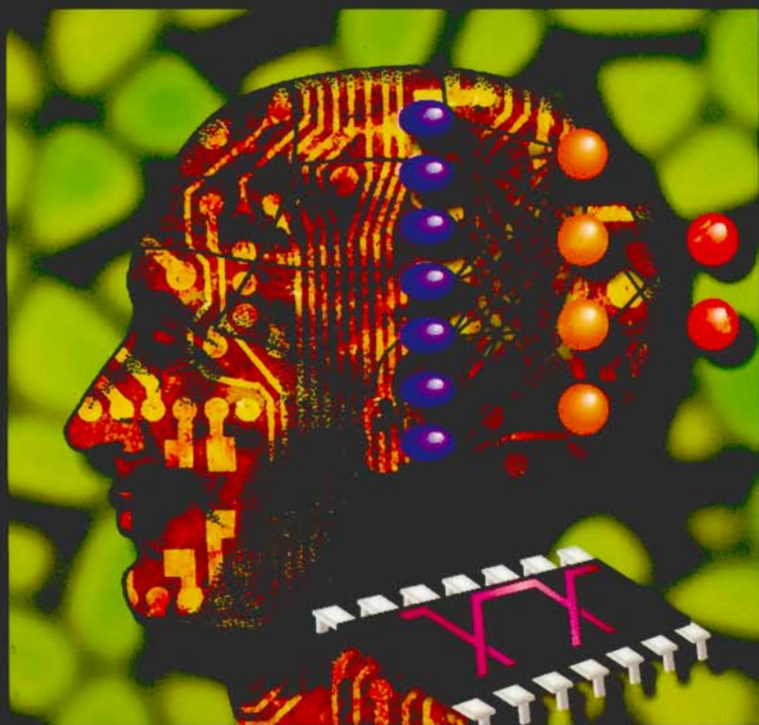


Redes Neuronales y Sistemas Difusos

2ª EDICIÓN AMPLIADA Y REVISADA



Bonifacio Martín del Brío
Alfredo Sanz Molina

Prólogo de Lotfi A. Zadeh

Alfaomega  Ra-Ma®



Incluye
CD-ROM

ÍNDICE

PRÓLOGO	xv
PREFACIO DE LOTFI A. ZADEH	xix
FOREWORD BY LOTFI A. ZADEH	xxi
INTRODUCCIÓN.....	xxiii
1 El largo y tortuoso camino hacia la construcción de máquinas inteligentes	xxiii
2 Microprocesadores, computadores y cerebro	xxvi
3 Redes neuronales artificiales	xxx
4 Sistemas borrosos.....	xxxii
5 Redes neuronales y sistemas borrosos.....	xxxiii
 PARTE I. REDES NEURONALES	 1
 CAPÍTULO 1. FUNDAMENTOS DE LAS REDES NEURONALES ARTIFICIALES	 3
1.1 Breve introducción biológica	3
1.2 Estructura de un sistema neuronal artificial	10
1.3 Modelo de neurona artificial	13

1.3.1 Modelo general de neurona artificial	13
1.3.2 Modelo estándar de neurona artificial	18
1.4 Arquitecturas de redes neuronales.....	21
1.5 Modos de operación: recuerdo y aprendizaje	26
1.6 Clasificación de los modelos neuronales.....	30
1.7 Computabilidad neuronal.	32
1.8 Un ejercicio de síntesis: sistemas conexionistas.....	33
1.9 Realización y aplicaciones de los ANS.....	35
1.A Apéndice: de la neurona biológica a la artificial.	37

CAPÍTULO 2. REDES NEURONALES SUPERVISADAS..... 41

2.1 Redes unidireccionales	41
2.2 El asociador lineal: aprendizaje hebbiano	42
2.3 El perceptrón simple (Rosenblatz, 1959).	47
2.3.1 Algoritmo de aprendizaje del perceptrón	51
2.4 La adalina (Widrow, 1961)	55
2.4.1 Regla LMS.....	56
2.5 El perceptrón multicapa (grupo PDP, 1986)	63
2.5.1 El MLP como aproximador universal de funciones	64
2.5.2 Aprendizaje por retropropagación de errores (BP)	66
2.5.3 Aceleración del aprendizaje BP. Otros algoritmos	69
2.6 Capacidad de generalización de la red	71
2.7 Pinceladas sobre la relación del MLP con los métodos estadísticos	76
2.8 Ejemplos de aplicación del MLP-BP	79

CAPÍTULO 3. REDES AUTOORGANIZADAS 85

3.1 Modelos neuronales no supervisados.	85
3.2 Modelo de mapas autoorganizados (Kohonen, 1982).	88
3.2.1 Introducción a los mapas autoorganizados.....	88
3.2.2 Algoritmo de aprendizaje.....	92
3.2.3 Algunas variantes de los SOFM.....	98
3.3 Ejemplos de aplicaciones	100
3.4 SOFM: cuantificación óptima de vectores	106
3.5 Análisis formal del proceso de autoorganización.....	108
3.6 Modelos de neurona de Kohonen. Medidas de similitud.....	112
3.7 Modelos de aprendizaje en mapas autoorganizados	113

CAPÍTULO 4. OTROS MODELOS DE REDES NEURONALES. 121

4.1 Redes neuronales realimentadas	121
4.2 Modelo de Hopfield	123
4.2.1 Modelo de neurona y arquitectura. Dinámicas.....	123
4.2.2 Memoria asociativa.....	127
4.2.3 Función energía de la red.....	129
4.3 Aprendizaje en la red de Hopfield.....	132
4.3.1 Regla de Hebb.....	133
4.3.2 Reglas óptimas.....	137
4.4 Ejemplo: reconocimiento de caracteres.....	140
4.5 Neuronas estocásticas. Máquina de Boltzmann.....	142
4.6 Modelo de Hopfield analógico (continuo)	145
4.6.1 Modelo de Hopfield de neuronas continuas.....	145
4.6.2 Aplicaciones del modelo de Hopfield analógico.....	148
4.7 Funciones de base radial (RBF)	152
4.8 LVQ	158
4.9 Otros modelos de redes neuronales.....	160

CAPÍTULO 5 IMPLEMENTACIÓN DE REDES NEURONALES 161

5.1 Introducción	162
5.2 Simulación (software) de ANS.....	163
5.3 Emulación (hardware) de ANS	166
5.4 Realización hardware de ANS	168
5.5 Neurocomputadores y chips neuronales	169
5.5.1 Especificaciones de un neuroprocesador.....	171
5.5.2 Aspectos generales de la realización VLSI.....	174
5.6 Bloques básicos en la realización de neuroprocesadores digitales	176
5.6.1 Sistema de control.....	176
5.6.2 Unidad de proceso.....	177
5.6.3 Unidad de almacenamiento.....	179
5.6.4 Unidad de comunicación.....	180
5.6.5 Arquitecturas reconfigurables	184
5.6.6 Realizaciones especiales: lógica de frecuencia de pulsos	185
5.7 Bloques básicos en la realización de neuroprocesadores analógicos.....	186
5.7.1 Unidad de proceso	187
5.7.2 Unidad de almacenamiento	190
5.8 ¿Realización analógica o digital?	193
5.9 Realizaciones analógicas de ANS	198

5.10 Realizaciones digitales de ANS	200
5.11 Ejemplo: neuroemulador basado en FPGA	203
5.12 Resumen. Situación comercial y tendencias.....	206

CAPÍTULO 6 APLICACIONES DE LAS REDES NEURONALES ARTIFICIALES 209

6.1 Motivación e interés del empleo de ANS.....	209
6.2 Desarrollo de una aplicación con ANS	212
6.3 Programas de simulación de ANS.....	220
6.3.1 Programas comerciales	220
6.3.2 Programas de libre distribución	222
6.4 Comparación con otras técnicas.....	223
6.4.1 Redes neuronales e inteligencia artificial.....	223
6.4.2 Redes neuronales y estadística.....	225
6.4.3 Inconvenientes de las redes neuronales.....	227
6.5 Aplicaciones reales de los ANS	228
6.5.1 Informes sobre el estado de la aplicación de ANS.....	228
6.5.2 Listado de aplicaciones	229
6.6 Ejemplo de aplicación de ANS: previsión de la demanda de consumo eléctrico.	235
6.7 Conclusiones	239

PARTE II. SISTEMAS BORROSOS.....241

CAPÍTULO 7. LÓGICA BORROSA..... 243

7.1 Introducción.	244
7.2 Conjuntos borrosos.....	248
7.3 Funciones de inclusión de conjuntos borrosos.	249
7.4 Variable lingüística.	253
7.5 Particiones borrosas.	254
7.6 Medidas borrosas.	254
7.7 Operaciones borrosas	255
7.8 Inferencia borrosa.....	257
7.8.1 Principio de extensión.....	258
7.8.2 Relación borrosa.	258
7.8.3 <i>Modus Ponens</i> Generalizado y <i>Modus Tolens</i> Generalizado. .	259
7.8.4 Implicación borrosa.	260

7.9 Reglas borrosas	261
7.10 Dispositivos de inferencia borrosa	262
7.11 Borrosificador (<i>fuzzifier</i>)	264
7.12 Desborrosificador (<i>defuzzifier</i>)	265
7.13 Desarrollo de sistemas borrosos	266
7.14 Borrosidad y probabilidad	268
 CAPÍTULO 8. SISTEMAS DE CONTROL BORROSO	269
8.1 Introducción al control borroso	269
8.2 Un primer ejemplo	271
8.3 Tipos de controladores borrosos	276
8.3.1 Controladores borrosos directos sin optimización	276
8.3.2 Controladores borrosos directos con optimización	278
8.3.3 Controladores borrosos híbridos	281
 CAPÍTULO 9. APRENDIZAJE EN SISTEMAS BORROSOS	283
9.1 Introducción	283
9.2 Retropropagación (BP)	284
9.3 Algoritmos genéticos	287
9.3.1 ¿Qué optimizar?	290
9.3.2 Codificación	293
9.3.3 Operadores cruzamiento y mutación	295
9.3.4 Diseño de la función de idoneidad	297
9.4 Algoritmos genéticos desordenados	299
9.4.1 Codificación	300
9.4.2 Operadores <i>corta</i> y <i>empalma</i>	303
 CAPÍTULO 10. IMPLEMENTACIÓN DE SISTEMAS BORROSOS	305
10.1 Introducción	305
10.2 Entornos de desarrollo	307
10.2.1 Entornos de tipo matemático	311
10.2.2 Entornos de lógica borrosa	316
10.3 Codificación en C	324
10.4 Codificación en C++	326
10.4.1 El modelo ARS (Sistemas de Respuesta Autónoma)	326
10.4.2 Objeto <i>Vinculo</i>	330
10.4.3 Objeto <i>World</i>	332

10.4.4 Objeto <i>FEN</i> (red borrosa equivalente)	332
10.4.5 Objeto <i>ANN</i>	334
10.5 Realización hardware de sistemas borrosos. Aceleradores	334

CAPÍTULO 11. APLICACIONES DE LOS SISTEMAS BORROSOS..... 341

11.1 Introducción. <i>Soft computing</i> , o imitando a la naturaleza	341
11.2 Interés del empleo de la lógica borrosa. Fusión de tecnologías.....	342
11.3 Algunas aplicaciones de los sistemas borrosos	344
11.4 Robots móviles y navegación autónoma	345
11.5 Conclusión final	348

APÉNDICES

A. CONTENIDO DEL CD-ROM..... 351

B. RECURSOS EN INTERNET 353

BIBLIOGRAFÍA 357

ÍNDICE ALFABÉTICO 391

PRÓLOGO

El paradigma de procesamiento de la información desarrollado a finales del siglo XIX y principios del XX que conforman el binomio lógica booleana-máquina de Turing (o, si se prefiere, arquitectura von Neumann), es la base de los actuales sistemas de procesamiento digitales. Sin embargo, y pese a sus indiscutibles logros, este esquema presenta problemas a la hora de abordar tareas como las denominadas del *mundo real*, donde la información se presenta masiva, imprecisa y distorsionada.

Para abordar este tipo de tareas se han propuesto modelos alternativos, de los cuales las redes neuronales artificiales (*artificial neural networks*) y los sistemas basados en lógica borrosa (*fuzzy logic*) son los que cuentan con mayor popularidad y utilización. Estos nuevos modelos de procesamiento y control, junto con algunos otros (como los algoritmos genéticos), se engloban con los términos **inteligencia computacional** (por oposición a la inteligencia artificial clásica) o *soft computing* (por oposición a la *hard computing* convencional, basada en computadores von Neumann). El denominador común de estas nuevas técnicas radica en su inspiración en las soluciones que la naturaleza ha encontrado a lo largo de millones de años de evolución para el tratamiento del tipo de información masiva y distorsionada procedente del entorno natural, soluciones que copiadas en sistemas artificiales se espera que contribuyan a resolver importantes problemas tecnológicos (visión, habla, control de sistemas complejos, inteligencia artificial, etc.).

El trabajo en este tipo de materias, hasta hace poco minoritarias, constituye hoy en día algunos de los esfuerzos más dinámicos dentro del I+D. En este texto nos centraremos especialmente en las dos que actualmente están causando un mayor impacto, debido a su aplicabilidad práctica: las redes neuronales y los sistemas borrosos. Las **redes neuronales artificiales** (*artificial neural networks*), mediante un estilo de computación paralelo, distribuido y adaptativo, son capaces de aprender a partir de ejemplos. Estos sistemas imitan esquemáticamente la estructura hardware (neuronal) del cerebro para tratar de reproducir algunas de sus capacidades. En la

práctica, una red neuronal artificial puede simularse mediante un programa de ordenador, o bien realizarse en circuitos electrónicos específicos.

Por su parte, los **sistemas borrosos o difusos** (*fuzzy systems*) se introducen para manejar eficazmente conceptos vagos e imprecisos como los empleados en la vida cotidiana, y que nuestro cerebro está acostumbrado a tratar. Por ejemplo, en la realidad el agua no se presenta en tan sólo dos estados, *caliente* o *fría*, como diría la lógica booleana, sino más bien *gélida*, *fría*, *templada*, *caliente* o *quemando*. A partir de estos conceptos, y como generalización de las reglas de la lógica booleana, base de nuestros sistemas digitales, los sistemas borrosos llevan a cabo un tipo de razonamiento aproximado semejante al desarrollado por el cerebro.

Ambas tecnologías, neuronales y borrosas, pese a su (relativa) juventud, son ya ampliamente utilizadas en la resolución de tareas relacionadas con el procesamiento de señal, reconocimiento de patrones y control. Por ejemplo, en la actualidad, se emplean redes neuronales en la cancelación de ecos en las señales telefónicas de larga distancia, reconocimiento óptico de caracteres, lectores de cheques, ayuda al pilotaje de aviones o predicción de crisis bancarias. Por otra parte, los sistemas borrosos gobiernan trenes metropolitanos, estabilizan la imagen en cámaras de vídeo, controlan las lavadoras automáticas más modernas, o aconsejan si conceder un crédito o no. En equipos de reciente comercialización se utiliza incluso como reclamo comercial el empleo de *inteligencia artificial* borrosa (*fuzzy*) o neuronal (por ejemplo, cámaras de vídeo, lavadoras, aire acondicionado o software de diseño electrónico).

Por lo tanto, estas nuevas tecnologías permiten incorporar un cierto tipo de *inteligencia* en un sistema de procesamiento y control. Cada día está más extendida la opinión de que sistemas como los neuronales y los borrosos van a desempeñar un papel importante en la construcción de máquinas que emulen la capacidad humana de toma de decisiones en entornos imprecisos y con ruido; las múltiples aplicaciones desarrolladas en la última década así lo avalan.

¿A quién se dirige *Redes Neuronales y Sistemas Difusos*?

Este libro se dirige a todo aquel interesado en iniciarse en ambos campos, especialmente estudiantes, docentes y personal de la empresa. El texto podrá ser seguido por quien cuente con una mínima base matemática (cálculo con matrices, derivadas, etc.), como la manejada en estudios de ciencias, ingenierías o económicas.

¿Qué aporta la segunda edición de *Redes Neuronales y Sistemas Difusos*?

La primera edición del libro (1997) surgió a partir de las notas realizadas para el curso de verano *Introducción a las Redes Neuronales y Lógica Borrosa*, impartido por los autores durante varios años en la **Universidad de Verano de Teruel** (www.unizar.es). El interés que despiertan estas nuevas tecnologías, por un lado, y la carencia de textos en castellano que traten ambos temas a la par, por otro, hicieron que

la primera edición se agotara con cierta rapidez. Por ello nos decidimos a realizar esta **segunda edición**, en la que se ha actualizado parte del texto (especialmente los capítulos 4, 5, 6 y 10) y la bibliografía. Además, se ha incluido un prólogo de Lotfi A. Zadeh (el “padre” de la lógica borrosa), escrito especialmente para esta edición, un apéndice con direcciones de Internet, así como un CD-ROM con las versiones de demostración de dos de los simuladores más interesantes del mercado.

Pensamos que *Redes Neuronales y Sistemas Difusos* es un digna **introducción a estos temas**; en la actualidad está siendo utilizado como libro de texto en la universidad española en diversas asignaturas correspondientes a los planes de estudios oficiales (en carreras de ingeniería, física y economía), así como en cursos de doctorado y especialización. También sabemos que el texto cuenta con una cierta difusión en Sudamérica (pese a la casi permanente crisis que lastra sus economías).

Orientación de *Redes Neuronales y Sistemas Difusos*

En el libro hemos incorporado los conocimientos adquiridos en nuestro trabajo de investigación, docencia y aplicación de estas nuevas tecnologías a problemas reales (predicción de demanda, economía, robótica, etc.), en colaboración con departamentos universitarios y empresas. De la abundante literatura que hemos manejado hemos extraído los aspectos más relevantes, dejando en todo momento claro la procedencia de las distintas fuentes, por lo que aparecen numerosas referencias bibliográficas.

Nuestra intención es facilitar los primeros pasos dentro de este apasionante campo, especialmente **a los interesados en su vertiente práctica**, tratando de huir de triunfalismos gratuitos e intentando ofrecer una visión realista. Por ello, queremos dejar claro desde el principio que las redes neuronales y los sistemas borrosos no son la panacea que resolverá todos los problemas tecnológicos, sino dos potentes herramientas a añadir a las ya existentes (sistemas digitales, inteligencia artificial, estadística, procesamiento de señal, etc.).

Contenido y estructura de *Redes Neuronales y Sistemas Difusos*

En la **Introducción** exponemos una breve perspectiva histórica del intento de construir máquinas y sistemas inteligentes, para presentar lo que aportan en este sentido los dos tópicos que centran nuestra atención en este libro. Tras esta perspectiva, comenzaremos la **Primera Parte**, dedicada a las **Redes Neuronales Artificiales**, en la que en sucesivos capítulos expondremos sus fundamentos, modelos más significativos, forma de implementarlos y su aplicación en la práctica. Aunque se citan muchos modelos neuronales, los que se estudian con mayor profundidad son los siguientes: adaline, perceptrón simple y multicapa (BP), mapas autoorganizados, LVQ, RBF y Hopfield.

En la **Segunda Parte** nos centraremos en los **Sistemas Borrosos**, describiendo sus bases conceptuales, desarrollo de sistemas de control borroso, cómo implementar

aprendizaje en estos sistemas (presentamos aquí las bases de los **Algoritmos Genéticos**), las maneras de realizar en la práctica un sistema borroso y, finalmente, sus aplicaciones. Concluye el libro con un apéndice sobre el contenido del CD-ROM y otro sobre recursos disponibles en Internet, así como con una amplia bibliografía.

El empleo de programas de simulación constituye un excelente apoyo para asimilar la teoría, por ello en esta segunda edición hemos incluido un CD-ROM con las **versiones de demostración** de dos de los entornos de trabajo más interesantes: **Neuro-Solutions**, de NeuroDimension (Gainesville, Florida; www.nd.com), y **fuzzyTECH**, de Inform GmbH (Aachen, Alemania; www.fuzzytech.com).

Agradecimientos

A *Nicolás Medrano, Carlos Serrano, José Barquillas, David Buldáin, Tomás Pollán, Armando Roy y Javier Blasco*, compañeros de la *Universidad de Zaragoza* que han colaborado con nosotros en estos temas.

A las empresas *NeuroDimension* (www.nd.com) e *Inform* (www.fuzzytech.com), por cedernos las versiones demostración de sus excelentes productos software.

A nuestras familias y amigos por su constante apoyo (muy especialmente a nuestras queridas y siempre comprensivas *Cármenes*).

Agradecemos enormemente el interés que *Lotfi A. Zadeh*, "padre" de la lógica *fuzzy*, ha mostrado hacia nuestro trabajo y su visita a nuestro Departamento, así como su aportación al texto materializada en el prefacio que sigue a continuación.

Finalmente, y antes de empezar en materia, justo es recordar que si alguien merece el título de "padre" de las redes neuronales este es sin duda *Santiago Ramón y Cajal*, uno de los genios (como Miguel Servet, Francisco de Goya o Luis Buñuel) que de cuando en cuando surgen en ese espacio existente entre los Pirineos y las sierras de Teruel que ya desde la Edad Media se denomina Aragón. En buena medida se trata de una región subdesértica, partida en dos por el río Ebro, y cuya dureza de clima marca el carácter de sus hijos (aquellos que se empeñan en incluirla en la "España húmeda" deberían darse una vuelta por Monegros, Belchite o el entorno de Zaragoza).

Hace poco más de un siglo *Ramón y Cajal* demostró (y se empeñó en convencer de ello al mundo) que, a diferencia de lo que se pensaba entonces, el sistema nervioso estaba compuesto por células individuales, las neuronas. Este será el punto de partida de nuestro viaje ...

Bonifacio Martín del Brío
Alfredo Sanz Molina

Departamento de Ingeniería Electrónica y Comunicaciones.
Universidad de Zaragoza

Zaragoza, España. Marzo de 2001

INTRODUCCIÓN

En esta introducción realizaremos un breve repaso a la historia del desarrollo de sistemas y máquinas dotadas de cierta inteligencia. A continuación, expondremos más ampliamente las motivaciones que desembocan en la introducción de las redes neuronales artificiales y de los sistemas borrosos, como alternativa o complemento a los sistemas de procesamiento disponibles en la actualidad, basados en el binomio lógica booleana-arquitectura serie von Neumann, en el que se basan nuestros actuales dispositivos de cómputo.

1 EL LARGO Y TORTUOSO CAMINO HACIA LA CONSTRUCCIÓN DE MÁQUINAS INTELIGENTES

Sin remontarnos a épocas históricas, donde los intentos resultaban prematuros en relación a la tecnología disponible, podemos considerar que el camino hacia la construcción de máquinas inteligentes comienza en la Segunda Guerra Mundial [Moravec 88], con el diseño de ordenadores analógicos ideados para controlar cañones antiaéreos o para navegación. Algunos investigadores observaron entonces que existía una semejanza entre el funcionamiento de estos dispositivos de control y los sistemas reguladores de los seres vivos. De este modo, combinando las nuevas teorías sobre la realimentación, los avances de la electrónica de la posguerra y los conocimientos disponibles sobre los sistemas nerviosos de los seres vivos, se construyeron máquinas capaces de responder y de aprender como los animales. Norbert Wiener acuñó el término **cibernética** para designar este estudio unificado del control y de la comunicación en los animales y las máquinas. Ejemplos clásicos desarrollados en los años cincuenta son las tortugas de W. Grey Walter, que exhibían comportamientos sociales, o *La Bestia* de J. Hopkins, que guiada por un sonar y un ojo fotoeléctrico era capaz de encontrar un enchufe para *alimentarse*.

Tras dos décadas de vigencia, la cibernética clásica comenzó su declive, coincidiendo con la primera época de auge de las **redes neuronales**, que surgieron también como un intento de emular estructuras biológicas.

A la par que los ordenadores analógicos y la cibernética se inicia el desarrollo de los ordenadores digitales tal y como los conocemos hoy en día, basados en la separación de estructura y función (hardware y software), corriente que en adelante denominaremos **computación algorítmica**. Un hito de esta tendencia es el año 1937, cuando Alan Turing emprendió el estudio formal de la computación, para lo que introdujo una máquina ideal conocida como máquina de Turing [Conrad 92, Hopcroft 84]. En torno a esta fecha comienza la construcción de los primeros ordenadores electrónicos, como el famoso ENIAC, desarrollado bajo los auspicios del ejército Norteamericano durante la Segunda Guerra Mundial para la ejecución de cálculos de interés militar (trayectorias balísticas, desarrollo de la bomba atómica, etc.). El siguiente paso conceptual importante en esta línea se debe al matemático de origen húngaro John von Neumann, quien en los años cuarenta concibe una computadora basada en lógica digital que opera ejecutando en serie (una tras otra) las instrucciones que componen un algoritmo que se codifica en forma de programa, el cual se encuentra almacenado en memoria. Debido a su eficacia, flexibilidad y versatilidad, y soportado por el impresionante desarrollo de las tecnologías electrónicas, éste ha resultado ser el enfoque dominante en las últimas décadas, de modo que el binomio lógica booleana-máquina Von Neumann es la base sobre la que se asientan la mayor parte de nuestros actuales computadores digitales.

La computación algorítmica se basa, por lo tanto, en resolver cada problema mediante un algoritmo, que se codifica en forma de programa y se almacena en memoria; el programa es ejecutado en una máquina secuencial. Como desarrollo natural de esta tendencia, algunos pioneros, como Turing o von Neumann, abrigaban la esperanza de que pudiera incorporarse en una de estas máquinas la capacidad de pensar racionalmente, en la forma de un complejo software. En este sentido, en 1950 Claude Shannon y el mismo Turing diseñaron los primeros programas que permitían a un ordenador digital *razonar* y jugar al ajedrez. A lo largo de los cincuenta se prosiguió trabajando en este sentido. En 1957 A. Newell, H. Simon y J. Shaw presentaron el Teórico Lógico, el primer programa capaz de razonar sobre temas arbitrarios. Hacia 1960 John McCarthy acuña el término **inteligencia artificial** o IA, para definir los métodos algorítmicos capaces de hacer pensar a los ordenadores. En 1965 Marvin Minsky, Newell y Simon habían creado programas de IA que demostraban teoremas de geometría. Aunque importantes, los desarrollos citados (y otros muchos también basados en la manipulación de información simbólica) solamente eran capaces de resolver aquellos problemas para los que habían sido contruidos. Sin embargo, los resultados eran tan alentadores que a finales de los sesenta los investigadores llegaron a pensar que en una década se conseguiría construir una máquina realmente inteligente.

El extraordinario desarrollo de la IA en la época supuso el eclipse de la cibernética¹ y en buena medida de las redes neuronales. En 1969, Minsky y Papert [Minsky 69] mostraron mediante un estudio riguroso las graves limitaciones de los perceptrones, el modelo neuronal por excelencia de los sesenta, lo que supuso una enorme pérdida de confianza en este nuevo campo, con el resultado de que la mayor parte de los recursos económicos y humanos se desplazaran hacia la IA, abocando a las redes neuronales a una época oscura de la que tardaría más de diez años en salir².

El rápido progreso de la IA culminó en los años setenta con la introducción de los **sistemas expertos**, complejos programas de computador en los que se codifica el conocimiento de expertos en una materia muy concreta (concesión de créditos, diagnóstico de enfermedades, etc.), en forma de reglas de decisión. No obstante, un cuarto de siglo más tarde nuestros ordenadores son miles de veces más potentes que los de la época de los pioneros de la IA y, sin embargo, en general, no resultan mucho más inteligentes. El problema radica en que al binomio lógica booleana-máquina von Neumann sobre el que se asienta la IA, pese a su gran potencia, presenta problemas a la hora de abordar ciertas tareas, como aquellas denominadas *del mundo real*, donde la información que se presenta es masiva, imprecisa y distorsionada. Para abordar este tipo de tareas, desde hace unos años (década de los ochenta, principalmente) se han vuelto a retomar (o han surgido) una serie de paradigmas de cómputo alternativos, como las redes neuronales, los sistemas borrosos, algoritmos genéticos o la computación evolutiva, de los cuales los dos primeros quizás sean los más relevantes y empleados.

Así, el resurgimiento de las redes neuronales, también denominadas sistemas neuronales artificiales o ANS (*Artificial Neural Systems*)³, se debe, por una parte, a la citada dificultad para resolver con la eficiencia deseada problemas como los de visión o aprendizaje. Además, y debido a los altos requerimientos computacionales de este tipo de tareas, la arquitectura von Neumann no es muy apropiada y la introducción de sistemas de cálculo paralelo resulta necesaria si se quiere lograr respuestas en tiempo real [Churchland 90]. Estas nuevas inquietudes coincidieron con el desarrollo de la integración VLSI [Mead 89a, 89b] (que permitió simular y realizar estos sistemas, mostrando su potencial de aplicabilidad práctica), y con el trabajo de investigadores como J. J. Hopfield o T. Kohonen. En particular, Hopfield [Hopfield 82] introdujo nuevos e interesantes puntos de vista en los que se combinaban las redes neuronales con la integración VLSI y los modelos de vidrios de espín (*spin-glass*) de la mecánica

¹ En realidad, se suele pensar en la cibernética como en una disciplina que trató de abordar por primera vez importantes problemas relacionados con la robótica, el control, la comunicación, el modelado biológico, etc., y que desapareció una vez cumplió su papel, dando paso a diversas áreas de trabajo nuevas.

² Minsky, uno de los padres de la IA, suele ser considerado el principal causante de ello, aunque según ha comentado recientemente [Connections 93] no fuese esa su intención. En [Horgan 94] se trata sobre la persona de Marvin Minsky, comentándose algunas de sus opiniones actuales, como por ejemplo, que poco a poco se ha ido apartando de la IA y su aprobación al actual desarrollo de las redes neuronales.

³ Existen otras denominaciones, como redes neuronales artificiales (*Artificial Neural Networks*), neurocomputación (*Neural Computing*) o sistemas conexionistas.

estadística. Por último, se encontró la forma de entrenar un perceptrón multicapa [Rumelhart 86a], con lo que se resolvían los problemas atribuidos al perceptrón simple y se eliminaban las antiguas objeciones a los ANS.

Por otro lado, los sistemas borrosos [Zadeh 65, 73, 88, Kosko 92a] inciden fundamentalmente sobre el otro miembro del binomio, la lógica booleana o digital. En ésta, una sentencia solamente puede ser verdadera o falsa (1 o 0), una cualidad está presente o no lo está, mientras que en el mundo real las cosas no son 1 o 0, verdaderas o falsas. Según el criterio de un ser humano, una habitación no solamente puede estar fría o caliente, sino que también puede definirse como gélida, fría, fresca, templada, caliente o muy caliente. Los sistemas borrosos pueden describirse como un tipo de lógica multivaluada, que permite manejar estos conceptos borrosos o difusos (*fuzzy*) típicos del mundo real, y que emula el tipo de razonamiento que los seres humanos realizamos con ellos haciendo uso de sentencias como *SI la habitación está muy fría ENTONCES pon la calefacción al máximo*. Podemos decir que mientras las redes neuronales artificiales emulan el hardware del cerebro, los sistemas borrosos se ocupan del lado más software.

En definitiva, hoy en día vuelven a coexistir dos corrientes importantes dentro de la búsqueda de la *inteligencia* que son, más que alternativas, **complementarias**. Por una parte la IA *convencional*, basada en algoritmos manipuladores de información simbólica, que se ejecutan sobre ordenadores von Neumann, que operan sobre la base de la lógica digital. Por otra parte, suelen agruparse las redes neuronales, los sistemas borrosos y otras técnicas, que se incluyen en lo que se ha dado en denominar **inteligencia computacional** [Marks 93, IEEE 99] o *soft computing* [Zadeh 94, Jang 97], que en cierta medida imitan las construcciones desarrolladas por la naturaleza. Así, con el término **ABC de la inteligencia** [Marks 93] pueden contemplarse sus tres facetas: artificial, biológica y computacional.

2 MICROPROCESADORES, COMPUTADORES Y CEREBRO

Tras este pequeño recorrido histórico profundizaremos en algunos de los aspectos ya introducidos. Consideremos la siguiente cuestión: *¿Por qué pese al espectacular desarrollo de la electrónica e informática en las últimas décadas existen todavía tareas que nuestras máquinas no han conseguido resolver con suficiente eficacia?* Nos referimos especialmente a aquellas relacionadas con el reconocimiento de patrones y respuesta en entornos imprecisos y con ruido, como es el medio que nos rodea, y que en ocasiones denominaremos mundo real. Ejemplos típicos de tales tareas son el reconocimiento de habla, visión, control motor, etc. Tengamos en cuenta que el hombre todavía no ha podido construir una máquina capaz de reconocer una mosca y atraparla al vuelo, funciones que el relativamente sencillo cerebro de la rana desempeña de una forma sorprendentemente eficaz. Como contrapartida, las máquinas

El extraordinario desarrollo de la IA en la época supuso el eclipse de la cibernética¹ y en buena medida de las redes neuronales. En 1969, Minsky y Papert [Minsky 69] mostraron mediante un estudio riguroso las graves limitaciones de los perceptrones, el modelo neuronal por excelencia de los sesenta, lo que supuso una enorme pérdida de confianza en este nuevo campo, con el resultado de que la mayor parte de los recursos económicos y humanos se desplazaran hacia la IA, abocando a las redes neuronales a una época oscura de la que tardaría más de diez años en salir².

El rápido progreso de la IA culminó en los años setenta con la introducción de los **sistemas expertos**, complejos programas de computador en los que se codifica el conocimiento de expertos en una materia muy concreta (concesión de créditos, diagnóstico de enfermedades, etc.), en forma de reglas de decisión. No obstante, un cuarto de siglo más tarde nuestros ordenadores son miles de veces más potentes que los de la época de los pioneros de la IA y, sin embargo, en general, no resultan mucho más inteligentes. El problema radica en que al binomio lógica booleana-máquina von Neumann sobre el que se asienta la IA, pese a su gran potencia, presenta problemas a la hora de abordar ciertas tareas, como aquellas denominadas *del mundo real*, donde la información que se presenta es masiva, imprecisa y distorsionada. Para abordar este tipo de tareas, desde hace unos años (década de los ochenta, principalmente) se han vuelto a retomar (o han surgido) una serie de paradigmas de cómputo alternativos, como las redes neuronales, los sistemas borrosos, algoritmos genéticos o la computación evolutiva, de los cuales los dos primeros quizás sean los más relevantes y empleados.

Así, el resurgimiento de las redes neuronales, también denominadas sistemas neuronales artificiales o ANS (*Artificial Neural Systems*)³, se debe, por una parte, a la citada dificultad para resolver con la eficiencia deseada problemas como los de visión o aprendizaje. Además, y debido a los altos requerimientos computacionales de este tipo de tareas, la arquitectura von Neumann no es muy apropiada y la introducción de sistemas de cálculo paralelo resulta necesaria si se quiere lograr respuestas en tiempo real [Churchland 90]. Estas nuevas inquietudes coincidieron con el desarrollo de la integración VLSI [Mead 89a, 89b] (que permitió simular y realizar estos sistemas, mostrando su potencial de aplicabilidad práctica), y con el trabajo de investigadores como J. J. Hopfield o T. Kohonen. En particular, Hopfield [Hopfield 82] introdujo nuevos e interesantes puntos de vista en los que se combinaban las redes neuronales con la integración VLSI y los modelos de vidrios de espín (*spin-glass*) de la mecánica

¹ En realidad, se suele pensar en la cibernética como en una disciplina que trató de abordar por primera vez importantes problemas relacionados con la robótica, el control, la comunicación, el modelado biológico, etc., y que desapareció una vez cumplió su papel, dando paso a diversas áreas de trabajo nuevas.

² Minsky, uno de los padres de la IA, suele ser considerado el principal causante de ello, aunque según ha comentado recientemente [Connections 93] no fuese esa su intención. En [Horgan 94] se trata sobre la persona de Marvin Minsky, comentándose algunas de sus opiniones actuales, como por ejemplo, que poco a poco se ha ido apartando de la IA y su aprobación al actual desarrollo de las redes neuronales.

³ Existen otras denominaciones, como redes neuronales artificiales (*Artificial Neural Networks*), neurocomputación (*Neural Computing*) o sistemas conexionistas.

inteligentes construidas por el hombre llevan a cabo otro tipo de acciones, como puedan ser el cálculo y el razonamiento lógico, mucho más eficientemente que el cerebro.

En resumen, parece ser que las tareas que peor llevan a cabo nuestros actuales computadores y robots son precisamente aquellas que más fáciles resultan a los organismos biológicos. Así, aunque es relativamente sencillo conseguir que los ordenadores resuelvan complejos problemas aritméticos o jueguen al ajedrez (pudiendo derrotar incluso al campeón del mundo), resulta sumamente difícil dotarles de la capacidad perceptiva y movilidad de una simple rana.

Para comprender la causa de este hecho, compararemos la arquitectura de las actuales computadores con la estructura que presenta el cerebro, cuyas capacidades pretendemos emular. Los actuales computadores son **máquinas de von Neumann**, en esencia una máquina de procesamiento (hardware) que actúa ejecutando en serie (una tras otra) una secuencia de instrucciones o programa (software), que almacena en su memoria. La máquina está compuesta por cuatro unidades básicas: unidad de entrada de información, unidad de salida, unidad de procesamiento (compuesta a su vez por la unidad lógico-aritmética y la unidad de control), y memoria. Este sistema constituye una máquina universal de cómputo en el sentido de Turing, por lo que puede llevar a cabo, en principio, cualquier tarea sólo con programarla de forma adecuada. Su versatilidad reside en que, sólo con cambiar el programa que almacena en su memoria, es capaz de ejecutar tareas de muy diversa índole (desde cálculos científicos, a edición de textos, control de automatismos, etc.).

La mayor parte de los ordenadores disponibles en la actualidad son máquinas manipuladoras de símbolos que se basan en este esquema. La unidad de entrada de datos es el teclado o las unidades de disco; la unidad de salida es muy a menudo la pantalla o la impresora; la unidad de procesamiento o CPU (*Central Processing Unit*) es el microprocesador; la memoria puede estar distribuida entre el disco duro, cintas, CD-ROM o la memoria central de semiconductor.

El verdadero corazón del computador es el **microprocesador**, potente y complejísimo circuito electrónico, que en la actualidad puede integrar varios millones de componentes electrónicos en un espacio de alrededor de un centímetro cuadrado [Martín del Brío 99]. El microprocesador es considerado en muchas ocasiones como la cumbre de la tecnología humana, tanto por el nivel de desarrollo tecnológico necesario para poder incluir millones de dispositivos electrónicos en tan reducido espacio, como por la potencia de cálculo que pueden desarrollar. Cualquiera de los microprocesadores que contienen nuestros ordenadores personales de sobremesa (Pentium, PowerPC) poseen más potencia que las grandes computadoras de hace tan solo una década. Además, se trata de componentes muy baratos y versátiles, hasta el punto de que en la actualidad la mayor parte de los microprocesadores no se encuentran dentro de los ordenadores, como se podría pensar, sino contenidos en todo equipo que contenga cierta cantidad de electrónica, como puedan ser hornos microondas, aparatos de televisión, vídeos, cámaras fotográficas, cámaras de vídeo,

cadenas de alta fidelidad, lavadoras, etc⁴. Por ejemplo, en la actualidad podemos encontrar en un automóvil más de una decena de microprocesadores que se encargan de realizar importantes tareas de control y seguridad, como por ejemplo en los frenos ABS, encendido electrónico, inyección de gasolina, ordenador de a bordo, bloqueo de puertas, etc. [Martín del Brío 99]. Se puede afirmar que la potencia de cálculo disponible en forma de microprocesadores en, digamos, una lavadora automática, es superior que la que disponían a bordo las naves Apolo (previas al nacimiento del microprocesador) que pusieron al hombre en la Luna.

Retornando al computador, para resolver un determinado problema, en primer lugar debe idearse un algoritmo, que se materializará en la forma de un programa escrito en un lenguaje de programación que el computador entienda (BASIC, Pascal, C), por último, el microprocesador deberá ejecutar el programa. Con un programa suficientemente largo y detallado, y con una memoria de suficiente capacidad, siguiendo este esquema podría resolverse en principio cualquier problema de cómputo, desde encontrar las soluciones de una ecuación a diseñar un edificio, o incluso intentar simular nuestro propio proceso de razonamiento. Como se comentó en la sección anterior, ésta es precisamente la idea de partida de la inteligencia artificial clásica para intentar dotar de algún tipo de comportamiento inteligente a los computadores. Así, si se requiere de un ordenador que diagnostique enfermedades, bastaría en principio con extraer a un médico todo su conocimiento e implantarlo en la forma de un programa (sistema experto). Si necesitamos una máquina que reconozca el habla, bastaría con programar de forma adecuada un algoritmo que a partir de los sonidos que pueda recibir el ordenador obtenga su significado. Es éste un enfoque descendente (desde arriba) hacia la inteligencia, y marcadamente funcionalista, en el sentido de centrarse en conseguir una cierta función entrada-salida, con independencia de la estructura hardware subyacente que la implemente.

Con este esquema de cómputo se progresó rápidamente, tanto en la resolución de problemas prácticos como en la emulación de un cierto comportamiento inteligente, alcanzándose importantes logros. Sin embargo, pronto se comprobó que existían tareas que requerían un esfuerzo computacional tremendo, por lo que su resolución era difícilmente viable siguiendo esta línea [Churchland 90]. Un ejemplo es la simulación del reconocimiento de objetos por el sistema visual, tarea que exige tiempos de cálculo enormes haciendo uso de un computador von Neumann y un programa al efecto. Ello es debido a que el sistema deberá tratar la ingente masa de datos que componen la imagen, y si se precisa una respuesta en tiempo real, incluso grandes computadoras que operen en serie resultarán ineficaces por requerir excesivo tiempo de cálculo. En los problemas de mundo real surge otro inconveniente, pues el ordenador deberá manejar enormes bases de datos. La información proveniente del mundo real es masiva, redundante e imprecisa, mientras que el computador está orientado a tratar datos precisos en serie (uno tras otro).

⁴ Estos microprocesadores que se dedican a tareas de control se denominan **microcontroladores**, integrando en una única pastilla todos los elementos del computador: CPU, memoria, entradas y salidas, etc [Martín del Brío 99].

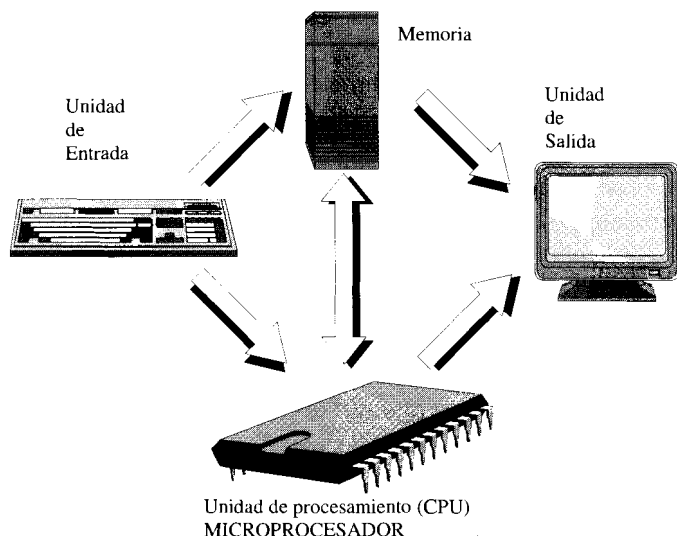


Figura 1. Arquitectura de un computador von Neumann

Parece obvio que para abordar esta clase de tareas debe recurrirse al procesamiento en paralelo. Una solución puede ser el empleo de varias CPU operando a la vez, pero el esfuerzo necesario para construir un programa orientado a una máquina de este tipo es tremendo, y muy dependiente de la estructura concreta de la máquina con la que se trabaje. Además, aparecen importantes cuestiones adicionales, como el espacio que ocupa una máquina de estas características, la energía que consume, y, sobre todo, su elevado coste económico, circunstancias que hacen inviable su aplicación a la construcción de, por ejemplo, un pequeño robot inteligente.

Por el contrario, el **cerebro**, cuyas capacidades queremos emular, no opera de acuerdo al marco descrito. El cerebro no es una arquitectura von Neumann, pues no está formado por un microprocesador (muy rápido y capaz de ejecutar en serie complejas instrucciones de una forma totalmente fiable), ni siquiera está constituido por unas cuantas CPU, sino que lo componen millones de procesadores elementales o neuronas, ampliamente interconectadas conformando redes de neuronas (tabla 1). La neurona es en realidad un pequeño procesador, sencillo, lento y poco fiable (a diferencia de nuestros potentes microprocesadores), sin embargo, en nuestro cerebro cohabitan unos cien mil millones de neuronas operando en paralelo. Es aquí donde reside el origen de su poder de cómputo. Aunque individualmente las neuronas sean capaces de realizar procesamiento muy simples, ampliamente interconectadas a través de las sinapsis (cada neurona puede conectarse con otras 10.000 en promedio) y trabajando en paralelo pueden desarrollar una actividad global de procesamiento enorme.

Por otra parte, las neuronas no deben ser *programadas*, éstas aprenden a partir de las señales que reciben del entorno, y operan siguiendo un esquema también muy diferente al de los computadores convencionales. En este punto nos encontramos por

primera vez con la idea de **autoorganización**. En una red de neuronas no existe un componente que gobierne el sistema (como la CPU), las neuronas se influyen mutuamente a través de sinapsis excitadoras e inhibidoras, lo que causa una compleja dinámica de activaciones y desactivaciones en ellas. Las neuronas, en definitiva, se autoorganizan, aprendiendo del entorno y adaptándose a él, y de esta autoorganización emergen ricas propiedades de procesamiento. Nuestra capacidad de percepción y, en última instancia, nuestro pensamiento, son producto de ello.

En resumen, el cerebro resulta ser un complejo sistema de procesamiento, no lineal, masivamente paralelo, y adaptativo, pero además es extraordinariamente eficiente desde un punto de vista energético. Por ejemplo, el cerebro emplea 10^{-16} Julios para ejecutar una operación por segundo, mientras que los ordenadores actuales emplean unos 10^{-6} [Haykin 99], diez órdenes de magnitud más.

	Cerebro	Computador
Velocidad de proceso	$\approx 10^{-2}$ seg. (100 Hz)	$\approx 10^{-9}$ seg. (1000 MHz)
Estilo de procesamiento	paralelo	secuencial
Número de procesadores	$10^{11} - 10^{14}$	pocos
Conexiones	10.000 por procesador	pocas
Almacenamiento del conocimiento	distribuido	direcciones fijas
Tolerancia a fallos	amplia	nula
Tipo de control del proceso	auto-organizado	centralizado

Tabla 1 Cerebro frente a computador convencional

3 REDES NEURONALES ARTIFICIALES

Así, tenemos que frente al acercamiento descendente hacia la inteligencia representado por los computadores convencionales y por la inteligencia artificial (en definitiva, por las máquinas procesadoras de símbolos [Churchland 90]), el cerebro adopta un enfoque ascendente (emergente), en el que el procesamiento es fruto de la autoorganización de millones de procesadores elementales (véase la tabla 2 y Figura 2). A causa de estos dos diferentes enfoques, existen problemas que resuelven muy bien los ordenadores y otros en los que el cerebro es mucho más eficiente. Los computadores fueron creados por el hombre específicamente para las tareas que podemos denominar de alto nivel, como el razonamiento o el cálculo, que pueden ser fácilmente resolubles mediante el procesamiento de símbolos; en este tipo de tareas nuestro cerebro no es excesivamente diestro, y actúa en clara desventaja frente a la

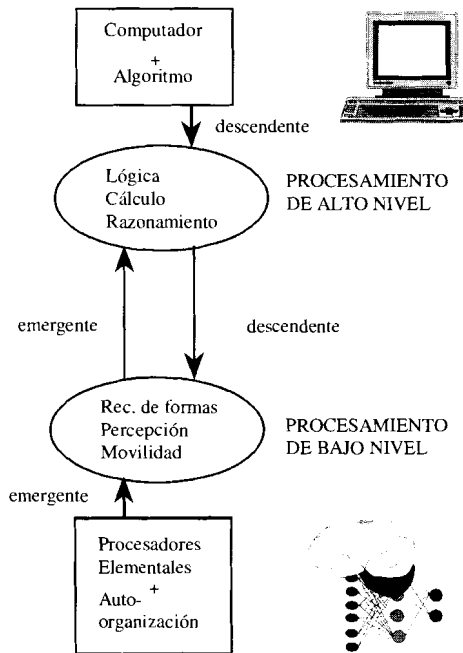
electrónica. Sin embargo, en tareas de procesamiento de bajo nivel, como las de reconocimiento de patrones, percepción, control, etc., los computadores se desenvuelven todavía torpemente, pues en origen no fueron ideados para ello (Figura 2). Para esta clase de problemas, esenciales para la supervivencia de un organismo vivo, la naturaleza encontró una excelente solución: el procesamiento autoorganizado que emerge de la interacción de numerosos procesadores elementales.

Por lo tanto, la idea de partida de las redes neuronales artificiales es que para ejecutar aquel tipo de tareas que más eficazmente resuelve el cerebro, puede resultar interesante copiar su estructura hardware, creando un sistema compuesto por múltiples neuronas ampliamente interconectadas, y estudiar si a partir de su autoorganización pueden reproducirse sus capacidades. Esta ruptura con el funcionalismo de la estructura de cómputo clásica aparece también en otros campos, que se suelen englobar dentro de la denominada computación emergente [Forrest 90], inteligencia computacional [IEEE 99] o *soft computing* [Jang 97] ya citadas.

