". Computer-Based Medical Systems, Proceedings of the Fith Annual IEEE Symposium, pp 639-647, 1992.

[Proakis-97] Proakis, J. Manolakis, D. "Tratamiento Digital de Señales", Prentice-Hall, 1997.

[Reed-93] Reed, R. "Pruning Algorithms: A Survey". IEEE Transactions on Neural Networks, vol 4, nº 5, pp 740-747, Septiembre 1993

[Reinhardt-96] Reinhardt, L. et alt. "Application of Learning Vector Quantization for Localization of Myocardial Infarction". IEEE Engineering in Medicine and Biology, paper 211, 1996.

[Ripley-96] Ripley, B.D. "Pattern Recognition and Neural Networks". Cambridge University Press, 1996.

[Robinson-91] Robinson, A.J. y F. Fallside, 1991. "A Recurrent Error Propagation Speech Recognition System." Computer Speech and Language 5, 259-274.

[Rohwer-87] Rohwer, R. y B. Forrest. "Training time-dependence in neural networks". Proc. IEEE Int. Conf. Neural Networks, vol. II, 701-708, San Diego, 1987.

[Rojas-95] Rojas, R. "Neural Networks: A Systematic Introduction". Springer-Verlag, 1995.

[Roli-96] Roli, F., Serpico, S.B., Vernazza, G. "Neural Networks for Classification of Remotely Sensed Images". Fuzzy Logic and Neural Network Handbook. IEEE Press, 1996.

[Rosenblatt-58] Rosenblatt, F. "The Perceptrón: A Probabilistic Model for Information Storage and Organization in the Brain". Neurocomputing: Foundations of Research, pp 92-113, MIT Press, 1989.

[Roy-90] Roy, S., Shynk, J.J. "Analysis of the Momentum LMS Algorithm". IEEE Transactions on Acoustics, Speech and Signal Processing, vol 38, nº 12, pp 2088-2098, Diciembre 1990.

[Rumelhart-86] Rumelhart, D.E., Hinton, G.E., Williams, R.J. "Learning Internal Representations by Error Propagation". Neurocomputing: Foundations of Research, pp 675-695. MIT Press, 1989.

[Sanger-89] Sanger, T.D. "Optimal Unsupervised Learning in a Single-Layer Linear Feedforward Neural Network". Neural Networks, vol 12, pp 459-473.

[Selfridge-58] Selfridge, O.G. "Pandemonium: a Paradigm for Learning". Mechanisation of Thought Processes: Proceedings of a Symposium Held at the National Physical Laboratory, Noviembre 1958 Recapitulado en Neurocomputing, pp 117-120, MIT Press, 1989.

[Serrano-98] Serrano, A.J. "Redes Neuronales aplicadas en el problema de la émesis posquimioterapia ". Tesis de Licenciatura, Universitat de València, Mayo 1998

[Sethares-88] Sethares, W.A., et alt. "Excitation Conditions for Signed Regressor Least Mean Squares Regressor". IEEE, Transc. on Circuits and Systems, vol 35, número 6, Junio, pp 613-624, año 1988.

[Sharkawi-96] M.A. El-Sharkawi, R.J. Marks II, S. Oh, C.M. Brace. "Data Partitioning for Training a Layered Perceptron to Forecast electric Load". IEEE Technology Update Series, Neural Networks Applications. Patrick K. Simpson Ed.-96.

[Shukla-93] Shukla, D.V. et alt. "Wavelet Transform for Small Dimension Neural Netwrok Pattern classification of Subtly Different ECGS". IEEE Engineering in Medicine & Biology, pp 729-730, 1993.

[Silipo-96] Silipo, R., Bortolan, G., Marchesi, C. "Supervised and Unsupervised Learning for Diagnostic ECG Classification". IEEE Engineering in Medicine and Biology, paper 1054, 1996.

[Slock-93] Slock, D.T.M. "On the Convergence Behavior of the LMS and the Normalized LMS Algorithms". IEEE Transactions on signal Processing, vol 41, nº 9, pp 2811-2825, Septiembre 1993.

[Soria-97] Soria, E. et alt. "Application of an Artificial Neural Network with a Piecewise-Linear Activation Function to Determine the End of the T-Wave in an ECG". World Congress on Medical Physics & Biomedical Engineering, Niza, Septiembre 1997.

[Stitson-96] Stitson M.O., Weston J.A.E., Gammerman A., Vovk V., Vapnik V. "Theory of Support Vector Machines". Technical Report, Royal Holloway College, Report Number CSD-TR-96-17.