Las redes neuronales o sistemas neuronales artificiales constituyen en la actualidad un activo campo multidisciplinar, en el que confluyen investigadores procedentes de muy diferentes áreas, como la electrónica, física, matemáticas, ingeniería, biología o psicología. No obstante, hay que tener muy presente que en las redes neuronales artificiales no se persigue ningún tipo de ambición *prometeica*, como las de la IA en sus inicios [Hèrault 94], sino que se utilizan en el control de procesos industriales, reconocimiento de vehículos en los peajes de las autopistas o la previsión del consumo eléctrico, **objetivos mucho más modestos** que la creación de un *cerebro artificial*, y extremadamente útiles desde un punto de vista tecnológico. Es en este tipo de problemas prácticos en los que los sistemas neuronales están alcanzando excelentes resultados.

Inteligencia artificial	Redes neuronales
Enfoque descendente	Enfoque ascendente
Basado en la psicología	Basado en la biología
Qué hace el cerebro	Cómo lo hace el cerebro
Reglas Si/Entonces	Generalización a partir de ejemplos
Sistemas programados	Sistemas entrenados
Lógica, conceptos, reglas	Reconocimiento de patrones, gestalt
Arquitectura von Neumann. Separación hardware/software	Arquitecturas paralelas, distribuidas, adaptativas. Autoorganización

Tabla 2 Características de la IA convencional y de los ANS [McCord 91]

**Figura 2**

Acercamiento ascendente (computador serie más algoritmo) y descendente (autoorganización) a la inteligencia

4 SISTEMAS BORROSOS

Hemos visto que las redes neuronales emulan el hardware del cerebro para reproducir algunas de sus capacidades asociadas a la *inteligencia*, especialmente la que denominamos de bajo nivel, relacionada con el reconocimiento de patrones, percepción, etc. Los sistemas basados en lógica borrosa, sin embargo, puede decirse que se orientan en otra dirección, en la de emular la parte más *software* del cerebro, tratando de reproducir las capacidades de más alto nivel, especialmente la de razonamiento aproximado.

Como ya hemos comentado, en el mundo real las cualidades no aparecen perfectamente definidas, no son 0 o 1, sino que resultan más bien imprecisas, borrosas (por ejemplo, *la habitación está templada*), por lo que puede resultar interesante introducir una lógica que trate de manejar estos conceptos imprecisos, como complemento a la lógica booleana (digital) tradicional. Siguiendo este razonamiento Lotfi Zadeh propuso y desarrolló en Estados Unidos la denominada lógica borrosa (*fuzzy logic*) durante los años sesenta (véase, por ejemplo, [Kosko 92a]). Aunque su trabajo fue recibido con gran frialdad (incluso rechazo) en Norteamérica, debido a que se apartaba totalmente de los muy bien asentados sistemas basados en lógica digital, sí fue muy bien recibido en Japón. En este país desde los años setenta se viene empleando lógica borrosa en el desarrollo de múltiples y variadas aplicaciones, como por ejemplo el control de sistemas de aire acondicionado, pilotaje automático de trenes metropolitanos, etc. En buena parte debido al éxito comercial que los japoneses han

sabido obtener de los sistemas borrosos, desde finales de los años ochenta se vienen realizando importantes esfuerzos de investigación y desarrollo en lógica borrosa tanto en Norteamérica como en Europa.

Como veremos detenidamente en la Segunda Parte del libro, la lógica borrosa asigna términos lingüísticos (borrosos) a propiedades físicas, como por ejemplo, temperatura *gélida*, *fría*, *templada*, etc., y proporciona un marco y unas herramientas para manejar estos conceptos, de forma paralela a como se manejan las variables booleanas que se sitúan únicamente en dos estados, 0 o 1 (*frío* o *caliente*). De hecho, se demuestra que la lógica booleana es en realidad un caso particular de la lógica borrosa.

En definitiva, la lógica borrosa razona a partir de estos términos lingüísticos borrosos, haciendo uso de sentencias del tipo *SI/ENTONCES*, como por ejemplo, *SI temperatura=fría ENTONCES enciende calefacción* o *SI presión=muy alta ENTONCES abrir la válvula*. Estas sentencias borrosas son en realidad representaciones naturales y compactas del conocimiento humano disponible sobre una materia (por ejemplo, el control de la temperatura ambiente de una habitación, el control de una caldera o el de unos frenos).

Esquemáticamente, en el desarrollo de un sistema borroso en primer lugar se proponen las variables lingüísticas (borrosas) que definen el sistema, junto a sus posibles estados y funciones de pertenencia, para formular entonces un conjunto de reglas que definan la operación (en ocasiones bastan unas pocas reglas), de cuya aplicación se infiere una respuesta. El sistema resultante suele ser tan sencillo, y los recursos de cálculo tan reducidos, que todo él puede incorporarse en un pequeñísimo y barato sistema microprocesador. De esta manera, la lógica borrosa permite incorporar de una manera relativamente sencilla y directa el conocimiento de un experto (o, simplemente, conocimientos intuitivos) en un determinado campo, y aplicarlas ejecutando un tipo de razonamiento aproximado a partir de la información imprecisa que suministra el entorno. El sistema experto borroso así desarrollado permite incorporar *inteligencia* en dispositivos de tamaño reducido, como por ejemplo, electrodomésticos.

5 REDES NEURONALES Y SISTEMAS BORROSOS

Tratando de resumir lo expuesto hasta el momento, podemos permitirnos una cierta licencia afirmando que

- a) las redes neuronales emulan el hardware del cerebro.
- b) los sistemas borrosos emulan el software del cerebro.

Ambos temas los desarrollaremos con cierta extensión en las dos partes en las que dividimos el texto. Comenzaremos la exposición con las redes neuronales artificiales, realizando una introducción al tema, exponiendo los modelos de mayor

interés desde un punto de vista práctico, y mostrando cómo se construye un ANS y cómo se desarrollan aplicaciones basadas en ellos. Un esquema similar seguiremos para el caso de los sistemas borrosos.

Al final del libro deberíamos quedarnos, por lo menos, con las siguientes **ideas básicas**: de entre las nuevas técnicas que tratan de complementar los sistemas tradicionales basados en el binomio lógica booleana-arquitectura von Neumann ...

... las redes neuronales artificiales son capaces de descubrir automáticamente relaciones entrada-salida (o rasgos característicos) en función de datos empíricos, merced a su capacidad de aprendizaje a partir de ejemplos.

... los sistemas borrosos permiten emplear el conocimiento disponible por los expertos para el desarrollo de sistemas inteligentes.

... resulta fundamental la integración de diversas técnicas, existiendo, en particular, estrechas relaciones entre redes neuronales y sistemas borrosos (neuro-fuzzy).

En relación con esto último, hay que destacar la clara tendencia actual hacia la fusión de técnicas, neuronales, borrosas y convencionales [Jang 97]. Los problemas de mundo real son lo suficientemente complejos como para que no haya una herramienta única que lo resuelva todo. No nos cansaremos de repetir que no existe una panacea, sino que debe emplearse en cada aspecto del problema aquella técnica que más eficaz resulte.

Cibernética, inteligencia artificial, redes neuronales, lógica borrosa, inteligencia computacional..., todas estas disciplinas tratan el problema de la inteligencia desde puntos de vista, a veces diferentes, en ocasiones paralelos. El proceso hasta lograr el desarrollo de una inteligencia artificial de nivel comparable a la de un ser humano es un largo camino que se prolongará durante muchos años⁵; no obstante, pese a llevarse tan sólo alrededor de cinco décadas en la investigación de la verdadera naturaleza de la computación y de la inteligencia, ya se han alcanzado importantes logros. Parafraseando a C. Frabetti en el prólogo de [Moravec 88], podemos decir que *mientras los filósofos discuten si es posible o no la inteligencia electrónica, los investigadores la construyen*.

⁵ Aunque algunos expertos opinan que hacia el año 2030 las máquinas igualarán la potencia de cálculo de nuestro cerebro [Kurzweil 99], y que en cincuenta años la existencia de robots realmente inteligentes será un hecho [Moravec 88, Kurzweil 99].

FUNDAMENTOS DE LAS REDES NEURONALES ARTIFICIALES

A lo largo de este capítulo expondremos los fundamentos básicos de las redes neuronales artificiales o **ANS** (*Artificial Neural Systems*). Así, describiremos los aspectos esenciales de los ANS, especialmente los relacionados con la estructura de la **neurona artificial** y de la **arquitectura de la red**. Posteriormente expondremos una serie de teoremas generales que proporcionan solidez teórica al campo de la computación neuronal. Por último, comentaremos brevemente las áreas de **aplicación práctica** de las redes neuronales.

El estudio de los ANS puede orientarse en dos direcciones, bien como modelos del sistema nervioso y los fenómenos cognitivos, bien como herramientas para la resolución de problemas prácticos; este último precisamente será el punto de vista que más nos interesará. En este sentido, consideraremos que las redes neuronales artificiales son sistemas, hardware o software, de procesamiento, que copian esquemáticamente la estructura neuronal del cerebro para tratar de reproducir sus capacidades. Los ANS son capaces así de aprender de la experiencia a partir de las señales o datos provenientes del exterior, dentro de un marco de computación paralela y distribuida, fácilmente implementable en dispositivos hardware específicos.

1.1 BREVE INTRODUCCIÓN BIOLÓGICA

Antes de abordar el estudio de los ANS creemos conveniente exponer algunos conceptos básicos de los sistemas neuronales biológicos, para poder establecer más fácilmente el paralelismo existente entre ambos.

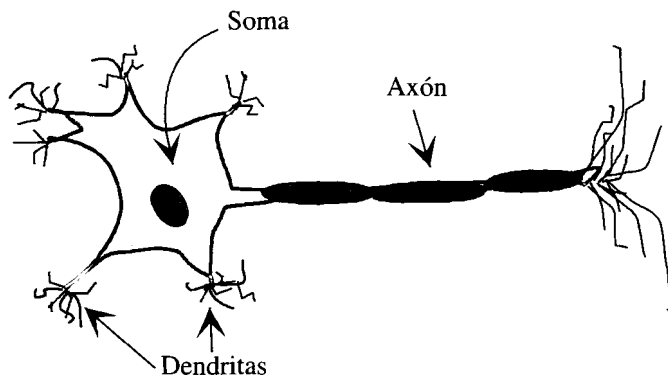


Figura 1.1 Estructura de una neurona biológica típica

La historia de las redes neuronales artificiales comenzaría con el científico aragonés **Santiago Ramón y Cajal** [Enciclopedia 82], descubridor de la estructura neuronal del sistema nervioso. A finales del siglo XIX la teoría reticularista, que sostenía que el sistema nervioso estaba formado por una red continua de fibras nerviosas, era la creencia extendida. Sin embargo, tras años de trabajo aplicando y perfeccionando la técnica de tinción de Golgi, en 1888 Ramón y Cajal demostró que el sistema nervioso en realidad estaba compuesto por una red de células individuales, las neuronas, ampliamente interconectadas entre sí. Pero no sólo observó al microscopio los pequeños espacios vacíos que separaban unas neuronas de otras, sino que también estableció que la información fluye en la neurona desde las dendritas hacia el axón, atravesando el soma. Este descubrimiento, básico para el desarrollo de las neurociencias en el siglo XX, causó en la época una verdadera conmoción en la forma de entender el sistema nervioso, concediéndose el premio Nobel de medicina a Ramón y Cajal en 1906 (compartido con Camilo Golgi). Hoy día el trabajo de Ramón y Cajal [López 93, DeFelipe, 99], el científico español más influyente de la historia, sigue siendo muy citada. En su obra *Textura* [Ramón y Cajal 1899-94], de la que se celebra el centenario [Marijuán 99], realiza un monumental estudio y descripción en detalle del sistema nervioso de los vertebrados (la calidad de sus láminas quizás no haya sido aún superada, pues además de excelente científico, Ramón y Cajal era un gran observador y consumado dibujante y fotógrafo); un siglo después *Textura* se sigue publicando en inglés para la comunidad científica [Ramón y Cajal 1999].

Obviamente, gracias al advenimiento de la microscopía electrónica y a la introducción de otras importantes técnicas, se ha llegado a profundizar mucho más en el estudio de la neurona (véase, por ejemplo, [Kandel 99]). Para el presente trabajo nos limitaremos a un nivel de descripción que nos permita trazar el paralelismo existente entre las redes neuronales biológicas y las artificiales.

Se estima que el sistema nervioso contiene alrededor de cien mil millones de neuronas. Vistas al microscopio, este tipo de células puede presentarse en múltiples

formas, aunque muchas de ellas presentan un aspecto similar muy peculiar (Figura 1.1), con un cuerpo celular o soma (de entre 10 y 80 micras de longitud), del que surge un denso árbol de ramificaciones (árbol dendrítico) compuesto por las dendritas, y del cual parte una fibra tubular denominada axón (cuya longitud varía desde las 100 micras hasta el metro en el caso de las neuronas motoras¹), que también se ramifica en su extremo final para conectar con otras neuronas.

Desde un punto de vista funcional, las neuronas constituyen procesadores de información sencillos. Como todo sistema de este tipo, poseen un canal de entrada de información, las dendritas, un órgano de cómputo, el soma, y un canal de salida, el axón². En las interneuronas el axón envía la información a otras neuronas, mientras que en las neuronas motoras lo hace directamente al músculo. Existe un tercer tipo de neuronas, las receptoras o sensoras, que en vez de recibir la información de otras neuronas, la reciben directamente del exterior (tal sucede, por ejemplo, en los conos y bastones de la retina). Se calcula que una neurona del córtex cerebral recibe información, por término medio, de unas 10.000 neuronas (convergencia), y envía impulsos a varios cientos de ellas (divergencia).

En el córtex cerebral se aprecia la existencia de una organización horizontal en capas (se suelen señalar unas seis capas), coexistiendo una organización vertical en forma de columnas de neuronas. Hay grupos neuronales, compuestos por millones de neuronas pertenecientes a una determinada región del cerebro, que constituyen unidades funcionales especializadas en ciertas tareas (por ejemplo, existe un área visual, un área auditiva, un córtex senso-motor, etc.); todos los subsistemas juntos conforman el encéfalo. Se tiene evidencia de que el procesamiento en el sistema nervioso involucra la actuación de muchos de tales subsistemas, que intercambian continuamente información.

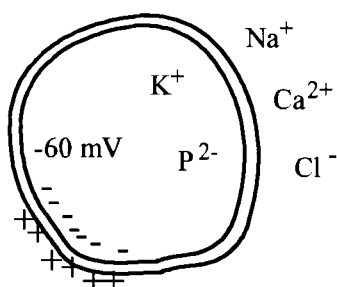


Figura 1.2
Distribución de los iones en la neurona dentro y fuera de la membrana celular

¹ Lo que constituye el nervio.

² Esta visión es algo simplista, aunque válida para nuestro propósito. En realidad, en el árbol dendrítico también se lleva a cabo una cierta computación; por otra parte, el soma también puede recibir información directamente de otros axones, sin la mediación de las dendritas.

Generación y transmisión de la señal nerviosa

La unión entre dos neuronas se denomina sinapsis. En el tipo de sinapsis más común no existe un contacto físico entre las neuronas, sino que éstas permanecen separadas por un pequeño vacío de unas 0.2 micras. En relación a la sinapsis, se habla de neuronas presinápticas (la que envía las señales) y postsinápticas (la que las recibe). Las sinapsis son direccionales, es decir, la información fluye siempre en un único sentido.

Las señales nerviosas se pueden transmitir eléctrica o químicamente. La transmisión química prevalece fuera de la neurona, mientras que la eléctrica lo hace en el interior. La transmisión química se basa en el intercambio de neurotransmisores, mientras que la eléctrica hace uso de descargas que se producen en el cuerpo celular, y que se propagan por el axón.

El fenómeno de la generación de la señal nerviosa está determinado por la membrana neuronal y los iones presentes a ambos lados de ella (Figura 1.2). La membrana se comporta como un condensador, que se carga al recibir corrientes debidas a las especies iónicas presentes. La membrana contiene canales iónicos selectivos al tipo de ión, algunos son pasivos (consisten en simples poros de la membrana) y otros activos (poros que solamente se abren ante ciertas circunstancias). En esencia, las especies iónicas más importantes, que determinan buena parte de la generación y propagación del impulso nervioso, son Na^+ , K^+ y Ca^{2+} , además de los iones de proteínas, que denotaremos genéricamente por P^{2-} , y que se originan por pérdida de los anteriores.

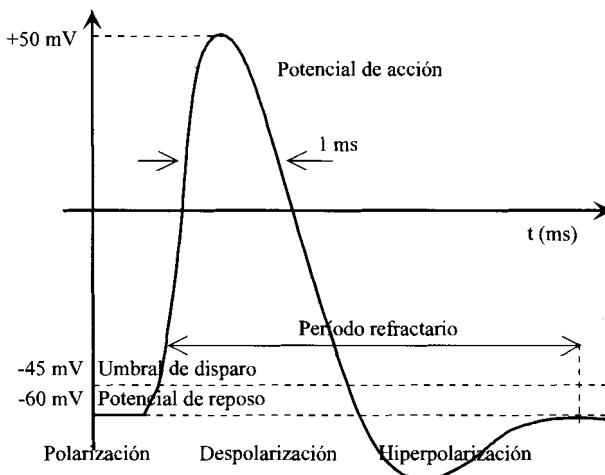


Figura 1.3
Potencial de acción

En estado de reposo el protoplasma del interior de la neurona permanece cargado negativamente en relación al medio externo, existiendo entre ambos una diferencia de potencial de unos -60 mV. La existencia de este potencial de reposo se debe a las concentraciones de Na^+ , K^+ y P^{2-} , y se mantiene mediante el flujo de iones Na^+ y K^+ a través de la membrana. El interior de la neurona está cargado negativamente puesto que, debido a su gran tamaño, los iones P^{2-} quedan dentro, al no poder atravesar la membrana. Los canales de K^+ son pasivos, y se comportan como simples poros. Por su parte, los de Na^+ son activos, y se convierten en permeables a este ión cuando el potencial del soma desciende por debajo de unos -45 mV. Por ello, en condiciones de reposo, la membrana es permeable al K^+ , pero no al Na^+ , y sus concentraciones se generan y mantienen por la acción de la denominada bomba de Na^+-K^+ , que por cada 2 iones K^+ que introduce extrae 3 iones Na^+ al exterior. Este bombeo de iones se realiza a costa de un gasto de energía, de ahí que la neurona sea una célula de alto consumo energético. El resultado final es que la concentración de K^+ y de P^{2-} es alta en su interior, y la de Na^+ lo es en el exterior, siendo la diferencia de potencial debida a sus concentraciones de unos -60 mV. Diferentes modelos eléctricos del comportamiento de la neurona se muestran en [Mead 89a, Kohonen 89] (véase el apéndice al final de este capítulo, página 37).

La forma de comunicación más habitual entre dos neuronas es de tipo químico. La neurona presináptica libera unas sustancias químicas complejas denominadas neurotransmisores (como el glutamato o la adrenalina), que atraviesan el vacío sináptico. Si la neurona postsináptica posee en las dendritas o en el soma canales sensibles a los neurotransmisores liberados, los fijarán, y como consecuencia de ello permitirán el paso de determinados iones a través de la membrana. Las corrientes iónicas que de esta manera se crean provocan pequeños potenciales postsinápticos, excitadores (positivos) o inhibidores (negativos), que se integrarán en el soma, tanto espacial como temporalmente; éste es el origen de la existencia de sinapsis excitatorias y de sinapsis inhibitorias³. Si se ha producido un suficiente número de excitaciones, la suma de los potenciales positivos generados puede elevar el potencial de la neurona por encima de los -45 mV (umbral de disparo): en ese momento se abren bruscamente los canales de sodio, de modo que los iones Na^+ , cuya concentración en el exterior es alta, entran masivamente al interior, provocando la despolarización brusca de la neurona, que pasa de un potencial de reposo de -60 mV a unos $+50$ mV. A continuación la neurona vuelve a la situación original de reposo de -60 mV; este proceso constituye la generación de un potencial de acción (Figura 1.3), que al propagarse a lo largo del axón da lugar a la transmisión eléctrica de la señal nerviosa. Tras haber sido provocado un potencial de acción, la neurona sufre un período refractario, durante el cual no puede generarse uno nuevo.

Un hecho importante es que el pulso así generado es "digital", en el sentido de que existe o no existe pulso, y todos ellos son de la misma magnitud. Por otra parte,

³ Existen evidencias experimentales que indican que un axón sólo puede generar sinapsis excitatorias o inhibitorias, pero no de ambos tipos (ley de Dale, [Müller 90])

ante una estimulación más intensa disminuye el intervalo entre pulsos, por lo que la neurona se disparará a mayor frecuencia cuanto mayor sea el nivel de excitación. Es decir, la excitación queda codificada en la frecuencia de los pulsos producidos. Por otra parte, la frecuencia de disparo de la neurona no puede crecer indefinidamente, sino que existe una frecuencia máxima de respuesta debida a la existencia del período refractario. En resumen, ante un estímulo mayor la frecuencia de respuesta aumenta, hasta que se alcanza una saturación conforme nos acercamos a la frecuencia máxima. De este modo, la función de respuesta de la neurona, frecuencia de disparo frente a intensidad de estimulación, tiene el aspecto mostrado en la Figura 1.4, que se emulará en muchos de los modelos de neurona artificial. La frecuencia de disparo oscila habitualmente entre 1 y 100 pulsos por segundo, aunque algunas neuronas pueden llegar a los 500 durante pequeños períodos de tiempo. Por otra parte, no todas las neuronas se disparan generando un tren de pulsos de una frecuencia aproximadamente constante, pues la presencia de otras especies iónicas hace que diferentes tipos de neuronas posean patrones de disparo distintos, en forma de trenes puros, paquetes de pulsos, o presentando patrones más complejos (Figura 1.5) [Shepherd 97].

Generado un pulso eléctrico por el soma, el transporte activo que se produce a lo largo del axón permite que pueda transmitirse a grandes distancias (hasta un metro) sin degradarse. En los extremos del axón existen unas pequeñas vesículas sinápticas que almacenan paquetes de neurotransmisores; así, ante la aparición de un pulso eléctrico proveniente del cuerpo celular, y por mediación de los iones Ca^{2+} , se produce la liberación de neurotransmisores en cantidades cuantificadas (correspondientes a un número entero de vesículas). El número de pulsos que llegan y su frecuencia determinan la cantidad de neurotransmisor liberado, que a su vez producirá nuevas excitaciones o inhibiciones en otras neuronas.

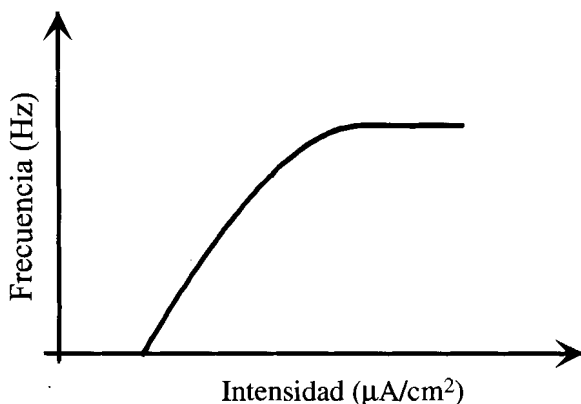


Figura 1.4
Función de respuesta de la neurona biológica ante estímulos del exterior
[Chapman 66]



Figura 1.5 Patrones de disparo: a) regular, b) en paquete de pulsos en una neurona piramidal del córtex, c) disparo de una célula de Purkinje del cerebelo [Shepherd 97]

El mecanismo aquí descrito constituye la forma más común de transmisión de la señal nerviosa, pero no el único. Cuando la distancia que debe recorrer la señal es menor de 1 mm la neurona puede no codificarla en frecuencia, sino enviar una señal puramente analógica. Es decir, la evolución biológica encontró que a distancias cortas la señal no se degradaba sustancialmente, por lo que podía enviarse tal cual, mientras que a distancias largas era preciso codificarla para evitar su degradación y la consiguiente pérdida de información. La naturaleza descubrió que la codificación en forma de frecuencia de pulsos digitales proporcionaba calidad, seguridad y simplicidad en la transmisión.

Aprendizaje

La intensidad de una sinapsis no viene representada por una cantidad fija, sino que puede ser modulada en una escala temporal mucho más amplia que la del disparo de las neuronas (horas, días o meses). Esta plasticidad sináptica se supone que constituye, al menos en buena medida, el aprendizaje [Shepherd 97, Arbib 98] tal y como postuló [Hebb 49], encontrándose posteriormente evidencias experimentales de ello [Nieto 89, Alkon 89, Kandel 92].

Durante el desarrollo de un ser vivo, el cerebro se modela, de forma que existen muchas cualidades del individuo que no son innatas, sino que se adquieren por la influencia de la información que del medio externo proporciona sus sensores. Existen diferentes formas de modelar el sistema nervioso: por el establecimiento de nuevas conexiones, ruptura de otras, modelado de las intensidades sinápticas (plasticidad) o incluso mediante muerte neuronal⁴. Este tipo de acciones (en especial la modificación de las intensidades sinápticas) serán las que utilicen los sistemas neuronales artificiales para llevar a cabo el aprendizaje.

⁴ La neurona es una célula muy especial, que, en general, únicamente posee capacidad para reproducirse en los primeros estadios de su vida, de modo que si una neurona muere, no nacerá otra que la reemplace (aunque recientemente se han encontrado evidencias de que en situaciones especiales sí podría reproducirse).

El esquema presentado a lo largo de esta sección ha sido simplificado en buena medida, pues la realidad es mucho más compleja. Citaremos como muestra unos cuantos ejemplos: hay más iones actuando que los citados, no hemos comentado el papel de los neuropéptidos (que realizan una función paralela a la de los neurotransmisores), no hemos discutido los detalles de cómo el impulso nervioso se propaga por el axón, existen muchísimos tipos de canales que operan de muy diferentes maneras, las sinapsis descritas son las químicas, pero también existen eléctricas, mixtas y recíprocas, etc. Por otra parte, no está claro todavía el papel que puedan desempeñar las células de la glía que, aunque habitualmente se describen como simples soportes de las neuronas (en el cerebro hay 10 células de glía por cada neurona, llenando los espacios interneuronales), parece ser que también intervienen en los procesos de memoria y aprendizaje [Travis 94, Nieto 89]. Tampoco hemos comentado otras formas diferentes de codificar la información en la señal nerviosa, como puedan ser el código en población, el probabilístico o el código en fases.

La bibliografía existente en este campo es inmensa, remitimos al lector interesado en profundizar en los aspectos comentados a, por ejemplo, [Kandel 99, Shepherd 97, Arbib 98]. El sistema nervioso es lo suficientemente complejo como para que todavía quede mucho por descubrir y comprender. Para mantenerse al tanto del progreso en la investigación, son recomendables los artículos divulgativos (aunque de nivel elevado) que aparecen periódicamente en la revista *Scientific American* (cuya versión española es *Investigación y Ciencia*). Por ejemplo, dos excelentes números monográficos sobre el tema son [Scientific 79] y [Scientific 92].

1.2 ESTRUCTURA DE UN SISTEMA NEURONAL ARTIFICIAL

Cerebro y computador

Los ANS imitan la estructura hardware del sistema nervioso, con la intención de construir sistemas de procesamiento de la información paralelos, distribuidos y adaptativos, que puedan presentar un cierto comportamiento "inteligente".

Recordemos una idea ya comentada en la Introducción: pese al extraordinario desarrollo de la electrónica y las ciencias de la computación, ni el ordenador más potente puede llevar a cabo tareas tales como reconocer una mosca y atraparla al vuelo, que un sistema tan simple como el cerebro de la rana es capaz de llevar a cabo con eficacia.

Como pudo apreciarse en la tabla 1.1, el cerebro y un computador convencional son mucho más diferentes de lo que suele suponerse cuando se habla de "cerebros electrónicos". Recordemos que un computador convencional es, en esencia, una máquina de von Neumann, construida en torno a una única CPU o procesador, que

ejecuta de un modo secuencial un programa almacenado en memoria. Por el contrario, el cerebro no está compuesto por un único procesador, sino por miles de millones de ellos (neuronas), aunque muy elementales. Curiosamente, las neuronas son mucho más simples, lentas y menos fiables que una CPU, y a pesar de ello, existen problemas difícilmente abordables mediante un computador convencional, que el cerebro resuelve eficazmente (reconocimiento del habla, visión de objetos inmersos en ambiente natural, respuesta ante estímulos del entorno, etc.).

Por lo tanto, la idea que subyace en los sistemas neuronales artificiales es que, para abordar el tipo de problemas que el cerebro resuelve con eficiencia, puede resultar conveniente construir sistemas que "copien" en cierto modo la estructura de las redes neuronales biológicas con el fin de alcanzar una funcionalidad similar.

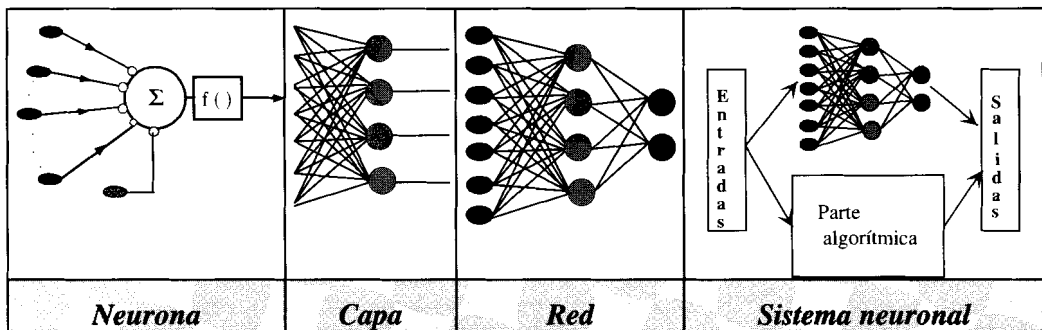


Figura 1.6 Estructura jerárquica de un sistema basado en ANS

Sistemas paralelos, distribuidos y adaptativos

Los tres conceptos clave de los sistemas nerviosos, que se pretende emular en los artificiales, son: paralelismo de cálculo, memoria distribuida y adaptabilidad, al entorno. De esta manera, podemos hablar de las redes neuronales como sistemas paralelos, distribuidos y adaptativos.

El **procesamiento paralelo** resulta esencial, como se deduce de un sencillo ejemplo. Un ordenador convencional tipo PC, que trabaja secuencialmente, instrucción a instrucción, emplearía varios minutos en realizar sobre una imagen compuesta por, digamos, 256x256 píxeles⁵, una sencilla tarea de tratamiento en bajo nivel (acentuar contrastes, extraer contornos, ...), mucho más simple que la que llevaba a cabo el sistema visual para reconocer una imagen. Un sistema basado en 16 DSP⁶ (por ejemplo, del modelo TMS32020, clásico DSP de Texas Instruments) operando en

⁵ Contracción de *picture element*, elemento de imagen.

⁶ *Digital Signal Processor*, o procesador digital de señales, un tipo de microprocesador especializado en la realización de cálculos matemáticos intensivos, como los que se emplean en el campo del procesamiento digital de señal.

paralelo emplearía del orden de 20 ms en la misma tarea, puesto que cada uno podría operar en paralelo sobre diferentes sectores de la imagen. Por otra parte, el cerebro tarda aproximadamente este mismo tiempo en preprocesar una imagen compuesta por millones de píxeles (los que representan los conos y bastones de la retina), extraer sus rasgos característicos, analizarla, e interpretarla. Ningún sistema creado por el hombre es capaz de realizar algo semejante. La clave reside en que en este último caso los miles de millones de neuronas que intervienen en el proceso de visión (solamente en la retina, y sin contar el córtex cerebral, intervienen millones de ellas) están operando en paralelo sobre la totalidad de la imagen.

Otro concepto importante que aparece en el cerebro es el de **memoria distribuida**. Mientras que en un computador la información ocupa posiciones de memoria bien definidas, en los sistemas neuronales se encuentra distribuida por las sinapsis de la red, de modo que si una sinapsis resulta dañada, no perdemos más que una parte muy pequeña de la información. Además, los sistemas neuronales biológicos son redundantes, de modo que muchas neuronas y sinapsis pueden realizar un papel similar; en definitiva, el sistema resulta tolerante a fallos (por ejemplo, cada día mueren miles de neuronas en nuestro cerebro, y sin embargo tienen que pasar muchos años para que se resientan nuestras capacidades).

El último concepto fundamental es el de **adaptabilidad**. Los ANS se adaptan fácilmente al entorno modificando sus sinapsis (y mediante otros mecanismos también), y aprenden de la experiencia, pudiendo generalizar conceptos a partir de casos particulares. En el campo de las redes neuronales llamaremos a esta propiedad **generalización a partir de ejemplos**.

Estructura de un sistema neuronal artificial

Los elementos básicos de un sistema neuronal biológico son las neuronas, que se agrupan en conjuntos compuestos por millones de ellas organizadas en capas, constituyendo un sistema con funcionalidad propia. Un conjunto de estos subsistemas da lugar a un sistema global (el sistema nervioso, en el caso biológico). En la realización de un sistema neuronal artificial puede establecerse una estructura jerárquica similar. El elemento esencial de partida será la neurona artificial, que se organizará en capas; varias capas constituirán una red neuronal; y, por último, una red neuronal (o un conjunto de ellas), junto con las interfaces de entrada y salida, más los módulos convencionales adicionales necesarios, constituirán el sistema global de proceso (Figura 1.6).

Formalmente, y desde el punto de vista del grupo PDP (*Parallel Distributed Processing Research Group*, de la Universidad de California en San Diego⁷), de D. E.

⁷ Grupo de investigación en ANS, responsables en gran medida del renacimiento de las redes neuronales a mediados de los ochenta, cuyo trabajo se publicó en dos volúmenes considerados clásicos [Rumelhart 86a, MacClelland 86].

Rumelhart y J. L. McClelland [Rumelhart 86a, McClelland 86], un **sistema neuronal o conexionista**, está compuesto por los siguientes elementos:

- Un conjunto de procesadores elementales o neuronas artificiales.
- Un patrón de conectividad o arquitectura.
- Una dinámica de activaciones.
- Una regla o dinámica de aprendizaje.
- El entorno donde opera.

Debido a que investigadores de numerosas áreas del conocimiento (neurobiólogos, psicólogos, matemáticos, físicos, ingenieros, etc.) trabajan en ANS, y a causa también de la relativa juventud de esta disciplina, coexisten diferentes terminologías y nomenclaturas. En este sentido, se están llevando a cabo esfuerzos encaminados a unificar conceptos, como los del comité de estandarización de la *Neural Network Society* del IEEE, o los de algunos investigadores [Fiesler 94].

1.3 MODELO DE NEURONA ARTIFICIAL

En esta sección se expone el modelo de neurona de los ANS. En primer lugar, describiremos la estructura de una neurona artificial muy genérica, para a continuación mostrar una versión simplificada, de amplio uso en los modelos orientados a aplicaciones prácticas, que posee una estructura más próxima a la neurona tipo McCulloch-Pitts [McCulloch 43] clásica.

Aunque el comportamiento de algunos sistemas neuronales biológicos sea lineal, como sucede en la retina del cangrejo *Limulus* [Brodie 78], en general, la respuesta de las neuronas biológicas es de tipo **no lineal**, característica que es emulada en los ANS ya desde la neurona formal original de McCulloch-Pitts. La formulación de la neurona artificial como dispositivo no lineal constituye una de sus características más destacables, y una de las que proporciona un mayor interés a los ANS, pues el tratamiento de problemas altamente no lineales no suele ser fácil de abordar mediante técnicas convencionales.

1.3.1 Modelo general de neurona artificial

En este punto describiremos la estructura genérica de neurona artificial en el marco establecido por el grupo PDP [Rumelhart 86a, McClelland 86].

Se denomina **procesador elemental o neurona** a un dispositivo simple de cálculo que, a partir de un vector de entrada procedente del exterior o de otras neuronas, proporciona una única respuesta o salida. Los elementos que constituyen la neurona de etiqueta *i* son los siguientes (véase la Figura 1.7):

- Conjunto de **entradas**, $x_j(t)$.
- **Pesos sinápticos** de la neurona i , w_{ij} que representan la intensidad de interacción entre cada neurona presináptica j y la neurona postsináptica i .
- **Regla de propagación** $\sigma(w_{ij}, x_j(t))$, que proporciona el valor del potencial postsináptico $h_i(t) = \sigma(w_{ij}, x_j(t))$ de la neurona i en función de sus pesos y entradas.
- **Función de activación** $f_i(a_i(t-1), h_i(t))$, que proporciona el estado de activación actual $a_i(t) = f_i(a_i(t-1), h_i(t))$ de la neurona i , en función de su estado anterior $a_i(t-1)$ y de su potencial postsináptico actual.
- **Función de salida** $F_i(a_i(t))$, que proporciona la salida actual $y_i(t) = F_i(a_i(t))$ de la neurona i en función de su estado de activación.

De este modo, la operación de la neurona i puede expresarse como

$$y_i(t) = F_i\left(f_i\left[a_i(t-1), \sigma_i(w_{ij}, x_j(t))\right]\right) \quad (1.1)$$

Este modelo de neurona formal se inspira en la operación de la biológica, en el sentido de integrar una serie de entradas y proporcionar cierta respuesta, que se propaga por el axón. En el apéndice 1.A mostraremos un poco más detalladamente la conexión entre el modelo de neurona artificial y la biológica.

Pasaremos a continuación a describir con mayor profundidad los conceptos introducidos.

Entradas y salidas

Las variables de entrada y salida pueden ser binarias (digitales) o continuas (analógicas), dependiendo del modelo y aplicación. Por ejemplo, un perceptrón multicapa o MLP (*Multilayer Perceptron*) admite ambos tipos de señales. Así, para tareas de clasificación poseería salidas digitales $\{0, +1\}$, mientras que para un problema de ajuste funcional de una aplicación multivariable continua, se utilizarían salidas continuas pertenecientes a un cierto intervalo.

Dependiendo del tipo de salida, las neuronas suelen recibir nombres específicos [Müller 90]. Así, las neuronas estándar (sección 1.2.3) cuya salida sólo puede tomar los valores 0 o 1 se suelen denominar genéricamente **neuronas de tipo McCulloch-Pitts**, mientras que aquellas que únicamente pueden tener por salidas -1 o +1 se suelen denominar **neuronas tipo Ising** (debido al paralelismo con los modelos físicos de partículas con espín que pueden adoptar únicamente dos estados, hacia arriba y hacia abajo). Si puede adoptar diversos valores discretos en la salida (por ejemplo, -2, -1, 0, +1, +2), se dice que se trata de una **neurona de tipo Potts**. En ocasiones, el rango de los valores que una neurona de salida continua puede proporcionar se suele limitar a un intervalo definido, por ejemplo, $[0, +1]$ o $[-1, +1]$.

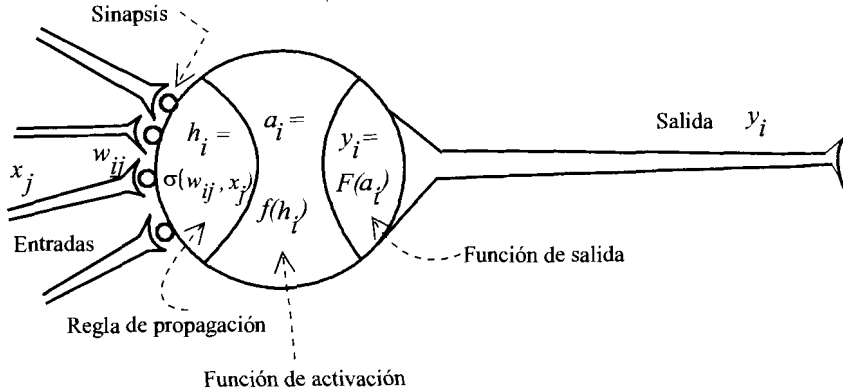


Figura 1.7 Modelo genérico de neurona artificial [Rumelhart 86a]

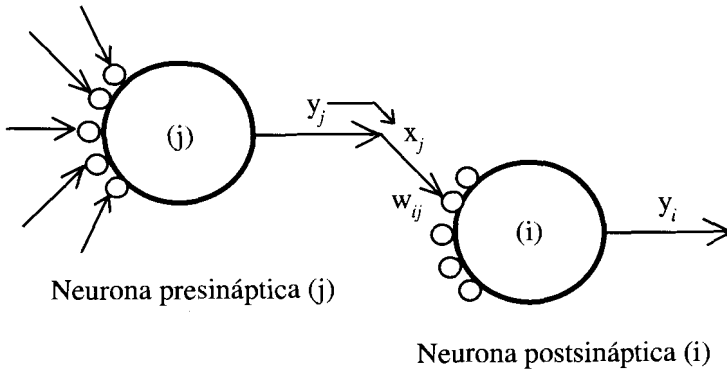


Figura 1.8
Interacción entre
una neurona
presináptica y otra
postsináptica

Regla de propagación

La **regla de propagación** permite obtener, a partir de las entradas y los pesos, el valor del potencial postsináptico h_i de la neurona

$$h_i(t) = \sigma_i(w_{ij}, x_j(t)) \quad (1.2)$$

La función más habitual es de tipo lineal, y se basa en la **suma ponderada** de las entradas con los pesos sinápticos

$$h_i(t) = \sum_j w_{ij} x_j \quad (1.3)$$

que formalmente también puede interpretarse como el producto escalar de los vectores de entrada y pesos

$$h_i(t) = \sum_j w_{ij} x_j = \mathbf{w}_i^T \cdot \mathbf{x} \quad (1.4)$$

El **peso sináptico** w_{ij} define en este caso la intensidad de interacción entre la neurona presináptica j y la postsináptica i . Dada una entrada positiva (procedente de un sensor o simplemente la salida de otra neurona), si el peso es positivo tenderá a excitar a la neurona postsináptica, si el peso es negativo tenderá a inhibirla. Así se habla de sinapsis excitadoras (de peso positivo) e inhibidoras (de peso negativo).

Una regla de tipo no lineal, de uso más limitado, es la siguiente:

$$h_i(t) = \sum_{j_1 j_2 \dots j_p} w_{i j_1 j_2 \dots j_p} x_{j_1} x_{j_2} \dots x_{j_p} \quad (1.5)$$

que implica una interacción de tipo multiplicativo entre las entradas de la neurona (como se ha observado realmente en determinadas sinapsis biológicas). El uso de esta última regla de propagación determina que una neurona se denomine de **orden superior** o neurona **sigma-pi** [Rumelhart 86] (por emplear sumas y productos), e implica una mayor complejidad, tanto en el estudio de la dinámica de la red neuronal, como en su realización hardware.

Otra regla de propagación habitual, especialmente en los modelos de ANS basados en el cálculo de distancias entre vectores (como RBF, mapas de Kohonen o LVQ), es la **distancia euclídea**

$$h_i^2(t) = \sum_j (x_j - w_{ij})^2 \quad (1.6)$$

que representa la distancia (al cuadrado) existente entre el vector de entradas y el de pesos. Cuando ambos vectores son muy similares, la distancia es muy pequeña; cuando son muy diferentes, la distancia crece. Por tanto, este tipo de regla opera de manera diferente a las anteriormente comentadas. Se pueden utilizar también otros tipos de distancia, como la de Manhattan o la de Mahalanobis, que se abordarán más adelante.

Separar el concepto de regla de propagación y función de activación permite considerar desde un punto de vista unificado muchos modelos que de otra manera habría que tratar como casos especiales de una neurona estándar, que definiremos más adelante (tal como sucede, por ejemplo, en el RBF o los mapas autoorganizados).

Función de activación o función de transferencia

La función de activación o de transferencia proporciona el estado de activación actual $a_i(t)$ a partir del potencial postsináptico $h_i(t)$ y del propio estado de activación anterior $a_i(t-1)$

$$a_i(t) = f_i(a_i(t-1), h_i(t)) \quad (1.7)$$

Sin embargo, en muchos modelos de ANS se considera que el estado actual de la neurona no depende de su estado anterior, sino únicamente del actual

$$a_i(t) = f_i(h_i(t)) \quad (1.8)$$

La función de activación $f(\cdot)$ se suele considerar determinista, y en la mayor parte de los modelos es monótona creciente y continua, como se observa habitualmente en las neuronas biológicas. La forma $y = f(x)$ de las funciones de activación más empleadas en los ANS se muestra en la tabla 1.1. Para abreviar, en ella designamos con x al potencial postsináptico, y con y el estado de activación. La más simple de todas es la función identidad (que se puede generalizar al caso de una función lineal cualquiera), empleada, por ejemplo, en la Adalina. Otro caso también muy simple es la función escalón, empleada en el Perceptrón Simple y en la red de Hopfield discreta, así como en la neurona clásica de McCulloch-Pitts. La función lineal a tramos se puede considerar como una lineal saturada en sus extremos, es de gran sencillez computacional y resulta más plausible desde un punto de vista biológico pues, como se ha explicado, las neuronas se activan más a mayor excitación, hasta saturarse a la máxima respuesta que pueden proporcionar.

En ocasiones los algoritmos de aprendizaje requieren que la función de activación cumpla la condición de ser derivable. Las más empleadas en este sentido son las funciones de tipo sigmoideo, como la del BP. Otra función clásica es la gaussiana, que se utiliza junto con reglas de propagación que involucran el cálculo de cuadrados de distancias (por ejemplo, la euclídea) entre los vectores de entradas y pesos. Por último, en ocasiones se emplean funciones sinusoidales, como en aquellos casos en los que se requiere expresar explícitamente una periodicidad temporal.

Función de salida

Esta función proporciona la salida global de la neurona $y_i(t)$ en función de su estado de activación actual $a_i(t)$. Muy frecuentemente la función de salida es simplemente la identidad $F(x)=x$, de modo que el estado de activación de la neurona se considera como la propia salida

$$y_i(t) = F_i(a_i(t)) = a_i(t) \quad (1.9)$$

Esto ocurre en los modelos más comunes, como el MLP o la adalina. La función de salida puede ser también de tipo escalón, lo que supone que la neurona no se dispare hasta que la activación supere un cierto umbral. En otros modelos, como es el caso de la **máquina de Boltzmann** [Hinton 86], se trata de una función estocástica de la activación, con lo que la neurona tendrá un comportamiento probabilístico.

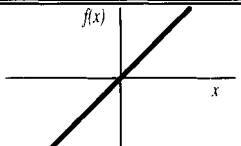
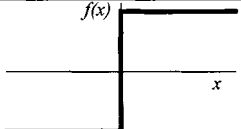
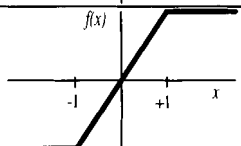
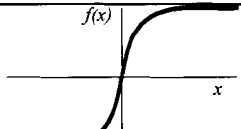
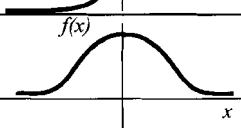
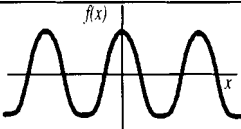
	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = \text{signo}(x)$ $y = H(x)$	$\{-1, +1\}$ $\{0, +1\}$	
Lineal a tramos	$y = \begin{cases} -1, & \text{si } x < -l \\ x, & \text{si } -l \leq x \leq +l \\ +1, & \text{si } x > +l \end{cases}$	$[-1, +1]$	
Sigmoidea	$y = \frac{1}{1 + e^{-x}}$ $y = \text{tgh}(x)$	$[0, +1]$ $[-1, +1]$	
Gaussiana	$y = A \cdot e^{-Bx^2}$	$[0, +1]$	
Sinusoidal	$y = A \cdot \text{sen}(\omega x + \varphi)$	$[-1, +1]$	

Tabla 1.1 Funciones de activación habituales (se han omitido algunas constantes)

1.3.2 Modelo estándar de neurona artificial

El modelo de neurona expuesto en la sección anterior resulta muy general. En la práctica suele utilizarse uno más simple, que denominaremos neurona estándar, que constituye un caso particular del modelo del PDP, considerando que la regla de propagación es la suma ponderada y que la función de salida es la identidad. De esta forma, la **neurona estándar** consiste en

- Un conjunto de **entradas** $x_j(t)$ y pesos sinápticos w_{ij} .
- Una **regla de propagación** $h_i(t) = \sigma(w_{ij}, x_j(t))$; $h_i(t) = \sum w_{ij}x_j$ es la más común.
- Una **función de activación** $y_i(t) = f_i(h_i(t))$, que representa simultáneamente la salida de la neurona y su estado de activación.

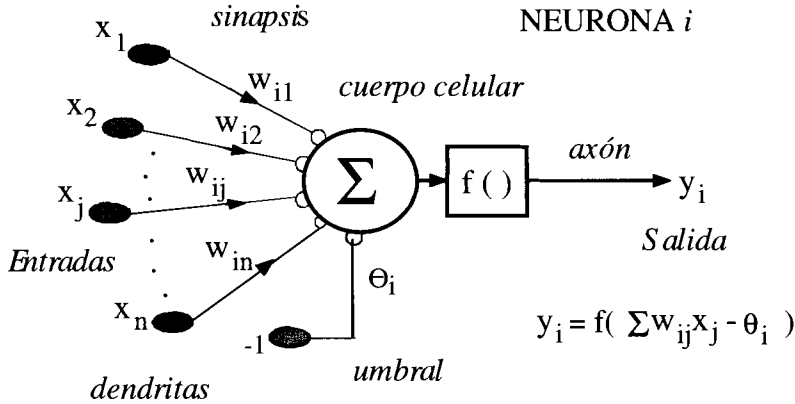


Figura 1.9 Modelo de neurona estándar

Con frecuencia se añade al conjunto de pesos de la neurona un parámetro adicional θ_i , que denominaremos umbral⁸, que se resta del potencial postsináptico, por lo que el argumento de la función de activación queda

$$\sum_j w_{ij}x_j - \theta_i \quad (1.10)$$

lo que representa añadir un grado de libertad adicional a la neurona. Veremos que en el caso de nodos de respuesta todo-nada este parámetro representará el umbral de disparo de la neurona, es decir, el nivel mínimo que debe alcanzar el potencial postsináptico (o potencial de membrana) para que la neurona se dispare o active.

En conclusión, el modelo de neurona que denominaremos estándar queda

$$y_i(t) = f_i\left(\sum_j w_{ij}x_j - \theta_i\right) \quad (1.11)$$

Ahora bien, si hacemos que los índices i y j comiencen en 0, podemos definir $w_{i0} \equiv \theta_i$ y $x_0 \equiv -1$ (constante), con lo que el potencial postsináptico (potencial local, o de membrana) se obtiene realizando la suma desde $j=0$

$$y_i(t) = f_i\left(\sum_{j=0}^n w_{ij}x_j\right) \quad (1.12)$$

Definida de esta manera la neurona estándar, basta con establecer la forma de la función de activación (tabla 1.1) para determinarla por completo. En los siguientes apartados mostraremos ejemplos de los modelos de neurona más habituales.

⁸ En algunos modelos es *threshold*, es decir, “umbral”, y en otros *bias*, que no tiene una traducción clara en este caso, motivo por el que siempre llamaremos umbral a este parámetro (aunque a veces no haga el papel de umbral).

Neuronas todo-nada (dispositivos de umbral)

Si en el modelo de neurona estándar consideramos que las entradas son digitales, por ejemplo $x_i = \{0, 1\}$, y la función de activación es la escalón $H(\cdot)$ (denominada también de Heaviside), definida entre 0 y 1, se tiene

$$y_i(t) = H\left(\sum_j w_{ij}x_j - \theta_i\right) \quad (1.13)$$

Como $H(x)=1$ cuando $x \geq 0$, y $H(x)=0$ cuando $x < 0$, se tiene

$$y_i = \begin{cases} 1, & \text{si } \sum w_{ij}x_j \geq \theta_i \\ 0, & \text{si } \sum w_{ij}x_j < \theta_i \end{cases} \quad (1.14)$$

Es decir, si el potencial de membrana supera un valor umbral θ_i (umbral de disparo), entonces la neurona se activa, si no lo supera, la neurona no se activa. Este es el modelo de **neurona del perceptrón** original, como se verá más adelante, que se denomina en ocasiones **dispositivo de tipo umbral**, existiendo una lógica definida a partir de elementos de esta clase.

Si en este modelo de neurona consideramos que la acción de las entradas inhibitoras es absoluta (ante la presencia de una sola señal inhibitora, la neurona ya no se dispara), y se introducen en el modelo retardos en la propagación de las señales, se obtiene el modelo original de **neurona de McCulloch-Pitts** [McCulloch 43], que introdujeron ambos investigadores en los años cuarenta, el cual se considera como el primer intento de modelado de la operación de la neurona.

Ambos autores demostraron ya en 1943 que mediante redes basadas en este modelo de neurona se podía realizar cualquier función lógica. Por ejemplo, considerando la neurona del perceptrón, y dada la configuración de pesos de la Figura 1.10, se tiene

$$y = H(w_1x_1 + w_2x_2 - \theta) = H(-2x_1 - 2x_2 + 3) \quad (1.15)$$

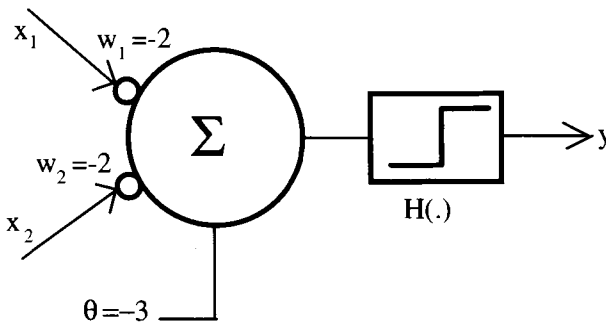


Figura 1.10
Neurona que implementa
una puerta lógica NAND
de dos entradas

Si se dan valores binarios a x_1 y x_2 , el lector puede construir fácilmente la llamada tabla de verdad de esta neurona y comprobar que implementa la función lógica NAND_2 . Hablando en términos generales, puede demostrarse que un nodo de tipo umbral solamente puede implementar funciones separables linealmente, como la NAND . Por ejemplo, la XOR (OR exclusivo) no es linealmente separable, por lo que no puede ser implementada por un nodo sencillo como el anterior. Para lograrlo pueden adoptarse varias soluciones, como introducir realimentación, o emplear estructuras multicapa, pero no profundizaremos en ello por el momento, pues supone adelantarnos a temas que serán ampliamente tratados en el capítulo siguiente.

Neurona continua sigmoidea

Si en el esquema de neurona estándar consideramos que las entradas puedan ser tanto digitales como continuas (analógicas), y las salidas exclusivamente continuas, puede emplearse como función de activación una sigmoidea (cuya gráfica tiene forma de letra 'S' inclinada y aplastada, tabla 1.1), que es una función continua y diferenciable en cierto intervalo, por ejemplo, en el $[-1, +1]$ o en el $[0, +1]$, dependiendo de la función concreta que elijamos. Las dos funciones más habituales de este tipo son las siguientes (tabla 1.1):

$$y = f(x) = \frac{1}{1 + e^{-x}}, \text{ con } y \in [0, 1] \quad (1.16)$$

$$y = f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \text{tgh}(x), \text{ con } y \in [-1, 1] \quad (1.17)$$

Este modelo de neurona es el utilizado en el perceptrón multicapa, como veremos más adelante. El requisito de trabajar con funciones diferenciables lo puede imponer la regla de aprendizaje, como sucede con la famosa BP (*backpropagation*).

Neurona estocástica (probabilística)

El modelo de neurona sigmoideo anterior puede interpretarse desde un punto de vista probabilístico, con lo que su operación deja de ser determinista. En la siguiente sección estudiaremos un ejemplo de modelo de neurona de este tipo (página 25).

1.4 ARQUITECTURAS DE REDES NEURONALES

Definiciones básicas. Tipos de arquitecturas

Se denomina arquitectura a la topología, estructura o patrón de conexionado de una red neuronal. En un ANS los nodos se conectan por medio de sinapsis, esta estructura de conexiones sinápticas determina el comportamiento de la red. Las

conexiones sinápticas son direccionales, es decir, la información solamente puede propagarse en un único sentido (desde la neurona presináptica a la postsináptica, Figura 1.8). En general, las neuronas se suelen agrupar en unidades estructurales que denominaremos **capas**. Las neuronas de una capa pueden agruparse, a su vez, formando **grupos neuronales** (*clusters*). Dentro de un grupo, o de una capa si no existe este tipo de agrupación, las neuronas suelen ser del mismo tipo. Finalmente, el conjunto de una o más capas constituye la **red neuronal**.

Se distinguen tres tipos de capas: de entrada, de salida y ocultas. Una **capa de entrada** o sensorial está compuesta por neuronas que reciben datos o señales procedentes del entorno (por ejemplo, proporcionados por sensores). Una **capa de salida** es aquella cuyas neuronas proporcionan la respuesta de la red neuronal (sus neuronas pueden estar conectadas a efectores). Una **capa oculta** es aquella que no tiene una conexión directa con el entorno, es decir, que no se conecta directamente ni a órganos sensores ni a efectores. Este tipo de capa proporciona a la red neuronal grados de libertad adicionales, gracias a los cuales puede encontrar representaciones internas correspondientes a determinados rasgos del entorno, proporcionando una mayor riqueza computacional.

Las conexiones entre las neuronas pueden ser excitatorias o inhibitorias: un peso sináptico negativo define una **conexión inhibitoria**, mientras que uno positivo determina una **conexión excitatoria**. Habitualmente, no se suele definir una conexión como de un tipo o de otro, sino que por medio del aprendizaje se obtiene un valor para el peso, que incluye signo y magnitud.

Por otra parte, se puede distinguir entre conexiones intra-capas e inter-capas. Las **conexiones intra-capas**, también denominadas **laterales**, tienen lugar entre las neuronas pertenecientes a una misma capa, mientras que las **conexiones inter-capas** se producen entre las neuronas de diferentes capas. Existen además **conexiones realimentadas**, que tienen un sentido contrario al de entrada-salida. En algunos casos puede existir realimentación incluso de una neurona consigo misma.

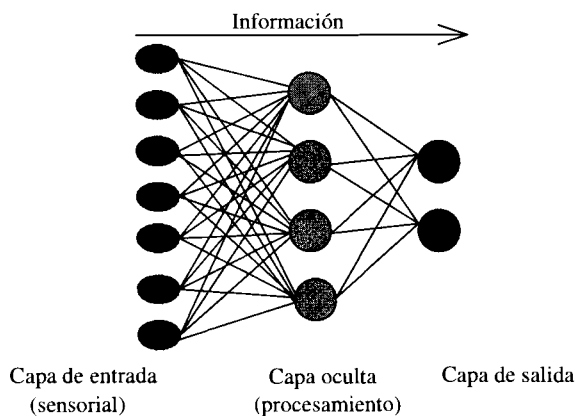
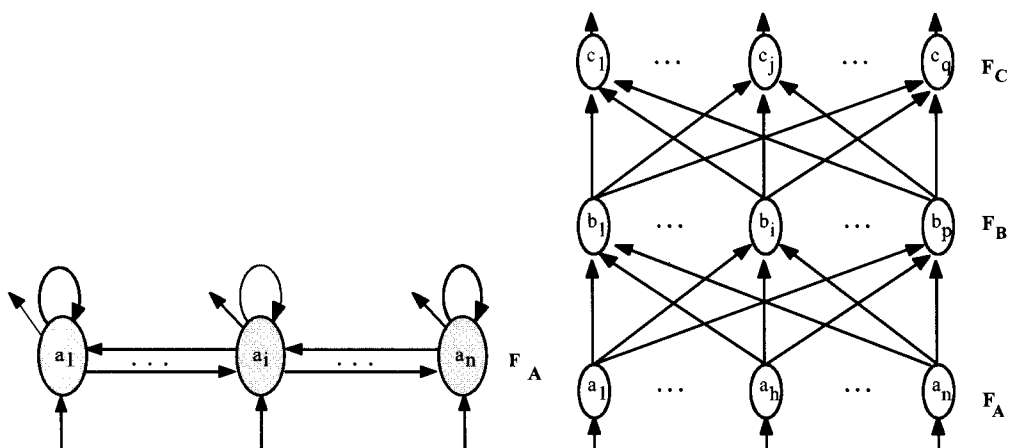


Figura 1.11
Arquitectura unidireccional de tres capas, de entrada, oculta y de salida



Monocapa y realimentada

Multicapa y unidireccional

Figura 1.12 Ejemplos de arquitecturas neuronales

Atendiendo a distintos conceptos, pueden establecerse diferentes tipos de arquitecturas neuronales (Figura 1.12). Así, en relación a su estructura en capas, podemos hablar de redes monocapa y de redes multicapa. Las **redes monocapa** son aquellas compuestas por una única capa de neuronas. Las **redes multicapa** (*layered networks*) son aquellas cuyas neuronas se organizan en varias capas.

Asimismo, atendiendo al flujo de datos en la red neuronal, podemos hablar de redes unidireccionales (*feedforward*) y redes recurrentes (*feedback*). En las **redes unidireccionales**, la información circula en un único sentido, desde las neuronas de entrada hacia las de salida. En las **redes recurrentes o realimentadas** la información puede circular entre las capas en cualquier sentido, incluido el de salida-entrada.

Por último, también se habla de redes autoasociativas y heteroasociativas. Con frecuencia se interpreta la operación de una red neuronal como la de una **memoria asociativa**, que ante un determinado patrón de entradas responde con un cierto patrón de salida. Si una red se entrena para que ante la presentación de un patrón **A** responda con otro diferente **B**, se dice que la red es **heteroasociativa**. Si una red es entrenada para que asocie un patrón **A** consigo mismo, se dice que es **autoasociativa** (el interés de este tipo de redes, como es el caso de la de Hopfield [Hopfield 82], reside en que ante la presentación del patrón $A' = A + \text{ruido}$, su respuesta sea el patrón original **A**, eliminando así el ruido presente en la señal de entrada).

Una definición formal de red neuronal

En el punto anterior se han introducido de una manera intuitiva diversos conceptos sobre las arquitecturas neuronales, con claros paralelismos a los utilizados

en las redes neuronales biológicas. Lógicamente, dichos conceptos pueden ser definidos de una manera más rigurosa desde un punto de vista matemático.

Una definición interesante de red neuronal hace uso del concepto matemático de **grafo**, objeto consistente en un conjunto de nodos (o vértices), más un conjunto de conexiones (o *links*) establecidas entre ellos. En este caso el grafo describe la arquitectura del sistema y proporciona los canales por los que puede discurrir su dinámica. Hay diferentes tipos de grafos, por ejemplo, grafos dirigidos (*directed*) y no dirigidos (*undirected*). En el primer tipo, las conexiones tienen asignado un sentido, mientras que en el otro son bidireccionales. Puede hablarse también de grafos densos (cuando casi todos los nodos están conectados con casi todos) y de grafos dispersos (cuando son pocas las conexiones entre los nodos). Un grafo puede componerse de diferentes tipos de nodos y diferentes tipos de conexiones.

Una forma de representar el grafo es, como su propio nombre indica, gráficamente, dibujando los nodos como círculos y las conexiones como líneas o flechas, según sean de un solo sentido o bidireccionales. Otra forma común de representación es mediante una matriz de conexiones. En el caso en que el grafo sea no dirigido, la matriz de conexiones será simétrica. Una tercera manera es mediante una lista de conexiones, que indican la manera en que los nodos se conectan entre sí (por ejemplo, si a , b y c son nodos, su lista de conexiones podría ser $a \rightarrow a$, $a \rightarrow b$, $a \rightarrow c$, $c \rightarrow b$). Por último, un grafo puede ser también representado mediante un algoritmo.

Una posible definición matemática de red neuronal utilizando el concepto de grafo toma la siguiente forma [Müller 90]:

Definición: una **red neuronal** es un grafo dirigido, con las siguientes propiedades:

- 1) A cada nodo i se asocia una variable de estado x_i .
- 2) A cada conexión (i,j) de los nodos i y j se asocia un peso $w_{ij} \in \mathfrak{R}$.
- 3) A cada nodo i se asocia un umbral θ_i .
- 4) Para cada nodo i se define una función $f_i(x_j, w_{ij}, \theta_i)$, que depende de los pesos de sus conexiones, del umbral y de los estados de los nodos j a él conectados. Esta función proporciona el nuevo estado del nodo.

#

En la terminología habitual de las redes neuronales, los nodos son las neuronas y las conexiones son las sinapsis. Algunos de los conceptos ya expuestos, quedarían redefinidos en este contexto de la siguiente manera:

- Se denomina **neurona de entrada** a las neuronas sin sinapsis entrantes.
- Se denomina **neurona de salida** a las neuronas sin sinapsis salientes.
- Las que no son ni de entrada ni de salida se denominan **neuronas ocultas**.

- Una red es **unidireccional** cuando no presenta bucles cerrados de conexiones.
- Una red es **recurrente** cuando el flujo de información puede encontrar un bucle de atrás hacia adelante, es decir, una realimentación.

Dinámica de la actualización del estado de las neuronas

En este punto estudiaremos la forma en la que las neuronas de una cierta red actualizan sus estados. Existen dos dinámicas fundamentales: **síncrona** y **asíncrona**. En los modelos con **dinámica síncrona**, los estados se actualizan en función de un cierto reloj común. Habitualmente, el proceso se realiza por capas, lo que significa que todas las neuronas pertenecientes a una misma capa se actualizan a la vez, comenzando desde la capa de entrada y continuando hasta la de salida. Ésta es la dinámica más habitual.

En los modelos con **dinámica asíncrona** no existe un reloj común, de manera que cada neurona actualiza su estado sin atender a cuándo lo hacen las demás. En general, una dinámica asíncrona se corresponde con neuronas de respuesta continua. Es éste el tipo de dinámica presente en los sistemas neuronales biológicos.

Ambas dinámicas aplicadas sobre una misma red neuronal y para un mismo patrón de entrada, pueden proporcionar diferentes resultados. Un ejemplo clásico es el modelo de Hopfield discreto (capítulo 4), sobre el que puede definirse una dinámica síncrona o asíncrona [Bruck 90]. En el caso de la dinámica asíncrona (dinámica de Glauber), si la matriz de pesos de la red es simétrica, la red siempre converge a un cierto estado estable. Sin embargo, si sobre esa misma red aplicamos una dinámica síncrona (dinámica de Little), la red puede, o bien converger a un estado estable, o bien permanecer en un ciclo límite de longitud dos.

Asimismo, puede introducirse una **dinámica no determinista** (estocástica) forzando que la salida de la neurona posea carácter probabilístico. Por ejemplo, si consideramos neuronas de activación sigmoidea, la salida de la neurona i es

$$y_i(t+1) = f(h_i(t)) = \frac{1}{1 + e^{-h_i(t)}} \quad (1.18)$$

que pertenece al rango $[0, +1]$; esta neurona es determinista. Ahora bien, si consideramos neuronas de salida discreta $\{0, +1\}$, podemos interpretar el valor proporcionado por (1.18) como la probabilidad de que su salida sea $+1$, es decir

$$p[y_i(t+1) = +1] = \frac{1}{1 + e^{-h_i(t)}} \quad (1.19)$$

así, hemos introducido una dinámica probabilística en la operación de la red neuronal. La estructura de **neurona probabilística** y de salida discreta que acabamos de describir es el utilizado en modelos neuronales como el denominado **máquina de Boltzmann** [Hinton 86].

1.5 MODOS DE OPERACIÓN: RECUERDO Y APRENDIZAJE

Clásicamente se distinguen dos modos de operación en los sistemas neuronales: el modo recuerdo o ejecución, y el modo aprendizaje o entrenamiento. Este último es de particular interés, pues una característica fundamental de los ANS es que se trata de sistemas entrenables, capaces de realizar un determinado tipo de procesamiento o cómputo aprendiéndolo a partir de un conjunto de patrones de aprendizaje o ejemplos.

Fase de aprendizaje. Convergencia

En el contexto de las redes neuronales puede definirse el **aprendizaje** como el proceso por el que se produce el ajuste de los parámetros libres de la red a partir de un proceso de estimulación por el entorno que rodea la red. El tipo de aprendizaje vendrá determinado por la forma en la que dichos parámetros son adaptados. En la mayor parte de las ocasiones el aprendizaje consiste simplemente en determinar un conjunto de pesos sinápticos que permita a la red realizar correctamente el tipo de procesamiento deseado.

Cuando se construye un sistema neuronal, se parte de un cierto modelo de neurona y de una determinada arquitectura de red, estableciéndose los pesos sinápticos iniciales como nulos o aleatorios. Para que la red resulte operativa es necesario entrenarla, lo que constituye el **modo aprendizaje**. El entrenamiento o aprendizaje se puede llevar a cabo a dos niveles. El más convencional es el de modelado de las sinapsis, que consiste en modificar los pesos sinápticos siguiendo una cierta regla de aprendizaje, construida normalmente a partir de la optimización de una función de error o coste, que mide la eficacia actual de la operación la red. Si denominamos $w_{ij}(t)$ al peso que conecta la neurona presináptica j con la postsináptica i en la iteración t , el algoritmo de aprendizaje, en función de las señales que en el instante t llegan procedentes del entorno, proporcionará el valor $\Delta w_{ij}(t)$ que da la modificación que se debe incorporar en dicho peso, el cual quedará actualizado de la forma

$$\Delta w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

El proceso de aprendizaje es usualmente **iterativo**, actualizándose los pesos de la manera anterior, una y otra vez, hasta que la red neuronal alcanza el rendimiento deseado.

Algunos modelos neuronales incluyen otro nivel en el aprendizaje, la creación o destrucción de neuronas, en el cual se modifica la propia arquitectura de la red. En cualquier caso, en un proceso de aprendizaje la información contenida en los datos de entrada queda incorporada en la propia estructura de la red neuronal, la cual almacena la representación de una cierta imagen de su entorno.

Los dos tipos básicos de aprendizaje son el supervisado y el no supervisado, cuya distinción proviene en origen del campo del reconocimiento de patrones. Ambas

modalidades pretenden estimar funciones entrada/salida multivariable o densidades de probabilidad, pero mientras que en el aprendizaje supervisado se proporciona cierta información sobre estas funciones (como la distribución de las clases, etiquetas de los patrones de entrada o salidas asociadas a cada patrón), en el autoorganizado no se proporciona información alguna. Las reglas de aprendizaje supervisadas suelen ser computacionalmente más complejas, pero también más exactos sus resultados.

Además de las dos formas básicas anteriores pueden distinguirse muchas otras [Haykin 99]; nosotros destacaremos aquí el aprendizaje híbrido y el reforzado. Los cuatro tipos citados pueden definirse de la forma siguiente:

a) Aprendizaje supervisado. Sea $E[W]$ un funcional que representa el error esperado de la operación de la red, expresado en función de sus pesos sinápticos W . En el aprendizaje supervisado se pretende estimar una cierta función multivariable desconocida $f: \mathcal{R}^n \rightarrow \mathcal{R}^m$ (la que representa la red neuronal) a partir de muestras $(\mathbf{x}, \mathbf{y})$ ($\mathbf{x} \in \mathcal{R}^n$, $\mathbf{y} \in \mathcal{R}^m$) tomadas aleatoriamente, por medio de la minimización iterativa de $E[W]$ mediante aproximación estocástica⁹.

Hablando en términos menos formales, en el aprendizaje supervisado se presenta a la red un conjunto de patrones, junto con la salida deseada u objetivo, e iterativamente ésta ajusta sus pesos hasta que su salida tiende a ser la deseada, utilizando para ello información detallada del error que comete en cada paso. De este modo, la red es capaz de estimar relaciones entrada/salida sin necesidad de proponer una cierta forma funcional de partida. Veremos más adelante que determinados modelos de red neuronal de relativa simplicidad (como el MLP) son estimadores universales de funciones.

b) Aprendizaje no supervisado o autoorganizado. El aprendizaje no supervisado se puede describir genéricamente como la estimación de la función densidad de probabilidad $p(\mathbf{x})$ que describe la distribución de patrones $\mathbf{x}$ pertenecientes al espacio de entrada $\mathcal{R}^n$ a partir de muestras (ejemplos).

En este tipo de aprendizaje se presentan a la red multitud de patrones sin adjuntar la respuesta que deseamos. La red, por medio de la regla de aprendizaje, estima $p(\mathbf{x})$, a partir de lo cual pueden reconocerse regularidades en el conjunto de entradas, extraer rasgos, o agrupar patrones según su similitud (clustering). Un ejemplo típico de modelo que emplea este tipo de aprendizaje es el de los mapas autoorganizados.

⁹ Las técnicas de aproximación estocástica estiman valores esperados a partir de cantidades aleatorias observadas. Usualmente se implementan en forma de algoritmo discreto del tipo de descenso por el gradiente (estocástico) [Kohonen 89, White 89b].

c) **Aprendizaje híbrido.** En este caso, coexisten en la red los dos tipos básicos de aprendizaje, el supervisado y el no supervisado, los cuales tienen lugar normalmente en distintas capas de neuronas. El modelo de contra-propagación y las RBF son ejemplos de redes que hacen uso de este tipo de aprendizaje.

d) **Aprendizaje reforzado** (*reinforcement learning*). Se sitúa a medio camino entre el supervisado y el autoorganizado. Como en el primero de los citados, se emplea información sobre el error cometido, pero en este caso existe una única señal de error, que representa un índice global del rendimiento de la red (solamente le indicamos lo bien o lo mal que está actuando, pero sin proporcionar más detalles). Como en el caso del no supervisado, no se suministra explícitamente la salida deseada. En ocasiones se denomina **aprendizaje por premio-castigo**.

Muchos de los algoritmos de aprendizaje (aunque no todos) se basan en métodos numéricos iterativos que tratan de minimizar una función coste, lo que puede dar lugar en ocasiones a problemas en la convergencia del algoritmo. Estos aspectos no pueden abordarse de un modo general, sino que deben ser estudiados para cada algoritmo concreto. En un sentido riguroso, la convergencia es una manera de comprobar si una determinada arquitectura, junto a su regla de aprendizaje, es capaz de resolver un problema, pues el grado de error que se mide durante el proceso de aprendizaje describe la precisión del ajuste del *mapping*.

En el proceso de entrenamiento es importante distinguir entre el nivel de error alcanzado al final de la fase de aprendizaje para el conjunto de datos de entrenamiento, y el error que la red ya entrenada comete ante patrones no utilizados en el aprendizaje, lo cual mide la capacidad de **generalización** de la red. Interesa más una buena generalización que un error muy pequeño en el entrenamiento, pues ello indicará que la red ha capturado correctamente el *mapping* subyacente en los datos. El problema de la generalización resulta fundamental en la resolución de problemas con ANS, por lo que se abordará más adelante con mayor amplitud.

Fase de recuerdo o ejecución. Estabilidad

Generalmente (aunque no en todos los modelos), una vez que el sistema ha sido entrenado, el aprendizaje "se desconecta", por lo que los pesos y la estructura quedan fijos, estando la red neuronal ya dispuesta para procesar datos. Este modo de operación se denomina **modo recuerdo** (*recall*) o **de ejecución**.

En las redes unidireccionales, ante un patrón de entrada, las neuronas responden proporcionando directamente la salida del sistema. Al no existir bucles de realimentación no existe ningún problema en relación con su estabilidad. Por el contrario, las redes con realimentación son sistemas dinámicos no lineales, que requieren ciertas condiciones para que su respuesta acabe convergiendo a un estado estable o punto fijo. Una serie de teoremas generales (Cohen-Grossberg [Cohen 83], Cohen-Grossberg-Kosko [Kosko 92a] y otros; véase, por ejemplo, [Simpson 89,

Haykin 99]) indican las condiciones que aseguran la estabilidad de la respuesta en una amplia gama de redes neuronales, bajo determinadas condiciones.

El lector no interesado en los detalles técnicos puede obviar lo que sigue a continuación y saltar directamente a la sección 1.6. Para demostrar la estabilidad del sistema, estos teoremas se basan en el **método de Lyapunov** [Simpson 89], como alternativa al mucho más tedioso método directo, consistente en integrar el sistema de ecuaciones diferenciales que lo describen.

Básicamente [Simpson 89], el método de Lyapunov establece que si en un sistema dinámico (como pueda ser una red neuronal) de variables de entrada $(x_1, x_2, \dots, x_n)$ y descrito por el siguiente sistema de ecuaciones diferenciales

$$\dot{x}_i \equiv \frac{dx_i}{dt} = F(t, x_1, x_2, \dots, x_n) \quad (1.20)$$

se cumplen las condiciones

- a) el sistema está en reposo solamente en el origen;
- b) existen las derivadas de las ecuaciones que lo describen en todo el dominio;
- c) las variables están acotadas;

y se puede encontrar una **función de Lyapunov** V de las variables x_i , $V: \mathfrak{R}^n \rightarrow \mathfrak{R}$, tal que

$$\dot{V} = \sum_{i=1}^n \frac{\partial V}{\partial x_i} \dot{x}_i \leq 0, \quad \forall \dot{x}_i \quad (1.21)$$

entonces el sistema converge para todas las posibles entradas $(x_1, x_2, \dots, x_n)$, y es globalmente estable.

La función de Lyapunov se denomina frecuentemente función **energía de Lyapunov**, pues constituye una generalización del concepto físico de energía.

El método de Lyapunov constituye una manera asequible de estudiar la estabilidad de un sistema dinámico. Es interesante observar que con esta formulación matemática simplemente se está expresando que si somos capaces de encontrar una cierta función energía del sistema, que disminuya siempre en su operación, entonces el sistema es estable. Una técnica similar empleó Hopfield [Hopfield 82, 84] para demostrar que su modelo de red completamente interconectada era estable en el caso de que la matriz de pesos sinápticos fuese simétrica y de diagonal nula (capítulo 4).

Esta técnica es también la que Cohen, Grossberg y Kosko han aplicado en los teoremas citados para demostrar la estabilidad de una amplia clase de redes neuronales realimentadas, autoasociativas y heteroasociativas. Así, el teorema de Cohen-Grossberg [Cohen 83], determina las condiciones de estabilidad para redes autoasociativas no adaptativas; el de Cohen-Grossberg-Kosko [Kosko 88], establece

las condiciones de estabilidad para redes autoasociativas adaptativas; y, por último, el teorema ABAM de Kosko [Kosko 92a], lo hace para redes adaptativas heteroasociativas (como el BAM de Kosko [Kosko 92a]). Como ilustración, enunciaremos el teorema de Cohen-Grossberg:

Teorema de Cohen-Grossberg. Para cualquier sistema dinámico no lineal que se pueda describir de la forma siguiente

$$\dot{x}_i = \alpha_i(x_i) \left[\beta_i(x_i) - \sum_{j=1}^n m_{ji} S_j(x_j) \right] \quad (1.22)$$

tal que

- a) la matriz $\|m_{ij}\|$ es simétrica y $m_{ij} \geq 0, \forall i, j$;
- b) la función $\alpha_i(\xi)$ es continua $\forall \xi \geq 0$;
- c) $\alpha_i(\xi) \geq 0, \forall \xi \geq 0; S_i(\xi) \geq 0, \forall \xi \geq 0$;
- d) la función $S_i(\xi)$ es diferenciable y no decreciente $\forall \xi \geq 0$;

la función

$$V = (1/2) \sum_{i=1}^n \sum_{j=1}^n m_{ij} S_i(x_i) S_j(x_j) - \sum_{i=1}^n \int_0^{x_i} S_i'(\theta_i) \beta(\theta_i) d\theta_i \quad (1.23)$$

es una función energía de Lyapunov para el sistema, y el sistema es estable.

#

Corolario. Si interpretamos la ecuación (1.22) como la descripción de la activación en el tiempo de una red neuronal autoasociativa de una o más capas, no adaptativa, con matriz de pesos simétrica, y que cumple las condiciones a), b), c) y d), entonces dicha red es estable.

#

Remitimos al lector a las referencias originales ya citadas para el enunciado de los demás teoremas y sus nada fáciles demostraciones. En [Simpson 89] y [Kosko 92a] se ofrecen perspectivas generales sobre el método de Lyapunov.

1.6 CLASIFICACIÓN DE LOS MODELOS NEURONALES

A partir de lo visto hasta el momento puede deducirse que dependiendo del modelo de neurona concreto que se utilice, de la arquitectura o topología de conexión, y del algoritmo de aprendizaje, surgirán distintos modelos de redes neuronales.

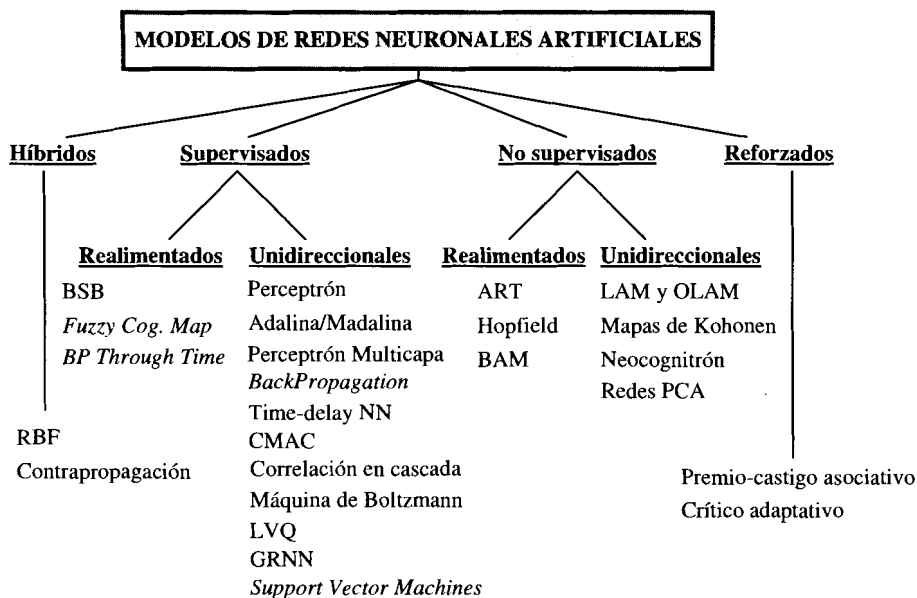


Figura 1.13 Clasificación de los ANS por el tipo de aprendizaje y la arquitectura

De la multitud de modelos y variantes que de hecho existen, unos cincuenta son medianamente conocidos, aunque tan sólo aproximadamente una quincena son utilizados con asiduidad en las aplicaciones prácticas. Por lo tanto, para llevar a cabo el estudio sistemático de los modelos se precisa algún tipo de clasificación.

Los dos conceptos que más caracterizan un modelo neuronal son el tipo de aprendizaje y la arquitectura de la red, por ello, consideramos interesante que la clasificación atienda ambos aspectos. Así se hace en la propuesta por Simpson [Simpson 89], que nosotros adoptamos, completamos y actualizamos.

De esta manera, en primer lugar, realizamos una distinción en cuanto al tipo de aprendizaje, por lo que aparece una primera clasificación en modelos supervisados, no supervisados, de aprendizaje híbrido y modelos de aprendizaje reforzado. A su vez, y dentro de cada uno de los grandes grupos, tendremos en cuenta el tipo de topología de la red, por lo que se distinguirá además entre redes realimentadas y redes unidireccionales (no realimentadas). La clasificación que así surge se muestra en la Figura 1.13.

Se puede apreciar que el conjunto de modelos de redes no realimentadas y de aprendizaje supervisado es el más numeroso. Dentro de este grupo trataremos en el capítulo 2 los casos del perceptrón simple, adalina y perceptrón multicapa o MLP (*Multilayer Perceptron*). Esta clase de modelos resulta especialmente importante por varias razones: por su interés histórico, generalidad, por ilustrar una amplia clase de aspectos que aparecen con frecuencia en todo el campo de las redes neuronales

(memoria asociativa, clasificación, aproximación funcional, etc.), y además por ser los sistemas neuronales más empleados en las aplicaciones prácticas.

A continuación, en el capítulo 3 trataremos los modelos no supervisados, en especial, uno de los más conocidos, el de mapas autoorganizados o mapas de Kohonen (*Self-organizing Feature Maps*). En el capítulo 4 estudiaremos como ejemplo de red híbrida las RBF, y como ejemplo de red realimentada el modelo de Hopfield, tanto en su versión discreta como continua.

Teniendo en cuenta que nuestro propósito es realizar una introducción al tema, en los próximos capítulos trataremos solamente algunos de los modelos más conocidos y habituales en las aplicaciones prácticas. Hemos estimado más conveniente estudiar unos pocos modelos con detenimiento que multitud de modelos más superficialmente (lo que se realiza en muchos textos introductorios) para no abrumar al lector no iniciado. Por otro lado, remitimos al lector interesado en profundizar en detalles o en estudiar muchos otros modelos a los excelentes libros de Haykin [Haykin 99] y [Principe 00], sin olvidar los ya clásicos textos de Hecht-Nielsen [Hecht-Nielsen 90] y de Hertz, Krogh y Palmer [Hertz 91],

1.7 COMPUTABILIDAD NEURONAL

Establecidos los ANS como un estilo de procesamiento alternativo-complementario al clásico basado en computadores digitales serie (tipo von Neumann), se hace necesario profundizar en sus características computacionales. Es bien sabido que *un ordenador digital constituye una máquina universal de Turing*, por lo que puede realizar cualquier cómputo¹⁰. Además, al estar construido en base a funciones lógicas, se deduce que *cualquier problema computacional puede ser resuelto con funciones booleanas* [Hopcroft 84].

Podemos preguntarnos si las redes neuronales, al igual que los ordenadores digitales, son también dispositivos universales de cómputo. Este asunto ha sido ampliamente tratado por autores como Abu-Mostafa [Abu-Mostafa 86, 87, 89].

En [Abu-Mostafa 86] se discute extensamente sobre las características computacionales de los ANS, demostrándose en particular que, **al igual que los computadores digitales convencionales, las redes neuronales son formalmente capaces de resolver cualquier problema computacional**. Una forma sencilla de verlo es la siguiente. Una red neuronal es capaz de implementar cualquier función booleana, pues basta recordar la neurona de la Figura 1.10, que realiza la función lógica $NAND_2$. Así, cualquier circuito digital puede ser realizado con una red neuronal, sólo con que escribirlo en función de puertas $NAND_2$, para a continuación

¹⁰ Una función matemática se dice computable si puede ser realizada por una máquina de Turing, de modo que la máquina de Turing se constituye en la referencia fundamental para aquellas cuestiones relacionadas con lo que es computable y lo que no lo es. Aquello que ninguna máquina de Turing pueda resolver se dice que no es computable.

sustituir cada puerta por una neurona como la anterior. Por lo tanto, se tiene que todo problema computacional puede ser resuelto mediante funciones booleanas. Por otra parte, toda función booleana puede realizarse con puertas NAND, y toda puerta NAND tiene su neurona equivalente. La conclusión es que *toda computación puede ser realizada por una red de neuronas*¹¹.

Por lo tanto, los ANS, como los ordenadores convencionales, son máquinas universales, por lo que para resolver un determinado problema, cualquiera de las dos aproximaciones sería perfectamente válida, en principio. La cuestión que entonces surge es, dado un problema, cuál de las dos alternativas, procesamiento neuronal o convencional, resulta más eficiente en su resolución. Estudiando en el campo de las redes neuronales los aspectos relacionados con la complejidad computacional, en [Abu-Mostafa 86] se deduce que *los problemas que requieren un extenso algoritmo o que precisan almacenar un gran número de datos, aprovechan mejor la estructura de una red neuronal que aquellos otros que requieren algoritmos cortos*. Así, un ordenador digital resulta más eficiente en la ejecución de tareas aritméticas y lógicas, mientras que un ANS resolverá mejor problemas que deban tratar con grandes bases de datos que almacenen ingentes cantidades de información, y en los que existan muchos casos particulares, como sucede en los problemas de reconocimiento de patrones en ambiente natural [Abu-Mostafa 87]. De esta manera podemos concluir que un estilo de computación no es mejor que el otro, simplemente para cada problema particular se deberá elegir el método más adecuado, y en el caso de problemas muy complejos, éstos deberían ser separados en partes, para resolver cada una mediante el método más idóneo.

1.8 UN EJERCICIO DE SÍNTESIS: SISTEMAS CONEXIONISTAS

En el presente capítulo, al exponer los fundamentos de los ANS hemos respetado el punto de vista tradicional: partiendo del modelo biológico se introduce un modelo de neurona artificial, para posteriormente establecer el concepto de arquitectura de la red y, por último, introducir las dinámicas de recuerdo y de aprendizaje como dos posibles modos de operación del sistema.

En el trabajo de J. D. Farmer, *A Rosetta Stone for Connectionism* [Farmer 90], se realiza un interesante ejercicio de síntesis de dichos conceptos, de aplicación tanto a las redes neuronales, como a muchos otros sistemas dinámicos que, por poseer modelos matemáticos similares, encajarían en los denominados **sistemas conexionistas**. Entre ellos se encuentran, además de los sistemas neuronales, las redes inmunes, sistemas clasificadores, redes booleanas, autómatas celulares, etc.

¹¹ MacCulloch y Pitts [McCulloch 43] demostraron ya que cualquier función lógica arbitraria puede ser construida con una apropiada combinación de elementos basados en su modelo de neurona, observando que la función NAND podía implementarse con un caso particular de ella.

Dicha síntesis parte de la idea de grafo (sección 1.4), sobre la que se desarrolla el marco común de los sistemas conexionistas. En los **modelos dinámicos** convencionales, la única parte del sistema que puede cambiar es su estado (que contiene toda la información necesaria para determinar su futuro); el grafo (la estructura), sin embargo, permanece inalterable, es decir, los parámetros que definen su patrón de conexiones no cambia en el tiempo.

Se define un **modelo conexionista** como un sistema dinámico en el que las interacciones entre variables están limitadas a un conjunto finito de conexiones, y en el que las conexiones son fluidas, en el sentido de que sus intensidades y/o su patrón de conectividad pueden cambiar con el tiempo. Es decir, en un sistema conexionista se permite que la propia estructura del grafo varíe, con lo que, además de la propia dinámica habitual del grafo establecida como cambios de estado, aparecen una o más dinámicas nuevas relacionadas con el cambio del propio grafo, y que pueden ocurrir en escalas temporales diferentes.

Las tres dinámicas que residen en el grafo de un sistema conexionista, y que corresponden a tres escalas temporales diferentes, son las siguientes:

- a) **Dinámica de los estados** (escala temporal rápida). Es una dinámica rápida, en la cual, a partir del estado actual y de las entradas del sistema, se obtiene su nuevo estado. En esta dinámica actúa la regla de activación, que determina la transición entre estados. En una red neuronal equivale al **modo recuerdo**, en la que el papel fundamental lo desempeña la función de activación de la neurona.
- b) **Dinámica de los parámetros** (escala temporal intermedia). Es una dinámica más lenta que la anterior, en la que se produce el cambio en los parámetros que definen las interacciones entre los nodos (intensidades de conexión). En una red neuronal se corresponde con la variación de los pesos sinápticos, es decir, con el **modo aprendizaje**, que lleva a cabo la regla de adaptación.
- c) **Dinámica del grafo** (escala temporal lenta). Es la más lenta de todas, en ella puede cambiar la estructura del propio grafo, es decir, no sólo la intensidad de la conexión entre los nodos, sino los propios nodos y su patrón de conexión. En una red neuronal se corresponde con posibilitar la **modificación de la arquitectura de la red** durante el aprendizaje, con la destrucción y creación dinámica de nodos.

El último tipo de dinámica no ha sido demasiado usual en los modelos neuronales clásicos, al menos explícitamente, pues implícitamente es la labor que se realiza cuando se ensaya con diferentes arquitecturas hasta encontrar la que mejor se adapta al problema. No obstante, desde hace algunos años se vienen introduciendo modelos en los que sí varía la propia arquitectura, que en ocasiones se denominan **arquitecturas evolutivas**. Uno de los primeros modelos que respondían a este esquema fue el de **Correlación en Cascada** (*cascade correlation* [Fahlman 90]); en

[Jutten 95] aparecen otros ejemplos de modelos evolutivos y en [IEEE 99] algunos modelos neuronales evolutivos basados en **algoritmos genéticos** (una clara tendencia dentro de la inteligencia computacional es la de fusión de modelos neuronales, borrosos, evolutivos y clásicos).

1.9 REALIZACIÓN Y APLICACIONES DE LOS ANS

Para concluir esta introducción a los ANS expondremos muy brevemente cómo éstos se implementan en la práctica y cuáles son sus aplicaciones. Estos dos temas se abordarán más extensamente en los capítulos 5 y 6, respectivamente.

Realización de redes neuronales

El modo más habitual de realizar una red neuronal consiste en **simularla en un ordenador convencional**, como un PC o una estación de trabajo, haciendo uso de programas escritos en lenguajes de alto nivel, como C o Pascal. Aunque de esta manera se pierde su capacidad de cálculo en paralelo, las prestaciones que ofrecen los ordenadores actuales resultan suficientes para resolver numerosos problemas prácticos, permitiendo la simulación de redes de tamaño considerable a una velocidad razonable. Ésta constituye la manera más barata y directa de realizar una red neuronal. Además, no es necesario que cada diseñador confeccione sus propios simuladores, pues hay disponible comercialmente software de simulación que permite el trabajo con multitud de modelos neuronales. En [Arbib 98, Hammerstrom 93^a, Principe 00] se muestran unos cuantos ejemplos (véase también el capítulo 6).

En el resto de las maneras de realizar un ANS se trata de aprovechar, en mayor o menor medida, su estructura de cálculo paralelo. Un paso adelante en este sentido consiste en simular la red sobre computadores con capacidad de cálculo paralelo (sistemas multiprocesador, máquinas vectoriales, masivamente paralelas...). Una orientación diferente consiste en llevar a cabo la **emulación hardware** de la red neuronal, mediante el empleo de sistemas de cálculo expresamente diseñados para realizar ANS basados, bien en microprocesadores de altas prestaciones (RISC, DSP...), bien en procesadores especialmente diseñados para el trabajo con redes neuronales. Estas estructuras se suelen denominar **placas aceleradoras, neuroemuladores o neurocomputadores de propósito general**. Algunos sistemas de desarrollo de redes neuronales, además de un software de simulación, incluyen dispositivos de este tipo, en forma de tarjetas conectables al bus de un PC.

El aprovechamiento a fondo de la capacidad de cálculo masivamente paralelo de los ANS conduce a la realización hardware de la estructura de la red neuronal, en forma de circuitos específicos que reflejan con cierta fidelidad la arquitectura de la red. La tecnología más habitualmente empleada para ello es la microelectrónica VLSI, denominándose **chips neuronales** a los circuitos integrados así construidos. La realización hardware de la red neuronal es la manera de resolver problemas que

involucran un gran número de datos y precisan respuestas en tiempo real (por ejemplo, un sistema de detección e interceptación de misiles enemigos).

La realización electrónica de redes neuronales es un campo muy activo, abordado tanto por grupos de investigación universitarios como por empresas de los sectores de la electrónica e informática. Compañías como Siemens, Philips, Hitachi, AT&T, IBM o Intel han puesto en marcha desde mediados de los años ochenta programas de investigación y desarrollo en este campo. Asimismo, se han creado diversas empresas que tratan de explotar comercialmente (con mejor o peor fortuna) estos nuevos desarrollos. No obstante, debido a la creciente potencia de los computadores de propósito general (por ejemplo, un PC convencional basado en procesador Pentium y Windows) y sus bajos precios, en muchas aplicaciones no merece la pena el desarrollo o adquisición de hardware neuronal [Granado 99], bastando con simular la red neuronal en un PC. Con el paso del tiempo la aplicación de circuitos neuronales está quedando restringida a aplicaciones muy concretas, donde se requieren muy altas prestaciones, o bien en sistemas específicos donde se necesita un chip barato. Un ejemplo claro es el de los chips neuronales que la empresa californiana Sensory Inc. comercializa para reconocimiento de habla [Sensory 00], que se utilizan en teléfonos, periféricos, juguetes, etc., que pueden llegar a constar menos de 5 dólares. Todos estos aspectos se tratarán ampliamente en el capítulo 5.

Aplicaciones de las redes neuronales

Ya hemos señalado que los objetivos que se persiguen mediante el empleo de redes neuronales son mucho más modestos que la creación de un *cerebro artificial*. Las redes neuronales se utilizan en la **resolución de problemas prácticos concretos**, que normalmente no han sido bien resueltos mediante sistemas más tradicionales, como pueda ser el caso del reconocimiento de vehículos en los peajes de las autopistas o la previsión de consumo eléctrico [Hèrault 94]. Gracias a su capacidad de aprendizaje, robustez, no linealidad y tolerancia a la imprecisión e incerteza del entorno, desde hace unos años las redes neuronales vienen alcanzando excelentes resultados en aplicaciones diversas [Hèrault 94, Lupo89, Hammerstrom 93].

Así como las aplicaciones prácticas de las redes neuronales resultaban hace unos años anecdóticas (o en fase experimental), como veremos en el capítulo 6 en la actualidad muchas compañías las aplican de un modo rutinario a numerosos problemas; en este sentido, la aplicación de redes neuronales puede considerarse que ha alcanzado ya su **madurez** [Werbos 98]. Los artículos [Hammerstrom 93a, 93b, Widrow 94] constituyen un excelente repaso a los numerosos **campos de aplicación** de los ANS. Los más habituales son los relacionados con clasificación, estimación funcional y optimización; en general, el del **reconocimiento de patrones** suele considerarse como un denominador común. Se pueden señalar, entre otras, las siguientes áreas de aplicación de los sistemas neuronales: reconocimiento del habla, reconocimiento de caracteres, visión, robótica, control, procesamiento de señal, predicción, economía, defensa, bioingeniería, etc. Asimismo, se están aplicando ANS

para incorporar aprendizaje en los sistemas borrosos (capítulo 9) y a la confección de sistemas expertos conexionistas [Gallant 93].

Aunque en el capítulo 6 se estudiarán numerosos casos de aplicación de ANS a problemas reales, presentaremos a continuación unos cuantos ejemplos. El del reconocimiento de caracteres es uno de los campos donde mayores éxitos han cosechado estos sistemas; se estima que aproximadamente el 50% de los sistemas de OCR (*Optical Character Recognition*) se basa en redes neuronales [Werbos 98]. Por ejemplo, Sharp ha desarrollado un sistema de reconocimiento de caracteres para el alfabeto Kanji (japonés) mediante una red jerárquica basada en LVQ [Kohonen 90]. Synaptics, empresa del *Silicon Valley*, ha desarrollado un chip neuronal para el reconocimiento de direcciones escritas en los sobres de las cartas [Hammerstrom 93a]. Por otro lado, Quicktionary, de la empresa Wizcom, es un pequeño escáner con forma de bolígrafo que lee y traduce textos escritos.

Un área de intenso trabajo es el del tratamiento de la información económica, siendo uno de los grupos punteros el de A.N. Refenes, de la *London Business School*; en [Refenes 95] se describen aplicaciones diversas en este campo.