[Suga-90] Suga, N. "The Extent to Which Biosonar Information is Represented in the Bat Auditory Cortex". Neurocomputing: Directions for Research. MIT Press, 1990.

[Suga-98] Suga, N., Kanwal, S. "Echolocation: Creating Computational Maps". Tha Handbook of Brain Theory and Neural Networks, pp 344-348, MIT Press, 1998.

[Tamura-90] Tamura, S. Nakamura, M. "Improvements to the Noise Reduction Neural Network". Proceedings IEEE ICASSP'90, vol 2, pp 825-828, 1990.

[Taur-93] J. S. Taur, S.Y. Kung, "Fuzzy-Decision Neural Networks", Princeton University. IEEE. pp. 73-76, 1993.

[Thomson-93] Thomson, D.C., Soraghan, J.J., Durrani, T.S. "An Automatic Neural-Network Based SVT/VT Classification System". Computers in Cardiology, pp 333-336, 1993.

[Tsai-90] Tsai, Y.S., Hung, B.N., Tung, S.F. "An Experiment on ECG Classification Using Back-Propagation Neural Network". IEEE Engineering in Medicine and Biology, vol 12, nº 3, pp 1463-1464, 1990]

[Tsao-93] Tsao J., Wolter, J., Wang, H. "Model-Based Understanding of Uncertain Observational Data for Oil Spill Tracking". IEEE Technology Update Series, Neural Networks Theory, Technology and Applications. Patrick K. Simpson Ed.-98.

[Ukrainec-96] Ukrainec, A., Haykin, S. "A Mutual Information-Based Learning Strategy and Its Application to Radar". Fuzzy Logic and Neural Network Handbook. IEEE Press, 1996.

[Vapnik-96] Vapnik, V.M. "The Nature of Statistical Learning Theory". Springer Verlag, 1996.

[Vincent-95] Vincent, J.M. "Face Finding in Images". Applications of Neural Networks, Kluwer Academic Publishers, 1995.

[Vonk-97] Vonk, E., Jain, L.C., Johnson, R.P. "Automatic Generation of Neural Network Architecture Using Evolutionary Computation". Advances in Fuzzy Systems, World Scientific, 1997.

[Von-Neumann-58] Von-Neumann, J. "The Computer and the Brain". Yale University Press, recopìlado en Neurocomputing: Foundations of Research, pp 83-87, MIT Press, 1989.

[Waibel-89] Waibel, A. et alt. "Phoneme Recognition Using Time-Delay Neural Networks". IEEE Transactions on Acoustics, Speech and Signal Processing, vol 37, nº 3, pp 3238-339, Marzo 1989.

[Wan-93] Wan, E. "Finite Impulse Response Neural Networks with Applications in Time Series Prediction". PhD dissertation, Universidad de Stanford, 1993.

[Werbos-74] Werbos, P. "Beyond Regression: New Tools for Predictio and Analysis in the Behavioral Sciences". Ph. D. Thesis, Harvard, University Press, Cambridge, 1974.

[Weigend-94] Weigend, A.S., Gershenfeld, N.A. "Time Series prediction: Forecasting the Future and Understanding the Past". Addison-Wesley, 1994.

[Weigend-97] Weigend, A.S., "Data Mining in Finance: Rreport from the Post-NNCM-96 Workshop on Teaching Computer Intensive Methods for Financial Modeling and Data Analysis." Decision Technologies for Financial Engineering (Proceedings of the Fourth International Conference on Neural Networks in the Capital MArkets, NNCM-96), pp. 339-412. 1997.

[Widrow-60] Widrow, B., Hoff, M.E. "Adaptive Switching Circuits". Neurocomputing: Foundations of Research, pp 126-134, MIT Press, 1989.

[Widrow-75] Widrow, B., et alt. "Adaptive Noise Cancelling: Principles and Applications". Proceedings IEEE, vol 63, no12, pp 1692-1716, 1975.

[Widrow-76] Widrow, B., McCool, J.M., Larimore, M.G., Johnson, C.R. "Stationary and Non Stationary Learning Characteristics of the LMS Adaptive Filter". Proceedings IEEE, vol 64, pp 1151-1162, 1976.

[Williams-90] Williams, R.J., Zisper, D. "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks". Neural Computation 1, 270-280.

[Williamson-93] Williamson, G.A., Clarkson, P.M., Sethares, W.A. "Perfomance Characteristics of the Median LMS Adaptive Filter". IEEE Transc. on Signal Processing, vol 41, número 2, pp 667-680, año 1993.

[Zadeh-65] Zadeh, L. A., "Fuzzy Sets". Information and Control, vol. 8, pp. 338-353, 1965.