Otra de las áreas importantes es la industria. Fujitsu, Kawasaki y Nippon Steel emplean ANS en el control de procesos industriales, como por ejemplo en plantas de producción de acero. Siemens aplica redes neuronales y sistemas borrosos en la fabricación de celulosa (por ejemplo, en la planta de Celulose do Caima, Portugal [Höhfeld 93, Poppe 95]), en laminadoras y en galvanizadoras. Citröen emplea redes neuronales en la determinación de la calidad del material utilizado en la confección de los asientos de los vehículos, Ford en reducción de contaminantes [James 98] y Renault para detectar averías en el encendido de los automóviles [Hèrault 94].

Finalmente, en la nueva versión del avión de combate F-15 se ha ensayado con un sistema neuronal para ayuda al piloto en caso de alcance por fuego enemigo [Freedman 94] (aprovechando que una red neuronal electrónica puede aprender a desenvolverse en las nuevas circunstancias que influyen en el pilotaje del avión miles de veces más rápidamente que el ser humano), y recientemente una red neuronal ha conseguido comandar el aterrizaje de un avión Jumbo sin intervención humana [Werbos 98].

1.A APÉNDICE: DE LA NEURONA BIOLÓGICA A LA ARTIFICIAL

En este apéndice vamos a mostrar cómo el sencillo modelo de neurona artificial que se emplea en las redes neuronales artificiales puede derivarse de modelos de neurona más complejos, que reflejan más fielmente la realidad biológica, y que sirven para modelar y estudiar su operación con detalle.

Un modelo clásico de la operación de la neurona biológica es el propuesto por Hodgkin y Huxley [Hodgin 52] (véase también, por ejemplo, [Kohonen 89]), que parte por considerar que la membrana celular actúa como un condensador que tiene asociada una capacidad eléctrica C_M . En este modelo a cada una de las especies iónicas que entran y salen a través de la membrana (iones de sodio, potasio, etc., véase la sección 1.1) se asigna cierta intensidad eléctrica; por otra parte, la membrana presenta diferente resistencia para cada tipo de ión (Figura 1.14). La capacidad eléctrica C_M asociada a la membrana celular integra dichas corrientes, y la tensión así obtenida es finalmente convertida en trenes de pulsos de una determinada frecuencia en el soma (concretamente en un lugar próximo a la raíz del axón). En definitiva, se lleva a cabo una conversión tensión a frecuencia, de manera que el nivel de activación (excitación) de la neurona queda codificado en su frecuencia de disparo (sección 1.1).

Este modelo trata con cierto nivel de detalle determinados aspectos biológicos, como el trasiego de corrientes de las distintas especies iónicas a través de la membrana, para así explicar algunas propiedades de la neurona biológica observadas experimentalmente, como la forma concreta que presentan los potenciales de acción. Sin embargo, si se pretende analizar y simular la operación de un sistema compuesto por un elevado número de neuronas debe recurrirse a modelos mucho más sencillos, idealizaciones que oculten muchos de los detalles, como puedan ser las corrientes concretas que corresponden a los diferentes iones; de otra manera el análisis del sistema inabordable por su complejidad. Ésta es la orientación que se toma en el campo de las redes neuronales artificiales, donde se estudia el comportamiento de toda una colectividad de neuronas, para poder así aplicar las propiedades que de ella surgen (emergentes) a la resolución de problemas prácticos de interés.

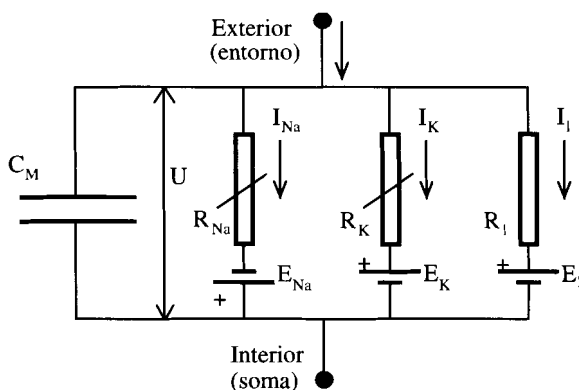


Figura 1.14 Modelo eléctrico de la membrana de la neurona [Hodgin 52]. Los subíndices Na y K hacen referencia a los iones sodio y potasio, y el l (leakage) a los iones que representan corrientes de fuga. U es el potencial de membrana

Vamos a obtener a continuación un primer modelo más sencillo de neurona que derivaremos del orientado a la biología que acabamos de presentar. Consideraremos por simplicidad que la salida de la neurona es una tensión analógica en vez de una frecuencia de pulsos, en otras palabras, representaremos la frecuencia de disparo por una tensión continua. Por otra parte, hemos visto que la membrana de la neurona se comporta como un condensador que recibe cargas procedentes del exterior o de otras neuronas, integrándolas y provocando una respuesta, generalmente de tipo no lineal. En definitiva, podemos modelar la neurona como si de un elemento integrador no lineal se tratase, con lo que la dinámica de cierta neurona i podría describirse utilizando la ley de conservación de la carga eléctrica de la siguiente manera

$$C \frac{dU_i}{dt} = -F_i + I_i \quad (1.24)$$

siendo U_i su potencial de membrana y C su capacidad, F_i representa el conjunto de las corrientes de pérdida o de fuga de la neurona, y I_i la suma de las corrientes sinápticas debidas a la acción de las entradas que recibe, provenientes, bien de otras neuronas, bien de fuentes externas.

Consideraremos que la acción de las demás neuronas sobre la i es lineal, y que vendrá descrita por ciertos acoplamientos w_{ij} (pesos sinápticos), que dan la intensidad de interacción entre la neurona j y la i . Por otra parte, denotaremos por b_i (*bias*) el resto de las corrientes provenientes del entorno. Así, podemos escribir la totalidad de las corrientes que la neurona recibe de la siguiente forma

$$I_i = \sum_{j=1}^n w_{ij} V_j + b_i \quad (1.25)$$

con V_j los potenciales generados por las neuronas j (o del exterior). Es interesante observar que los pesos sinápticos w_{ij} , en definitiva, representan conductancias eléctricas.

Por otra parte, supondremos que las corrientes de fuga son función únicamente de la salida de la neurona y_i , de manera que si una neurona presenta una mayor actividad de salida (mayor frecuencia de disparo) poseerá mayores pérdidas. Consideraremos una dependencia no lineal

$$F = g(y_i) \quad (1.26)$$

siendo $g(\cdot)$ una función monótona creciente, que consideraremos tiene inversa. De este modo el **modelo dinámico de la neurona** queda

$$C \frac{dU_i}{dt} = -g(y_i) + \sum_{j=1}^n w_{ij} V_j + b_i \quad (1.27)$$

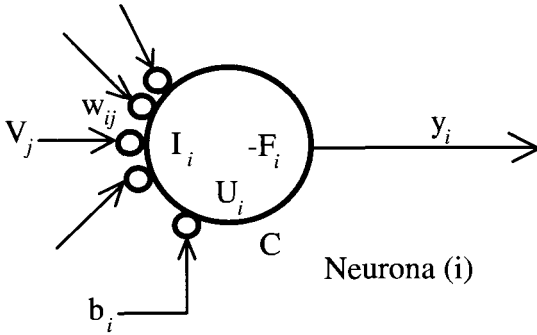


Figura 1.15
Parámetros del modelo
dinámico de neurona

Cuando estudiemos el modelo de Hopfield analógico (capítulo 4) veremos que el modelo de neurona que propone, basado en amplificadores, toma prácticamente la misma forma.

Si consideramos ahora que las entradas varían lentamente, podrán considerarse estables durante un tiempo suficiente, de modo que puede realizarse la aproximación $dU_i/dt \approx 0$, por lo que despejando en (1.27) obtenemos

$$g(y_i) = \sum_{j=1}^n w_{ij} V_j + b_i \quad (1.28)$$

y si calculamos la salida de la neurona, se tiene

$$y_i = g^{-1} \left(\sum_{j=1}^n w_{ij} V_j + b_i \right) = f \left(\sum_{j=1}^n w_{ij} V_j + b_i \right) \quad (1.29)$$

donde hemos definido $f(.) \equiv g^{-1}(.)$. La expresión obtenida es el **modelo estático de la neurona**, que resulta similar al que propusimos como **modelo de neurona formal estándar** (ecuación 1.11), comúnmente empleada en numerosos modelos de redes neuronales artificiales. De este modo hemos comprobado que el modelo convencional de neurona artificial resulta ser una simplificación de otros modelos mucho más elaborados y más fieles a la realidad biológica. En el modelo de neurona artificial se ocultan multitud de detalles, teniéndose en cuenta únicamente los rasgos esenciales de la operación de la neurona biológica, para así disponer de un modelo más sencillo que permita trabajar con colectividades de neuronas.

Para elaborar este apéndice nos hemos inspirado en las referencias [Hopfield 84, Vemuri 88, Gutfreund 92, Hush 92a, 92b, Kohonen 89, 97], con la intención de mostrar la conexión entre los modelos de neurona orientados a la biología y los modelos simplificados de neurona artificial. Para alcanzar una visión más rigurosa recomendamos especialmente la consulta de [Kohonen 89, 97] y las referencias allí citadas.

REDES NEURONALES SUPERVISADAS

En los próximos capítulos trataremos algunos de los modelos de redes neuronales más populares. En primer lugar, comenzaremos estudiando la amplia clase de redes unidireccionales organizadas en capas (*feed-forward*) y con aprendizaje supervisado, que son empleadas como clasificadores de patrones y estimadores de funciones. Estos modelos en la literatura son denominados *mapping neural networks* [Hecht-Nielsen 90], o redes neuronales para representación (ajuste) funcional.

Dentro de este gran grupo de redes trataremos el **perceptrón simple**, la **adalina** y el **perceptrón multicapa**. El popular algoritmo de aprendizaje denominado *back-propagation* (retropropagación) o **BP** se aplica precisamente a este último modelo. El perceptrón multicapa con aprendizaje BP (o alguna de sus variantes) es el modelo neuronal más empleado en las aplicaciones prácticas (se estima que el 70% de los desarrollos con redes neuronales hacen uso alguna de sus variantes [Gedeon 95]).

Por último, señalaremos algunas referencias para el lector interesado en ampliar conocimientos. En primer lugar destacaremos el artículo [Hush 93], que constituye una excelente actualización del clásico de Lippmann [Lippmann 87]. Para un estudio en mayor profundidad de estos modelos recomendamos el texto [Bishop 95] y, especialmente, los recientes [Haykin 99] y [Principe 00], donde se recopilan las últimas investigaciones.

2.1 REDES UNIDIRECCIONALES

Muchos problemas del mundo real pueden interpretarse desde el punto de vista de la estimación o aproximación funcional, en el sentido de tratar de encontrar la función que a partir de un conjunto de entradas proporciona la salida deseada. Por ejemplo, si queremos desarrollar un reconocedor de caracteres manuscritos el objetivo será encontrar un sistema que implemente la función que asocia la imagen de una determinada letra o carácter escrito con la clase a la que pertenece. Otro ejemplo

ilustrativo sería el de la predicción de cotizaciones bursátiles, en el que mediante una red neuronal se trataría de encontrar la función que relaciona diversas variables de entrada (cotizaciones previas, tipos de interés, inflación, etc.) con la actual cotización en bolsa de una determinada entidad o empresa.

Como hemos adelantado, dentro del grupo de modelos de redes neuronales unidireccionales trataremos especialmente los casos del perceptrón simple, adalina y perceptrón multicapa o MLP (*Multilayer Perceptron*). En primer lugar, hay que destacar que estos modelos presentan un gran interés histórico, pues su evolución representa la historia misma de las redes neuronales. Así, el perceptrón simple y la adalina se propusieron a finales de los años cincuenta, alcanzando una gran popularidad durante los años sesenta, para a continuación sufrir un duro revés a finales de esa década, debido fundamentalmente al riguroso trabajo de Minsky y Papert [Minsky 69], en el que pusieron claramente de manifiesto sus limitaciones. El contraste con el gran interés que el tema había despertado hasta entonces hizo que el campo de las redes neuronales en general entrase en una época oscura durante la década de los setenta, desviándose la mayor parte de los recursos económicos al prometedor campo de la inteligencia artificial. Sin embargo, el tema resurgió en los ochenta debido a diversas circunstancias, como la disponibilidad de ordenadores con potencia suficiente para llevar a cabo simulaciones antes difícilmente abordables, el desarrollo de la integración VLSI, que permitió realizar electrónicamente redes neuronales, y la introducción de nuevos modelos, especialmente el MLP entrenado mediante el algoritmo BP, que superaba los viejos problemas de los modelos predecesores, anulando buena parte de las objeciones "históricas" de Minsky y Papert.

No obstante, el principal interés de los modelos que trataremos en este capítulo es su generalidad y aplicabilidad práctica, además de ilustrar muy bien una amplia clase de problemas y aspectos que aparecen con frecuencia en todo el campo de los ANS. Por todo ello, este capítulo puede considerarse como uno de los centrales de la parte correspondiente a redes neuronales. Por otro lado, resulta interesante saber que algunos de los modelos de aprendizaje que expondremos también son empleados en el entrenamiento de sistemas borrosos, como veremos en la segunda parte.

2.2 EL ASOCIADOR LINEAL: APRENDIZAJE HEBBIANO

Antes de comenzar con los modelos citados estudiaremos el **asociador lineal**, un sencillo ejemplo de red unidireccional que servirá para introducir los conceptos relacionados con el aprendizaje en redes neuronales. Este modelo, mediante una transformación lineal, asocia un conjunto de patrones de entrada a otros de salida.

El asociador lineal consta únicamente de una capa de neuronas lineales, cuyas entradas las denotamos por $\mathbf{x}$ y sus salidas por $\mathbf{y}$, vector que constituye además la respuesta de la red neuronal. Asimismo, denotaremos por $\mathbf{W}=\{w_{ij}\}$ a la matriz de pesos sinápticos; cada fila de $\mathbf{W}$ contiene los pesos de una neurona $\mathbf{w}_i$

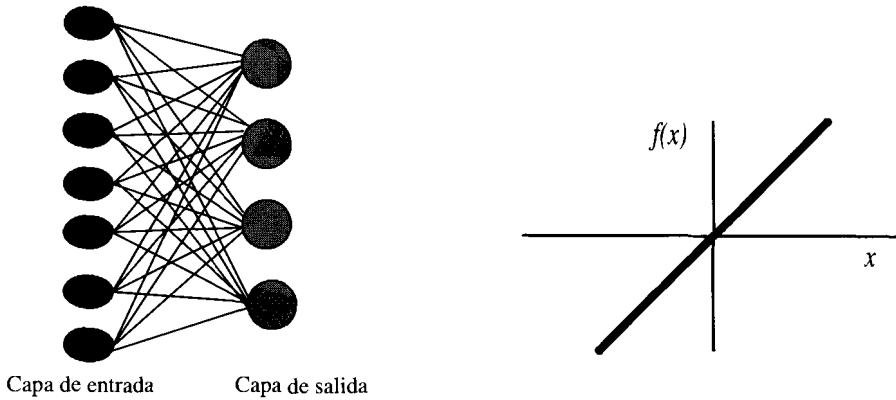


Figura 2.1 Asociador lineal (función de activación identidad)

$$W = (w_1 \quad w_2 \quad \dots \quad w_m)^T \quad (2.1)$$

La operación del asociador lineal es simplemente

$$y = Wx \quad (2.2)$$

o bien

$$y_i = \sum_{j=1}^n w_{ij} x_j \quad (2.3)$$

Por lo tanto, cada neurona i del asociador lineal lleva a cabo la suma ponderada de las entradas con sus pesos sinápticos. Es decir, dentro del marco de neurona estándar descrito en el capítulo 1, esta neurona calcula el potencial postsináptico por medio de la convencional suma ponderada, cantidad a la que aplica finalmente una función activación de tipo identidad.

El asociador lineal debe aprender a asociar p pares entrada-salida¹, $\{(x^\mu, t^\mu) / 1 \leq \mu \leq p\}$, ajustando sus pesos W de modo que ante un cierto patrón de entrada x^μ responda con t^μ , y que ante entradas similares, $(x^\mu + \epsilon)$, responda con salidas también próximas $(t^\mu + \delta)$ (con ϵ y δ cantidades pequeñas). El problema se centra en encontrar la matriz de pesos W óptima en el sentido descrito. Para ello, en el campo de las redes neuronales normalmente se hace uso de una **regla de aprendizaje**, que a partir de las entradas y de las salidas deseadas (en el caso del aprendizaje supervisado), proporcione el conjunto óptimo de pesos W .

¹ Usaremos el símbolo t (target) cuando nos refiramos a las salidas deseadas u objetivo.

Regla de aprendizaje de Hebb

Se trata de uno de los modelos clásicos de aprendizaje en redes neuronales. Donald Hebb en 1949 [Hebb 49] postuló un mecanismo de aprendizaje para la neurona biológica, cuya idea básica consiste en que *cuando un axón presináptico causa la activación de cierta neurona postsináptica, la eficacia de la sinapsis que las relaciona se refuerza*. El trabajo experimental posterior ha confirmado en parte esta teoría [Kandel 92], demostrando la presencia de este tipo de aprendizaje en la neurona biológica, aunque en coexistencia con otros esquemas [Alkon 89] (interacción a nivel presináptico, crecimiento y debilitamiento del axón, modificaciones del metabolismo, desarrollo y muerte celular, etc.).

Este tipo de aprendizaje es simple y local. Gran parte de su importancia radica en que fue pionero, tanto en neurociencias como en neurocomputación, y muchos otros algoritmos más complejos (y más potentes) lo toman como punto de partida.

De una manera general, se denomina **aprendizaje hebbiano** a aquellas formas de aprendizaje que involucran una modificación en los pesos Δw_{ij} proporcional al producto de una entrada j por la salida i de la neurona

$$\Delta w_{ij} = \varepsilon y_i x_j \quad (2.4)$$

siendo ε un parámetro denominado **ritmo de aprendizaje**, que suele ser una cantidad entre 0 y 1. Esta expresión puede considerarse la representación matemática del modelo de aprendizaje descrito por Hebb.

Consideremos nuestro asociador lineal. La regla de Hebb se expresa en este caso particular así

$$\Delta w_{ij}^{\mu} = t_i^{\mu} x_j^{\mu} \quad (2.5)$$

y, por lo tanto

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}^{\mu} \quad (2.6)$$

Si los pesos de partida son nulos, el valor final de W para las p asociaciones será

$$W = \mathbf{t}^1 \mathbf{x}^{1T} + \mathbf{t}^2 \mathbf{x}^{2T} + \dots + \mathbf{t}^p \mathbf{x}^{pT} \quad (2.7)$$

Empleando la regla de Hebb para el entrenamiento del asociador lineal, si los vectores de entrada $\{\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^p\}$ son ortonormales (ortogonales y de longitud unidad) se cumple

$$W \mathbf{x}^{\mu} = (\mathbf{t}^1 \mathbf{x}^{1T} + \dots + \mathbf{t}^p \mathbf{x}^{pT}) \mathbf{x}^{\mu} = \mathbf{t}^1 (\mathbf{x}^{1T} \cdot \mathbf{x}^{\mu}) + \dots + \mathbf{t}^p (\mathbf{x}^{pT} \cdot \mathbf{x}^{\mu}) = \mathbf{t}^{\mu} \quad (2.8)$$

y por tanto, ante la entrada $\mathbf{x}^{\mu}$ se reproduce la respuesta aprendida $\mathbf{t}^{\mu}$, es decir, la regla de Hebb ha conseguido en este caso que la red aprenda a realizar las asociaciones deseadas. El problema reside en que las condiciones son muy restrictivas, pues para

que las asociaciones sean correctas los patrones de entrada deben ser ortonormales. Por ello, si la dimensión del espacio de entrada es n , solamente podrá aprender hasta n asociaciones. Para almacenar más pares entrada-salida será preciso utilizar otras estrategias.

Eliminando la condición de ortogonalidad, se tiene (manteniendo el requisito de vectores de longitud 1)

$$\mathbf{W}\mathbf{x}^\mu = \mathbf{t}^1(\mathbf{x}^1{}^T\mathbf{x}^\mu) + \mathbf{t}^2(\mathbf{x}^2{}^T\mathbf{x}^\mu) + \dots + \mathbf{t}^p(\mathbf{x}^p{}^T\mathbf{x}^\mu) = \mathbf{t}^\mu + \sum_{v \neq \mu} \mathbf{t}^v(\mathbf{x}^v{}^T\mathbf{x}^\mu) = \mathbf{t}^\mu + \delta \quad (2.9)$$

expresión denominada **expansión señal-ruido**, pues proporciona la salida deseada más un término adicional, que interpretamos como el ruido superpuesto a la señal. Empleando reglas algo más sofisticadas que la de Hebb, como la de la pseudoinversa o la de Widrow-Hoff, se obtendrá una matriz de pesos que logrará además que el ruido δ sea pequeño comparado con la señal.

Regla de la pseudoinversa

La regla de aprendizaje de Hebb ha sido introducida debido a su plausibilidad biológica. Sin embargo, en general se tratará de deducir los algoritmos de aprendizaje a partir de un cierto criterio a optimizar; *el aprendizaje usualmente se planteará como un procedimiento para alcanzar el conjunto de pesos óptimo que resuelva un problema dado*. Para ello se hace necesario definir el significado de "óptimo" en cada caso concreto, es decir, hay que proponer un criterio que mida el rendimiento de la red neuronal para encontrar una regla de actualización de pesos que lo optimice. Una forma habitual de definir el rendimiento es el error cuadrático medio de las salidas actuales de la red respecto de las deseadas. Para el asociador lineal se tendría

$$E\{w_{ij}\} = (1/p) \sum_{\mu=1}^p |\mathbf{t}^\mu - \mathbf{W}\mathbf{x}^\mu|^2 = (1/p) \sum_{\mu=1}^p \sum_{i=1}^n (t_i^\mu - \mathbf{W}x_i^\mu)^2 \quad (2.10)$$

De este modo, un algoritmo de aprendizaje para el asociador lineal debería obtener un conjunto de pesos que minimicen esta expresión del error. Si los vectores $\mathbf{x}^\mu$ son ortonormales, el error que proporciona la regla de Hebb (2.5) según (2.10) es cero, lo que indica que esta regla es óptima respecto de la medida de error propuesta si se dispone de vectores de entrada $\mathbf{x}^\mu$ ortonormales.

Si denominamos $\mathbf{X}$ a una matriz $n \times p$ que tiene por columnas los vectores de entrada $\mathbf{x}^\mu$, $\mathbf{X}=(\mathbf{x}^1 \ \mathbf{x}^2 \ \dots \ \mathbf{x}^p)$, y si llamamos $\mathbf{Y}$ a la matriz $m \times p$ cuyas columnas son los vectores de salida $\mathbf{y}^\mu$, $\mathbf{Y}=(\mathbf{y}^1 \ \mathbf{y}^2 \ \dots \ \mathbf{y}^p)$, la ecuación (2.10) se transforma en²

² Definiendo la norma $\|\mathbf{M}\|^2$ de una matriz $\mathbf{M}$ ($p \times q$) de la siguiente forma $\|\mathbf{M}\|^2 = \|(m_{ij})\|^2 = \sum_{i=1}^p \sum_{j=1}^q m_{ij}^2$

$$E\{w_{ij}\} = (1/p) \|Y - WX\|^2 \quad (2.11)$$

Con esta nomenclatura, la regla de Hebb se expresa de la forma siguiente

$$W = YX^T \quad (2.12)$$

Una regla de aprendizaje basada en la utilización de la **matriz pseudoinversa** puede escribirse como

$$W = YX^+ \quad (2.13)$$

donde X^+ denota la pseudoinversa³ de X [Kohonen 89, Hecht-Nielsen 90]. Se puede demostrar que esta elección para W minimiza el error cuadrático medio (ecuaciones 2.10 y 2.11), es decir, que es óptima respecto de este error [Hecht-Nielsen 90]. En [Ritter 91a] se deduce (2.13) a partir de la minimización algebraica de (2.11).

Así como con la regla de Hebb se podían almacenar hasta n vectores ortonormales, con la pseudoinversa se pueden almacenar hasta n vectores linealmente independientes (por ejemplo, en la página 82 de [Hecht-Nielsen 90], puede verse su demostración). Si pretendemos almacenar más pares entrada-salida, surgirán errores, pero el *mapping* lineal implementado seguirá siendo óptimo en el sentido del error cuadrático medio (se alcanzará el menor error posible).

Debido a que ambas reglas son óptimas según el mismo criterio, la regla de Hebb y la de la pseudoinversa deben estar muy relacionadas. Esta circunstancia es fácil de apreciar, pues si consideramos un conjunto de vectores de entrada ortonormales, la regla de la pseudoinversa (2.13) se convierte en la de Hebb. Por otra parte, si se realiza la expansión en serie de la ecuación (2.13) de la pseudoinversa (véase, por ejemplo, la página 455 de [Rumelhart 86]), el primer término de la serie es precisamente la ecuación (2.12) de la regla de Hebb. Es decir, la regla de Hebb representa en el fondo un caso particular de la más general regla de la pseudoinversa.

Habitualmente para el cálculo de la pseudoinversa se utiliza el **teorema de Greville** [Kohonen 89, Hecht-Nielsen 90], aunque presenta el inconveniente de que para aprender un nuevo patrón se debe recalcular toda la matriz de pesos, lo que no resulta común dentro de la filosofía de los ANS [Ritter 91a], en la se tiende a que todos los modelos sean locales y operen incrementalmente. Se ha mostrado que en la

³ Toda matriz tiene una pseudoinversa. La pseudoinversa A^+ de una matriz A ($p \times q$) arbitraria se define como la única matriz que cumple las siguientes propiedades:

$$\begin{aligned} AA^+A &= A \\ A^+AA^+ &= A^+ \\ AA^+ &= (AA^+)^T \\ A^+A &= (A^+A)^T \end{aligned}$$

Se puede demostrar que la pseudoinversa de una matriz cuadrada no singular es igual a su inversa. En este sentido, la pseudoinversa representa la generalización del concepto de inversa de una matriz, para el caso de matrices no cuadradas [Kohonen 89].

práctica [Hecht-Nielsen 90] la siguiente forma aproximada del teorema de Greville, local e iterativa, suele proporcionar resultados correctos

$$\mathbf{w}_i^{new} = \mathbf{w}_i^{old} + \varepsilon \cdot (t_i^\mu - (\mathbf{w}_i^{old})^T \mathbf{x}^\mu) \mathbf{x}^\mu \quad (2.14)$$

siendo ε el ritmo de aprendizaje, parámetro que indica la rapidez en la actualización ($0 < \varepsilon < 1$). En este esquema iterativo, los patrones deben ser presentados a la red neuronal repetidamente, obteniéndose de esta manera una aproximación a la matriz pseudoinversa mediante cálculos simples y locales.

Esta expresión coincide con la famosa regla de Widrow-Hoff de la adalina que veremos más adelante. Asimismo, en [Rumelhart 86a] se relacionan ambos algoritmos de aprendizaje, mostrándose que en realidad ambas reglas son equivalentes cuando se trata de realizar un aprendizaje estadístico, consistente en asignar en vez de un patrón de entradas a uno de salidas, toda una clase de vectores de entrada (compuesta por un conjunto de patrones estocásticos que se distribuyen gaussianamente en torno al prototipo de la clase) a una clase del espacio de salidas (de similares características). Es decir, la regla de la pseudoinversa y la de Widrow-Hoff coinciden asintóticamente cuando se consideran distribuciones de vectores estocásticos.

La regla de la pseudoinversa se ha aplicado en redes más complejas, como la de Hopfield [Personnaz 86], proporcionando mejores resultados que la de Hebb, inicialmente propuesta y más conocida y extensamente estudiada (véase, por ejemplo, [Müller 90] y referencias allí citadas).

2.3 EL PERCEPTRÓN SIMPLE (ROSENBLATT, 1959)

Este modelo neuronal fue introducido por Rosenblatt a finales de los años cincuenta [Rosenblatt 62, Hertz 91, Principe 00]. La estructura del perceptrón se inspira en las primeras etapas de procesamiento de los sistemas sensoriales de los animales (por ejemplo, el de visión), en los cuales la información va atravesando sucesivas capas de neuronas, que realizan un procesamiento progresivamente de más alto nivel.

El perceptrón simple es un modelo unidireccional, compuesto por dos capas de neuronas, una sensorial o de entradas, y otra de salida (Figura 2.2). La operación de una red de este tipo, con n neuronas de entrada y m de salida, se puede expresar como

$$y_i(t) = f\left(\sum_{j=1}^n w_{ij} x_j - \theta_i\right), \quad \forall i, 1 \leq i \leq m \quad (2.15)$$

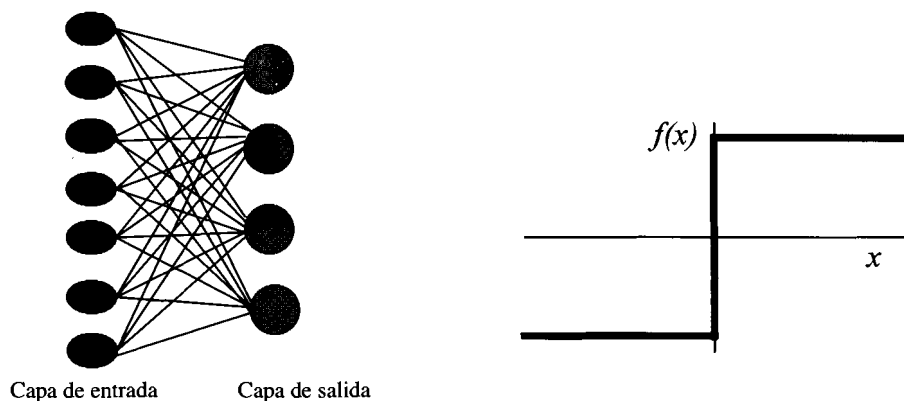


Figura 2.2 Perceptrón simple y función de transferencia de su neurona

Las neuronas de entrada no realizan ningún cómputo, únicamente envían la información (en principio consideraremos señales discretas $\{0, +1\}$) a las neuronas de salida (en el modelo original estas neuronas de entrada representaban información ya procesada, no datos directamente procedentes del exterior). La función de activación de las neuronas de la capa de salida es de tipo escalón. Así, la operación de un perceptrón simple puede escribirse

$$y_i = H\left(\sum_{j=1}^n w_{ij}x_j - \theta_i\right), \quad \forall i, 1 \leq i \leq m \quad (2.16)$$

con $H(\cdot)$ la función de Heaviside o escalón (capítulo 1). El perceptrón puede utilizarse tanto como clasificador, como para la representación de funciones booleanas, pues su neurona es esencialmente de tipo MacCulloch-Pitts, de salida binaria. La importancia histórica del perceptrón radica en su carácter de dispositivo entrenable, pues el algoritmo de aprendizaje del modelo introducido por Rosenblatt, y que describiremos más adelante, permite determinar automáticamente los pesos sinápticos que clasifican un conjunto de patrones a partir de un conjunto de ejemplos etiquetados.

Mostraremos a continuación que un perceptrón permite realizar tareas de clasificación. Cada neurona del perceptrón representa una determinada clase, de modo que dado un vector de entrada, una cierta neurona responde con 0 si no pertenece a la clase que representa, y con un 1 si sí pertenece. Es fácil ver que una neurona tipo perceptrón solamente permite discriminar entre dos clases **linealmente separables** (es decir, cuyas regiones de decisión pueden ser separadas mediante una única condición lineal o hiperplano⁴). Sea una neurona tipo perceptrón de dos entradas, x_1 y x_2 , con salida y , cuya operación se define por lo tanto

⁴Una línea recta, si trabajamos en dos dimensiones.

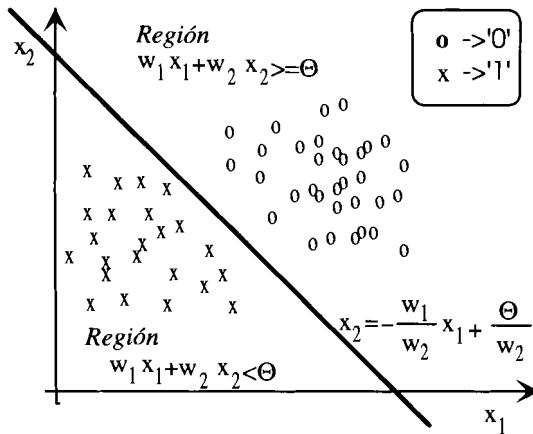


Figura 2.3 Regiones de decisión en el plano

$$y = H(w_1x_1 + w_2x_2 - \theta) \quad (2.17)$$

o bien

$$y = \begin{cases} 1, & \text{si } w_1x_1 + w_2x_2 \geq \theta \\ 0, & \text{si } w_1x_1 + w_2x_2 < \theta \end{cases} \quad (2.18)$$

Si consideramos x_1 y x_2 situadas sobre los ejes de abscisas y ordenadas en el plano, la condición

$$w_1x_1 + w_2x_2 - \theta = 0 \Rightarrow x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_2} \quad (2.19)$$

representa una recta (hiperplano, si trabajamos con n entradas) que divide el plano (espacio) en dos regiones, aquellas para las que la neurona proporciona una salida '0' o '1', respectivamente (Figura 2.3). Luego, efectivamente, una neurona tipo perceptrón representa un discriminador lineal, al implementar una condición lineal que separa dos regiones en el espacio, que representan dos diferentes clases de patrones.

Consideremos la función lógica NAND_2 (AND negada de dos entradas), que representamos sobre el plano (Figura 2.4a). En este caso pueden encontrarse unos parámetros w_1 , w_2 y θ que determinen una recta que separa perfectamente las regiones correspondientes a los valores lógicos 0 y 1. Por ello, la función lógica NAND se dice separable linealmente, puesto que hemos podido encontrar una única condición lineal que divida ambas regiones⁵. Por ejemplo, un perceptrón con los siguientes parámetros implementa la función NAND: $w_1=w_2=-2$, y $\theta=-3$ (capítulo 1).

⁵Una definición más rigurosa sería: una función se dice **linealmente separable** cuando su espacio de variables de entrada puede ser dividido en regiones de igual salida mediante una única condición lineal (un hiperplano).

Sin embargo, consideremos la función lógica or-exclusivo o XOR (su salida es el 0 lógico si las variables de entrada son iguales y 1 si son diferentes), y representémosla también en el plano (Figura 2.4b). En este caso podemos apreciar que no se puede encontrar una única condición lineal que separe las regiones correspondientes a los valores de salida 0 y 1, por lo que se dice que la XOR no es separable linealmente. Como la neurona del perceptrón representa en el fondo un discriminador lineal, esta neurona por sí sola no puede implementar la función XOR. Por lo tanto, concluimos con que *la clase de funciones no separables linealmente no puede ser representada por un perceptrón simple*.

Por lo tanto, pese a su gran interés, el perceptrón presenta serias limitaciones, pues solamente puede representar funciones linealmente separables. Así, aunque pueda aprender automáticamente a representar complejas funciones booleanas o resolver con éxito muchos problemas de clasificación (mediante el algoritmo que expondremos más adelante), en otras ocasiones fallará estrepitosamente.

Minsky (uno de los padres de la IA) y Papert [Minsky 69] estudiaron en profundidad el perceptrón, y en 1969 publicaron un exhaustivo trabajo en el que se subrayaba sus limitaciones, lo que resultó decisivo para que muchos de los recursos que se estaban invirtiendo en redes neuronales se desviasen hacia otros campos más prometedores entonces, como era en la época el de la inteligencia artificial.

Reflexionemos un poco sobre el problema de la función XOR para intentar encontrar una solución. Una neurona tipo perceptrón implementa una decisión lineal, observando la Figura 2.4 podemos considerar dos neuronas perceptrón, una implementa la decisión lineal DL1, y la otra la DL2. Consideremos una capa adicional, compuesta por una única neurona perceptrón encargada de componer las regiones en las que el plano queda dividido por las dos neuronas anteriores: si esta neurona se activa únicamente cuando la neurona correspondiente a DL1 está activada y DL2 desactivada, tendremos una red de tres capas (una de ellas oculta, es decir, sin conexión directa al exterior) que implementa la función XOR. Luego una solución a las limitaciones del perceptrón simple puede consistir en incluir más capas en la arquitectura, con lo que tendremos un **perceptrón multicapa**.

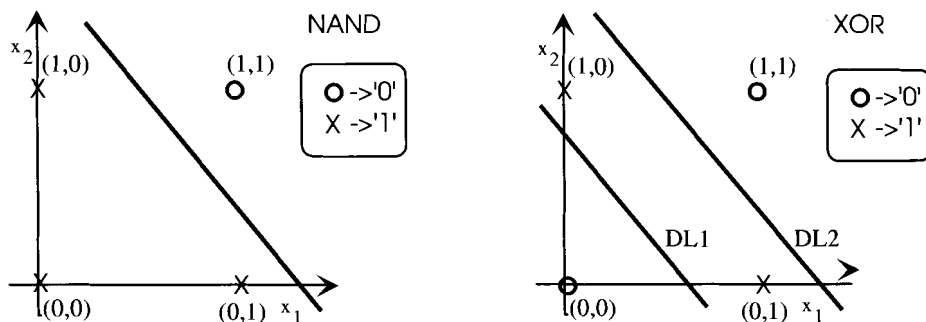


Figura 2.4 Funciones lógicas NAND (a) y XOR (b)

Los tipos de regiones de decisión que pueden formarse mediante estructuras simples y multicapa se muestran en la Figura 2.5. El problema a resolver en este caso es la discriminación entre dos clases de patrones, la clase A y la B. Los contornos que se forman con estructuras multicapa son polinomiales, puesto que consisten en la composición de discriminaciones lineales correspondientes a diferentes neuronas de tipo umbral. Si consideramos neuronas de respuesta continua, por ejemplo de tipo sigmoideo, los contornos serían similares, aunque sin esquinas. De la figura podemos apreciar que mediante una estructura de dos capas, sin capa oculta (la que corresponde a un perceptrón simple), la región de decisión es un hiperplano que separa en dos el espacio de las variables. Haciendo uso de tres capas, con una oculta, se pueden discriminar regiones convexas, sean cerradas o abiertas. Con una estructura de cuatro capas, dos de ellas ocultas, se puede discriminar regiones de forma arbitraria, cuyo único límite viene impuesto por el número de nodos empleados.

Por ejemplo, en [Müller 90] pueden encontrarse demostraciones más formales relacionadas con los conceptos expuestos. Por ejemplo, allí se demuestra que *toda función booleana puede ser representada por una red neuronal unidireccional con una única capa oculta*, lo que fue demostrado por primera vez por Denker y otros a mediados de los ochenta [Denker 87].

A finales de los sesenta ya se apuntaba como solución a las limitaciones del perceptrón introducir capas ocultas, pero el problema residía en que si bien se disponía de un algoritmo de aprendizaje para el perceptrón simple (el denominado algoritmo del perceptrón), no se disponía de ningún procedimiento que permitiese obtener automáticamente los pesos en una multicapa, con neuronas ocultas. Este problema, denominado de "asignación de crédito" a las neuronas sin conexión directa con el exterior (consistente en cómo medir la contribución al error en la salida de la red neuronal de cada uno de los nodos ocultos que precisamente no tienen una conexión directa con ella) fue resuelto no mucho más tarde por Paul Werbos [Werbos 74], pero fue preciso esperar hasta mediados de los años ochenta para que el grupo PDP (junto con otros grupos de forma independiente) redescubriera un algoritmo similar, que denominaron *back-propagation* o BP [Rumelhart 86a], y diera a conocer a la comunidad internacional su gran potencial para la resolución de problemas prácticos.

2.3.1 Algoritmo de aprendizaje del perceptrón

La importancia del perceptrón radica en su carácter de dispositivo entrenable, pues el algoritmo de aprendizaje introducido por Rosenblatt permite que el perceptrón determine automáticamente los pesos sinápticos que clasifican un determinado conjunto de patrones etiquetados.

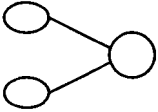
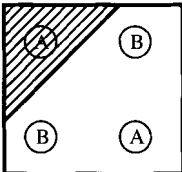
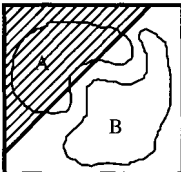
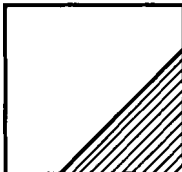
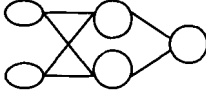
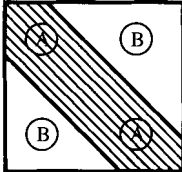
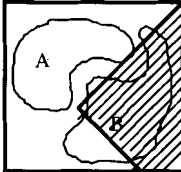
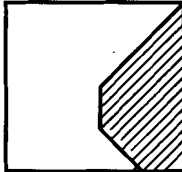
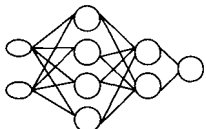
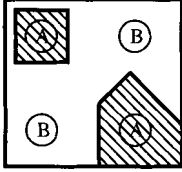
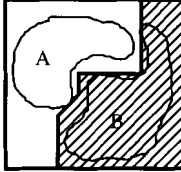
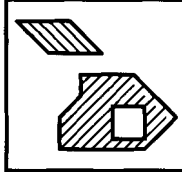
Arquitectura	Región de decisión	Ejemplo 1: XOR	Ejemplo 2: clasificación	Regiones más generales
Sin capa oculta 	Hiperplano (dos regiones)			
Una capa oculta 	Regiones polinomiales convexas			
Dos capas ocultas 	Regiones arbitrarias			

Figura 2.5 Tipos de regiones de decisión en el perceptrón [Lippmann 87]

El del perceptrón [Rosenblatt 62, Hertz 91] es un algoritmo de aprendizaje de los denominados por **corrección de errores**. Los algoritmos de este tipo (en el que incluiríamos también el de la adalina y el BP) ajustan los pesos en proporción a la diferencia existente entre la salida actual de la red y la salida deseada, con el objetivo de minimizar el error actual de la red.

Introduciremos sin más dilación la regla de aprendizaje. Sea un conjunto de p patrones $\mathbf{x}^\mu$, $\mu=1, \dots, p$, con sus salidas deseadas $\mathbf{t}^\mu$. Tanto las entradas como las salidas solamente pueden tomar los valores -1 o 1 (o bien, 0 o 1, según definamos los niveles lógicos). Se tiene una arquitectura de perceptrón simple, con pesos iniciales aleatorios, y se requiere que clasifique correctamente todos los patrones del conjunto de aprendizaje (lo cual es posible solamente si son separables linealmente). Actuaremos del siguiente modo, ante la presentación del patrón μ -ésimo, si la respuesta que proporciona el perceptrón es correcta, no actualizaremos los pesos; si es incorrecta, los modificaremos según la regla de Hebb de la sección 2.2. Se tiene

$$\Delta w_{ij}^\mu(t) = \begin{cases} 2\epsilon I_i^\mu x_j^\mu, & \text{si } y_i^\mu \neq t_i^\mu \\ 0, & \text{si } y_i^\mu = t_i^\mu \end{cases} \quad (2.20)$$

que se puede reescribir del siguiente modo

$$\Delta w_{ij}^{\mu}(t) = \varepsilon \cdot (t_i^{\mu} - y_i^{\mu}) x_j^{\mu} \quad (2.21)$$

que es la forma habitual de expresar la **regla del perceptrón**. En su utilización práctica, se debe llegar a un compromiso para el valor del ritmo de aprendizaje ε , puesto que un valor pequeño implica un aprendizaje lento, mientras que uno excesivamente grande puede conducir a oscilaciones en el entrenamiento, al introducir variaciones en los pesos excesivamente amplias. Al ser las entradas y las salidas discretas $\{-1, +1\}$, también lo será la actualización de los pesos (2.21), que únicamente podrá tomar los valores 0 o $\pm 2\varepsilon$.

Una forma mucho más gráfica de introducir la regla del perceptrón es la siguiente. Sea la neurona i tipo perceptrón $\{-1, +1\}$, cuyo vector de pesos es $\mathbf{w}_i$. Se presenta el patrón de entrada $\mathbf{x}^{\mu}$, la salida objetivo de la neurona i ante este patrón es t_i^{μ} . La operación de la neurona la escribimos como

$$y_i^{\mu}(t) = \text{signo}\left(\sum_{j=1}^n w_{ij} x_j^{\mu} - \theta_i\right) = \text{signo}(\mathbf{w}_i \cdot \mathbf{x}^{\mu}) = \text{signo}(\|\mathbf{w}_i\| \cdot \|\mathbf{x}^{\mu}\| \cos(\phi)) \quad (2.22)$$

considerando el umbral como un peso adicional de entrada -1 (véase el capítulo 1), y siendo ϕ el ángulo que forman los vectores de pesos y entradas. La hipersuperficie $\mathbf{w}_i \cdot \mathbf{x}^{\mu} = 0$ establece la condición lineal que separa el espacio en dos regiones, etiquetadas por -1 y +1, respectivamente. En el proceso de aprendizaje, ante la presentación del patrón μ -ésimo en la iteración t pueden darse los siguientes casos:

- a) La salida objetivo de la neurona es $t_i^{\mu} = +1$, pero su salida actual es $y_i^{\mu} = -1$. En este caso, el producto escalar $\mathbf{w}_i \cdot \mathbf{x}^{\mu}$ debería ser positivo, pero es negativo, lo cual indica que el ángulo existente entre $\mathbf{w}_i$ y $\mathbf{x}^{\mu}$ es mayor de 90° ($\phi \in [\pi/2, 3\pi/2]$, Figura 2.6). Así, la regla de aprendizaje del perceptrón debería en este caso acercar $\mathbf{w}_i$ a $\mathbf{x}^{\mu}$ para reducir el ángulo que forman, y eventualmente conseguir que sea inferior a 90° ($\mathbf{w}_i \cdot \mathbf{x}^{\mu} > 0$), lo cual se puede realizar del siguiente modo (véase la Figura 2.6a)

$$\mathbf{w}_i^{\mu}(t+1) = \mathbf{w}_i^{\mu}(t) + \alpha \cdot \mathbf{x}^{\mu} \quad (2.23)$$

- b) La salida objetivo de la neurona es $t_i^{\mu} = -1$, pero su salida actual es $y_i^{\mu} = +1$. Razonando al revés que en el caso anterior, la regla de aprendizaje deberá alejar $\mathbf{w}_i$ de $\mathbf{x}^{\mu}$, por lo tanto en este caso (Figura 2.6b)

$$\mathbf{w}_i^{\mu}(t+1) = \mathbf{w}_i^{\mu}(t) - \alpha \cdot \mathbf{x}^{\mu} \quad (2.24)$$

- c) La salida objetivo de la neurona t_i^{μ} coincide con su salida actual y_i^{μ} . En este caso la regla de aprendizaje no actúa.

Es fácil comprobar que los tres casos se resumen en la siguiente regla

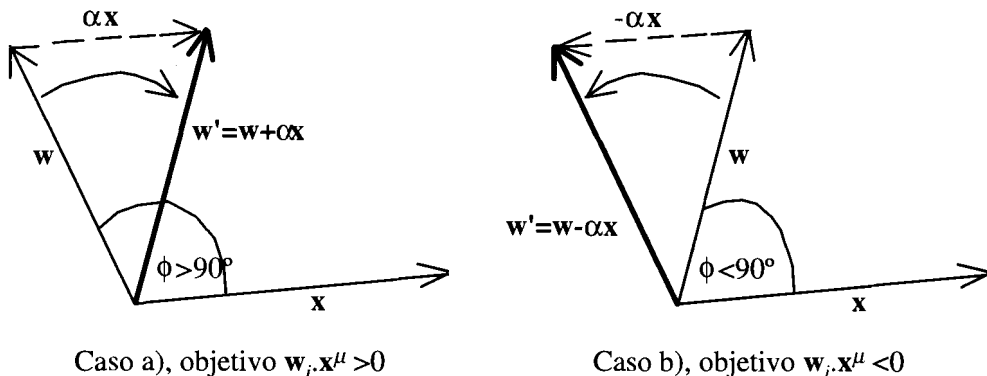


Figura 2.6. Regla del perceptrón, cuando salida actual y objetivo no coinciden

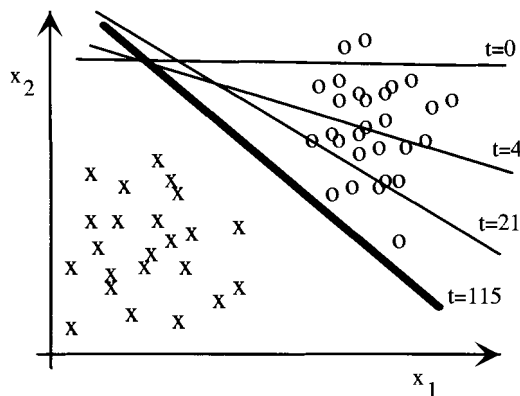


Figura 2.7 Regiones de decisión que establece iterativamente el perceptrón durante el aprendizaje (en la iteración 115 ha conseguido separar ya las dos clases)

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + (\alpha/2) \cdot \mathbf{x}^\mu(t_i^\mu - y_i^\mu) \quad (2.25)$$

y llamando $\varepsilon \equiv \alpha/2$, se tiene

$$\Delta \mathbf{w}_i(t) = \varepsilon \cdot \mathbf{x}^\mu(t_i^\mu - y_i^\mu) \quad (2.26)$$

que es la regla del perceptrón (2.21) ya conocida.

Es importante remarcar que el proceso de aprendizaje es iterativo: se parte de una configuración sináptica de partida (de pesos pequeños aleatorios, habitualmente), y se presentan una y otra vez los patrones, para que los pesos se ajusten iterativamente según (2.21), hasta que todos queden bien clasificados. El hiperplano que establece el límite entre dos clases se desplaza lentamente hasta conseguir separarlas por completo

(si ello es posible), como se puede apreciar en la Figura 2.7. El ajuste de los pesos en la iteración t debido a todo el conjunto de aprendizaje será

$$w_{ij}(t+1) = w_{ij}(t) + \sum_{\mu=1}^p \Delta w_{ij}^{\mu}(t) \quad (2.27)$$

Roseblatt demostró que si la función a representar es linealmente separable, este algoritmo siempre converge en un tiempo finito y con independencia de los pesos de partida. Por otra parte, si la función no es linealmente separable, el proceso de entrenamiento oscilará. Una prueba de la convergencia del algoritmo puede encontrarse, por ejemplo, en [Hertz 91]. Por otro lado, el algoritmo del perceptrón se detiene tan pronto como consigue clasificar correctamente todos los ejemplos, por lo que con frecuencia la línea de discriminación queda muy cerca de las muestras de uno de los grupos (en la Figura 2.7 ha quedado cerca de los patrones '0'). Para obtener una discriminación óptima ("en medio" de ambos grupos) se han introducido algoritmos como el denominado **Adatron** (véase, por ejemplo, [Principe 00]).

2.4 LA ADALINA (WIDROW, 1961)

Otro de los modelos clásicos es la **Adalina** (*Adaline*), introducida por Widrow en 1959 [Widrow 60, 88], cuyo nombre proviene de *ADaptive Linear Neuron*⁶. Este modelo utiliza una neurona similar a la del perceptrón, pero de respuesta lineal (Figura 2.8), cuyas entradas pueden ser continuas. Por otra parte, a diferencia del nodo del asociador lineal, el de la adalina incorpora un parámetro adicional denominado *bias*, que traduciremos como umbral, aunque debe tenerse en cuenta que no se trata de un umbral de disparo como el del perceptrón, sino de un parámetro que proporciona un grado de libertad adicional⁷. De este modo, la ecuación de la adalina queda

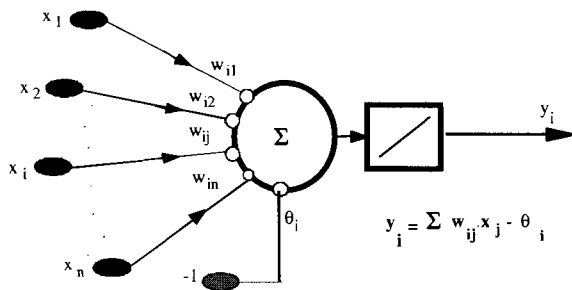


Figura 2.8 Neurona lineal de la adalina

⁶ Re-bautizado *ADaptive LInear Element* cuando los ANS perdieron popularidad, según comentó el propio Widrow en la conferencia que impartió en la *Neural Network Computation Conference* (Snowbird, Utah, 1987) [Anderson 88].

⁷ Un asociador lineal realiza combinaciones lineales de las entradas (rotaciones y dilataciones); la adalina, al incorporar un *bias*, realiza transformaciones afines (las cuales, además de las anteriores, incluyen traslaciones).

$$y_i(t) = \sum_{j=1}^n w_{ij} x_j - \theta_i, \quad \forall i, 1 \leq i \leq m \quad (2.28)$$

No obstante, la diferencia más importante con el perceptrón y con el asociador lineal reside en la regla de aprendizaje que implementa. En la adalina se utiliza la **regla de Widrow-Hoff**, también conocida como **regla LMS** (*Least Mean Squares*, mínimos cuadrados), que conduce a actualizaciones de tipo continuo, siendo la actualización de los pesos proporcional al error que la neurona comete.

Este ANS es un modelo muy conocido y ampliamente utilizado, aunque en ocasiones se hace más referencia a su carácter de dispositivo adaptativo lineal que a su naturaleza neuronal. La adalina se viene utilizando con asiduidad desde los años sesenta como filtro adaptativo, por ejemplo, para cancelar el ruido en la transmisión de señales (un ejemplo clásico es su empleo como supresor de ecos en las comunicaciones telefónicas por satélite [Widrow 88]; para el interesado en profundizar en el tema, una interesante introducción al tratamiento de señal con la adalina se expone en [Freemann 92]). De este modo, y desde hace años, millones de módems en todo el mundo incluyen una adalina.

Su utilidad se ve limitada por tratarse de un sistema lineal. Así, solamente podrá separar correctamente patrones linealmente independientes, fallando en ocasiones ante patrones linealmente separables, que el perceptrón siempre discrimina. No obstante, ante patrones no separables linealmente, los resultados que proporciona son en promedio mejores que los del perceptrón [Widrow 90, Hertz 90, Gallant 93], pues la adalina siempre opera reduciendo el error cuadrático medio al mínimo posible.

2.4.1 Regla LMS

La regla de Widrow-Hoff o LMS, que en un caso particular es conocida como regla delta, constituye el algoritmo de aprendizaje asociado a la adalina. Así como la regla de Hebb es capaz de almacenar sin errores pares de patrones cuyos vectores de entrada sean ortogonales, la regla LMS conducirá a asociaciones perfectas cuando sean linealmente independientes, proporcionando cuando no lo sean una matriz de pesos óptima desde el punto de vista de los mínimos cuadrados. Así, la regla delta puede considerarse en realidad como una versión iterativa aproximada de la basada en la pseudoinversa [Rumelhart 86a], para el caso de vectores estocásticos. No obstante, la forma de derivar la regla LMS a partir de la optimización de cierta función coste, que presentaremos a continuación, ilustrará la forma habitual de obtener algoritmos de aprendizaje en el campo de la computación neuronal.

Aprendizaje como optimización de una función coste

Expondremos en este punto una metodología que permita derivar de forma sistemática reglas de aprendizaje para arquitecturas concretas. El método consistirá en

proponer una función error o coste que mida el rendimiento actual de la red, función que dependerá de los pesos sinápticos. Dada esta función error, introduciremos un procedimiento general de optimización que sea capaz de proporcionar una configuración de pesos que correspondan a un extremal (en general, mínimo) de la función propuesta. El método de optimización aplicado a la función coste proporcionará una regla de actualización de pesos, que en función de los patrones de aprendizaje modifique iterativamente los pesos hasta alcanzar el punto óptimo de la red neuronal.

El método de optimización (minimización) más habitualmente empleado es el denominado **descenso por el gradiente**. Para ello, se comienza definiendo una función coste $E(.)$ que proporcione el error actual E que comete la red neuronal, que será una función del conjunto de pesos sinápticos W , $E=E(W)$, $E: \mathbb{R}^n \rightarrow \mathbb{R}$. De esta manera, podemos imaginarnos la representación gráfica de esta función, como una hipersuperficie con montañas y valles (Figura 2.9), en la que la posición ocupada por un valle se corresponde con una configuración de pesos W' localmente óptima, al tratarse de un mínimo local de la función error. El objetivo del aprendizaje será encontrar la configuración de pesos que corresponde al mínimo global de la función error, aunque con frecuencia en una red genérica deberemos conformarnos con un mínimo local suficientemente bueno.

Para encontrar la configuración de pesos óptima mediante descenso por el gradiente se opera del siguiente modo. Se parte en $t=0$ de una cierta configuración $W(0)$, y se calcula el sentido de la máxima variación de la función $E(W)$ en $W(0)$, que vendrá dado por su gradiente en $W(0)$. El sentido de la máxima variación (máximo gradiente) apuntará hacia una colina del paisaje de la hipersuperficie de $E(.)$. A continuación se modifican los parámetros W siguiendo el sentido contrario al indicado por el gradiente de la función error. De este modo se lleva a cabo un descenso por la hipersuperficie del error, aproximándose en una cierta cantidad al valle, un mínimo (local); el proceso se itera hasta alcanzarlo (véase [Rumelhart 86a] para más detalles). Matemáticamente, se expresa del siguiente modo

$$W(t+1) = W(t) - \varepsilon \cdot \nabla E(W) \quad (2.29)$$

donde ε (que puede ser diferente para cada peso) indica el tamaño del paso tomado en cada iteración, que idealmente debe ser infinitesimal. Como una elección de este tipo conduciría a un proceso de entrenamiento extremadamente lento, se toma de un tamaño lo suficientemente grande como para que cumpla el compromiso de rápida actualización sin llevar a oscilaciones (por ejemplo, en la Figura 2.9 se puede ver que una actualización excesivamente grande de los pesos nos llevaría lejos de nuestro objetivo, el mínimo).

Es fácil comprobar matemáticamente que, efectivamente, una actualización de este tipo conduce a un mínimo del funcional de error $E(W)$. Supongamos una matriz de pesos $W = \{w_{ij}\}$, y calculemos la variación que en $E(W)$ se produce en la iteración t

$$\delta(E(w_{ij})) = \sum_{ij} \frac{\partial E(w_{ij})}{\partial w_{ij}} \delta w_{ij} \quad (2.30)$$

pero, por (2.24), la variación en los pesos es $-\epsilon$ (idealmente infinitesimal, puesto que por el cálculo de δw_{ij} debe serlo) multiplicado por el gradiente, por lo tanto

$$\delta(f(w_{ij})) = \sum_{ij} \frac{\partial E(w_{ij})}{\partial w_{ij}} \left(-\epsilon \frac{\partial E(w_{ij})}{\partial w_{ij}} \right) = -\epsilon \sum_{ij} \left(\frac{\partial E(w_{ij})}{\partial w_{ij}} \right)^2 \leq 0 \quad (2.31)$$

luego la variación en la función error es siempre menor que cero, por lo que siempre disminuye. Mediante este procedimiento se asegura alcanzar un mínimo local de la función, aunque puede no coincidir, en general, con el mínimo global (veremos que en el caso de la adalina sí se cumple este hecho).

Primera derivación de la regla de Widrow-Hoff. Aproximación estocástica

A continuación, aplicaremos este método a la adalina, que por ser una red de tipo lineal permite un análisis teórico detallado. Más adelante lo aplicaremos al caso del perceptrón multicapa. La respuesta de una neurona de la adalina es lineal

$$y_i = \sum_{j=1}^n w_{ij} x_j - \theta_i, \quad \forall i, 1 \leq i \leq m \quad (2.32)$$

y sus salidas son continuas, por ejemplo en $[0, +1]$. Si definimos $w_{i0} = \theta_i$, y $x_0 = -1$, podemos reescribir esta expresión de la manera siguiente

$$y_i = \sum_{j=0}^n w_{ij} x_j = \mathbf{w}_i^T \mathbf{x} \quad (2.33)$$

de modo que podemos tratar al umbral θ_i como un peso adicional, pero cuya entrada es constante e igual a -1.

Deduciremos la regla de Widrow-Hoff en el marco del formalismo de la **aproximación estocástica**, que se introdujo para la resolución de complejos problemas de regresión no lineal en los que, o bien se desconoce la estadística del problema, o bien ésta es complicada (véase [Kohonen 89, White 89b] y referencias allí citadas).

Se dispone de un conjunto de patrones de entrada $\mathbf{x}$, y salidas deseadas $\mathbf{t}$. En general, son variables estocásticas que presentan una cierta dispersión estadística. En la situación más habitual, se desconoce la forma estadística del problema, solamente puede obtenerse un conjunto de muestras $(\mathbf{x}^\mu, \mathbf{t}^\mu)$, $\mu=1, \dots, p$.

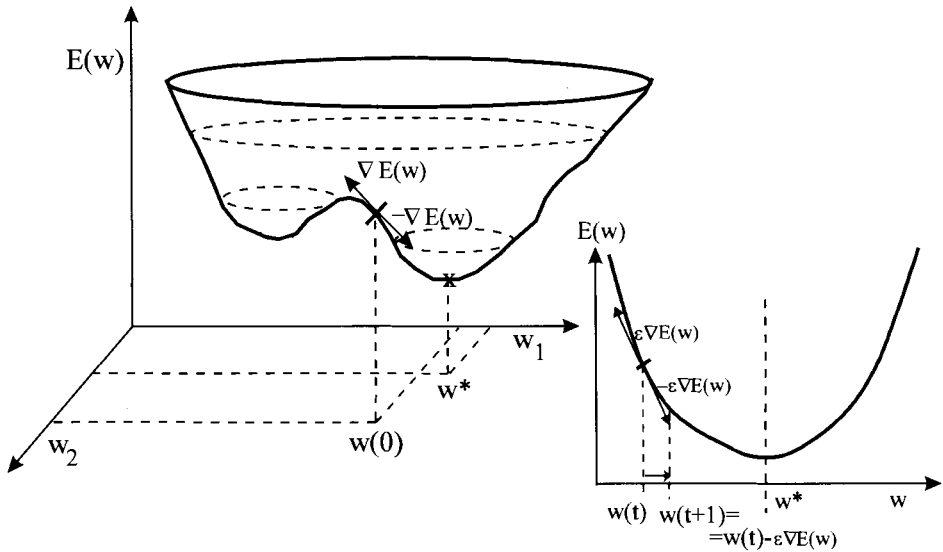


Figura 2.9 Superficie de error $E(w)$ en el espacio de los pesos w , y descenso por el gradiente hacia un mínimo (local)

Ante una entrada x , la adalina responde con una salida y dada por (2.28) que, en principio, no coincidirá con la salida deseada t . Planteamos la siguiente función error, que compara las salidas actuales con las objetivo, y que depende de los pesos de la red

$$J \equiv \langle E[w_{ij}] \rangle = \langle (t_i - y_i)^2 \rangle = \lim_{p \rightarrow \infty} (1/p) \sum_{\mu=1}^p \sum_{i=1}^n (t_i^\mu - y_i^\mu)^2 \quad (2.34)$$

Obsérvese que al tratar con variables estocásticas, el error se ha definido como un valor esperado. Aunque se ha utilizado como criterio de error el cuadrático medio (lo que se hace con asiduidad en los ANS), debe tenerse en cuenta que el error puede definirse también de otras maneras. Se puede decir que, en general, *la medida de error refleja las hipótesis realizadas sobre la forma del ruido presente en los datos*. El error anterior, como promedio de la suma de errores cuadráticos, supone que los errores de cada variable se distribuyen gaussianamente, y que éstos son independientes [Weigend 93]. De hecho, esta medida del error puede derivarse haciendo uso del principio de máxima verosimilitud a partir de la suposición de que el ruido en los datos del espacio de salida es gaussiano [White 89a]. Por ejemplo, en [Hertz 91, Weige 93] se muestra otro tipo de función error, basada en un criterio de **entropía**, y que se aplica cuando los patrones de salida son binarios, con lo que los errores se distribuirán binómicamente.

El problema que nos planteamos consiste en encontrar la configuración de pesos sinápticos w_{ij}^* que minimiza (2.34), es decir

$$\nabla J|_{w_{ij}=w_{ij}^*} = \nabla \langle E[w_{ij}] \rangle|_{w_{ij}=w_{ij}^*} = \frac{\partial \langle E[w_{ij}] \rangle}{\partial w_{ij}} \bigg|_{w_{ij}=w_{ij}^*} = 0 \quad (2.35)$$

Pero si la estadística que rige el fenómeno físico en estudio es desconocida, habrá que realizar los cálculos a partir de las muestras $(\mathbf{x}^\mu, \mathbf{t}^\mu)$. En general, la resolución directa de este conjunto de ecuaciones no es posible (en [Freemann 93] puede verse un caso muy simple en el que sí es factible), y hay que recurrir al método iterativo de descenso por el gradiente descrito (2.29), en el que la actualización en los pesos adquiere la forma

$$\Delta w_{ij} = -\varepsilon \nabla \langle E[w_{ij}] \rangle = -\varepsilon \frac{\partial \langle E[w_{ij}] \rangle}{\partial w_{ij}} \quad (2.36)$$

Calculando el gradiente

$$\begin{aligned} \nabla \langle E[w_{ij}] \rangle &= \nabla \left[\lim_{p \rightarrow \infty} (1/p) \sum_{\mu=1}^p \sum_{i=1}^n (t_i^\mu - y_i^\mu)^2 \right] = \lim_{p \rightarrow \infty} (1/p) \sum_{\mu=1}^p \sum_{i=1}^n \nabla [(t_i^\mu - y_i^\mu)^2] = \\ &= \lim_{p \rightarrow \infty} (1/p) \sum_{\mu=1}^p \sum_{i=1}^n 2(t_i^\mu - y_i^\mu) (-x_j^\mu) = -2 \langle (t_i - y_i) x_j \rangle \quad (2.37 \text{ y } 2.38) \end{aligned}$$

Por lo tanto, para estimar el valor esperado del error, habría que tomar una numerosa cantidad de muestras, y promediar. La idea de la aproximación estocástica (y la idea que Widrow y Hoff propusieron) consiste en lo siguiente: en vez de utilizar un valor promedio del gradiente de la función error $\langle E[w_{ij}] \rangle$, se aproximará su valor en cada iteración por el valor concreto proporcionado por el par $(\mathbf{x}^\mu, \mathbf{t}^\mu)$ actualmente presentado a la red, es decir

$$\nabla \langle E[w_{ij}] \rangle \approx -2 \langle (t_i - y_i) x_j \rangle \approx -2(t_i^\mu - y_i^\mu) x_j^\mu \quad (2.39)$$

El promedio se irá realizando de un modo automático durante el transcurso de las iteraciones, empleando un conjunto amplio de patrones de aprendizaje. De este modo, se van tomando muestras $(\mathbf{x}^\mu, \mathbf{t}^\mu)$, y se actualizan los pesos de la forma

$$\Delta w_{ij} = -\varepsilon \nabla \langle E[w_{ij}] \rangle \approx -\varepsilon [-2(t_i^\mu - y_i^\mu) x_j^\mu] = \alpha (t_i^\mu - y_i^\mu) x_j^\mu \quad (2.40)$$

con α el ritmo de aprendizaje (que puede depender de la iteración, $\alpha = \alpha(t)$).

La actualización de los pesos queda

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} = w_{ij}(t) + \alpha (t_i^\mu - y_i^\mu) x_j^\mu \quad (2.41)$$

Ésta es la **regla LMS** o de **Widrow-Hoff**, que puede considerarse la versión iterativa de la regla de la pseudoinversa para el caso de vectores estocásticos. Ambas reglas proporcionan resultados asintóticamente coincidentes.

Si en lugar de considerar la función activación lineal la consideramos sigmoidea, el algoritmo de aprendizaje se denomina **regla delta** [Widrow 90]. Obsérvese que aunque hemos encontrado una forma de actualizar los pesos, esta misma regla se aplica a los umbrales, considerándolos como pesos especiales, con entrada constante e igual a -1.

Widrow y Hoff demostraron [Widrow 60] la convergencia del algoritmo (2.41) a la configuración w_{ij}^* que minimiza el valor esperado del error, a condición de que $\alpha(t)$ cumpla

$$\sum_{t=1}^{\infty} \alpha(t) = \infty, \quad \sum_{t=1}^{\infty} \alpha^2(t) < \infty \quad (2.42)$$

por ejemplo $\alpha = \alpha(t) = t^{-1}$ satisface ambas condiciones [Kohonen 89]. En muchas ocasiones es suficiente con que α tome un valor pequeño ($0 < \alpha < 1$). La interpretación de las condiciones (2.42) es que éstas garantizan que el aprendizaje no se lleve a cabo ni excesivamente rápido ni muy lentamente. Como señaló Grossberg [Grossberg 82], el sistema que aprende debe ser suficientemente estable como para recordar los patrones antiguos (un ritmo de aprendizaje excesivamente grande los *borraría*, pues la presente actualización sería de gran magnitud), pero suficientemente plástico como para aprender los nuevos (un ritmo muy pequeño, provoca actualizaciones diminutas, por lo que el proceso de entrenamiento se dilataría excesivamente en el tiempo). Esto constituye el denominado **dilema de la plasticidad frente a la estabilidad** [Grossberg 82]. La primera condición (2.42) asegura que el sistema sea plástico, mientras que la segunda asegura la condición de estabilidad.

Una deducción simple de la regla de Widrow-Hoff

Vamos a exponer ahora una formulación alternativa que, por no considerar explícitamente la estocasticidad del problema, resultará mucho más simple. Para una muestra finita, planteamos la siguiente función error u objetivo

$$E[w_{ij}] = \frac{1}{2} \sum_{\mu=1}^p \sum_{i=1}^n \left(t_i^{\mu} - y_i^{\mu} \right)^2 \quad (2.43)$$

Mediante esta función se obtiene el error cuadrático medio correspondiente a las salidas actuales de la red respecto de los objetivos. El proceso de optimización es, de nuevo, el del descenso por el gradiente. Para ello calculamos

$$\frac{\partial E[w_{ij}]}{\partial w_{ij}} = -(1/2) \cdot 2 \sum_{\mu=1}^p (t_i^{\mu} - y_i^{\mu}) \frac{dy_i^{\mu}}{dw_{ij}} = - \sum_{\mu=1}^p (t_i^{\mu} - y_i^{\mu}) x_j^{\mu} \quad (2.44)$$

y el incremento en los pesos queda

$$\Delta w_{ij} = -\varepsilon \frac{\partial E[w_{ij}]}{\partial w_{ij}} = \varepsilon \sum_{\mu=1}^p (t_i^{\mu} - y_i^{\mu}) x_j^{\mu} \quad (2.45)$$

Esta expresión es, de nuevo, la regla LMS. Conviene observar que mientras que en la regla del perceptrón se llevan a cabo actualizaciones discretas en los pesos, en la adalina la regla LMS produce actualizaciones de tipo continuo, de modo que a un mayor error se tiene una actualización mayor. Otra diferencia entre ambos algoritmos es que la regla del perceptrón converge en un número finito de iteraciones (en cuanto consigue clasificar correctamente todos los patrones), mientras que la regla LMS se acerca asintóticamente a la solución, pues el tamaño de los incrementos se hace cada vez menor. Es importante remarcar que ante patrones no linealmente separables, la adalina proporciona mejores resultados que el perceptrón [Widrow 90, Hertz 90], pues realiza un ajuste óptimo en el sentido de los mínimos cuadrados, mientras que el perceptrón no alcanzará ninguna solución.

Debido a la linealidad de la neurona de la adalina, la función error $E[w_{ij}]$ es cuadrática en los pesos, por lo que define una superficie en forma de paraboloide (una demostración rigurosa aparece, por ejemplo, en [Hecht-Nielsen 90]). Un paraboloide, como la parábola en el plano, usualmente posee un único mínimo, aunque en ocasiones puede presentar una forma degenerada, con uno o más *canales*, pero todos de la misma profundidad. En cualquiera de los dos casos la función $E[w_{ij}]$ es mínima en ese punto o en cualquiera de los de los canales, y la regla (2.45) nos lleva directamente a él, puesto que siempre desciende por la superficie de error [Hecht-Nielsen 90]. Por ello, *la regla LMS alcanza siempre el mínimo global*, sin importar la configuración de pesos de partida, constituyendo uno de los pocos casos en redes neuronales en el que se puede realizar una afirmación de este tipo.

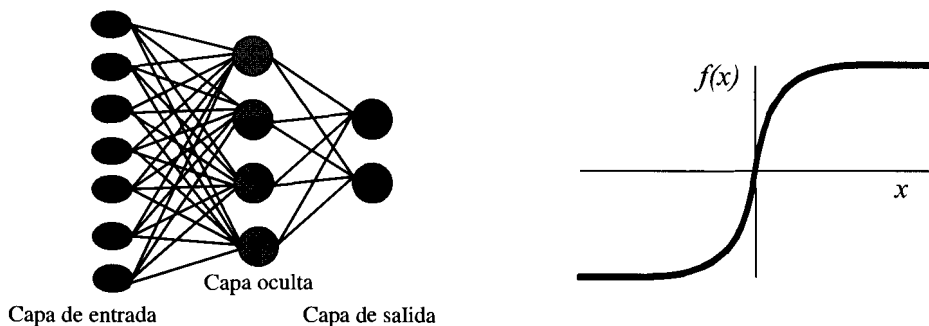


Figura 2.10 Perceptrón Multicapa y función de transferencia de la neurona

Por último, no queremos concluir la sección dedicada a la adalina sin citar también la denominada **madalina** (*madaline=many adalines*, es decir *muchas adalinas*), que constituye la versión multicapa de la adalina [Widrow 88, Freeman 92], introducida para superar los problemas de la adalina, en cuanto que red monocapa. Es decir, podemos afirmar que la madalina es a la adalina lo que el perceptrón simple es al multicapa. Recomendamos las referencias citadas al lector interesado en estudiar más ampliamente el modelo.

2.5 EL PERCEPTRÓN MULTICAPA (GRUPO PDP, 1986)

Si añadimos capas intermedias (ocultas) a un perceptrón simple, obtendremos un perceptrón multicapa o MLP (*Multi-Layer Perceptron*). Esta arquitectura suele entrenarse mediante el algoritmo denominado retropropagación de errores o BP, o bien haciendo uso de alguna de sus variantes o derivados, motivo por el que en muchas ocasiones el conjunto *arquitectura MLP + aprendizaje BP* suele denominarse **red de retropropagación**, o simplemente BP.

Como se describe en [Hecht-Nielsen 90], el proceso de desarrollo del BP resulta una curiosa historia de redescubrimientos y olvidos. Al parecer, fue Werbos quien introdujo por primera vez el BP en su tesis doctoral en 1974 [Werbos 74], pero el hecho no tuvo demasiada repercusión en la época. Años más tarde, hacia 1984, el BP fue redescubierto por D. Parker, y casi a la vez (1985) por el grupo del PDP (Rumelhart, Hinton, MacClelland..., [Rumelhart 86b, 86a]), quienes realmente lo popularizaron. Además, existe un procedimiento matemático recursivo empleado en control, de apariencia similar al BP, que data de 1969.

Pese a todo, el mérito del éxito del BP se debe al trabajo del grupo PDP, que lo presentaron a la comunidad internacional como una técnica útil de resolución de problemas complejos [Rumelhart 86a], lo que despertó el interés, no sólo por el perceptrón, sino por el campo de la neurocomputación en general. Los importantes requisitos de cómputo que el algoritmo BP precisa no podían ser satisfechos con los medios disponibles a principios de los setenta, por lo que el *primer descubrimiento* del BP [Werbos 74] era quizás algo prematuro. Por fin en los años ochenta los computadores eran suficientemente potentes como para permitir la aplicación del BP a problemas de interés, lo cual permitió que el grupo PDP pudiera mostrar su gran potencial de aplicabilidad a la resolución de tareas complejas.

La estructura del MLP se presenta en las Figuras 2.10 y 2.11. Denominaremos x_i a las entradas de la red, y_j a las salidas de la capa oculta y z_k a las de la capa final (y globales de la red); t_k serán las salidas objetivo (*target*). Por otro lado, w_{ij} son los pesos de la capa oculta y θ_j sus umbrales, w'_{kj} los pesos de la capa de salida y θ'_k sus umbrales. La operación de un MLP con una capa oculta y neuronas de salida lineal (estructura que constituye, como veremos, un aproximador universal de funciones) se expresa matemáticamente de la siguiente manera

$$z_k = \sum_j w'_{kj} y_j - \theta'_i = \sum_j w'_{kj} f\left(\sum_i w_{ji} x_i - \theta_j\right) - \theta'_i \quad (2.46)$$

siendo $f(\cdot)$ de tipo sigmoideo (Figura 2.10), como por ejemplo, las siguientes

$$f(x) = \frac{1}{1 + e^{-x}} \quad (2.47a)$$

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \tanh(x) \quad (2.47b)$$

proporcionando la primera una salida en el intervalo $[0, +1]$, y en el $[-1, +1]$ la segunda.

Ésta es la arquitectura más común de MLP, aunque existen numerosas variantes, como incluir neuronas no lineales en la capa de salida (del mismo tipo que las (2.47), solución que se adopta especialmente en problemas de clasificación), introducir más capas ocultas, emplear otras funciones de activación, limitar el número de conexiones entre una neurona y las de la capa siguiente, introducir dependencias temporales o arquitecturas recurrentes [Werbos 90], etc.

2.5.1 El MLP como aproximador universal de funciones

El desarrollo del MLP durante los últimos treinta años ha resultado curioso. Partiendo de un perceptrón monocapa y observando sus limitaciones computacionales, se llegó a la arquitectura perceptrón multicapa, y aplicándolo a numerosos problemas, se comprobó experimentalmente que éste era capaz de representar complejos *mappings* y de abordar problemas de clasificación de gran envergadura, de una manera eficaz y relativamente simple. Sin embargo, faltaba una demostración teórica que permitiese explicar sus aparentemente enormes capacidades computacionales.

Este proceso histórico comienza con McCulloch y Pitts, quienes mostraron [McCulloch 43] que mediante su modelo de neurona (esencialmente un dispositivo de umbral) podría representarse cualquier función booleana; mucho más tarde, Denker y otros [Denker 87] demostraron que toda función booleana podía ser representada por una red unidireccional multicapa de una sola capa oculta. Por las mismas fechas, Lippmann [Lippmann 87] mostró que un perceptrón con dos capas ocultas bastaba para representar regiones de decisión arbitrariamente complejas (véase la sección 1.3). Por otra parte, Lapedes y Farber demostraron [Lapedes 87] que un perceptrón de dos capas ocultas es suficiente para representar cualquier función arbitraria (no necesariamente booleana). Más tarde, Hecht-Nielsen [Hecht-Nielsen 87, 90] aplicando el teorema de Kolmogorov demostró que una arquitectura de características similares al MLP, con una única capa oculta, resultaba ser un aproximador universal de funciones. Por fin, a finales de la década, diversos grupos propusieron casi a la par teoremas muy similares que demostraban matemáticamente que un MLP convencional, de una única capa oculta (ecuación 2.46), constituía, en efecto, un aproximador universal de funciones [Funahashi 89, Hornik 89]. A título de ejemplo, enunciaremos uno de estos teoremas.

Teorema [Funahashi 89]. Sea $f(x)$ una función no constante, acotada y monótona creciente. Sea K un subconjunto compacto (acotado y cerrado) de $\mathfrak{R}^n$. Sea un número real $\varepsilon \in \mathfrak{R}$, y sea un entero $k \in \mathbb{Z}$, tal que $k \geq 3$, que fijamos. En estas condiciones, se tiene que:

Cualquier *mapping* $\mathbf{g}: \mathbf{x} \in K \rightarrow (g_1(\mathbf{x}), g_2(\mathbf{x}), \dots, g_m(\mathbf{x})) \in \mathfrak{R}^m$, con $g_i(\mathbf{x})$ sumables en K , puede ser aproximado en el sentido de la topología L_2 en K por el *mapping* entrada-salida representado por una red neuronal unidireccional (MLP) de k capas ($k-2$ ocultas), con $f(x)$ como función de transferencia de las neuronas ocultas, y funciones lineales para las de las capas de entrada y de salida. En otras palabras:

$\forall \varepsilon > 0, \exists$ un MLP de las características anteriores, que implementa el mapping

$$\mathbf{g}': \mathbf{x} \in K \rightarrow (g'_1(\mathbf{x}), g'_2(\mathbf{x}), \dots, g'_m(\mathbf{x})) \in \mathfrak{R}_m \quad (2.48)$$

de manera que

$$d_{L_2(K)}(\mathbf{g}, \mathbf{g}') = \left(\sum_{i=1}^m \int_K |g_i(x_1, \dots, x_n) - g'_i(x_1, \dots, x_n)|^2 d\mathbf{x} \right)^{1/2} < \varepsilon \quad (2.49)$$

#

Es fácil observar que las funciones sigmoideas empleadas habitualmente en el MLP (ecuación (2.47)) cumplen las condiciones exigidas a $f(x)$. En [Hornik 89] se llega a un resultado similar, considerando funciones de activación sigmoideas, no necesariamente continuas.

En resumen, *un MLP de una única capa oculta puede aproximar hasta el nivel deseado cualquier función continua en un intervalo*⁸, por lo tanto, las redes neuronales multicapa unidireccionales son **aproximadores universales de funciones**. A partir de la expresión que define la operación de este tipo de red

$$g_k'(\mathbf{x}) = \sum_j w'_{kj} y_j - \theta'_i = \sum_j w'_{kj} f\left(\sum_i w_{ji} x_i - \theta_j\right) - \theta'_i \quad (2.50)$$

podemos observar que la $\mathbf{g}'(\mathbf{x})$ dada por el MLP representa una cierta función $\mathbf{g}(\mathbf{x})$, como un desarrollo en funciones sigmoideas $f(x)$, lo cual posee una clara analogía con la representación convencional de una función periódica como un desarrollo en serie de Fourier de sinusoides [Principe 00]. También se han establecido paralelismos entre el MLP y otros tipos de transformaciones, como la de Gabor o las *wavelets*.

Los teoremas citados resultan de vital importancia, puesto que proporcionan una sólida base teórica al campo de las redes neuronales, al incidir sobre un aspecto (la

⁸ El teorema permite elegir k con la restricción $k \geq 3$; si se elige $k=3$ se tiene una sola capa oculta. No obstante, pueden emplearse más capas ocultas, obteniéndose en ocasiones resultados más eficientes o una mejor generalización.

aproximación funcional) y un modelo (el MLP) centrales en la teoría de las redes neuronales artificiales. No obstante, todavía quedan muchos asuntos abiertos. Por ejemplo, estos teoremas no informan sobre el número de nodos ocultos necesarios para aproximar una función determinada, simplemente se afirma que hay que colocar los necesarios para lograr el nivel de aproximación requerido [Hornik 89]. Para un problema concreto, muy bien pudiera ocurrir que el número de neuronas ocultas para alcanzar una cierta cota de error sea tan elevado que su aplicación resulte inabordable en la práctica. Este tipo de cuestiones están siendo investigadas intensamente; en [Haykin 99, Principe 00] se puede encontrar un más amplio (y actualizado) estudio sobre este importante asunto.

2.5.2 Aprendizaje por retropropagación de errores (BP)

Una solución al problema de entrenar los nodos de las capas ocultas pertenecientes a arquitecturas multicapa la proporciona el algoritmo de retropropagación de errores o BP (*backpropagation*) [Rumelhart 86a, 86b, Hecht-Nielsen 91].

En el marco conceptual que estamos describiendo la deducción del BP aparece como una consecuencia natural de extender el algoritmo LMS a las redes multicapa (aunque transcurrieron muchos años antes de que se llegase a esta conclusión). Para ello, se planteará un funcional de error similar al (2.43), y se derivará, no solo en función de los pesos de la capa de salida, sino también en función de los pesos de las neuronas ocultas, haciendo uso de la regla de la cadena; en consecuencia, habrá que exigir que las funciones de transferencia de las neuronas sean derivables.

Sea un MLP de tres capas, cuya arquitectura se presenta en la Figura 2.11, con las entradas, salidas, pesos y umbrales de las neuronas definidas en la sección anterior. Dado un patrón de entrada $\mathbf{x}^\mu$, ($\mu=1, \dots, p$), recordemos que la operación global de esta arquitectura se expresa del siguiente modo

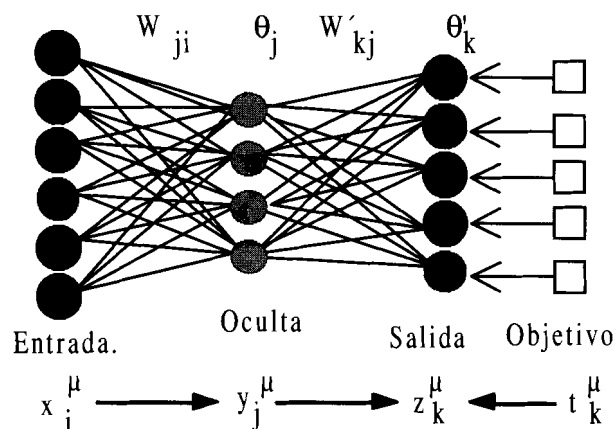


Figura 2.11 Arquitectura del MLP

$$z_k^\mu = \sum_j w'_{kj} y_j^\mu - \theta'_k = \sum_j w'_{kj} f\left(\sum_i w_{ji} x_i^\mu - \theta_j\right) - \theta'_k \quad (2.51)$$

Las funciones de activación de las neuronas ocultas $f(h)$ son de tipo sigmoideo, con h el potencial postsináptico o local, siendo típicas las (2.47). Como en el caso de la adalina, la función coste de la que se parte es el error cuadrático medio

$$E(w_{ji}, \theta_j, w'_{kj}, \theta'_k) = (1/2) \sum_\mu \sum_k \left[t_k^\mu - f\left(\sum_j w'_{kj} y_j^\mu - \theta'_k\right) \right]^2 \quad (2.52)$$

La minimización se lleva a cabo mediante descenso por el gradiente, pero en esta ocasión habrá un gradiente respecto de los pesos de la capa de salida y otro respecto de los de la oculta

$$\delta w'_{kj} = -\varepsilon \frac{\partial E}{\partial w'_{kj}} \quad \delta w_{ji} = -\varepsilon \frac{\partial E}{\partial w_{ji}} \quad (2.53)$$

Las expresiones de actualización de los pesos se obtienen sólo con derivar, teniendo en cuenta las dependencias funcionales y aplicando adecuadamente la regla de la cadena

$$\delta w'_{kj} = \varepsilon \sum_\mu \Delta'^\mu_k y_j^\mu, \text{ con } \Delta'^\mu_k = [t_k^\mu - f(v'^\mu_k)] \frac{\partial f(v'^\mu_k)}{\partial v'^\mu_k} \quad (2.54)$$

$$\delta w_{ji} = \varepsilon \sum_\mu \Delta^\mu_j x_i^\mu, \text{ con } \Delta^\mu_j = \left(\sum_k \Delta'^\mu_k w'_{kj} \right) \frac{\partial f(v_j^\mu)}{\partial v_j^\mu} \quad (2.55)$$

La actualización de los umbrales (*bias*) se realiza haciendo uso de estas mismas expresiones, considerando que el umbral es un caso particular de peso sináptico, cuya entrada es una constante igual a -1, como vimos ya en la adalina (sección 2.4.1)

En estas expresiones está implícito el concepto de propagación hacia atrás de los errores que da nombre al algoritmo. En primer lugar se calcula la expresión Δ'^μ_k (2.54), que denominaremos **señal de error**, por ser proporcional al error de la salida actual de la red, con el que calculamos la actualización $\delta w'_{kj}$ de los pesos de la capa de salida. A continuación se *propagan hacia atrás* los errores Δ'^μ_k a través de las sinapsis, proporcionando así las señales de error Δ^μ_j (2.55), correspondientes a las sinapsis de la capa oculta; con éstas se calcula la actualización δw_{ji} de las sinapsis ocultas. El algoritmo puede extenderse fácilmente a arquitecturas con más de una capa oculta siguiendo el mismo esquema.

En resumen, el **procedimiento** a seguir para entrenar mediante BP una arquitectura MLP dada es el siguiente:

- 1) Establecer aleatoriamente los pesos y umbrales iniciales ($t:=0$).
- 2) Para cada patrón μ del conjunto de aprendizaje
 - 2.1) Llevar a cabo una fase de ejecución para obtener la respuesta de la red ante el patrón μ -ésimo (2.51)
 - 2.2) Calcular las señales de error asociadas Δ^{μ}_k y Δ^{μ}_j según (2.54-55).
 - 2.3) Calcular el incremento parcial de los pesos y umbrales debidos a cada patrón μ (elemento de los sumatorios (2.54 y 2.55))
- 3) Calcular el incremento total (para todos los patrones) actual de los pesos $\delta w'_{kj}$ y δw_{ji} según (2.54-55). Hacer lo mismo para los umbrales.
- 4) Actualizar pesos y umbrales.
- 5) Calcular el error actual (2.52), $t:=t+1$, y volver a 2) si todavía no es satisfactorio.

Se debe comenzar siempre con **pesos iniciales** aleatorios (normalmente números pequeños, positivos y negativos), ya que si se parte de pesos y umbrales iniciales nulos el aprendizaje no progresará (puesto que las salidas de las neuronas y el incremento en los pesos serán siempre nulos). En la siguiente sección se explicará una heurística que permite elegir unos pesos iniciales adecuados.

En el esquema presentado, que surge de forma natural del proceso de descenso por el gradiente, se lleva a cabo una fase de ejecución para todos y cada uno de los patrones del conjunto de entrenamiento, se calcula la variación en los pesos debida a cada patrón, se acumulan, y solamente entonces se procede a la actualización de los pesos. Este esquema se suele denominar **aprendizaje por lotes** (*batch*). Una variación común al algoritmo consiste en actualizar los pesos sinápticos tras la presentación de cada patrón (en vez de presentarlos todos y luego actualizar), esquema denominado **aprendizaje en serie** (*on line*). Aunque el *verdadero* BP es el que se ejecuta por lotes, el aprendizaje en serie es habitualmente empleado en aquellos problemas en los que se dispone de un muy numeroso conjunto de patrones de entrenamiento (compuesto por cientos o miles de patrones), en el que habrá mucha redundancia en los datos. Si se emplease en este caso el modo por lotes, el tener que procesar todos los patrones antes de actualizar los pesos demoraría considerablemente el entrenamiento (además de precisar el almacenamiento de numerosos resultados parciales). Por ejemplo, podemos imaginar un conjunto de entrenamiento compuesto por 10.000 patrones, en el que cada patrón aparece repetido cien veces, entrenando por lotes el aprendizaje duraría cien veces más que en modo serie.

En el aprendizaje en serie se debe tener presente que *el orden en la presentación de los patrones debe ser aleatorio*, puesto que si siempre se siguiese un mismo orden el entrenamiento estaría viciado en favor del último patrón del conjunto de entrenamiento, cuya actualización, por ser la última, siempre predominaría sobre

las anteriores. Además, esta aleatoriedad presenta una importante ventaja, puesto que puede permitir escapar de mínimos locales en determinadas ocasiones, por lo que al final del proceso puede alcanzarse un mínimo más profundo [Bishop 94].

EL BP, sea en versión por lotes o en serie, constituye un método de gran generalidad, lo que presenta ventajas e inconvenientes. Su ventaja principal es que se puede aplicar a multitud de problemas diferentes, proporcionando con frecuencia buenas soluciones con no demasiado tiempo de desarrollo. No obstante, si se requiere una solución realmente excelente, habrá que dedicar más tiempo al desarrollo del sistema neuronal, teniendo en cuenta diferentes cuestiones adicionales que todavía no hemos abordado (partir de una arquitectura óptima, selección de los pesos iniciales, estilo de aprendizaje, preprocesamiento de los datos de entrada, conjunto de patrones de aprendizaje empleado, utilización de técnicas que eviten el sobreajuste, etc.).

Como desventaja se encuentra, entre otras, su lentitud de convergencia, uno de los precios que hay que pagar por disponer de un método general de ajuste funcional que no requiere (en principio) información apriorística. Sin embargo, se debe tener en cuenta que el BP no requiere tanto esfuerzo computacional como el que sería necesario si se tratasen de obtener los pesos de la red mediante la evaluación directa de las derivadas; en ese sentido se ha comparado el BP con la transformada rápida de Fourier, que permite calcular la transformada de Fourier con un muy inferior esfuerzo computacional [Bishop 94]. Otro problema del BP es que puede incurrir en el denominado sobreaprendizaje (o sobreajuste), fenómeno directamente relacionado con la capacidad de generalización de la red a partir de los ejemplos presentados, y sobre el que profundizaremos en la próxima sección. Por otra parte, debe tenerse en cuenta que el algoritmo BP no garantiza alcanzar el mínimo global de la función error, tan sólo un mínimo local, por lo que el proceso de aprendizaje puede quedarse estancado en uno de estos mínimos locales.

2.5.3 Aceleración del aprendizaje BP. Otros algoritmos

Para resolver algunos de los inconvenientes del BP se plantean continuamente correcciones o variantes. Buena parte de estas modificaciones tratan de resolver el problema de su lenta convergencia, mientras que otras se centran en conseguir una mejor generalización; en esta sección nos ocuparemos del primer caso. Así, la primera variante la propusieron los propios *inventores* del modelo [Rumelhart 86b], al incluir en el algoritmo un término denominado **momento** (*momentum*), consistente en añadir al cálculo de la variación de los pesos (ecuaciones 2.53) un término adicional proporcional al incremento de la iteración anterior (*inercia*)

$$\delta w'_{kj}(t+1) = -\varepsilon \left. \frac{\partial E}{\partial w'_{kj}} \right|_t + \alpha \cdot \delta w'_{kj}(t-1) \quad \delta w_{ji}(t+1) = -\varepsilon \left. \frac{\partial E}{\partial w_{ji}} \right|_t + \alpha \cdot \delta w_{ji}(t-1) \quad (2.56)$$

con α un parámetro entre 0 y 1, que se suele tomar próximo a 1 ($\alpha \approx 0.9$). De esta manera, si los incrementos en un determinado peso tienen siempre el mismo signo, las actualizaciones en cada iteración serán mayores; sin embargo, si los incrementos en cierto peso oscilan (a veces son positivos, otras negativos), el incremento efectivo (acumulado) se reduce al cancelarse. Así, en zonas estrechas y profundas de la hypersuperficie de error (con forma de valle angosto), los pesos correspondientes a la dimensión estrechas (que sin el término de momento oscilarían de un lado al otro del valle) sufren incrementos pequeños, mientras que los de las direcciones que descienden directamente al fondo se ven potenciados [Bishop 94]. Es ésta una manera de aumentar el ritmo de aprendizaje efectivo en determinadas direcciones.

En [Fahlman 88] se realizó uno de los primeros estudios experimentales sobre la velocidad de aprendizaje en el BP, proponiéndose distintas **recetas para acelerarlo**. Un detalle tan simple como utilizar **sigmoides bipolares**, por ejemplo en el rango $[-1, +1]$ (función tangente hiperbólica, ecuación 2.47b), en vez de en el intervalo $[0, +1]$ (ecuación 2.47a), puede acelerar considerablemente el aprendizaje. En [LeCun 98, Haykin 99] pueden encontrarse también muchas recetas; por ejemplo, allí se aconseja la función de activación $f(x) = a \cdot \tanh(b \cdot x)$, con $a = 1.7159$ y $b = 2/3$.

Otra circunstancia a tener en cuenta es la magnitud de los **pesos iniciales**, pues una correcta elección puede suponer un menor tiempo de entrenamiento [Thimm 95]. Para el caso de la función de activación tangente hiperbólica, simplemente el elegir los pesos aleatoriamente en el intervalo $[-2.4/nin, +2.4/nin]$ (siendo nin el número de entradas de la neurona) ya suele dar buenos resultados [Principe 00]. En [LeCun 98] se propone otra heurística parecida.

Existen otras técnicas que permiten acelerar el aprendizaje, consistentes en preprocesar las entradas, asignar un ritmo de aprendizaje diferente a cada peso, ritmos adaptativos, etc. [LeCun 98, Haykin 99]. A partir de modificaciones de este tipo se han introducido algunos algoritmos de aprendizaje que pueden considerarse **variantes del BP** (véase, por ejemplo, [Wassermann 93]), siendo quizás los más conocidos **Quickprop** (el cual incorpora una heurística que da información sobre la forma de las derivadas segundas sin tener que derivar) y **SuperSAB** (en el que cada peso tiene su ritmo de aprendizaje, los cuales además van cambiando).

Como hemos visto, el BP estándar se basa en la técnica de descenso por el gradiente, es decir, calculando las derivadas de la superficie de error respecto de cada peso, $\partial E(W)/\partial w_{ij}$, se obtiene la dirección de máxima pendiente de la superficie $E(W)$, la cual se sigue para actualizar los pesos; sin embargo, nadie garantiza que sea ése precisamente el camino más rápido hacia el mínimo. Los denominados **métodos de segundo orden** se basan en realizar el descenso utilizando también la información proporcionada por el ritmo de cambio de la pendiente, $H = \partial^2 E(W)/\partial w_{ij} \partial w_{kl}$. Los algoritmos de **gradientes conjugados**, **gradientes conjugados escalados** y **Levenberg-Marquardt** son ejemplos de ello; cada uno emplea distintas aproximaciones que eviten el gran esfuerzo computacional que representaría el cálculo directo de la matriz H , llamada Hessiana. Son técnicas más robustas que el BP, que

pueden acelerar en uno o dos órdenes de magnitud la convergencia, aunque como contrapartida son mucho más complejas de implementar y precisan más recursos de cálculo. Estas técnicas pueden estudiarse en, por ejemplo, [Wassermann 93, Bishop 94, Demuth 98, Haykin 99, Principe 00].

Para finalizar, debemos dejar claro que desgraciadamente ninguno de los algoritmos o recetas descritos puede considerarse superior en general: un buen método para una aplicación puede proporcionar un rendimiento pobre en otra diferente. No obstante, en el manual de las herramientas de redes neuronales para el entorno matemático MATLAB® [Demuth 98] se proporcionan algunas pistas que pueden hacernos inclinarnos por uno u otro, recomendándose especialmente el algoritmo de Levenberg-Marquardt como primera opción a ensayar.

2.6 CAPACIDAD DE GENERALIZACIÓN DE LA RED

Uno de los aspectos fundamentales de los ANS es su capacidad de generalizar a partir de ejemplos, lo que constituye el problema de la **memorización frente a generalización**. Por **generalización** se entiende la capacidad de la red de dar una respuesta correcta ante patrones que no han sido empleados en su entrenamiento. Una red neuronal correctamente entrenada generalizará, lo que significa que ha aprendido adecuadamente el *mapping* no sólo los ejemplos concretos presentados, por lo que responderá correctamente ante patrones nunca vistos con anterioridad.

Validación cruzada (*cross-validation*)

En un proceso de entrenamiento se debe considerar, por una parte, un **error de aprendizaje**, que se suele calcular como el error cuadrático medio de los resultados proporcionados por la red para el conjunto de patrones de aprendizaje (ecuación 2.52). Con una red suficientemente grande, puede reducirse tanto como se quiera sólo con llevar a cabo más iteraciones. Por otra parte, existe un **error de generalización**, que se puede medir empleando un conjunto representativo de patrones diferentes a los utilizados en el entrenamiento. De esta manera, podemos entrenar una red neuronal haciendo uso de un conjunto de aprendizaje, y comprobar su eficiencia real, o error de generalización, mediante un **conjunto de test**.

Un hecho experimental, fácilmente observable con cualquier simulador, es que si se entrena una red hasta alcanzar un muy pequeño error en aprendizaje (por ejemplo, inferior a un 1%), la eficacia real del sistema o generalización (medido como error en test) se degrada. Si representamos a la vez el error en aprendizaje y el error en test durante el transcurso del aprendizaje, se obtiene una gráfica como la representada en la Figura 2.12 (izquierda): tras una fase inicial, en la que pueden aparecer oscilaciones en el valor del error, el de aprendizaje tiende a disminuir monótonamente, mientras que el error de generalización a partir de cierto punto comienza a incrementarse, lo que indica una degradación progresiva del aprendizaje.

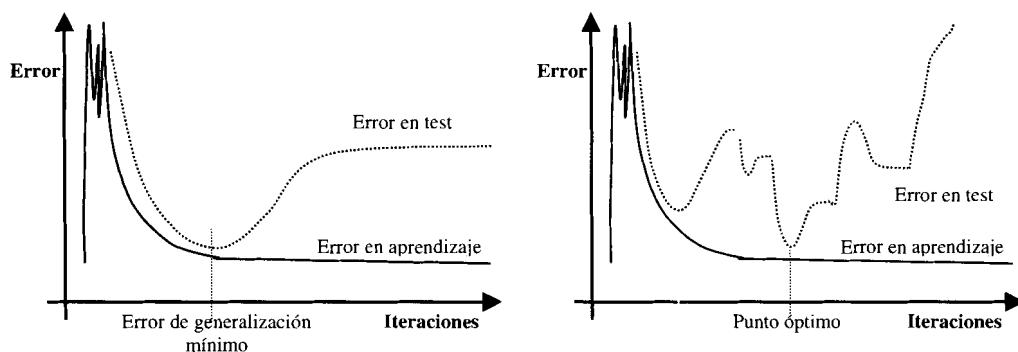


Figura 2.12 Evolución del error de aprendizaje y del error de generalización. A la izquierda, situación idealizada, a la derecha situación real [Prechelt 98]

La explicación de este fenómeno es la siguiente. Al principio la red se adapta progresivamente al conjunto de aprendizaje, acomodándose al problema y mejorando la generalización. Sin embargo, en un momento dado el sistema se ajusta demasiado a las particularidades de los patrones empleados en el entrenamiento, aprendiendo incluso el ruido en ellos presente, por lo que crece el error que comete ante patrones diferentes a los empleados en el entrenamiento (error de generalización). En este momento la red no ajusta correctamente el *mapping*, sino que simplemente está memorizando los patrones del conjunto de aprendizaje, lo que técnicamente se denomina **sobreaprendizaje o sobreajuste** (*overtraining* o *overfitting*), pues la red está *aprendiendo demasiado* (incluso el ruido presente en los patrones-ejemplo). Idealmente, dada una arquitectura de red neuronal, ésta debería entrenarse hasta un punto óptimo (Figura 2.12, izquierda) en el que el error de generalización es mínimo. El procedimiento consistente en entrenar y validar a la vez para detenerse en el punto óptimo se denomina **validación cruzada** (*cross validation*), y es ampliamente utilizado en la fase de desarrollo de una red neuronal supervisada (como el MLP).

No obstante, la situación descrita ha sido en cierta medida idealizada; una situación más realista sería la de la parte derecha de la Figura 2.12: en realidad pueden presentarse varios mínimos para el conjunto de test, debiendo detener el aprendizaje en el punto óptimo de mínimo error de generalización, y no quedarnos en el primer mínimo en test que aparezca. Distintas técnicas de **parada temprana** (*early stopping*) se describen en [Prechelt 98], aunque muchas veces basta con dejar que el aprendizaje discurra hasta una cota razonable de error (0.5%, 0.1%, 0.01%..., depende del problema), guardando periódicamente las distintas configuraciones intermedias de pesos, para luego quedarnos con la que proporcionó un error en test mínimo.

La clave de este asunto está en que las redes neuronales son estimadores no lineales poderosos, capaces de modelar situaciones muy complejas. En las herramientas lineales (por ejemplo, en ajuste mediante polinomios) la **complejidad del modelo** viene dada simplemente por el número de parámetros libres a ajustar

(coeficientes), mientras que en los sistemas no lineales (como son las redes neuronales), la complejidad del modelo depende tanto del número de parámetros como de su valor actual [Principe 00]. Los modelos que implementan las redes neuronales son de elevada complejidad, por lo que pueden aprender (memorizar) casi cualquier cosa, motivo por el cual incurren fácilmente en sobreaprendizaje.

Para comprenderlo mejor, pensemos en lo que sucede cuando se trata de ajustar un conjunto de medidas a un polinomio mediante el método de mínimos cuadrados [Bishop 94]. Supongamos un conjunto de medidas que se distribuyen como se muestra en la Figura 2.13. Si tratamos de ajustarlas a un polinomio de grado 1 (Figura 2.13a), el error que obtendremos será alto, y el valor que este polinomio proporcionará para valores diferentes de los empleados se apartará de la realidad. Si llevamos a cabo un ajuste con un polinomio de grado 3 (Figura 2.13b) los resultados que se obtienen son mucho mejores; si ajustamos con un polinomio de grado excesivo, por ejemplo superior a 10 (Figura 2.13c), la representación que obtenemos también se aparta de la correcta. En el caso (a) ajustamos con muy pocos parámetros, por lo que el modelo que obtenemos no se corresponde con la realidad, en (b) el número de parámetros es adecuado y el polinomio ajusta bien los datos; finalmente, en el caso (c) el polinomio tiene demasiados parámetros y el resultado que se obtiene se aparta de la realidad: no está capturando la tendencia del problema (una forma casi senoidal), sino que está interpolando (memorizando) los datos, ajustando también el ruido en ellos presente.

En una red neuronal sucede algo similar. En este caso los parámetros de ajuste no son los coeficientes del polinomio, sino los pesos y umbrales de las neuronas. En principio podemos suponer que una red de gran tamaño dispondrá de muchos grados de libertad. Hablando en términos generales, si el problema es *sencillo*, bastarán pocos parámetros para su ajuste, luego deberá utilizarse una red pequeña. Si el problema es complejo se necesitarán más parámetros de ajuste, luego se necesitará una red de mayor tamaño. Por lo tanto, *debe ajustarse el tamaño de la red a la complejidad del problema que se está tratando*, debiéndose limitar en lo posible su tamaño (principio de Occam o de la máxima economía de medios [Haykin 99]), dificultando así la aparición de sobreentrenamiento.

No obstante, como hemos comentado más arriba, el número de parámetros efectivos de la red depende del número de pesos y también de sus valores actuales. Se ha demostrado que el número efectivo de parámetros es generalmente menor que el número de pesos [Bishop 94], y su número crece conforme el aprendizaje progresa [Weigend 91, 93]. *Ése es el motivo por el que la parada temprana del aprendizaje evita el sobreentrenamiento, pues es equivalente a limitar el número de parámetros de la red* (sin modificar para nada la arquitectura actual, es decir, el número de pesos).

Cuando se entrena una red unidireccional supervisada debe tenerse muy en cuenta todo esto, y la técnica de validación cruzada suele ser un buen remedio (aunque, como veremos a continuación, no es el único disponible); usualmente, de todo el conjunto de entrenamiento se emplea aproximadamente un 80% de los patrones para entrenar, **reservándose un 20% como conjunto de test** [Haykin 99].

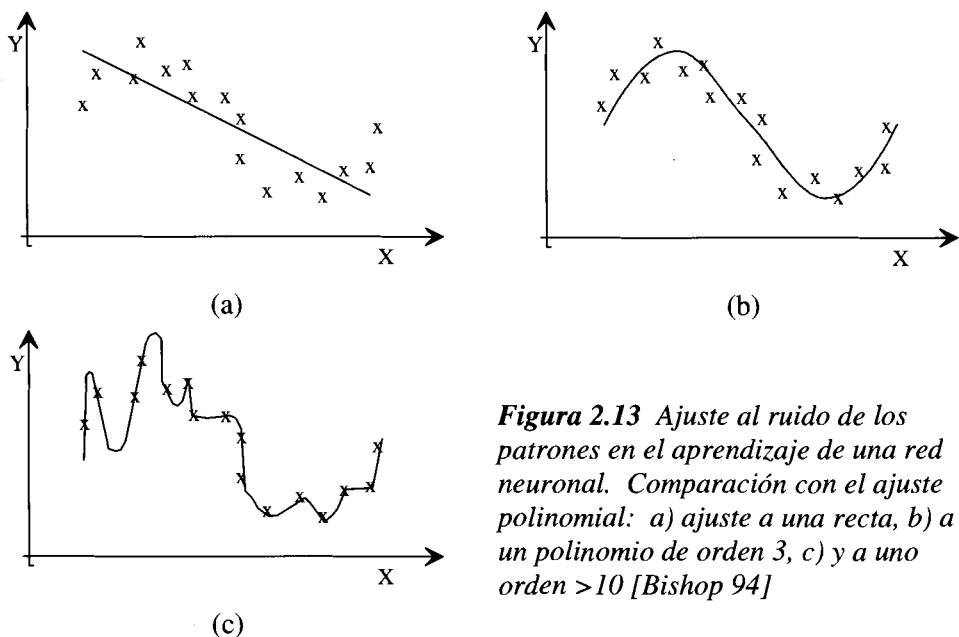


Figura 2.13 Ajuste al ruido de los patrones en el aprendizaje de una red neuronal. Comparación con el ajuste polinomial: a) ajuste a una recta, b) a un polinomio de orden 3, c) y a uno orden >10 [Bishop 94]

Número de ejemplos de entrenamiento

En definitiva, la capacidad de generalización de la red la determinan en buena medida las siguientes tres circunstancias: 1) la arquitectura de la red, 2) el número de ejemplos de entrenamiento y 3) la complejidad del problema [Haykin 99]. Los tres puntos están muy relacionados; en términos generales, cuanto más complejo sea el problema a modelar, más grande deberá ser la red (con más parámetros a ajustar) y, por lo tanto, más ejemplos se necesitarán para entrenarla (ejemplos que deberán cubrir todo el espacio de entrada, contemplando todas las situaciones posibles).

A menudo el número de patrones-ejemplo disponibles es limitado (y reducido), y en proporción el **número de parámetros efectivos de la red** elegida (grados de libertad) suele ser muy grande. En [Baum 89] se demostraba ya (lo cual ha sido corroborado por otros autores [Haykin 99]) que una red de n entradas y h neuronas ocultas, con un total de w pesos, requiere un número de patrones de aprendizaje del orden de $p=w/\epsilon$ para proporcionar un error de generalización del orden de ϵ . Así, si queremos que la red alcance un error de generalización de, por ejemplo, $\epsilon=0.1$ (un 10%), el número de patrones de aprendizaje necesarios p será del orden de $p=10.w$, expresión que se suele dar como indicativa del número aproximado de patrones que serían necesarios para entrenar adecuadamente una red neuronal de w pesos. Por ejemplo, para una red 10-5-1 (10 neuronas de entrada, 5 ocultas y 1 de salida.), que dispone de 61 parámetros, entre pesos y umbrales, el número de patrones necesarios para alcanzar un error del 10% será de unos 610 (!!), lo que representa un cifra de patrones muy alta, no disponible en muchas aplicaciones prácticas. Ello ilustra de nuevo la facilidad de incurrir en sobreaprendizaje al entrenar una red neuronal.

Reducción del tamaño de la arquitectura de red

Además, hay que tener presente la llamada **maldición de la dimensionalidad** (*curse of dimensionality*) [Bishop 94, Haykin 99], que consiste en que el número de datos necesarios para especificar un *mapping*, en general crece exponencialmente con la dimensión del espacio de entrada, lo que agrava en los problemas de dimensión de entrada elevada el disponer de un número de patrones para el aprendizaje escaso. Disminuyendo el número de parámetros de la red (tamaño) se tendrá una relación $p \approx w/\epsilon$ más favorable. Una forma de reducirlo consiste en **limitar el número de las entradas** de la red, pues ello implica la disminución drástica del número de pesos. Por ejemplo, una red con 200 entradas, 100 neuronas ocultas y 3 salidas, contendrá del orden de 20.000 pesos, con lo que se necesitarían unos 200.000 patrones para entrenarla adecuadamente. Si reducimos el número de entradas a 10 (por ejemplo, realizando un análisis de componentes principales a las variables de entrada, empleando ratios, etc.), el número de pesos se reduce a 143, con lo que se precisarían *únicamente* unos 1400 patrones de aprendizaje.

Otras técnicas empleadas en la reducción del número de parámetros de la red se relacionan con eliminar algunos de sus pesos; algunas bien conocidas son las de **compartir pesos** (*weight sharing*), **podado** de la red (*pruning*) o **decaimiento de pesos** (*weight decay*) [Hertz-Nielsen 91, Haykin 99, Principe 00]. En la primera de las citadas, diversas neuronas comparten sus pesos, de modo que el número total disminuye. En el proceso de podado la red es entrenada hasta un cierto nivel, para luego eliminar aquellos pesos que no aportan prácticamente nada a su operación. El decaimiento es un caso especial del podado; durante el aprendizaje se deja a los pesos tender poco a poco a cero, para que aquellos que no sean actualizados periódicamente, se anulen y desaparezcan.

Resumen

Recapitulando lo expuesto en esta sección, cuando se entrena supervisadamente (por ejemplo, con BP) una red unidireccional está muy presente la posibilidad de aparición de sobreajuste (sobreentrenamiento), lo cual degrada enormemente la capacidad de generalización de la red. Existen *dos formas de luchar contra el fenómeno del sobreentrenamiento*: la parada temprana (validación cruzada) y limitar el tamaño de la arquitectura de la red. Aplicar la validación cruzada en los entrenamientos es casi obligatorio; se suele recomendar el empleo de un 80% de los ejemplos disponibles para entrenar y reservar un 20% para test. Por otro lado, debe limitarse siempre el tamaño de la red, debiendo elegir la arquitectura de tamaño mínimo que permita aprender los ejemplos disponibles. La relación $p=10.w$ (con p el número de ejemplos y w el número de pesos de la red) puede ser particularmente útil, pudiéndose utilizar de dos formas distintas: para establecer el tamaño aproximado de la red ideal para el conjunto de ejemplos disponible, o, dada una arquitectura de red, para hacernos una idea sobre si es fácil que incurramos en sobreentrenamiento.

2.7 PINCELADAS SOBRE LA RELACIÓN DEL MLP CON LOS MÉTODOS ESTADÍSTICOS

Desde un punto de vista estadístico, muchos de los problemas que se intentan resolver con un MLP entran en la categoría de los denominados **problemas mal planteados** (*ill-posed*), en los que el espacio de trabajo es tan amplio y los datos disponibles tan escasos, que resulta difícil encontrar la red neuronal que los ajusta correctamente, puesto que la información contenida en los datos de entrenamiento no es suficiente para determinar unívocamente el *mapping*, de manera que las posibles soluciones que en principio permiten ajustar los datos son virtualmente infinitas.

Hablando en términos estadísticos, las redes neuronales son estimadores no paramétricos [Vapnik 99a], que realizan estimaciones denominadas de modelo libre. Por ejemplo, el método convencional de ajuste a una línea recta mediante mínimos cuadrados es un estimador paramétrico, pues se impone al problema un determinado modelo de partida (la línea recta), cuyos parámetros se deben ajustar según las muestras disponibles. A diferencia de los paramétricos, el MLP (y muchas otras redes neuronales) sería un estimador de modelo libre, pues no se impone ninguna forma funcional de partida.

Las limitaciones de las redes neuronales, y del MLP en particular, pueden entenderse e interpretarse mediante el bien conocido problema que también surge en la estimación paramétrica, el denominado **dilema de la varianza y el sesgo** (*bias and variance dilemma*) [Bishop 94, 95], que involucra muchos de los aspectos discutidos en la sección anterior. El asunto es el siguiente. Hablando en términos neuronales, para realizar un ajuste óptimo el número de patrones de entrenamiento debería tender a infinito pues, para un conjunto finito, los estimadores no paramétricos suelen ser muy sensibles a los casos particulares de pares entrada-salida seleccionados para realizar el aprendizaje. La causa es que la red neuronal, estimador de modelo libre, posee inherentemente una gran varianza. Hablando coloquialmente, la red neuronal puede implementar muchísimos diferentes *mappings*, pero solamente implementará el correcto si utilizamos un conjunto de entrenamiento de tamaño idealmente infinito. La única forma de controlar la elevada varianza que la red neuronal posee inicialmente es introducir en su arquitectura algún tipo de sesgo (*bias*) o información apriorística sobre el problema a resolver; hablando coloquialmente, sería como proporcionar a la red neuronal alguna *pista* que le indique cómo pensamos que debe ser la solución. Es decir, se trata de conseguir que el estimador (la red neuronal), en principio no paramétrico, tienda a ser paramétrico en un cierto grado, de manera que de partida se encuentre en cierta medida viciado hacia el tipo de solución que nos interesa.

El dilema que se plantea es precisamente en qué medida se debe viciar el modelo neuronal. Si se emplea una red de bajo sesgo y alta varianza (es decir, con muchos parámetros o pesos y sin introducir información apriorística), el número de ejemplos necesarios para entrenarla correctamente será altísimo. Si en el problema a resolver no se dispone de suficientes patrones, se deberá introducir cierto sesgo en la

arquitectura, el cual se debería corresponder fielmente con la realidad, de otro modo se fracasará en la resolución del problema (se estarían presentando *pistas falsas* a la red). Por lo tanto, una red de alto sesgo tendrá poca flexibilidad, pero una de alta varianza ajustará también el ruido presente en los datos. La solución ideal del dilema varianza-sesgo sería encontrar el punto de la Figura 2.12, que daría la arquitectura óptima.

Un resumen de este asunto, junto con algunos métodos que pueden resolver el problema planteado, se muestran en [Gedeon 95]. Ya hemos apuntado una forma de resolverlo: introducir en la red neuronal restricciones que de antemano se sabe debe cumplir el problema, contribuyendo así a reducir los grados de libertad y conseguir que el problema pase a estar bien planteado (*well posed*). Estas restricciones suponen *introducir información apriorística disponible sobre el problema en la propia arquitectura de la red*. Las denominadas **técnicas de regularización** abordan el problema introduciendo restricciones que implican que el *mapping* que implementa la red neuronal sea suave, es decir, que a entradas similares haga corresponder resultados próximos [Poggio 90, Girosi 93]. Éstas pueden ser incluidas en la función coste en forma de términos adicionales $\phi(w_\alpha)$ (siendo w_α los pesos), que miden la desviación de los resultados actuales respecto de la restricción planteada

$$E[w_\alpha] = \frac{1}{2} \sum_{\mu} \sum_k \left[t_k^\mu - F_k^\mu(\mathbf{x}) \right]^2 + \lambda \sum_{\alpha} \phi(w_\alpha) \quad (2.57)$$

siendo λ el denominado parámetro de regularización, que controla el compromiso entre el grado de suavidad de la solución frente al nivel de ajuste de los datos de entrenamiento que alcanza [Poggio 90].

Otro tipo de restricción que puede introducirse es que los pesos sean de tamaño reducido, evitando que crezcan indefinidamente; en este caso se elige una función de regularización de la forma $\phi(w_\alpha) = (w_\alpha)^2$, lo que conduce a una regla que implementa el bien conocido decaimiento de pesos (sección 2.6). Otros ejemplos de restricciones se muestran en [Hertz 91, Moody 92].

La aplicación de las técnicas de regularización sobre el modelo MLP-BP puede mejorar su nivel de generalización. No obstante, también ha inspirado el trabajo con otros modelos neuronales, como las denominadas redes de regularización (*regularization networks*), de las que se puede deducir las bien conocidas **funciones de base radial** o RBF (*Radial Basis Functions*) como un caso particular [Poggio 90]. Las RBF son en la actualidad un modelo neuronal de gran aplicabilidad práctica, que estudiaremos en el capítulo 4.

Un área de gran trabajo hoy en día se relaciona con la clásica **teoría estadística de aprendizaje** (*statistical learning theory*), que ha llevado a la introducción durante los años noventa de conceptos tan importantes como la **dimensión VC** (Vapnik-Chervonenkis), o el modelo de las **máquinas de vectores soporte** o SVM (*Support Vector Machines*), empleadas en clasificación y ajuste funcional. No nos

extenderemos en el tratamiento de esta teoría (muy complicada, por otra parte), remitiendo al lector interesado a las referencias [Haykin 99, Principe 00, IEEE 99b], o directamente al trabajo de uno de sus inventores [Vapnik 99b].

Es muy importante resaltar la existencia de claras relaciones entre el MLP y las técnicas estadísticas convencionales, como la regresión o el análisis discriminante, u otras más recientes, como la teoría de la regularización. El MLP, en definitiva, realiza un tipo de regresión multidimensional y no lineal, en la que no es necesaria la suposición inicial de una determinada forma funcional para el ajuste; en este sentido se suele apreciar en su aplicación práctica que el MLP con frecuencia supera a la regresión u otras técnicas lineales tradicionales para espacios de dimensión alta (mayor que tres) [Hetch-Nielsen 90], aspecto que otros estudios rigurosos han corroborado [Principe 00]. También ha sido comparado con el análisis discriminante, mostrándose que proporciona mejores resultados en problemas no lineales.

La comparación del MLP en particular, y de los ANS en general, con los modelos estadísticos, así como su estudio riguroso desde este punto de vista, resulta de importancia capital a la hora de comprender cómo operan los modelos neuronales. De hecho, la relación redes neuronales/estadística es hoy en día una de las áreas de trabajo más importantes, tanto por el número de publicaciones, como por su impacto, pues estos trabajos se sitúan en el núcleo teórico básico de los ANS. Los artículos de H. White [White 89a, 89b] constituyen una excelente introducción al tema; en [NATO 93, Cherkaski 94] pueden encontrarse una interesante recopilación de trabajos. Un reciente (y excelente) estudio de las relaciones de los modelos de redes supervisadas con el reconocimiento estadístico de patrones se da en [Bishop 95]. La referencia [Haykin 99] realiza una completa puesta al día de estos temas.

En la época de desarrollo *explosivo* del trabajo en redes neuronales, la idea que se trataba de *vender* consistía en que estos modelos eran más eficaces que las técnicas convencionales. Sin embargo, en la actualidad, con el campo ya más asentado, se dispone de estudios serios y maduros, llegándose a la conclusión de que algunos de los problemas que aparecen en el campo de los ANS pueden ser resueltos por la estadística o bien ésta ya los ha resuelto con anterioridad. En la actualidad, pasada la época de considerar las redes neuronales como objeto de moda, empiezan a ser tratadas como un área cercana a la madurez, constituyendo un conjunto de técnicas más a añadir a la amplia panoplia de métodos de tratamiento de datos y señal ya existentes, que en ocasiones proporcionarán mejores resultados que los convencionales.

Evidentemente, la realidad es lo suficientemente compleja como para que no exista una solución milagrosa que resuelva todos los problemas. *Los problemas complejos deben ser divididos en partes, y cada parte debe ser resuelta con la técnica apropiada*, convencional (estadística, sistemas expertos, reglas heurísticas, etc.) o no tan convencional (redes neuronales, lógica borrosa, algoritmos genéticos, etc.). La tendencia hacia sistemas híbridos es clara en la actualidad, y clave en la Inteligencia Computacional [IEEE 99].

2.8 EJEMPLOS DE APLICACIÓN DEL MLP-BP

Para concluir, en esta última sección mostraremos dos casos prácticos que ilustrarán el desarrollo de aplicaciones mediante la arquitectura perceptrón multicapa entrenada con BP.

La función OR-exclusivo

Como ya hemos comentado, la función OR-exclusivo o XOR no es separable linealmente, por lo que un perceptrón simple no puede representarla, mientras que una arquitectura MLP sí es capaz de hacerlo, en principio. Vamos a ver además que un MLP puede aprender automáticamente a representarla haciendo uso de aprendizaje BP, empleando como patrones de aprendizaje la tabla de verdad de la función XOR. Consideraremos una XOR de dos entradas; llamaremos x_1 y x_2 a las entradas lógicas, y será la salida. La arquitectura MLP deberá tener dos neuronas de entrada, una por variable, y una de salida. Por otro lado, el número de nodos ocultos debe ser propuesto por el que desarrolla el sistema; como se trata de un problema sencillo estableceremos, en principio, dos neuronas ocultas. Por lo tanto, la arquitectura resultante es una 2-2-1 (Figura 2.14).

Disponemos de cuatro patrones de aprendizaje: a la pareja de entradas 0 y 0 le corresponde una salida 0; a 0 y 1, una salida 1; a 1 y 0, 1; y, por último, a 1 y 1, una salida 0. Las neuronas del MLP no operan directamente con variables lógicas, sino con valores reales, por lo que a cada valor lógico (0 y 1) hay que asignarle un número real que hará el papel de entrada o salida; la elección más directa sería asignar al valor lógico 0 el real 0.0, y al valor lógico 1, el real +1.0; no obstante, el entrenamiento de una red que opera con números bipolares (positivos y negativos) suele ser más rápido, por lo que en este caso decidimos hacer corresponder al valor lógico 0 el real -0.5, y al valor lógico 1 el +0.5. Debido a esta elección, la función de activación para las neuronas ocultas que emplearemos será la siguiente

$$f(v) = \frac{1}{1 + e^{-v}} - 0.5 \quad (2.58)$$

sigmoidea bipolar, que trabaja en el intervalo [-0.5,+0.5]. La variable v representa para cada neurona oculta la suma ponderada de las entradas con sus pesos, cantidad a la que se resta el umbral (*bias*). Ya que los valores de salida deben ser digitales (+0.5 o -0.5), emplearemos esta misma función saturada como función de activación de la neurona de la capa de salida.

También debe establecerse el valor de los pesos de partida, que normalmente son pequeños y aleatorios, por lo que los tomaremos en el intervalo [-0.3, +0.3]. Entrenaremos la arquitectura mediante el BP estándar con término de momento. También deben establecerse los valores de los parámetros de entrenamiento: elegimos el ritmo de aprendizaje $\epsilon = 0.3$ y el correspondiente al término de momento $\alpha = 0.9$. El

aprendizaje es iterativo, las salidas de la red asintóticamente se acercarán a los valores objetivo +0.5 o -0.5, por lo que también deberemos decidir cuándo detener el entrenamiento, por ejemplo, cuando el error cuadrático medio de la neurona de salida sea inferior a un 1%. En este caso se trata de memorizar la tabla de verdad de la función XOR (Figura 2.14), cuatro ejemplos de entrenamiento en total, y no existen más posibilidades, por lo que en este caso no puede haber sobreaprendizaje ni es necesario realizar una validación cruzada.

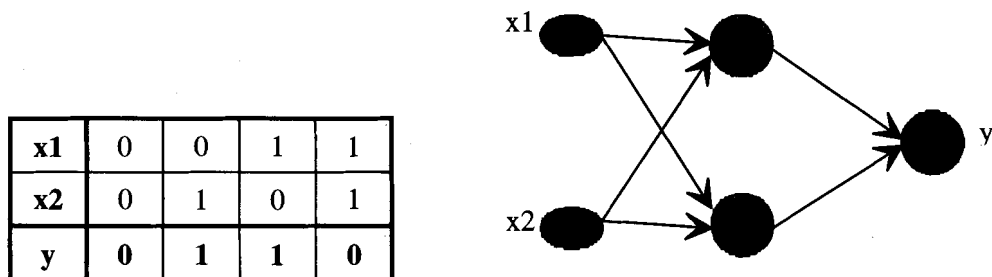


Figura 2.14. Tabla de verdad de la función XOR y arquitectura de MLP propuesta

Dadas todas estas condiciones, y para los pesos aleatorios concretos que se seleccionaron, se necesitaron 38 iteraciones para completar el entrenamiento de la arquitectura, obteniéndose los resultados mostrados en la Figura 2.15. Dependiendo de las elecciones descritas, y de los pesos de partida, el número de iteraciones requerido para entrenar la arquitectura para este problema puede ir desde las varias decenas (como nuestro caso), hasta varios centenares. El criterio para elegir los parámetros propuestos, y no otros, es una de las cuestiones importantes que el desarrollador debe dilucidar, para lo cual debe apoyarse en su experiencia previa en el trabajo con el BP, y con frecuencia también deberá experimentar con diversas posibilidades para el propio problema en desarrollo, ejecutando varios entrenamientos (completos o parciales). La prueba con diferentes configuraciones es habitual, casi inevitable, en el desarrollo de una aplicación con redes neuronales; de hecho, algunos programas de simulación de redes neuronales pueden realizarlo automáticamente.

Acabado el entrenamiento, dicha red implementa la función XOR, lo cual se comprueba introduciendo todos los posibles valores de entrada, y observando las salidas. En este sencillo ejemplo podemos intentar llevar a cabo un **análisis exhaustivo de la operación de la red**, y tratar de comprender qué es lo que la red ha aprendido, y cómo lo ha aprendido (es decir, qué tarea desempeña cada una de las neuronas individuales). Si presentamos las cuatro parejas de entradas (Figura 2.14) y anotamos las salidas $h1$ y $h2$ de las dos neuronas ocultas, nos encontramos con que las funciones lógicas que han aprendido cada una son las siguientes

$$h1 = \overline{x1.x2} \qquad h2 = \overline{x1.x2} \qquad (2.59)$$

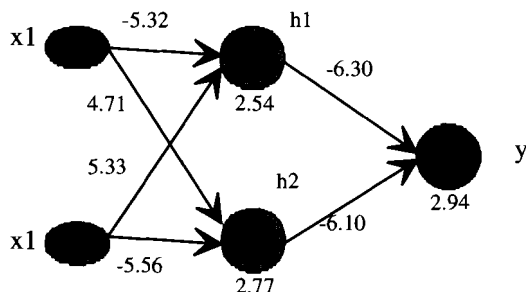


Figura 2.15. MLP entrenado con la tabla de verdad de la función XOR. Los números sobre las conexiones indican los valores de los pesos sinápticos obtenidos, y los valores bajo cada neurona los de los umbrales (bias)

Si hacemos lo mismo con la neurona de salida, teniendo en cuenta que sus entradas son $h1$ y $h2$, se tiene

$$y = \overline{h1.h2} \quad (2.60)$$

y sustituyendo los valores de (2.59) y simplificando, se tiene

$$y = x1.\overline{x2} + \overline{x1}.x2 \quad (2.61)$$

que es justamente la expresión lógica conocida correspondiente a la función XOR.

En este sencillo ejemplo (tan solo dos neuronas ocultas) hemos podido estudiar detenidamente cómo actúa cada neurona, lo que resulta de gran interés, aunque suele ser mucho más dificultoso en el caso de arquitecturas de cierto tamaño.

La crisis bancaria española 1977-85

El segundo caso de estudio consiste en el análisis de la crisis bancaria española de 1977-85 mediante redes neuronales [Serrano 93, Martín del Brío 93a, 95c], el cual fue desarrollado en colaboración con el Departamento de Contabilidad y Finanzas de la Universidad de Zaragoza. Esta crisis resultó tan grave que se ha comparado al crack de 1929 en los Estados Unidos (afectó nada menos que a 58 de los 108 bancos españoles de la época).

En primer lugar, se trataba de obtener un sistema capaz de determinar automáticamente si una entidad bancaria se encuentra en situación solvente o crítica en función de sus ratios económicos. Para ello nos apoyamos en la referencia [Pina 89], donde se estudiaba mediante técnicas estadísticas los ratios financieros más significativos para este problema concreto. De este estudio se desprende que de los múltiples ratios que pueden definirse, resultan ser solamente nueve los más importantes a la hora de determinar una situación de crisis; algunos de ellos tienen que

ver con términos de rentabilidad, otros de liquidez, otros se relacionan con el *cash-flow*, etc. [Serrano 93] (véase la tabla 2.1).

Para modelar las situaciones de crisis/solvencia hicimos uso de un perceptrón multicapa entrenado mediante BP. Si empleamos los nueve ratios indicados como entradas, y una sola neurona de salida que nos indique con un 1 si un banco es solvente, y con 0 si está en crisis, solamente nos quedará determinar el número más adecuado de neuronas ocultas.

Disponíamos de una base de datos correspondiente a 66 bancos de la época, 29 de los cuales quebraron [Serrano 93] (véase la Figura 2.16). En diferentes pruebas preliminares entrenamos distintas arquitecturas con los datos de la mitad de los bancos (seleccionados al azar), presentando como entradas sus 9 ratios, y como salidas un 0 o un 1 según su estado de crisis o solvencia. El resto de los datos, no utilizados en el aprendizaje del sistema, se emplean para estimar objetivamente el rendimiento de la arquitectura ya entrenada.

R1	Activo circulante/Activo total
R2	(Activo circulante-Caja)/Activo total
R3	Activo circulante/Deudas
R4	Reservas/Deudas
R5	Beneficio neto/Activo total
R6	Beneficio neto/Fondos propios
R7	Beneficio neto/Deudas
R8	Coste de ventas/Ventas
R9	Cash Flow/Deudas

Tabla 2.1 Definición de los ratios utilizados en el estudio de la crisis bancaria

1 B Union	15 B Valladolid	29 B Garriga N.	43 B Guipuzc.	57 Sind. Banq.
2 B Mas Sardá	16 B Cred. Com.	30 B de Progreso	44 B de Galicia	58 B de Europa
3 B Levante	17 B Prést. y Ahor.	31 B Ind. Bilbao	45 B Hisp. Ind.	59 B de Vasconia
4 B Catalana	18 B Descuento	32 B Int. Español	46 B March	60 B Pop. Español
5 B Ind. Catal.	19 B Com. Occid.	33 B Com. Tran.	47 B Depósitos	61 B Hisp. Amer.
6 B Barcelona	20 B Occidental	34 B Comercio	48 B Herrero	62 B Españ. Créd.
7 B Gerona	21 B Ind. Medit.	35 B Jover	49 B Sabadell	63 B Santander
8 B Alicante	22 B Catal. Desarr.	36 B de Vitoria	50 Bankpyme	64 B Central
9 B Créd. e Inv.	23 B Prom. Neg.	37 B Pueyo	51 B Int. de Com.	65 B Bilbao
10 B Pirineos	24 B López Ques.	38 B Créd Balear	52 B Zaragoza	66 B Vizcaya
11 B Madrid	25 B Asturias	39 B Huesca	53 B Com. Espa.	
12 B de Navarra	26 B Granada	40 B Fomento	54 B Merc. Tarr.	
13 B Cantábrico	27 B Simeón	41 B Pastor	55 B Abel Matutes	
14 B Meridional	28 B. Exp. Indust.	42 B de Castilla	56 B Fin. Indust.	

Figura 2.16. Relación de bancos empleados en el estudio. Los bancos del 1 al 29 entraron en crisis. Los bancos 60 a 66 son los siete grandes de la época

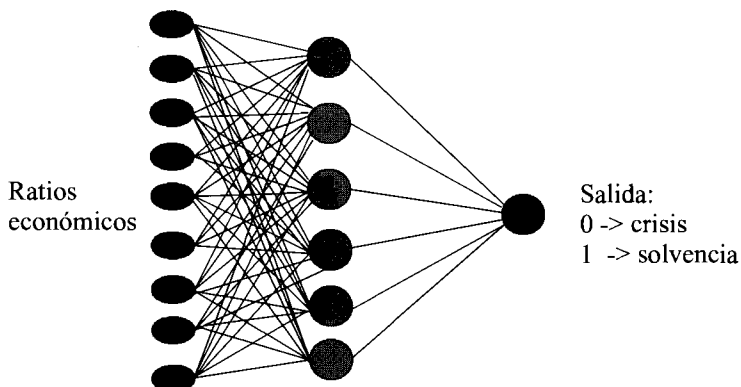


Figura 2.17. Arquitectura de MLP 9-6-1 empleada para la determinación de crisis bancarias (9 ratios bancarios de entrada, y una única salida 1 o 0, indicando solvencia o quiebra)

Tras una serie de pruebas determinamos que un número de neuronas ocultas entre 4 y 9 daba buenos (y similares) resultados, por lo que finalmente nos quedamos con una arquitectura 9-6-1, es decir, con seis neuronas ocultas. Obsérvese que con una arquitectura 9-6-1 se tiene un total de 67 parámetros (entre pesos y umbrales), con lo que la regla $p=10.w$ nos indicaría que se necesitarían aproximadamente ¡670 ejemplos de bancos quebrados y no quebrados para entrenar esta red adecuadamente! Obviamente en España no hay tantos bancos, y nosotros disponíamos en total de 66 ejemplos, por lo que esta arquitectura fácilmente incurrirá en sobreaprendizaje, por lo que se deberá realizar un entrenamiento con *validación cruzada*.

En el entrenamiento, la red aprende la asociación de cada grupo de 9 ratios con el 0 o el 1 que los califica; con un número suficiente de patrones de entrenamiento el MLP, generalizando los datos aprendidos, deberá encontrar la *ley de quiebra* subyacente (que los economistas desconocían). El entrenamiento se realizó mediante BP; el proceso iterativo ejecutado sobre un ordenador tipo PC con procesador i80486 empleó típicamente del orden de media hora, en un Pentium este tiempo se reduce a algunos minutos.

Realizando un estudio exhaustivo de la eficacia del sistema neuronal mediante técnicas de *bootstrap* o dejar-uno-fuera, encontramos que el MLP propuesto entrenado con BP es capaz de determinar correctamente las situaciones de quiebra/solvencia en el 94% de las ocasiones (las herramientas estadísticas llegaban aproximadamente a un 85%, pero además este índice incluía los propios patrones empleados en el desarrollo del modelo; si nosotros entrenáramos y testáramos la red con los mismos ejemplos obtendríamos un 100% de acierto, pero ello no es honesto).

De este modo, con el sistema neuronal así construido, y dados los 9 ratios de cualquier banco, podremos establecer su nivel de solvencia/crisis sólo con alimentar con ellos la red y observar la salida: cuanto más próxima sea la salida al valor 1.0 en mejor situación se encontrará el banco, mientras que cuanto más cercana a 0.0 se sitúe, será más crítica. Es importante resaltar que en redes neuronales, aunque el proceso de entrenamiento puede ser lento, la respuesta de la red una vez entrenada puede considerarse en muchas ocasiones instantánea a efectos prácticos.

El mismo problema fue analizado también haciendo uso de un modelo neuronal no supervisado [Serrano 93, Martín del Brío 93a, 95c], el de los mapas autoorganizados, que será descrito en el siguiente capítulo.

REDES AUTOORGANIZADAS

Estudiadas algunas de las redes supervisadas más importantes, en este capítulo trataremos el otro gran grupo de modelos neuronales, los no supervisados o autoorganizados. Éstos se caracterizan porque en su entrenamiento no se presentan las salidas objetivo que se desean asociar a cada patrón de entrada. La red, a partir de un proceso de autoorganización, proporcionará cierto resultado, el cual será reflejo de las relaciones de similitud existentes entre dichos patrones de entrada. La principal aplicación de estos modelos será la realización de agrupamiento de patrones (*clustering*), análisis exploratorio, y visualización y minería de datos (*data mining*).

Una vez sean expuestos los aspectos generales más relevantes de los modelos no supervisados, nos centraremos en el estudio de uno de los más populares, el de los **mapas autoorganizados de Kohonen**. Así, se introducirá su arquitectura y forma de operación, presentándose a continuación una serie de ejemplos que la ilustran. Finalmente, se hará énfasis en los trabajos teóricos llevados a cabo sobre el modelo para mostrar su capacidad de procesamiento, y se mostrarán algunos de sus algoritmos de aprendizaje.

3.1 MODELOS NEURONALES NO SUPERVISADOS

A diferencia de lo que sucede en el aprendizaje supervisado tratado hasta el momento, en el no supervisado (autoorganizado) no existe ningún *maestro* externo que indique si la red neuronal está operando correcta o incorrectamente, pues no se dispone de ninguna salida objetivo hacia la cual la red neuronal deba tender. Así, durante el proceso de aprendizaje la red autoorganizada debe descubrir por sí misma rasgos comunes, regularidades, correlaciones o categorías en los datos de entrada, e incorporarlos a su estructura interna de conexiones (pesos). Se dice en este caso que las neuronas deben **autoorganizarse** en función de los estímulos (señales o datos) procedentes del exterior. Para obtener resultados de calidad la red requiere un cierto

nivel de redundancia en las entradas procedentes del espacio sensorial, en definitiva, un número de patrones de aprendizaje suficientemente amplio.

El tipo de procesamiento que una red neuronal no supervisada puede realizar depende en buena medida de su arquitectura. En [Hertz 91] se enumeran algunos:

- a) **Análisis de similitud entre patrones (o novedad).** Este tipo de procesamiento aparece cuando existe una única neurona cuya salida es continua, indicando el grado de similitud o parecido entre el patrón de entrada actual y el promedio de los presentados en el pasado. Dicho promedio queda representado durante el entrenamiento en el vector de pesos sinápticos de la neurona, de modo que la neurona aprende lo que es típico dentro de un conjunto de patrones.
- b) **Análisis de componentes principales.** Si se extiende el caso anterior a varias neuronas de salida continua, el proceso de aprendizaje supone encontrar una cierta base del espacio de entrada, representada por el conjunto de los vectores de pesos sinápticos de todas las neuronas, que se corresponderán con los rasgos más sobresalientes del espacio sensorial.
- c) **Agrupamiento/clasificación (*clustering*).** La red realiza tareas de agrupamiento o clasificación cuando se compone de neuronas de salida discreta $\{0,1\}$, donde cada una representa una categoría, y solamente una de ellas puede permanecer activada a la vez. Ante un patrón de entrada, la neurona que se activa indica a qué categoría o grupo (*cluster*) pertenece. Durante el aprendizaje la red deduce las categorías presentes en el espacio de entrada a partir de la medida de distancias (euclídea o de otro tipo) entre los patrones presentados.
- d) **Memoria asociativa.** Representa una generalización del caso anterior de redes para agrupamiento; en este caso la salida proporcionada por la neurona activada no es solamente un 0 o un 1, sino el vector prototipo de la clase en cuestión.
- e) **Codificación.** Es análogo al anterior, sólo que en este caso la neurona no proporciona como salida el vector prototipo de la clase, sino una versión codificada (por ejemplo, una etiqueta), habitualmente empleando menos bits y manteniendo la información relevante (compresión de datos).
- f) **Mapas de rasgos.** Se trata de modelos no supervisados en los que las neuronas se ordenan geoméricamente (por ejemplo, en forma de matriz bidimensional o mapa), llevando a cabo una proyección del espacio sensorial de entrada sobre la red (mapa), proyección que preserva en lo posible la topología del espacio original, pero reduciendo sus dimensiones.

Los casos citados no son necesariamente diferentes ni excluyentes; un mismo modelo de red no supervisada puede estar capacitado para efectuar varios de estos procesamientos. Por ejemplo, la codificación puede realizarse mediante un análisis de componentes principales o mediante clustering. Este último caso recibe el nombre de **cuantificación vectorial** (*vector quantization*).

Aunque los modelos autoorganizados tienen un campo de aplicación propio, relacionado en general con el **análisis exploratorio de datos**, en ocasiones pueden sustituir a los modelos supervisados. Esta circunstancia puede darse, por ejemplo, cuando la aplicación del aprendizaje BP a un problema que requiere aprendizaje supervisado es demasiado lenta, en cuyo caso puede resultar de utilidad el empleo de un **modelo multicapa híbrido** (con una capa autoorganizada y otra supervisada), pues al romper el entrenamiento en dos fases se gana en velocidad [Hrycej 92]. Un ejemplo de este tipo de redes híbridas lo estudiaremos en el capítulo siguiente.

Tipos de redes no supervisadas

Los modelos no supervisados suelen ser simples, monocapa y con algoritmos sencillos y rápidos, más próximos a la biología. En general, se pueden clasificar en dos grandes grupos. El primero lo denominaremos **redes no supervisadas hebbianas**, pues su aprendizaje es de tipo Hebb. Como característica general de dichas redes puede señalarse que en ellas un número elevado de neuronas de salida pueden activarse simultáneamente. Algunos modelos utilizan reglas de aprendizaje directamente basadas en la regla de Hebb, como las redes PCA (que realizan análisis de componentes principales o *Principal Component Analysis*) [Oja 82, Hertz 91]. Otros se derivan de determinadas propiedades a optimizar, como pueda ser la maximización del contenido de información [Linsker 88].

Existe otra amplia clase de modelos autoorganizados, denominados **redes no supervisadas competitivas**, en las que solamente una neurona (o grupo de vecinas) puede quedar finalmente activadas. La base de la operación de estos modelos es la competición entre las neuronas, materializada en forma de inhibiciones laterales, a través de las cuales cada una trata de inhibir a las demás (idea que históricamente parte del trabajo de Von der Malsburg [Malsburg 73]). En este proceso de competición la neurona más activada conseguirá inhibir a todas las demás, por lo que será la única que permanezca activada, motivo por el cual estas redes también se denominan **redes WTA** (*winner-take-all*).

En los modelos competitivos, durante la fase de aprendizaje las neuronas vencedoras obtienen como premio el refuerzo de sus conexiones sinápticas. La competición es un comportamiento básico en muchos de los modelos neuronales autoorganizados más conocidos, como el ART [Carpenter 88], Neocognitrón¹ [Fukushima 80] o los mapas autoorganizados [Kohonen 82a, 82b, 89]. La idea común a todos ellos es el agrupamiento o categorización de patrones, pues esencialmente desarrollan clusters que agrupan patrones. No obstante, también pueden resultar útiles para muchas otras tareas, como cuantificación vectorial, aproximación funcional, procesamiento de imagen, análisis estadístico y optimización combinatorial [Hertz 91].

¹ Una versión del neocognitrón implementa también aprendizaje supervisado [Fukushima 83].

Un estudio en profundidad del aprendizaje competitivo implica la lectura detenida de las referencias de von der Malsburg [Malsburg 73], Fukushima [Fukushima 75, 80], Grossberg [Grossberg 76a, b], Kohonen [Kohonen 82a, b] y Rumelhart y Zipser [Rumelhart 86c]. Una muy buena introducción se encuentra en el conocido y excelente libro de Hertz, Krogh y Palmer [Hertz 91]. Una referencia actualizada sobre el modelo de mapas autoorganizados es [Kohonen 97].

3.2 MODELO DE MAPAS AUTOORGANIZADOS (KOHONEN, 1982)

Expuestos los conceptos básicos relacionados con las redes no supervisadas, en lo que resta de capítulo nos centraremos en los mapas autoorganizados [Kohonen 82a, 82b, 89, 90, 97, Ritter 91a], uno de los sistemas neuronales más conocidos y empleados, y cuyo estudio además ilustra muy bien la operación de muchas otras arquitecturas no supervisadas. En esta sección introduciremos el modelo y presentaremos alguno de sus algoritmos de aprendizaje.

3.2.1 Introducción a los mapas autoorganizados

Se observa que en muchas regiones del córtex de los animales superiores aparecen zonas donde las neuronas detectoras de rasgos (o características) se distribuyen topológicamente ordenadas [Kohonen 90]. Por ejemplo, en el área somatosensorial (Figura 3.1), las neuronas que reciben señales de sensores que se encuentran próximos en la piel se sitúan también próximas en el córtex, de manera que reproducen (aunque distorsionadamente) el mapa de la superficie de la piel en una zona de la corteza cerebral; lo mismo sucede, por ejemplo, en el córtex motor. Por lo que respecta al sentido del oído, existen en el cerebro áreas que representan mapas tonotópicos, donde los detectores de determinados rasgos relacionados con el tono de un sonido se encuentran ordenados en dos dimensiones. En resumen, la representación de la información en la corteza cerebral aparece con frecuencia organizada espacialmente, circunstancia que el modelo neuronal de **mapas de rasgos autoorganizados o SOFM** (*Self-Organizing Feature Maps*) [Kohonen 89, 90] trata de reproducir.

Los SOFM o mapas de Kohonen fueron desarrollados a lo largo de la década de los ochenta por el físico finlandés Teuvo Kohonen [Kohonen 82a, 82b, 88a, 88b, 89, 90], como una continuación natural de la línea de desarrollo de las redes competitivas iniciada por von der Malsburg. Aparte de su interés como una sencilla modelización de redes neuronales naturales [Kohonen 93b], los SOFM poseen un gran potencial de aplicabilidad práctica. De entre las clases de problemas del mundo real en los que han demostrado su eficacia caben citar: clasificación de patrones, cuantificación vectorial, reducción de dimensiones, extracción de rasgos, monitorización de procesos, análisis

exploratorio, visualización y minería de datos [Kohonen 90, 97, Vesanto 99]. Por ejemplo, los SOFM han sido empleados en reconocimiento del habla [Kohonen 88b], control robot [Martinetz 93], optimización combinatorial [Fort 88], clasificación de proteínas [Ferrán 91] y de restos arqueológicos [Castro 98], monitorización de procesos industriales [Tryba 91], emplazamiento de componentes en tarjetas o circuitos integrados (*placement*) [Heman 90], ayuda al diseño de circuitos integrados [Tryba 89], reconocimiento de patrones financieros [Martín del Brío 93a, 95c] y minería de grandes bases de datos en Internet [Kohonen 00]. En [Kohonen 97] hay un completo compendio de aplicaciones.

En este modelo, las neuronas se organizan en una arquitectura unidireccional de dos capas (Figura 3.2). La primera es la capa de entrada o sensorial, que consiste en m neuronas, una por cada variable de entrada, que se comportan como *buffers*, distribuyendo la información procedente del espacio de entrada a las neuronas de la segunda capa. Las entradas son muestras estadísticas $\mathbf{x}(t) \in \mathcal{R}^m$ del espacio sensorial. El procesamiento se realiza en la segunda capa, que forma el mapa de rasgos, consistente habitualmente en una estructura rectangular de $n_x \times n_y$ neuronas (Figura 3.2), que operan en paralelo. Aunque la arquitectura rectangular es la más corriente, a veces también se utilizan capas de una sola dimensión (cadena lineal de neuronas) o de tres dimensiones (paralelepípedo). Etiquetaremos las m neuronas de entrada con el índice k ($1 \leq k \leq m$), y las $n_x \times n_y$ neuronas del mapa con un par de índices $\mathbf{i} = (i, j)$ ($1 \leq i \leq n_x$, $1 \leq j \leq n_y$) que determinan su localización espacial². Cada neurona de entrada k está conectada a todas las neuronas (i, j) del mapa mediante un peso sináptico w_{ijk} .

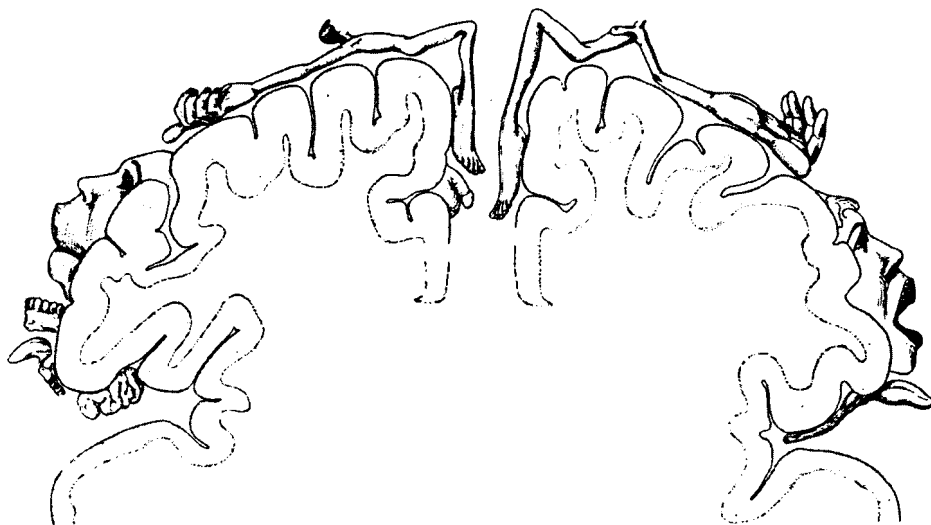


Figura 3.1 *Córtex somatosensorial y motor [Geschwind 79]*

² En ocasiones representaremos un vector de índices mediante un único índice escrito en **negrita**.

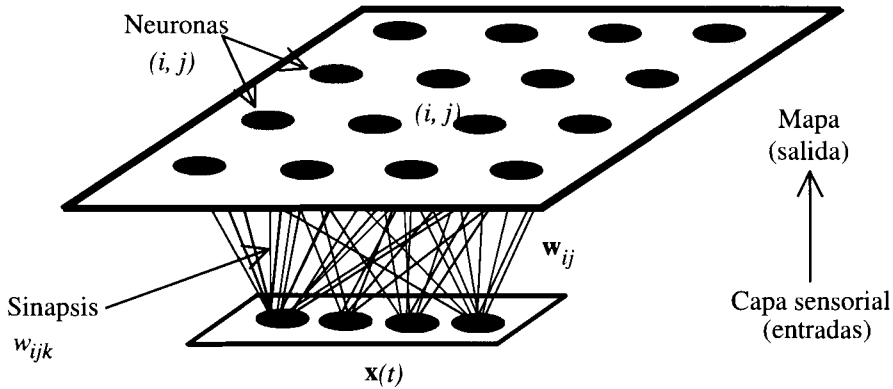


Figura 3.2 Arquitectura del SOFM

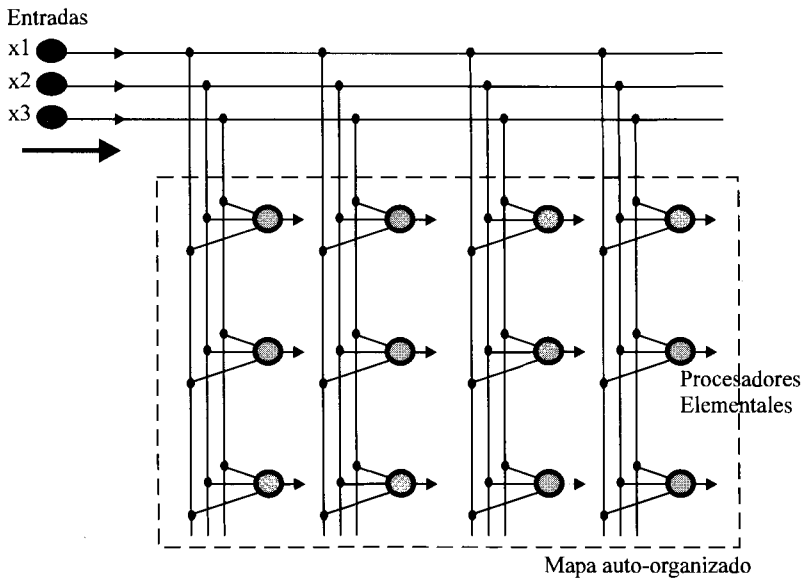


Figura 3.3 Disposición de los procesadores elementales en un SOFM bidimensional

En resumen, el mapa puede describirse como una matriz de procesadores elementales (i, j) ordenados en dos dimensiones (Figura 3.3), que almacenan un vector de pesos sinápticos o vector de referencia (*codebook*) $w_{ij}(t)$, con $\{w_{ij}(t); w_{ij} \in \mathbb{R}^m, 1 \leq i \leq nx, 1 \leq j \leq ny\}$.

En **fase de ejecución** (operación normal de la red), los pesos permanecen fijos. En primer lugar, cada neurona (i, j) calcula la similitud entre el vector de entradas x , $\{x_k \mid 1 \leq k \leq m\}$ y su propio vector de pesos sinápticos w_{ij} , según una cierta medida de distancia o criterio de similitud establecido. A continuación, se declara vencedora la

neurona $\mathbf{g}=(g1,g2)$, cuyo vector de pesos $\mathbf{w}_{\mathbf{g}}$ es más similar al de entradas. De esta manera, cada neurona actúa como un detector de rasgos específicos, y la neurona ganadora nos indica el tipo de rasgo o patrón detectado en el vector de entradas.

$$d(\mathbf{w}_{\mathbf{g}}, \mathbf{x}) = \min_{ij} \{d(\mathbf{w}_{ij}, \mathbf{x})\} \quad (3.1)$$

En la **fase de aprendizaje** cada neurona del mapa sintoniza con diferentes rasgos del espacio de entrada. El proceso es el siguiente. Tras la presentación y procesamiento de un vector de entradas $\mathbf{x}(t)$, la neurona vencedora modifica sus pesos de manera que se parezcan un poco más a $\mathbf{x}(t)$. De este modo, ante el mismo patrón de entrada, dicha neurona responderá en el futuro todavía con más intensidad. El proceso se repite para numerosos patrones de entrada, de forma que al final los diferentes vectores de referencia sintonizan con dominios específicos de las variables de entrada (dominios de Voronoi [Kohonen 88b]), y tienden a representar la función densidad de probabilidad $p(\mathbf{x})$ (o función de distribución) del espacio sensorial. Si dicho espacio está dividido en grupos, cada neurona se especializará en uno de ellos, y la operación esencial de la red se podrá interpretar entonces como un análisis cluster.

Lo descrito hasta el momento responde a un esquema competitivo clásico de relativa sencillez, en el que cada neurona actúa en solitario. Sin embargo, el modelo de mapa de Kohonen aporta una importante novedad, pues incorpora a este esquema relaciones entre las neuronas próximas en el mapa. Para ello introduce una **función de vecindad**, que define un entorno alrededor de la neurona ganadora actual (vecindad); su efecto es que durante el aprendizaje se actualizan tanto los pesos de la vencedora como los de las neuronas pertenecientes a su entorno. De esta manera, en el modelo de SOFM se logra que neuronas próximas sintonicen con patrones similares, quedando de esta manera reflejada sobre el mapa una cierta imagen del orden topológico presente en el espacio de entrada (ordenación de los detectores de rasgos).

En esencia, por medio del proceso descrito los SOFM realizan la **proyección no lineal** de un espacio multidimensional de entrada $\mathbb{R}^m$ sobre un espacio discreto de salida, representado por la capa de neuronas. El mapa representa una imagen del espacio sensorial, pero de menor número de dimensiones, reflejando con mayor fidelidad aquellas dimensiones del espacio de entrada de mayor varianza (que suelen coincidir con los rasgos más importantes de las entradas [Ritter 88]). La proyección se realiza de manera óptima, en el sentido de que la topología del espacio de entrada se preserva en lo posible sobre la superficie de la red (entradas $\mathbf{x}$ similares se corresponden con neuronas próximas). Así, la distribución de las neuronas sobre el mapa resulta ser un reflejo de la función densidad de probabilidad $p(\mathbf{x})$: regiones en el espacio sensorial cuyos representantes $\mathbf{x}$ aparecen con más frecuencia ($p(\mathbf{x})$ mayor) serán proyectadas sobre un número mayor de neuronas en el mapa.

La función vecindad representa matemáticamente de una forma sencilla el efecto global de las interacciones laterales existente entre las neuronas en el cerebro,

pues en vez de considerar en detalle que una neurona trata de activar a sus vecinas y de inhibir a las alejadas (como sucede en el córtex), modelamos esta situación mediante una sencilla función que define el tamaño de la vecindad en torno a la vencedora, dentro de la cual todas las neuronas son premiadas actualizando sus pesos, y fuera de ella son castigadas al no actualizar sus pesos o al hacerlo en sentido contrario. La utilización de la función vecindad en el modelo de mapas auto-organizados aporta respecto del modelo competitivo sencillo dos ventajas adicionales: el ritmo efectivo de convergencia se mejora y el sistema es más robusto frente a variaciones en los valores iniciales de los pesos [Ritter 91a].

Hay que señalar que el tratamiento teórico del modelo de Kohonen es complejo, entre otras razones, porque se demuestra que la dinámica de aprendizaje no puede ser descrita mediante una función de Lyapunov [Erwin 92b] (lo que simplificaría su estudio). Sin embargo, si se elimina la función vecindad, cada neurona individual representa un *mapping* lineal, haciendo factible un análisis teórico que puede proporcionar ideas sobre la operación del modelo completo [Ritter 91a].

3.2.2 Algoritmo de aprendizaje

Como hemos visto, la principal novedad de los SOFM consiste en que la modificación de los pesos no se aplica solamente a una neurona específica (la ganadora), sino también a su vecindad. Al comienzo del entrenamiento la vecindad comprende una amplia región del mapa, lo que permite una ordenación global de los pesos sinápticos. Con el transcurso de las iteraciones, el tamaño de la vecindad se reduce, y finalmente solamente se modifican los pesos de la neurona ganadora. Así, el proceso de aprendizaje comprende dos fases fundamentales: una ordenación global, en la que se produce el despliegue del mapa; y un ajuste fino, en el que las neuronas se especializan.

Un algoritmo de aprendizaje autoorganizado

Se debe tener en cuenta que no existe un algoritmo de aprendizaje totalmente estándar para los SOFM. No obstante, el resultado final es bastante independiente de los detalles de su realización concreta, como puedan ser los pesos sinápticos de partida, el esquema de la actualización del ritmo de aprendizaje, o la forma establecida para la vecindad. A continuación exponemos **un algoritmo autoorganizado** habitual [Martín del Brío 93a, Kohonen 90]:

- 1) **Inicialización de los pesos** sinápticos w_{ijk} . Se puede partir en $t=0$ de diferentes configuraciones: pesos nulos, aleatorios de pequeño valor absoluto (lo más habitual), o con un valor de partida predeterminado.
- 2) En cada iteración, **presentación de un patrón** $\mathbf{x}(t)$ tomado de acuerdo con la función de distribución $p(\mathbf{x})$ del espacio sensorial de entrada (en la muy

habitual situación de disponer solamente de un conjunto finito de patrones de entrenamiento, basta con tomar al azar uno de ellos y presentarlo a la red).

- 3) Cada neurona $\mathbf{i}=(i,j)$ en paralelo del mapa calcula la **similitud** entre su vector de pesos sinápticos $\mathbf{w}_{ij}$ y el actual vector de entradas $\mathbf{x}$. Un criterio de medida de similitud muy utilizado es la **distancia euclídea**:

$$d^2(\mathbf{w}_{ij}, \mathbf{x}) = \sum_{k=1}^n (w_{ijk} - x_k)^2 \quad (3.2)$$

- 4) Determinación de la **neurona ganadora** $\mathbf{g}=(g1,g2)$, cuya distancia sea la menor de todas.
- 5) **Actualización de los pesos** sinápticos de la neurona ganadora $\mathbf{g}=(g1,g2)$, y los de sus neuronas vecinas. La regla más empleada es

$$w_{ijk}(t+1) = w_{ijk}(t) + \alpha(t) \cdot h(|\mathbf{i} - \mathbf{g}|, t) \cdot (x_k(t) - w_{ijk}(t)) \quad (3.3)$$

donde $\alpha(t)$ es un parámetro denominado **ritmo de aprendizaje**. La función $h(\cdot)$ se denomina **función de vecindad**, puesto que establece qué neuronas son las vecinas a la actualmente ganadora. Esta función depende de la distancia entre la neurona $\mathbf{i}$ y la ganadora $\mathbf{g}$, valiendo cero cuando $\mathbf{i}$ no pertenece a la vecindad de $\mathbf{g}$ (con lo que sus pesos no son actualizados), y un número positivo cuando sí pertenece (sus pesos sí son modificados). Como veremos, la vecindad es un conjunto de neuronas centrado en la ganadora. Tanto α como el radio de la vecindad disminuyen monótonamente con t .

- 6) Si se ha alcanzado el número máximo de iteraciones establecido, entonces el proceso de aprendizaje finaliza. En caso contrario, se vuelve al paso 2.

Se puede realizar a continuación **una segunda fase** en el aprendizaje, en la que se produce el **ajuste fino del mapa**, de modo que la distribución de los pesos sinápticos se ajuste más a la de las entradas. El proceso es similar al anterior, tomando $\alpha(t)$ constante e igual a un pequeño valor (por ejemplo, 0.01), y radio de vecindad constante e igual a uno.

En el aprendizaje, el número de iteraciones debe ser suficientemente grande por requerimientos estadísticos, así como proporcional al número de neuronas del mapa (a más neuronas, son necesarias más iteraciones), e independiente del número de componentes de $\mathbf{x}$. Aunque 500 iteraciones por neurona es una cifra adecuada, de 50 a 100 suelen ser suficientes para la mayor parte de los problemas [Kohonen 90]. Entre 20.000 y 100.000 iteraciones representan cifras habituales en la simulación por ordenador del entrenamiento de un SOFM³.

³ Pese a parecer cifras muy altas, las simulaciones de un SOFM usualmente son más rápidas que las del BP, pues su algoritmo es computacionalmente mucho más sencillo.

Una cuestión a tener presente es que el criterio de similitud y la regla de aprendizaje que se utilicen deben ser métricamente compatibles; así ocurre con la distancia euclídea (3.2) y la regla de aprendizaje (3.3). El empleo de diferentes métricas para la fase de recuerdo y para la actualización de los pesos puede causar problemas en el desarrollo del mapa [Demartines 92, Kohonen 97]. En las secciones. 3.6 y 3.7 discutiremos ampliamente las diferentes posibilidades para la elección del criterio de distancia o métrica, y los algoritmos de aprendizaje que de cada una de ellas se derivan. No obstante, podemos adelantar que una medida de similitud alternativa, más simple que la euclídea, es la correlación o **producto escalar**

$$C_{ij} = \sum_{k=1}^n w_{ijk} x_k \quad (3.4)$$

que suele incorporarse al algoritmo, junto con la regla de adaptación (3.3). Sin embargo, dicha regla procede de la métrica euclídea, y la correlación solamente es compatible con esta métrica si se utilizan *vectores normalizados de norma 1* (en cuyo caso distancia euclídea y correlación coinciden). Por esta razón, en ocasiones se realiza la afirmación errónea de que el modelo de Kohonen precisa vectores normalizados. *Si utilizamos la distancia euclídea (3.2) y la regla (3.3), no es necesario tratar con vectores normalizados* (otra cuestión diferente es que en determinado problema dicha normalización pueda ser aconsejable para mantener las entradas dentro de determinado rango dinámico).

Interpretación del algoritmo de aprendizaje

La siguiente interpretación del proceso de aprendizaje puede resultar interesante para comprender la operación de los SOFM. El efecto de la regla de aprendizaje (3.3) no es otro que en cada iteración acercar en una pequeña cantidad el vector de pesos $\mathbf{w}$ de la neurona de mayor actividad (ganadora) al vector de entrada $\mathbf{x}$, como puede apreciarse en la Figura 3.4, donde la expresión

$$\Delta \mathbf{w}(t) = \alpha (\mathbf{x} - \mathbf{w}) \quad (3.5)$$

representa el incremento del vector de pesos de la ganadora ($0 < \alpha < 1$). Así, en cada iteración el vector de pesos de la neurona vencedora rota hacia el presentado, y se aproxima a él en una cantidad que depende del tamaño del ritmo de aprendizaje α . Este hecho también podemos verlo re-escribiendo (3.5) como

$$\Delta \mathbf{w}(t) = \alpha (\mathbf{x} - \mathbf{w}) = \alpha \cdot \mathbf{x} - \alpha \cdot \mathbf{w}$$

de modo que podemos observar que en cada iteración se elimina una cierta fracción del antiguo vector de pesos $\mathbf{w}(t)$ (es decir, la cantidad $-\alpha \cdot \mathbf{w}$ representa un término de olvido), el cual es sustituido por una fracción del vector actual $\alpha \cdot \mathbf{x}$, de modo que en cada paso el vector de pesos de la ganadora $\mathbf{w}$ se parece un poco más al vector de entradas $\mathbf{x}$ que la hace ganar.

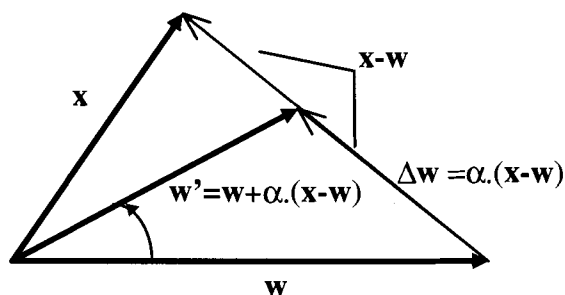


Figura 3.4
Interpretación geométrica de la regla de aprendizaje euclídea

Comienzo del aprendizaje

Fase intermedia

Fin del aprendizaje

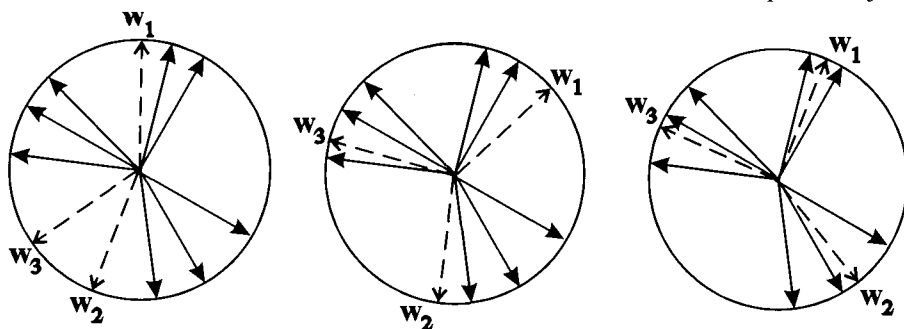


Figura 3.5. Proceso de aprendizaje en dos dimensiones

En la Figura 3.5 se muestra cómo opera la misma regla de aprendizaje para el caso de varios patrones pertenecientes al espacio de entrada, representados en la figura por vectores de trazo continuo (por simplicidad todos los vectores poseerán la misma longitud, el radio de la circunferencia). Supongamos que los vectores del espacio de entrada se agrupan en tres clusters y que el número de neuronas de la red es también tres. Al principio del entrenamiento ($t=0$) los vectores de pesos de las tres neuronas (representados por vectores de trazo discontinuo) son aleatorios y se distribuyen por la circunferencia. Conforme avanza el aprendizaje, y en virtud de la regla (3.3), éstos se van acercando progresivamente a las muestras procedentes del espacio de entrada, para quedar finalmente estabilizados como centroides de los tres clusters.

Consideraciones prácticas: ritmo de aprendizaje y función vecindad

El **ritmo de aprendizaje** $\alpha(t)$ es una función monótonamente decreciente con el tiempo, siendo habitual su actualización mediante una función lineal

$$\alpha(t) = \alpha_0 + (\alpha_f - \alpha_0) \frac{t}{t_\alpha} \quad (3.6)$$

con α_0 el ritmo de aprendizaje inicial (<1.0), α_f el final ($\cong 0.01$) y t_α el máximo número de iteraciones hasta llegar a α_f . Una alternativa es utilizar una función que decrece exponencialmente

$$\alpha(t) = \alpha_0 \left(\frac{\alpha_f}{\alpha_0} \right)^{t/t_\alpha} \quad (3.7)$$

El empleo de una u otra función no suele influir demasiado en el resultado final.

La **función vecindad** $h(|\mathbf{i}-\mathbf{g}|, t)$ define en cada iteración t si una neurona $\mathbf{i}$ pertenece o no a la vecindad de la vencedora $\mathbf{g}$. La vecindad es simétrica y centrada en $\mathbf{g}$, de ahí que representemos como uno de sus argumentos la distancia entre la neurona genérica $\mathbf{i}=(i,j)$ y la vencedora $\mathbf{g}=(g1,g2)$, pues

$$|\mathbf{i}-\mathbf{g}| = \sqrt{(i-g1)^2 + (j-g2)^2} \quad (3.8)$$

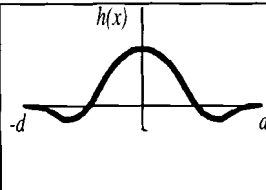
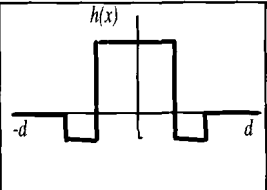
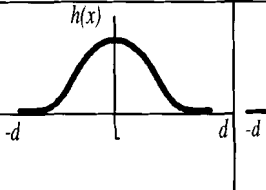
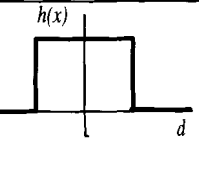
			
Sombrero mejicano	Función pipa	Gaussiana	Rectangular

Tabla 3.1 Diferentes formas de las funciones vecindad

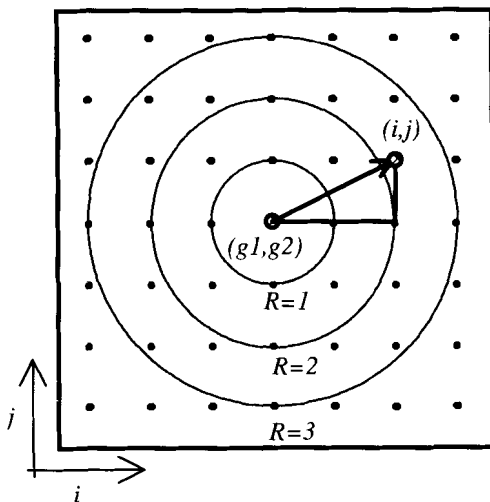


Figura 3.6. Vecindades delimitadas por la función escalón en un mapa de Kohonen para diferentes radios

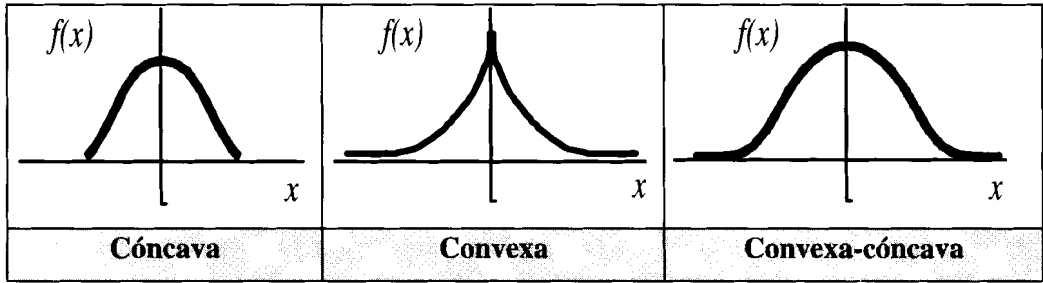


Figura 3.7. Funciones convexa (a), cóncava (b) y convexa-cóncava (c)

En general, $h(\cdot)$ decrece con la distancia a la vencedora, y depende de un parámetro denominado radio de vecindad $R(t)$, que representa el tamaño de la vecindad actual. En realidad, bajo la forma funcional de $h(\cdot)$ se encapsula el complejo sistema de interacciones laterales existente entre las neuronas del mapa, que sí aparecerían explícitamente en una formulación del modelo más orientada a la biología, estableciendo la forma y la intensidad de la interacción entre neuronas.

La función vecindad más simple es del tipo escalón, que denominaremos rectangular

$$h(|\mathbf{i} - \mathbf{g}|, t) = \begin{cases} 0, & \text{si } |\mathbf{i} - \mathbf{g}| > R(t) \\ 1, & \text{si } |\mathbf{i} - \mathbf{g}| \leq R(t) \end{cases} \quad (3.9)$$

Por tanto, en este caso una neurona $\mathbf{i}=(i,j)$ pertenece a la vecindad de la ganadora solamente si su distancia es inferior a $R(t)$. Con este tipo de función las vecindades adquieren forma circular, de bordes nítidos, en torno a la vencedora (Figura 3.6) y la ecuación (3.3) se reduce a

$$\Delta w_{ijk}(t) = \begin{cases} 0, & \text{si } |\mathbf{i} - \mathbf{g}| > R(t) \\ \alpha(t)(x_k(t) - w_{ijk}(t)), & \text{si } |\mathbf{i} - \mathbf{g}| \leq R(t) \end{cases} \quad (3.10)$$

por lo que en cada iteración únicamente se actualizan las neuronas que distan de la vencedora menos de $R(t)$.

También se utilizan a veces funciones gaussianas o en forma de sombrero mejicano (tabla 3.1), continuas y derivables en todos sus puntos, que al delimitar vecindades decrecientes en el dominio espacial, establecen niveles de pertenencia en lugar de fronteras nítidas. En [Lo 91] se demuestra que con una función vecindad decreciente en el dominio espacial se asegura un más rápido ordenamiento en el mapa y la preservación del orden en cada iteración. Con una función tipo escalón, el valor de $R(t)$ debe decrecer más lentamente para alcanzar mapas correctamente ordenados. En [Erwin 92a] se distingue entre funciones vecindad cóncavas y convexas (véase la Figura 3.7), y se muestra que se logra una convergencia más rápida utilizando

funciones vecindad convexas, a la vez que se garantiza la no aparición de estados metaestables.

La función vecindad posee una forma definida, pero su **radio** $R(t)$ varía con el tiempo. Se parte de un valor inicial R_0 grande, que determina vecindades amplias, con el fin de lograr la ordenación global del mapa. $R(t)$ disminuye monótonamente con el tiempo, hasta alcanzar un valor final de $R_f=1$, por el que solamente se actualizan los pesos de la neurona vencedora y las adyacentes. Una posible función de actualización de $R(t)$ es la siguiente, linealmente decreciente

$$R(t) = R_0 + (R_f - R_0) \frac{t}{t_R} \quad (3.11)$$

donde t es la iteración y t_R el número de iteraciones para alcanzar R_f . Existen otras expresiones, como funciones exponencialmente decrecientes, de aspecto similar a (3.7).

Hemos comentado algunas cuestiones prácticas relacionadas con distintos aspectos de los SOFM, ofreciendo diversas alternativas para la realización del algoritmo. No obstante, hay que señalar que la operación del modelo es bastante robusta, y que el resultado es habitualmente correcto si se realiza una actualización suficientemente lenta, independientemente de las alternativas seleccionadas.

En [Kohonen 90, 97] se pueden encontrar **recomendaciones adicionales** para la utilización práctica de este algoritmo. Otra fuente de interesantes consejos es [SOM_PACK 95], el manual de usuario del *SOM_PACK*, software de dominio público desarrollado en la Universidad de Tecnología de Helsinki por el grupo de Kohonen, para el trabajo y experimentación con el modelo de SOFM. De entre las recomendaciones prácticas allí señaladas, merece especialmente la pena exponer la relativa a la elección de la dimensiones n_x y n_y del mapa. Como se señala en [SOM_PACK 95], en determinadas ocasiones un mapa cuadrado, con $n_x=n_y$, puede causar inestabilidades en su desarrollo, por lo que **se recomienda el empleo de mapas rectangulares**, en los que las dimensiones n_x y n_y sean proporcionales al tamaño de los dos ejes principales de la función $p(\mathbf{x})$, es decir, que se ajusten a la forma como se distribuyen los patrones en la realidad. Una manera de determinar esta forma es el empleo del denominado *mapping* de Sammon [Sammon 69, SOM_PACK 95, Kohonen 97], aunque puede bastar un simple análisis de componentes principales.

3.2.3 Algunas variantes de los SOFM

Hasta el momento nos hemos limitado a exponer el modelo de mapas auto-organizados clásico, pero es posible incorporar modificaciones que mejoren algunos aspectos de su operación.

Una de las más interesantes consiste en comenzar el proceso de aprendizaje con un mapa de tamaño pequeño, para ir introduciendo progresivamente nuevas neuronas a medida que se desarrolla el entrenamiento, cuyos pesos se obtienen de la interpolación de los valores de los pesos de las neuronas vecinas, ya presentes en el mapa. Esquemas de **mapas autoorganizados crecientes** se proponen en [Rodrigues 91, Fritzke 93, Alahakoon 00]. Ésta es una forma de resolver el problema de cómo seleccionar de entrada las dimensiones óptimas del mapa autoorganizado, asunto que también trata de resolver Demartines en su interesante modelo autoorganizado **VQP** (*Vector Quantization and Projection*) [Demartines 94].

Un aspecto no tratado hasta el momento es el relacionado con el **efecto de los bordes del mapa**. Debido a que las neuronas situadas en las orillas del mapa poseen menos vecinas, presentan una cierta asimetría respecto de las demás, puesto que se actualizan menos frecuentemente; como resultado los mapas desarrollados aparecerán algo encogidos en sus orillas. Una forma de resolver este problema consiste en establecer un ritmo de aprendizaje mayor para las neuronas de los bordes [Rodrigues 91]. Otra solución es *cerrar la red*, es decir, considerar que una neurona de un extremo del mapa es vecina de la neurona correspondiente del extremo opuesto, obteniendo así una superficie esferoïdal o toroidal (dependiendo de cómo cerremos la red) donde están situadas las neuronas. De este modo se resuelve también el problema relacionado con patrones que durante el proceso de entrenamiento puedan quedar aislados al situarse en un extremo del mapa, con los posibles caminos de salida cortados por prototipos pertenecientes a otras clases.

Una de las variantes de los mapas de Kohonen más conocidas se orienta en el sentido de conseguir que todas las neuronas ganen con la misma frecuencia, de modo que no exista un conjunto de neuronas que monopolicen el proceso y no permitan un correcto desarrollo del mapa. Para lograr esta equiprobabilidad en el proceso WTA, se ha introducido una modificación al algoritmo denominada **mecanismo de consciencia** [DeSieno 88], cuya idea consiste en contabilizar durante el proceso de aprendizaje el número de veces que cada neurona gana, y penalizar a aquellas que ganan en demasiadas ocasiones [DeSieno 88].

Diversas variantes del modelo se muestran en [Kangas 94]. Por ejemplo, allí se presenta el denominado **hipermapa** (*hypermap*), que operando sobre las señales a diferentes niveles (primero trata la información contextual del patrón actual, para procesar a continuación el propio vector de entradas) permite tratar con secuencias temporales. En el mismo trabajo los mapas autoorganizados se generalizan asociando a cada neurona un cierto operador (que puede hacer, por ejemplo, el papel de filtro), de modo que el nuevo modelo de SOFM permita el análisis de efectos dinámicos en las entradas. En esta línea hay que citar el reciente modelo de **ASSOM** (Adaptive-Subspace SOM) [Kohonen 97].

La **red de contra-propagación** (*counter-propagation*), de Hetch-Nielsen [Hetch-Nielsen 90], podría ser considerado una variante de los mapas autorganizados. Se trata de un modelo híbrido, constando en su versión simple de tres capas de

neuronas: capa de entrada, capa oculta no supervisada (de tipo Kohonen) y capa supervisada de salida (de tipo *instar* de Grossberg, de similitudes con la adalina) [Hetch-Nielsen 90]. La idea de este modelo es poder utilizar una red de Kohonen para representación funcional. En esta red el aprendizaje queda dividido en dos fases: una primera no supervisada (para la capa de Kohonen) y una segunda supervisada (para la de Grossberg), lo que hace que su convergencia sea más rápida que la del BP. Sus características son comparables al MLP, aunque al parecer presenta peor generalización. En su versión completa de cinco capas implementa tanto el *mapping* para el que es entrenado, como su inverso.

Por último, la red **LVQ** (*Learning Vector Quantization*) [Kohonen 90] es un modelo supervisado, compuesto por una capa simple de neuronas de Kohonen sin relaciones de vecindad, que en ocasiones se emplea como procedimiento supervisado de ajuste fino del mapa de Kohonen [Kohonen 90, 97]. Este modelo, orientado a **clasificación de patrones**, se basa en premiar a aquellas neuronas que clasifican correctamente un determinado patrón, actualizando sus pesos con la regla convencional, y castigar a las que realizan una clasificación errónea, modificando sus pesos en sentido contrario (restando la cantidad $\Delta w(t)$ en vez de sumarla). Debido a su simplicidad y eficacia en tareas de clasificación, este modelo de red neuronal se utiliza mucho en la actualidad y será estudiado en detalle en el capítulo 4.

El empleo del modelo SOFM en tareas de *clustering* es un error bastante extendido. Por ello Kohonen insiste a menudo [Kohonen 90, 97] en que *el modelo SOFM es útil para visualizar datos, pero no debe emplearse para clasificar (clustering); para este tipo de tareas debe recurrirse al LVQ*, de tipo supervisado.

3.3 EJEMPLOS DE APLICACIONES

Los mapas autoorganizados, como modelo no supervisado, pueden llevar a cabo, entre otros, los siguientes tipos de procesamiento:

- Reducción de dimensiones.
- Preprocesamiento de datos para otros sistemas.
- Monitorización de procesos.
- Cuantificación vectorial.
- Modelado de funciones densidad.
- Análisis cluster (aunque se recomienda emplear LVQ).

Para entender su operación, comentaremos a continuación algunos ejemplos prácticos concretos de su aplicación. Se recuerda que en las siguientes referencias se incluyen **cuestiones muy útiles en las aplicaciones prácticas** de este modelo [Kohonen 90, 97, SOM_PACK 95, Vesanto 99].

Representación de funciones densidad de probabilidad

En primer lugar mostramos la capacidad del SOFM para representar funciones densidad de probabilidad, y cómo preserva la topología del espacio de entrada. Como el número de neuronas que se especializan en reconocer un tipo de patrón está en relación a la probabilidad de aparición de su clase, la distribución de los vectores de referencia aproxima la forma de la función densidad de probabilidad del espacio sensorial⁴.

En un primer caso, clásico en la literatura de los SOFM [Kohonen 89, 90], supondremos un espacio sensorial de dimensión dos, cuyos vectores se distribuyen uniformemente en el interior de un cuadrado de lado unidad, siendo nula la función densidad en el exterior. Para representar esta distribución haremos uso de un mapa de Kohonen cuadrado. Con el fin de visualizar la evolución de los pesos sinápticos en el entrenamiento, representaremos en cada iteración los vectores $\mathbf{w}_{ij}(t)$ de cada neurona (i,j) por un punto, y uniremos con rectas los puntos que corresponden a neuronas vecinas (Figura 3.8). Al principio, los puntos que representan cada procesador aparecen desordenados, pero poco a poco el mapa se despliega; al final la distribución de los pesos sinápticos representa la distribución de los patrones del espacio sensorial, pues ocupan uniformemente el cuadrado. Así, el mapa auto-organizado ha proyectado correctamente un espacio de dimensión dos sobre una red de la misma dimensión.

En ocasiones, debido a una elección de los parámetros de aprendizaje indebida (normalmente, un pequeño radio de vecindad inicial), o a causa de la aleatoriedad del proceso, puede alcanzarse como solución mapas incorrectamente desarrollados, como el de la Figura 3.9. En [Varfis 92] se propone un método sistemático para tratar este tipo de problemas, que consiste en realizar diferentes entrenamientos a partir de diferentes configuraciones iniciales de pesos y, comparando los mapas resultantes, desechar aquellos que se apartan claramente de los demás, por considerarse incorrectamente desarrollados.

En la Figura 3.10 se muestra cómo el mapa de Kohonen aprende a representar una función densidad de probabilidad algo más compleja, en forma de anillo circular. Además, los SOFM pueden modelar densidades de probabilidad de geometrías muy complicadas, cuya definición analítica sea desconocida o inabordable en la práctica.

En [Kohonen 89, 90] se muestran diversos ejemplos en los que el espacio de entrada y el de salida son de diferente dimensión. Por ejemplo, el mapa es capaz de proyectar sobre su superficie un espacio de entrada de dimensión mayor que dos. Para ello los vectores de pesos deben *doblarse*, intentando acomodar el espacio de alta dimensión sobre su superficie, quedando el mapa ordenado por regiones, pero no globalmente.

⁴ Sin embargo, el mapa de Kohonen presenta una cierta tendencia en favor de las regiones de menor densidad de probabilidad, a las que dedica más neuronas de las que en principio les debería corresponder, como veremos en la sección 4.5.

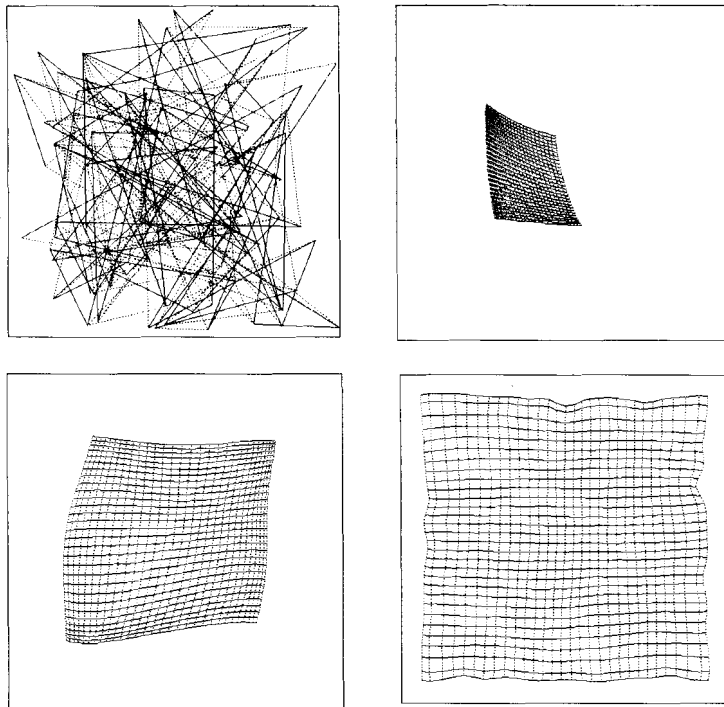


Figura 3.8 Ajuste de una densidad de probabilidad uniforme mediante SOFM. Los vectores de pesos evolucionan hasta ajustar fielmente el espacio sensorial. Se representan las iteraciones 0, 500, 2000 y 20.000

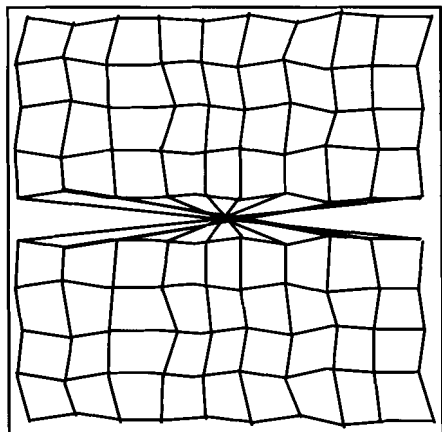


Figura 3.9 Ejemplo de una red de Kohonen incorrectamente desarrollada, en forma de malla retorcida por el centro (twisted mesh)

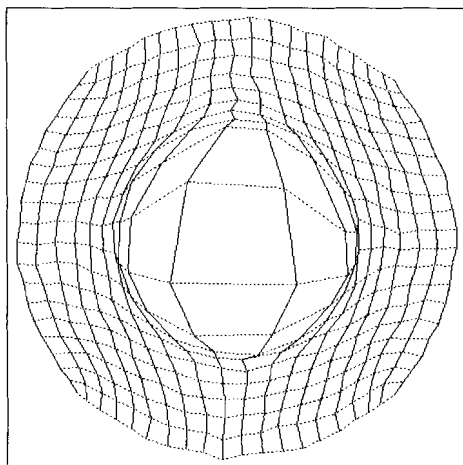


Figura 3.10. Representación de una densidad de probabilidad en forma de anillo circular (mapa 20x20) [Müller 90]

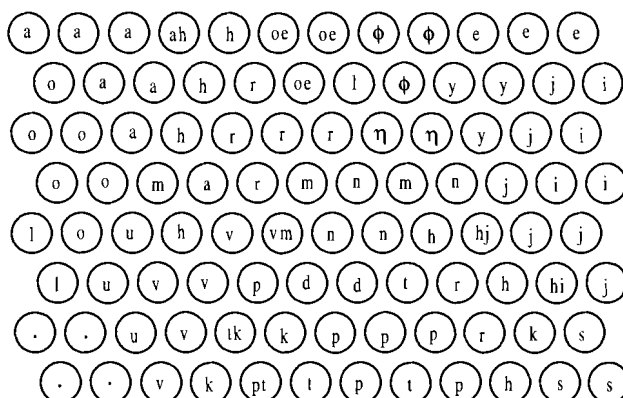


Figura 3.11 Mapa auto-organizado fonético [Kohonen 90]

El mapa fonético

Es un ejemplo muy conocido de aplicación de los SOFM [Kohonen 88b, 90], en el que un mapa auto-organizado se entrena con los fonemas (en realidad su transformada de Fourier) de un cierto idioma (en el trabajo original se empleó el finlandés). Tras la fase de aprendizaje, los vectores de entrada se presentan a la red, quedando representados sobre la superficie del mapa (Figura 3.11), obteniéndose como resultado final un mapa fonético en el que cada neurona se ha especializado en reconocer un tipo de fonema, y en el que sonidos similares o pequeñas variaciones quedan representados sobre neuronas vecinas.

Posteriormente, se aplicó al mapa entrenado el algoritmo LVQ (*Learning Vector Quantization*), que puede utilizarse también como un procedimiento supervisado de ajuste fino del mapa [Kohonen 90]. El sistema así construido alcanza un rendimiento muy alto en el reconocimiento de fonemas, especialmente cuando se introduce en el modelo información contextual. Por otra parte, es de destacar el parecido del mapa fonético con los mapas fonotópicos del córtex temporal del cerebro.

Análisis de datos y monitorización de procesos: la crisis bancaria 1977-85

Un ejemplo muy ilustrativo de la aplicación de mapas autoorganizados al **análisis exploratorio de datos** es el estudio sobre la crisis bancaria española de 1977-1985 que realizamos en [Serrano 93, Martín del Brío 93a, 95c], al que nos hemos referido ya en el capítulo 2. Haciendo uso de los mismos 9 ratios pertenecientes a 66 entidades financieras de la época, 37 solventes y 29 quebradas, se entrenó un mapa autoorganizado de 14×14 neuronas. La gran diferencia con el caso BP (modelo supervisado) es que a la red no supervisada se presentan solamente las entradas, pero no las salidas deseadas. El resultado del aprendizaje debe ser la especialización de cada neurona hacia algún tipo de banco; el empleo de un mapa con mayor número de neuronas (196) que patrones de entrenamiento (66) tendrá como resultado que los 66 bancos queden distribuidos (proyectados) por su superficie con cierta separación.

El resultado final se puede apreciar en la Figura 3.12, en la que se muestra qué neurona responde con más intensidad ante cada patrón. Las neuronas que sintonizaron a bancos quebrados aparecen en dicha figura enmarcados con trazo más oscuro; podemos observar que aparecen definidas con claridad dos regiones, una de crisis y otra de solvencia. Es decir, sin haber proporcionado a la red el dato referente a su situación final de crisis, sólo sus parámetros financieros, los bancos quebrados han quedado situados próximos en una misma zona del mapa (la derecha); el proceso de autoorganización ha encontrado distintos patrones tipo, y el correspondiente a banco quebrado se diferencia claramente del solvente. Éste es un ejemplo muy claro de cómo un conjunto de patrones multidimensionales queda proyectado sobre la superficie bidimensional del mapa de Kohonen.

Suele resultar muy útil **delimitar regiones sobre el mapa**. Por ejemplo, en la Figura 3.13 se ha realizado una delimitación supervisada: se han presentado los 66 patrones a cada neurona, etiquetando como neurona de quiebra (oscura) o de solvencia (clara) dependiendo del tipo de patrón que se activa más. Aplicando distintas técnicas no supervisadas de análisis de los vectores de pesos de las neuronas (por ejemplo, realizando *clustering* sobre ellos) se pueden delimitar automáticamente otras regiones de interés sobre el mapa [Martín del Brío 95b, Vesanto 00], como la zona de situación más crítica, más solvente, zonas intermedias, etc.

Si tras el entrenamiento presentamos los ratios de determinado banco a la red, la situación sobre el mapa de la neurona más activada indicará su estado de crisis o solvencia, de una forma muy visual. El mapa así desarrollado puede emplearse

también para la **monitorización de la evolución temporal** del estado financiero de una entidad, sólo con proporcionarle como entradas los ratios financieros de una compañía a lo largo de un período de tiempo.

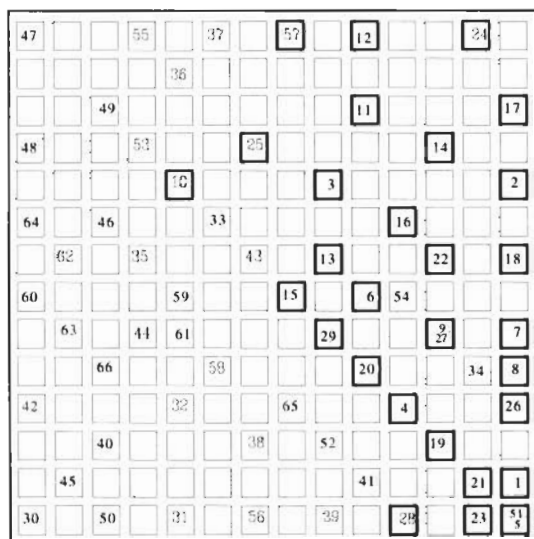


Figura 3.12 Mapa 14x14 entrenado con los datos de quiebra; se señala la neurona que más se activa ante cada uno de los 66 patrones (en trazo ancho los bancos quebrados, del 1 al 29). Puede apreciarse la existencia de zonas de crisis y solvencia

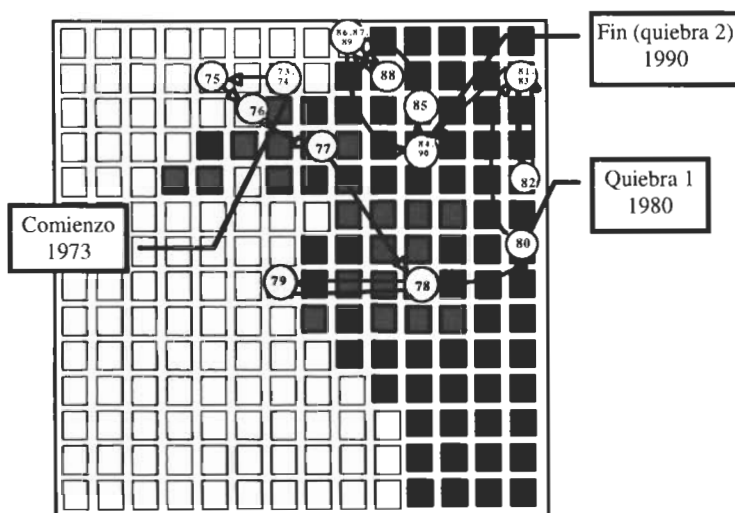


Figura 3.13 Mapa de quiebra bancaria (zona oscura, quiebra; clara, solvencia). Se ha superpuesto la evolución del Banco de Descuento en el período 1973-1990

A modo de ejemplo, en la Figura 3.13 sobre el mapa autoorganizado aparece la trayectoria del Banco de Descuento, entre 1973 y 1990 [Serrano 93, Martín del Brío 93a]. Este banco parte de una situación relativamente buena en 1973, para adentrarse poco a poco en la zona peligrosa del mapa, quebrando (por primera vez) en 1980. Posteriormente es reflotado y cambia de nombre; tras una serie de malos años mejora su situación y trata de salir de la zona peligrosa, aunque en 1990 vuelve a la zona más crítica (Figura 3.13) y quiebra definitivamente.

El caso de estudio de la crisis bancaria ilustra la forma de aplicar el SOFM a la monitorización de procesos complejos, como el de una planta industrial, central de generación de energía, etc., actualmente en desarrollo [Kangas 94, Kohonen 97].

Minería de datos: WEBSOM

Un modelo de SOFM modificado denominado **WEBSOM** se ha propuesto para la organización de la ingente masa de datos disponible en Internet. Remitimos al lector interesado directamente a las referencias [Kohonen 97, Kohonen 00].

3.4 SOFM: CUANTIFICACIÓN ÓPTIMA DE VECTORES

Hasta ahora se han descrito los SOFM como sistemas capaces de realizar *clustering*, representar densidades de probabilidad o proyectar un espacio de alta dimensión sobre otro de dimensión mucho menor. En esta sección, se establecerá la relación existente entre los mapas autoorganizados y la cuantificación óptima de vectores o VQ (*vector quantization*). Esta interpretación, aunque a primera vista muy diferente de las anteriores, guarda estrecha relación con ellas; la exposición aquí desarrollada es paralela a la que se realiza en [Ritter 91b]. El lector no interesado en este tema puede saltar directamente a las sección siguiente.

El objetivo de la cuantificación vectorial consiste en tratar de reducir la cantidad de información requerida para almacenar o transmitir un conjunto numeroso de vectores de un cierto espacio multidimensional, reemplazando cada vector por otro elegido de entre un conjunto (mucho menor) de vectores de referencia o prototipos (*codebook vectors*) que se almacenan en el sistema de VQ. El principal problema de la VQ es la elección del conjunto de vectores de referencia óptimo. La VQ es de vital importancia en, por ejemplo, la transmisión de imágenes.

Consideremos un espacio sensorial de entrada V , compuesto por un gran número de patrones (posiblemente infinito) $v \in V$; el espacio imagen W lo forman un pequeño conjunto de prototipos o vectores de referencia $w_i \in W$, con $W = \{w_{i1}, w_{i2}, \dots, w_{in}\}$. Tanto las redes competitivas (por ejemplo, los SOFM) como la VQ pretenden almacenar un conjunto grande de patrones V encontrando uno mucho menor W de prototipos que sea una buena aproximación del original V . Matemáticamente

$$\text{si } \mathbf{v} \in V \Rightarrow \|\mathbf{v} - \mathbf{w}_{\mathbf{m}(\mathbf{v})}\| \ll \varepsilon, \text{ con } \mathbf{m}(\mathbf{v}) = \arg \min_i \|\mathbf{v} - \mathbf{w}_i\| \text{ y } \varepsilon \text{ pequeño} \quad (3.12)$$

es decir, los vectores de referencia deben ser elegidos de manera que la diferencia entre un vector de V dado y el vector de referencia más próximo sea pequeña. Esta diferencia se relaciona con el concepto de error de cuantificación, que pasamos a definir de manera más precisa. Si los patrones se distribuyen en V según una función densidad $p(\mathbf{v})$, los vectores de referencia deben ser determinados de manera que el valor esperado dado por

$$E(\mathbf{w}) = \int \|\mathbf{v} - \mathbf{w}_{\mathbf{m}(\mathbf{v})}\|^2 p(\mathbf{v}) d\mathbf{v}, \text{ con } \mathbf{w} = (\mathbf{w}_{i1}, \mathbf{w}_{i2}, \dots, \mathbf{w}_{in}) \quad (3.13)$$

sea minimizado. El funcional $E(\mathbf{w})$ es denominado error de cuantificación, y su minimización representa el problema de la **cuantificación óptima de vectores**. Si en vez de transmitir un conjunto de patrones V se envían las etiquetas $\mathbf{m}(\mathbf{v})$ de los prototipos $\mathbf{w}_i$, se reducirá la cantidad de datos a transmitir, manteniendo en lo posible la información original.

Una forma directa de encontrar el mínimo de $E(\mathbf{w})$ es el descenso por el gradiente, modificando iterativamente los prototipos siguiendo el gradiente de $E(\mathbf{w})$

$$\dot{\mathbf{w}}_i \propto \int (\mathbf{v} - \mathbf{w}_{\mathbf{m}(\mathbf{v})}) p(\mathbf{v}) d\mathbf{v} \quad (3.14)$$

$i = \mathbf{m}(\mathbf{v})$

lo que puede escribirse en formulación discreta en la forma

$$\Delta \mathbf{w}_i = \alpha \delta_{i, \mathbf{m}(\mathbf{v})} (\mathbf{v} - \mathbf{w}_{\mathbf{m}(\mathbf{v})}) \quad (3.15)$$

aplicada a una secuencia aleatoria de muestras $\mathbf{v} \in V$, distribuidas según $p(\mathbf{v})$ (con $\alpha \ll 1$). El símbolo δ_{rs} representa la función delta de Dirac. Es interesante observar que la expresión (3.15) coincide con la regla de actualización de pesos de los SOFM (3.3), salvo la función vecindad. Así, la VQ es similar al modelo de Kohonen, pero sin establecer ninguna relación topológica entre los vectores de referencia.

Los dos rasgos adicionales que los SOFM introducen al esquema clásico de VQ descrito son los siguientes: por una parte, los vectores de referencia $\mathbf{w}_i$ se asocian a puntos $\mathbf{i} = (i, j)$ en un dominio de la imagen A (mapa de rasgos), habitualmente ordenados en forma de malla bidimensional; por otra, la regla de aprendizaje (3.15) se completa con una función vecindad, por la que se actualiza tanto el prototipo vencedor como sus vecinos en A .

$$\Delta \mathbf{w}_i = \alpha h(\mathbf{i}, \mathbf{m}(\mathbf{v})) (\mathbf{v} - \mathbf{w}_{\mathbf{m}(\mathbf{v})}) \quad (3.16)$$

con lo que se tiene la regla convencional de Kohonen (3.3). De este modo, además de conseguir un almacenamiento óptimo, se logra que los pesos $\mathbf{w}_i$ se ordenen de modo que la topología de V quede reflejada en A con una mínima distorsión. Así, los SOFM generalizan el esquema VQ de almacenamiento óptimo de la información al

crear una imagen de V en un espacio de pocas dimensiones, en el que además se preservan las relaciones de similitud más importantes de los vectores de V , transformados en relaciones de vecindad en el dominio de la imagen.

La regla (3.16) propuesta minimiza localmente el siguiente funcional

$$F(\mathbf{w}) = \sum_{\mathbf{i}, \mathbf{i}'} h(\mathbf{i}, \mathbf{i}') \int_{m(\mathbf{v})=\mathbf{i}'} \|\mathbf{v} - \mathbf{w}_{\mathbf{i}}\|^2 p(\mathbf{v}) d\mathbf{v} \quad (3.17)$$

Según esta interpretación, *el mapa autoorganizado constituye una memoria o tabla de búsqueda de tamaño limitado que, al no poder almacenar cantidades masivas de datos, guarda en su lugar un conjunto de representantes de los patrones de entrada (los vectores de referencia) que aseguren un aprovechamiento óptimo de la memoria, preservando además en cierto modo su topología.*

Para finalizar, merece la pena citar el trabajo de H. P. Luttrell [Luttrell 90, 89a, 89b], que ha realizado una interpretación más general de la relación del SOFM con la VQ. Incluyendo la probabilidad de aparición de ruido en el canal de transmisión (lo que se traduce en la aparición de errores en las etiquetas enviadas), Luttrell ha demostrado que la expresión $F(\mathbf{w})$ (3.17) representa entonces el valor esperado del error presente en la información transmitida.

3.5 ANÁLISIS FORMAL DEL PROCESO DE AUTOORGANIZACIÓN

En esta sección realizaremos una síntesis del abundante y disperso trabajo teórico que sobre el SOFM se ha desarrollado, desde su introducción a principios de la década de los ochenta.

A pesar de constituir uno de los modelos neuronales más conocidos y de más amplio uso en aplicaciones prácticas, los mapas autoorganizados se resisten a un estudio matemático completo. Hasta el momento, solamente ha podido ser estudiado completamente el caso de un espacio unidimensional de entrada para una red en cadena lineal [Kohonen 82a, 82b, 89, Cottrell 87]. La dificultad de su tratamiento teórico se puede asociar fundamentalmente a dos causas: por una parte, no se ha podido definir adecuadamente, para dimensiones mayores que la unidad, en qué consiste exactamente una configuración correctamente ordenada⁵; por otra, se ha demostrado la imposibilidad de asociar una función energía globalmente decreciente al modelo general de SOFM [Erwin 92b].

Existen multitud de trabajos publicados durante la última década sobre el análisis formal del modelo de mapas de Kohonen. Entre ellos, hay que resaltar tres

⁵ Ha habido intentos, como el de [Lo 91], pero han resultado fallidos, como se muestra en [Kangas 94].

excelentes resúmenes del estado actual [Kangas 94, Cottrell 94, Kohonen 97], donde se incluyen además multitud de referencias.

Existencia de una función energía de Lyapunov para el SOFM

El estudio formal del SOFM se simplificaría notablemente si se demostrase que sus reglas de aprendizaje pueden deducirse de la minimización de una cierta función energía, pues entonces podrían aplicarse las técnicas convencionales, como la de Lyapunov (capítulo 1), para asegurar la convergencia de los algoritmos. Esta posibilidad ha sido extensamente estudiada [Kohonen 91b, Erwin 92b, Tolat 90, Heskes 93], demostrándose en [Erwin 92b] que no es posible encontrar, para un caso general del modelo, una función energía global. En [Tolat 90] se establece la existencia de al menos un sistema de funciones energía, una por cada neurona, que son minimizadas independientemente, y que se estudian en profundidad en [Erwin 92b]. En [Heskes 93] se propone una función error para algoritmos auto-organizados, a partir de la cual se puede deducir una versión particular de los SOFM.

En otro sentido, en el trabajo de [Ritter 88b], se plantea una función energía para el denominado caso discreto de los SOFM, es decir, cuando el conjunto de entrada es un conjunto finito de patrones. Para deducir la función error se destaca que el aprendizaje en la red de Kohonen es un proceso de Markov, con ciertas probabilidades de transición entre los estados que determinan los vectores de pesos. En promedio, la función error planteada es minimizada por el proceso de adaptación. En [Kohonen 91b] se propone una función energía similar, que se estudiará detenidamente en la sección. 3.7, y a partir de la cual derivaremos diferentes algoritmos de aprendizaje.

Ordenación de los vectores de referencia

Otro aspecto interesante consiste en averiguar si los vectores de pesos, inicialmente aleatorios, se ordenan merced a la acción de la regla de aprendizaje definida, de modo que preserven la topología del espacio de entrada. Desde que fue introducido el modelo se han sucedido numerosas demostraciones, pero, hasta hoy, el único caso que ha podido ser completamente estudiado es el de un espacio unidimensional de entrada, para una red en forma de cadena lineal. Las primeras demostraciones se encuentran en [Kohonen 82a, 82b, 89], pero la prueba completa de las propiedades de ordenamiento y convergencia aparece por vez primera en [Cottrell 87], donde se interpreta el aprendizaje como un proceso markoviano. Otra prueba para el caso unidimensional aparece en [Erwin 92b]. Como ya se adelantó, el principal problema para abordar este análisis para dimensiones mayores que uno es que en este caso se desconoce una forma correcta de definir la ordenación.

La propiedad general demostrada del ordenamiento de una mapa de Kohonen unidimensional se establece de la forma siguiente:

Dado un conjunto de n vectores de pesos unidimensionales (escalares) $\{w_i\}$ en un mapa unidimensional, y dados un conjunto de muestras $x(t)$ del espacio de entrada, el proceso de auto-organización definido por las ecuaciones (3.2) y (3.3) adaptará los valores de los pesos de modo que quedarán ordenados, es decir, $w_i < w_{i+1}$, $\forall i < n$, o $w_i > w_{i+1}$, $\forall i < n$; y, tras alcanzar la ordenación, ya no la perderán.

Teoremas de convergencia

En este caso se trata de determinar hacia qué valores convergen los vectores de referencia en el aprendizaje, la posible existencia de mínimos locales [Erwin 92a], y también si la distribución de los pesos es reflejo de la función distribución del espacio sensorial.

En cuanto a los valores hacia los que convergen los pesos, en [Kohonen 82b] se estudia el valor esperado de los pesos para el caso unidimensional, demostrando que, con independencia de la asignación inicial, siempre se llega a una misma solución. Un caso similar y su extensión a dos dimensiones aparecen en [Lo 93], donde se demuestra asimismo la existencia de una única solución.

El grupo alemán de H. Ritter ha tratado extensamente el modelo de Kohonen durante la última década, y en particular sus propiedades de convergencia para casos más generales que los comentados anteriormente. Para mapas con infinito número de nodos y función vecindad de rango también infinito, demostraron que el número de neuronas por área unidad del espacio de entrada $M(\cdot)$ o **factor de magnificación local** se relaciona con la función distribución $p(\mathbf{x})$ según la expresión siguiente [Ritter 86]

$$M(w) \propto p(\mathbf{x})^{2/3} \quad (3.18)$$

por lo que, al representar la distribución del espacio sensorial, el SOFM presentará una cierta desviación en favor de las clases de menor probabilidad.

En [Ritter 88] se estudia la convergencia de los SOFM a partir de una ecuación tipo Fokker-Planck, que describe el comportamiento del mapa durante la fase de aprendizaje, así como sus fluctuaciones en torno al equilibrio.

Un asunto interesante es el fenómeno que aparece cuando dos de las componentes del vector de entradas poseen una varianza mucho mayor que las demás, es decir, cuando existen dos componentes principales. En este caso, los vectores de pesos del mapa bidimensional se orientan de modo que esas dos variables se distribuyen en el mapa tendiendo a seguir las direcciones de los ejes X e Y de la matriz de procesadores. Este fenómeno denominado **selección automática de las dimensiones de los rasgos**, es también comentado e ilustrado con diversos ejemplos en [Kohonen 89]. Si se aumenta la varianza del resto de las componentes hasta determinado nivel, de modo que ya no pueda considerarse la existencia de únicamente dos componentes principales, se producen entonces en el mapa *hinchazones y baches* perpendiculares al plano determinado por las dos componentes señaladas. Estas

ondulaciones representan la tendencia del mapa de encajar un espacio que tiene más de dos dimensiones efectivas sobre otro de únicamente dos: el espacio multivariado es *doblado* para que pueda ser acomodado en dos dimensiones. Además, si el nivel excede un punto crítico, surgirán grandes distorsiones en el mapa. En este caso el mapa puede llevar a cabo una gran reorganización para tratar de encontrar las nuevas dimensiones fundamentales, lo que se manifiesta en forma de una gran sacudida o inestabilidad.

Posteriormente, los mismos autores estudiaron el caso de la red de tamaño finito, limitando la vecindad a n neuronas. En esas condiciones, la densidad de magnificación queda [Ritter 91c]

$$M(w) \propto p(\mathbf{x})^\alpha \quad (3.19)$$

con

$$\alpha = \frac{2}{3} - \frac{1}{3n^2 + 3(n+1)^2} \quad (3.20)$$

Obsérvese que si en esta expresión hacemos tender a infinito el número n de neuronas de la vecindad, se reproduce (3.18).

En [Lo 91] se estudió experimentalmente la velocidad de convergencia del SOFM en relación a la función vecindad, observándose que una vecindad gaussiana proporcionaba una mayor velocidad de convergencia que la habitual función rectangular.

En [Erwin 92a] se estudia la aparición de mínimos locales en función del tipo de vecindad elegida, y se distingue entre funciones de vecindad cóncavas y convexas, demostrándose que en las convexas no aparecen estados metaestables (que sí aparecen en las cóncavas) y que la convergencia es más rápida. Los resultados teóricos mostraban la conveniencia de comenzar con una vecindad amplia, que asegure la ordenación global y evite la aparición de estados metaestables, para luego contraerla al proceder al ajuste fino.

Cuantificación de los pesos

Un asunto fundamental al realizar la implementación física (hardware) de una red neuronal es el estudio de la influencia en la dinámica y propiedades de la red debidas a las limitaciones que la realización hardware impone al modelo. La forma de almacenamiento físico de los pesos sinápticos constituye un ejemplo en este sentido, pues sea analógica o digital, existirá siempre un límite en su precisión y por tanto en sus variaciones incrementales. Este punto es tratado en las referencias [Thiran 92, 93, 94, Martín del Brío 94a].

3.6 MODELOS DE NEURONAS DE KOHONEN. MEDIDAS DE SIMILITUD

El modelo de neurona de Kohonen se basa en el cálculo de la similitud entre el vector de entradas y el de pesos. Así, dependiendo del criterio de distancia que se seleccione, se tendrá un modelo u otro. Expondremos a continuación algunos de los criterios más habituales.

Uno de los más comunes es la **correlación** o **producto escalar**:

$$C_{ij} = \sum_{k=1}^n w_{ijk} x_k \quad (3.21)$$

según el cual, dos vectores serán más similares cuanto mayor sea su correlación. Es interesante observar que una neurona SOFM que utilice este criterio de distancia coincide básicamente con el modelo de neurona estándar de los ANS. Sin embargo, esta medida es sensible al tamaño de los vectores; grandes diferencias en sus longitudes pueden introducir una importante distorsión en la medida de similitud. Para resolver este problema puede dividirse por las normas de los vectores, con lo que se tiene el denominado criterio del **coseno**

$$\cos(\mathbf{w}_{ij}, \mathbf{x}) = \frac{\sum_{k=1}^n w_{ijk} x_k}{\|\mathbf{w}_{ij}\| \cdot \|\mathbf{x}\|} \quad (3.22)$$

Su interés radica en que esta medida se basa en una característica relativa a ambos vectores, como es su ángulo, independientemente de sus tamaños.

Otro de los criterios de más amplio uso es la **distancia euclídea**

$$d^2(\mathbf{w}_{ij}, \mathbf{x}) = \sum_{k=1}^n (w_{ijk} - x_k)^2 \quad (3.23)$$

De acuerdo con este criterio, dos vectores serán más similares cuanto menor sea su distancia. Si se utiliza una red de Kohonen para análisis cluster, la distancia euclídea es más adecuada cuando los grupos a extraer están compuestos por nubes esféricas de puntos en torno a un centro. Si no es así, el algoritmo tratará de ajustar los datos en múltiples grupos esféricos.

La **métrica de Minkowski** representa una generalización del caso anterior

$$d(\mathbf{w}_{ij}, \mathbf{x}) = \left(\sum_{k=1}^n |w_{ijk} - x_k|^\lambda \right)^{1/\lambda}, \quad \lambda \in \mathbb{R} \quad (3.24)$$

coincidiendo con la euclídea para $\lambda=2$. Un caso particularmente interesante de esta métrica es cuando $\lambda=1$, que se corresponde con la **distancia de Manhattan**

$$d(\mathbf{w}_{ij}, \mathbf{x}) = \sum_{k=1}^n |w_{ijk} - x_k| \quad (3.25)$$

Por su sencillez, este criterio es empleado en muchas realizaciones electrónicas del SOFM. Es interesante observar que ni siquiera requiere multiplicaciones, lo que desde el punto de vista del hardware es de sumo interés [Medrano 99]. Veremos que la regla de aprendizaje que se deduce de este criterio es también muy simple.

La correlación, el coseno y la distancia euclídea son los criterios más utilizados, siendo fácil demostrar que coinciden para el caso de vectores normalizados. Por una parte, si las normas son iguales a uno en la ecuación (3.21) se obtiene (3.22), y por otra, desarrollando la ecuación de la distancia euclídea (3.23) y haciendo las normas igual a uno, se obtiene

$$d^2(\mathbf{w}_{ij}, \mathbf{x}) = \|\mathbf{w}_{ij} - \mathbf{x}\|^2 = \|\mathbf{w}_{ij}\|^2 + \|\mathbf{x}\|^2 - 2\mathbf{w}_{ij}^T \cdot \mathbf{x} = 2(1 - \mathbf{w}_{ij}^T \cdot \mathbf{x}) \quad (3.26)$$

de lo que se deduce que una correlación máxima se corresponde con una distancia euclídea mínima, luego ambas medidas también coinciden.

Para vectores normalizados se puede realizar una neurona de Kohonen empleando la correlación (que no es más que una suma ponderada) y la regla de actualización habitual (3.3), que veremos se deduce del criterio de distancia euclídea (sección 3.7). La forma de este modelo coincide con el de neurona estándar definida en el capítulo 1, y resulta además de gran sencillez.

En [Kohonen 89] se comentan otros criterios de distancia, como la medida de similitud de Tanimoto, la de Mahalanobis o la de Hamming, y se discute su empleo para el caso de patrones cuyas componentes no sean números reales, sino variables lógicas o cadenas de caracteres. Por último, en [Martín del Brío 96] se propone un **nuevo modelo de neurona de Kohonen de suma ponderada**, similar a la neurona estándar (por ejemplo, la empleada en el MLP), deduciéndose una simple regla de aprendizaje; su principal interés es que gracias a este modelo puede implementarse un mapa autoorganizado siguiendo el mismo esquema empleado en implementaciones hardware del MLP.

3.7 MODELOS DE APRENDIZAJE EN MAPAS AUTOORGANIZADOS

A continuación se deducen mediante procedimientos sistemáticos distintas reglas de aprendizaje para el modelo de SOFM. De este modo, multitud de algoritmos que aparecen dispersos en la literatura, muchas veces propuestos simplemente por criterios de plausibilidad, quedan descritos aquí en un marco conceptual común.

En primer lugar, presentaremos un procedimiento sistemático para la deducción de reglas de aprendizaje para los SOFM, de acuerdo con las directrices aplicadas ya al

caso de redes supervisadas (capítulo 2). Es decir, *se propone una cierta función objetivo o error, dependiente de los pesos de la red, y se obtiene la regla de actualización a partir de su optimización mediante descenso por el gradiente*, dentro del marco del formalismo de la aproximación estocástica.

Para la definición de una función error, sea $d(\mathbf{x}, \mathbf{w}_{ij})$ una distancia genérica definida en el espacio de las señales, que supondremos diferenciable, y que mide el error de cuantificación para el vector de entrada $\mathbf{x}$. La neurona ganadora $\mathbf{g}$ será la que cumple

$$\mathbf{g} = \arg \min_{ij} \{d(\mathbf{w}_{ij}, \mathbf{x})\} \quad (3.27)$$

La definición de una función error en el caso supervisado resultaba bastante obvia, pues lo que se pretendía era que las salidas actuales tendieran a las deseadas, con lo cual una función objetivo a minimizar consistía en la suma de los errores asociados a cada patrón. En el caso no supervisado la definición no resulta tan evidente, puesto que no se dispone de un objetivo explícito al que deban tender las salidas de la red.

El problema de la imposibilidad de definir una función objetivo para el modelo general de SOFM ya ha sido comentado. Aquí, nuestra intención será encontrar una función error que, aunque no sea una verdadera función potencial, nos permita deducir sistemáticamente reglas de aprendizaje. Como se pretende que los pesos ajusten la distribución de las entradas, un objetivo puede ser que los pesos sinápticos tiendan a ellas, es decir, que los errores de cuantificación sean lo más pequeños posibles. Con esta premisa, puede definirse una función objetivo global de la red de la manera siguiente [Kohonen 93a]

$$E = \int \sum_i h(|\mathbf{i} - \mathbf{g}|) f(d(\mathbf{x}, \mathbf{w}_i)) p(\mathbf{x}) d\mathbf{x} \quad (3.28)$$

con $p(\mathbf{x})$ la función de distribución del espacio sensorial, $f(\cdot)$ una cierta función del error de cuantificación (introducida por generalidad), y $h(\cdot)$ la función vecindad. Esta función objetivo global al mapa se basa en la suma a todas las neuronas de los errores de cuantificación, ponderada en la vecindad, y promediado por medio de la función de distribución para todas las posibles entradas. Como se puede observar, posee un cierto paralelismo con las definidas para redes supervisadas, pero en realidad no se trata de una verdadera función potencial, porque E depende de la neurona vencedora $\mathbf{g}$ y, por tanto, de la entrada actual $\mathbf{x}$ y los pesos $\mathbf{w}_i$ [Kohonen 91b]. En los modelos supervisados la función potencial posee un cierto *paisaje*, que es recorrido durante el aprendizaje hasta alcanzar un mínimo, mientras que aquí la función error cambia su paisaje ante la presentación de cada vector de entrada $\mathbf{x}$. Por lo tanto, el formalismo de la aproximación estocástica aplicado en este caso encontrará como mucho un punto cercano al óptimo o, digamos, un óptimo aproximado [Kohonen 91b].

Para aplicar la aproximación estocástica definiremos la siguiente función

$$E_1(t) = \sum_i h(\|\mathbf{i} - \mathbf{g}\|, t) f(d(\mathbf{x}(t), \mathbf{w}_i(t))) \quad (3.29)$$

que es una muestra tomada en t de la función global objetivo E . Para esta muestra, $\mathbf{g}$ es constante, por lo que su paisaje no varía, y una solución aproximada se obtiene mediante descenso por el gradiente

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) - \lambda(t) \nabla_{\mathbf{w}_i} E_1(t) \quad (3.30)$$

con $\lambda(t)$ el ritmo de aprendizaje, que debe cumplir las dos condiciones habituales⁶

$$\sum_{t=0}^{\infty} \lambda(t) = \infty, \quad \sum_{t=0}^{\infty} \lambda^2(t) < \infty \quad (3.31)$$

Al estar realizando descensos por los gradientes locales proporcionados por $E_1(t)$, y no descensos por la máxima pendiente del paisaje global que proporciona E , se obtienen soluciones no globalmente óptimas. No obstante, en [Kohonen 91b] se muestra que el punto que se alcanza está muy próximo al óptimo, y puede considerarse que las soluciones que proporciona son *casi* óptimas. Este procedimiento permite deducir sistemáticamente algoritmos de aprendizaje solo con cambiar el criterio de distancia $d(\cdot)$ y la función $f(\cdot)$.

Regla de aprendizaje euclídea

Si consideramos como criterio la distancia euclídea

$$d(\mathbf{w}_{ij}, \mathbf{x}) = \sqrt{\sum_{k=1}^n (w_{ijk} - x_k)^2} \quad (3.32)$$

y como función $f(d)=d^2$, la muestra $E_1(t)$ de la función objetivo queda

$$E_1(t) = \sum_{ij} h(\|\mathbf{i} - \mathbf{g}\|, t) \left[\sum_{k=1}^n (w_{ijk} - x_k)^2 \right] \quad (3.33)$$

Si calculamos su gradiente

$$\nabla_{\mathbf{w}_{ijk}} E_1(t) = \frac{\partial}{\partial w_{ijk}} \left\{ \sum_{ij} h(\|\mathbf{i} - \mathbf{g}\|, t) \left[\sum_{k=1}^n (w_{ijk} - x_k)^2 \right] \right\} = \quad (3.34)$$

por ser $\mathbf{g}$ constante para la muestra $E_1(t)$, se tiene

⁶ En [Ritter 88] se llegan a unas expresiones parecidas, pero no idénticas, describiendo el SOFM con una ecuación tipo Fokker-Planck.

$$= \sum_{ij} h(\mathbf{i} - \mathbf{g} \mid, t) \frac{\partial}{\partial w_{ijk}} \left[\sum_{k=1}^n (w_{ijk} - x_k)^2 \right] = 2h(\mathbf{i} - \mathbf{g} \mid, t)(w_{ijk} - x_k) \quad (3.35)$$

y, llamando $\alpha(t) = 2\lambda(t)$, de (3.35) se obtiene

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \alpha(t)h(\mathbf{i} - \mathbf{g} \mid, t)(x_k - w_{ijk}) \quad (3.36)$$

que es la regla de aprendizaje de Kohonen (3.3). Por lo tanto, *la regla convencionalmente utilizada en el aprendizaje de una red de Kohonen procede de la métrica euclídea.*

Regla de aprendizaje de Manhattan

Otro de los criterios de distancia comentados es la norma de Manhattan

$$d(\mathbf{w}_{ij}, \mathbf{x}) = \sum_{k=1}^n |w_{ijk} - x_k| \quad (3.37)$$

que resulta de interés para la **realización hardware** de los SOFM. Para obtener su regla de aprendizaje asociada tomaremos $f(d) = d$,

$$E_1(t) = \sum_{ij} h(\mathbf{i} - \mathbf{g} \mid, t) \left[\sum_{k=1}^n |w_{ijk} - x_k| \right] \quad (3.38)$$

y calculando el gradiente

$$\begin{aligned} \nabla_{w_{ijk}} E_1(t) &= \left\{ \sum_{ij} h(\mathbf{i} - \mathbf{g} \mid, t) \frac{\partial}{\partial w_{ijk}} \left[\sum_{k=1}^n |w_{ijk} - x_k| \right] \right\} = \\ \nabla_{w_{ijk}} E_1(t) &= h(\mathbf{i} - \mathbf{g} \mid, t) \frac{\partial}{\partial w_{ijk}} |w_{ijk} - x_k| = \end{aligned} \quad (3.39)$$

La función valor absoluto no es derivable en el origen. Si consideramos el caso $w_{ijk} > 0$, se tiene

$$\nabla_{w_{ijk}} E_1(t) = h(\mathbf{i} - \mathbf{g} \mid, t) \frac{\partial}{\partial w_{ijk}} (w_{ijk} - x_k) = h(\mathbf{i} - \mathbf{g} \mid, t) \quad (3.40)$$

y para el caso $w_{ijk} < 0$

$$\nabla_{w_{ijk}} E_1(t) = h(\mathbf{i} - \mathbf{g} \mid, t) \frac{\partial}{\partial w_{ijk}} [- (w_{ijk} - x_k)] = -h(\mathbf{i} - \mathbf{g} \mid, t) \quad (3.41)$$

Podemos agrupar ambas expresiones haciendo uso de la función signo

$$y = \text{sign}(x) = \begin{cases} -1, & \text{si } x < 0 \\ 0, & \text{si } x = 0 \\ +1, & \text{si } x > 0 \end{cases} \quad (3.42)$$

y llamando $\alpha(t)=\lambda(t)$ la regla de aprendizaje queda

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \alpha(t)h(\|\mathbf{i} - \mathbf{g}\|, t) \text{sign}(x_k - w_{ijk}) \quad (3.43)$$

La fórmula obtenida, aunque con una apariencia similar a la euclídea, es mucho más simple de realizar, como se pueda apreciar sólo con reescribirla así

$$\Delta w_{ijk}(t) = \begin{cases} +\alpha.h & \text{si } x_k(t) > w_{ijk}(t) \\ 0 & \text{si } x_k(t) = w_{ijk}(t) \\ -\alpha.h & \text{si } x_k(t) < w_{ijk}(t) \end{cases} \quad (3.44)$$

En [Onodera 93] se compara el modelo basado en la distancia de Manhattan con el convencional euclídeo, haciendo uso en ambos de la regla de actualización euclídea (3.3). Realizando numerosas simulaciones se llega a la conclusión de que, aunque ambos alcanzan resultados parecidos, los del modelo euclídeo son alrededor de un 2 % mejores (no obstante, esta pequeña diferencia puede deberse a que en la referencia citada en el modelo de Manhattan se hace uso de (3.3) en vez de (3.37), con lo que la regla de aprendizaje no es compatible con la métrica empleada). De nuestras simulaciones [Martín del Brío 94a, Medrano 99] concluimos que este modelo proporciona resultados similares a los de la regla euclídea, aunque es más sensible a la variación de los parámetros de aprendizaje, que deben ser más cuidadosamente elegidos.

Regla de aprendizaje derivada de la correlación o producto escalar

Si tomamos como base el criterio de la correlación o producto escalar, la neurona vencedora es aquella cuyo vector de pesos presenta la máxima correlación con el vector de entrada actual, dado por (3.21)

$$C_{ij} = \sum_{k=1}^n w_{ijk} x_k$$

Definiremos la muestra de una función error para un tiempo t en la forma

$$E_2(t) = \sum_i h(\|\mathbf{i} - \mathbf{g}\|, t) f(c(\mathbf{x}(t), \mathbf{w}_i(t))) \quad (3.45)$$

con $c(\cdot)$ un cierto criterio de similitud, que es mayor cuanto más parecidos sean $\mathbf{x}$ y $\mathbf{w}_i$, y $f(\cdot)$, una cierta función que se introduce por generalidad. En esta ocasión, se trata de maximizar $E_2(t)$, y se obtiene una solución aproximada iterando de la forma conocida

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \lambda(t) \nabla_{\mathbf{w}_i} E_2(t) \quad (3.46)$$

con $\lambda(t)$ el ritmo de aprendizaje. Obsérvese que en este caso hemos cambiado el signo en el gradiente, puesto que para maximizar debe efectuarse un ascenso por él.

Consideremos que $f(c)=c$, la ecuación (3.45) se convierte en

$$E_2(t) = \sum_i h(\|\mathbf{i} - \mathbf{g}\|, t) \left[\sum_{k=1}^n w_{ijk} x_k \right] \quad (3.47)$$

y calculando el gradiente

$$\nabla_{w_{ijk}} E_2(t) = h(\|\mathbf{i} - \mathbf{g}\|, t) \frac{\partial}{\partial w_{ijk}} \left[\sum_{k=1}^n w_{ijk} x_k \right] = h(\|\mathbf{i} - \mathbf{g}\|, t) x_k \quad (3.48)$$

se obtiene la regla de actualización

$$w_{ijk}(t+1) = w_{ijk}(t) + \alpha(t) h(\|\mathbf{i} - \mathbf{g}\|, t) x_k \quad (3.49)$$

Esta regla presenta el problema de que con las sucesivas presentaciones de $\mathbf{x}$ los pesos pueden crecer indefinidamente, para evitarlo hay que normalizar los pesos en cada iteración

$$w_{ijk}(t+1) = \frac{w_{ijk}(t) + \alpha(t) h(\|\mathbf{i} - \mathbf{g}\|, t) x_k}{\|\mathbf{w}_i(t) + \alpha(t) h(\|\mathbf{i} - \mathbf{g}\|, t) \mathbf{x}\|} \quad (3.50)$$

Esta es, junto con la euclídea (3.3), una de las reglas de aprendizaje más conocidas y empleadas, que se propone en [Kohonen 90], sin ser deducida a partir de criterio alguno.

Un grave problema de este algoritmo de aprendizaje es que la normalización de los pesos que se debe efectuar en cada paso supone un alto coste computacional. Por ello, vamos a deducir a partir de (3.50) un nuevo algoritmo de aprendizaje que preserve la norma de los vectores de pesos; así, si están normalizados inicialmente, también lo estarán en cada iteración, de manera que no sea preciso normalizarlos en cada paso.

Para ello, supondremos que $\|\mathbf{w}_{ij}(t)\|=1$, y pretendemos obtener una regla para que también lo estén en $t+1$. La expresión del desarrollo en serie de Taylor de una cierta función $f(x)$ en torno a un punto a es

$$f(x) = f(a) + f'(a) \cdot (x-a) + \frac{1}{2!} f''(a) \cdot (x-a)^2 + \dots \quad (3.51)$$

Si escribimos la regla de aprendizaje (3.50) en la forma

$$w_{ijk}(t+1) = \frac{w_{ijk}(t) + \alpha h x_k}{\|\mathbf{w}_i(t) + \alpha h \mathbf{x}\|} = \frac{w_{ijk}(t) + \alpha h x_k}{L(\mathbf{w}_i(t) + \alpha h \mathbf{x})} \quad (3.52)$$

y consideramos la norma $L(x)$ como una función dependiente del parámetro α , desarrollando en serie en torno a $\alpha=0$, resulta

$$L(\alpha) = L(0) + \left(\frac{dL}{d\alpha} \right)_{\alpha=0} \alpha + O(\alpha^2) = 1 + \left(\frac{dL}{d\alpha} \right)_{\alpha=0} \alpha + O(\alpha^2) \quad (3.53)$$

donde se han supuesto pesos iniciales normalizados, $L(0)=1$.

De la expresión de la norma al cuadrado

$$\begin{aligned} L^2(\mathbf{w}_i + \alpha h \mathbf{x}) &= (\mathbf{w}_i + \alpha h \mathbf{x})^T (\mathbf{w}_i + \alpha h \mathbf{x}) = \|\mathbf{w}_i\|^2 + \alpha^2 h^2 \|\mathbf{x}\|^2 + 2\alpha h \mathbf{w}_i^T \cdot \mathbf{x} \\ &= 1 + \alpha^2 h^2 \|\mathbf{x}\|^2 + 2\alpha h \mathbf{w}_i^T \cdot \mathbf{x} \end{aligned} \quad (3.54)$$

se obtiene su derivada

$$\frac{dL}{d\alpha} = \frac{2\alpha h^2 \|\mathbf{x}\|^2 + 2h \mathbf{w}_i^T \cdot \mathbf{x}}{2\sqrt{1 + \alpha^2 h^2 \|\mathbf{x}\|^2 + 2\alpha h \mathbf{w}_i^T \cdot \mathbf{x}}} \quad (3.55)$$

y por tanto

$$\left. \frac{dL}{d\alpha} \right|_{\alpha=0} = h \mathbf{w}_i^T \cdot \mathbf{x} \quad (3.56)$$

Así, de (3.53) resulta

$$L(\alpha) = 1 + \alpha h \mathbf{w}_i^T \cdot \mathbf{x} + O(\alpha^2) \quad (3.57)$$

con lo que los pesos en $t+1$ quedan

$$\begin{aligned} w_{ijk}(t+1) &= \frac{w_{ijk}(t) + \alpha h x_k}{L(\mathbf{w}_i(t) + \alpha h \mathbf{x})} = \frac{w_{ijk}(t) + \alpha h x_k}{(1 + \alpha h \mathbf{w}_i^T \cdot \mathbf{x} + O(\alpha^2))} \\ &= (w_{ijk}(t) + \alpha h x_k) (1 - \alpha h \mathbf{w}_i^T \cdot \mathbf{x} + O(\alpha^2)) \end{aligned} \quad (3.58)$$

y desarrollando

$$\begin{aligned} w_{ijk}(t+1) &= w_{ijk}(t) + \alpha h x_k - \alpha h w_{ijk}(t) \mathbf{w}_i^T \cdot \mathbf{x} + O(\alpha^2) \equiv \\ &\equiv w_{ijk}(t) + \alpha h (x_k - (\mathbf{w}_i^T \cdot \mathbf{x}) w_{ijk}(t)) \end{aligned} \quad (3.59)$$

considerando despreciables los términos $O(\alpha^2)$ por ser α pequeño. La expresión (3.59) coincide con la regla de aprendizaje que se propone en [Kohonen 93a], y que se puede escribir en la forma

$$w_{ijk}(t+1) = w_{ijk}(t) + \alpha h(x_k - y_{ij}(t) \cdot w_{ijk}(t)) \quad (3.60)$$

donde se denomina $y_{ij}(t)$ al producto escalar del vector de entradas por el de pesos de la neurona (i,j) , que consideraremos como salida de la neurona (i,j) . El algoritmo obtenido guarda un cierto parecido con la regla de Oja para redes PCA [Oja 92, Kröser 93], que también mantiene normalizados los vectores de pesos.

En [Kohonen 89] se estudian otras propiedades de este algoritmo, aunque la regla no es deducida, sino propuesta a partir de un criterio de plausibilidad.

Por último, en la tabla 3.2 resumimos los diferentes modelos de neurona presentados, junto a sus correspondientes algoritmos de aprendizaje, derivados en esta sección.

MODELO DE NEURONA	FUNCIÓN DISTANCIA	REGLA DE APRENDIZAJE
1) Manhattan	$d(\mathbf{w}_{ij}, \mathbf{x}) = \sum_{k=1}^n w_{ijk} - x_k $	$\Delta w_{ijk}(t) = \alpha(t) \cdot \text{sign}(x_k(t) - w_{ijk}(t))$
2) Produc. escalar		
2a) $\ \mathbf{x}\ = \text{cte} = 1$	$d(\mathbf{w}_{ij}, \mathbf{x}) = \sum_{k=1}^n w_{ijk} x_k$	$\Delta w_{ijk}(t) = \alpha(t) \cdot (x_k(t) - w_{ijk}(t))$
2b) $\ \mathbf{x}\ \neq \text{cte.}$	$d(\mathbf{w}_{ij}, \mathbf{x}) = \sum_{k=1}^n w_{ijk} x_k$	$w_{ijk}(t+1) = \frac{w_{ijk}(t) + \alpha(t) \cdot x_k(t)}{\ \mathbf{w}_{ij}(t) + \alpha(t) \cdot \mathbf{x}(t)\ }$
2c) $\ \mathbf{x}\ \neq \text{cte.}$	$d(\mathbf{w}_{ij}, \mathbf{x}) = \sum_{k=1}^n w_{ijk} x_k \equiv y_{ij}$	$\Delta w_{ijk}(t) = \alpha(t) \cdot (x_k - y_{ij}(t) \cdot w_{ijk}(t))$
2d) $\ \mathbf{x}\ \neq \text{cte.}$	$d(\mathbf{w}_{ij}, \mathbf{x}) = \frac{\sum_{k=1}^n w_{ijk} x_k}{\ \mathbf{w}_{ij}\ \cdot \ \mathbf{x}\ }$	$\Delta w_{ijk}(t) = \alpha(t) \cdot (x_k(t) - w_{ijk}(t))$
3) Euclídea	$d^2(\mathbf{w}_{ij}, \mathbf{x}) = \sum_{k=1}^n (w_{ijk} - x_k)^2$	$\Delta w_{ijk}(t) = \alpha(t) \cdot (x_k(t) - w_{ijk}(t))$

Tabla 3.2. Tabla resumen de los diferentes modelos de mapas autoorganizados

OTROS MODELOS DE REDES NEURONALES

Hasta ahora se han tratado algunos de los modelos más extendidos en el campo de las redes neuronales artificiales, como puedan ser la adalina y el perceptrón (ejemplos de redes no supervisadas), y los mapas autoorganizados (no supervisada). Concluiremos la parte del libro correspondiente a modelos de redes neuronales con algunos otros importantes desde un punto de vista práctico e histórico.

Comenzaremos el capítulo estudiando algunos modelos pertenecientes al grupo de **redes realimentadas**, siendo quizás el **modelo de Hopfield** uno de los más representativos. Trataremos en primer lugar la red de Hopfield discreta (de entradas y salidas digitales), empleada como memoria asociativa o en la eliminación de ruido en imágenes. Cuando el modelo de neurona de Hopfield se interpreta desde un punto de vista probabilístico se obtiene la denominada **máquina de Boltzmann**, que se estudiará brevemente. Posteriormente realizaremos una breve introducción a la **red de Hopfield continua** (de entradas y salidas analógicas), que se emplea en la resolución de problemas de optimización.

A continuación estudiaremos algunos otros ANS que, pese a ser relativamente recientes, están siendo en la actualidad muy utilizados en las aplicaciones prácticas, como las **funciones de base radial o RBF** (*Radial Basis Functions*) y el **LVQ** (*Learning Vector Quantization*). Finalmente, y por completar el panorama, haremos referencia a algunos **otros modelos de ANS** no tratados en este libro.

4.1 REDES NEURONALES REALIMENTADAS

Los modelos que hemos estudiado hasta el momento son básicamente de tipo unidireccional (*feedforward*), en los cuales la ausencia de realimentación hace que la información se propague en un único sentido (de las entradas a las salidas). Estos

modelos neuronales relativamente sencillos son ya difíciles de analizar, pero si añadimos además la posibilidad de que exista realimentación, el análisis de su operación se complicará más aún. No obstante, el papel de la realimentación es fundamental en las redes neuronales biológicas, por lo que se ha incorporado a algunos modelos de redes neuronales artificiales.

Ante un determinado patrón de entradas una red unidireccional proporciona inmediatamente una respuesta de salida, por lo que su operación es siempre **estable** en fase de recuerdo o ejecución. Pero si introducimos realimentaciones en este esquema, la información se propagará tanto hacia adelante como hacia atrás, comportándose por lo tanto como un **sistema dinámico**, de difícil análisis, en general, y en el que **deberá garantizarse la estabilidad de su respuesta**.

Por ese motivo, en las aplicaciones prácticas a menudo se hace uso de redes no realimentadas, de estudio más simple y comportamiento más *fiable*, ya que en fase de ejecución siempre convergen. Además, debemos recordar que se ha demostrado que modelos no realimentados de relativa sencillez, como el BP o las RBF, que estudiaremos más adelante, son aproximadores universales, por lo que pueden utilizarse para abordar una muy amplia clase de problemas.

Como ya hemos apuntado, la respuesta de la red realimentada en ocasiones se estabilizará tras un determinado número de iteraciones, convergiendo a un **estado estable, punto fijo o atractor**. En otros casos la respuesta puede no converger, describiendo una trayectoria caótica en el espacio de las salidas. En definitiva, en el caso de las redes realimentadas surge el problema fundamental relacionado con la **estabilidad de la respuesta de la red neuronal**. En este sentido, como se expuso en el capítulo 1, existen teoremas generales [Cohen 83, Simpson 89] que establecen las condiciones que debe cumplir una red realimentada para que su respuesta sea estable.

La demostración de la estabilidad de la respuesta suele hacer uso del **método de Lyapunov** [Simpson 89], como alternativa a la más tediosa integración del sistema de ecuaciones diferenciales que describen su operación. Recordemos que, esencialmente, el método de Lyapunov establece que si en un sistema dinámico de variables de entrada $(x_1, x_2, \dots, x_n)$ y descrito por el sistema de ecuaciones diferenciales

$$\dot{x}_i \equiv \frac{dx_i}{dt} = F(t, x_1, x_2, \dots, x_n) \quad (4.1)$$

se cumple que el sistema está en reposo solamente en el origen, existen en todo el dominio las derivadas de las ecuaciones que lo describen, y las variables están acotadas, entonces, si se puede encontrar una **función de Lyapunov** V de las variables x_i , $V: \mathcal{R}^n \rightarrow \mathcal{R}$, tal que

$$\dot{V} = \sum_{i=1}^n \frac{\partial V}{\partial x_i} \dot{x}_i \leq 0, \quad \forall x_i \quad (4.2)$$

se garantiza que el sistema converge para todas las posibles entradas $(x_1, x_2, \dots, x_n)$, y es además globalmente estable.

La función de Lyapunov se denomina también **función energía de Lyapunov**, pues constituye una generalización del concepto físico de energía. Es interesante recalcar que con este formalismo matemático simplemente se está expresando que si somos capaces de encontrar una cierta función energía de la red que disminuya con su operación, entonces ésta será estable, de modo que disponemos de una manera relativamente simple de estudiar sus propiedades de estabilidad. Veremos más adelante que Hopfield [Hopfield 82, 84] empleó precisamente un esquema similar para demostrar que su modelo de red recurrente era estable en el caso de que la matriz de pesos sinápticos fuese simétrica y de diagonal nula.

Esta técnica es también la que Cohen, Grossberg y Kosko han aplicado en los teoremas señalados para demostrar la estabilidad de una amplia clase de redes neuronales realimentadas, autoasociativas (teorema de Cohen-Grossberg [Cohen 83], y de Cohen-Grossberg-Kosko [Kosko 88] para las adaptativas) y heteroasociativas (teorema ABAM¹ de Kosko [Kosko 92a], para adaptativas).

En este capítulo centraremos nuestra atención en uno de los modelos de redes realimentadas más populares, el **modelo de Hopfield**, que cuenta con el interés añadido de que su análisis y aplicación fue una de las causas del renacimiento de los ANS a principios de los años ochenta. Este modelo supuso la chispa que provocó un ingente trabajo posterior relacionado con la aplicación de los potentes métodos de la física estadística al estudio de sus propiedades, tratamiento que posteriormente se ha extendido a otros modelos neuronales. Los libros de Amit [Amit 89], Müller y Reinhardt [Müller 90], Domany y otros [Domany 91], y Hertz y otros [Hertz 91], representan una síntesis muy completa del trabajo llevado a cabo; en [Haykin 99] se encuentra una versión resumida. Para una breve visión de las relaciones entre física estadística y redes neuronales es recomendable el artículo [Sompolinsky 88].

4.2 MODELO DE HOPFIELD

4.2.1 Modelo de neurona y arquitectura. Dinámicas

En la literatura existen distintos modelos de redes realimentadas (ART, BAM, ABAM, etc. [Simpson 89]), aunque el denominado **red de Hopfield** (por haber sido popularizado por el norteamericano J. J. Hopfield) quizás sea el más conocido. No obstante, hay que tener presente que otros investigadores, como Little, Amari o Grossberg, trataron ya previamente a Hopfield modelos neuronales muy similares

¹BAM≡*Bidirectional Associative Memory*; ABAM≡*Adaptive BAM*. El BAM es una red recurrente heteroasociativa discreta, y el ABAM es su versión analógica (continua) [Kosko 92a].

(véase, por ejemplo, [Grossberg 87], o las referencias en [Hopfield 82]). Por ello, el mérito de Hopfield no consistió en la propuesta de una nueva arquitectura de red, sino en las nuevas ideas que sobre un viejo modelo propuso en su trabajo original [Hopfield 82]: por una parte, resultaba novedoso el enfoque empleado en su tratamiento, basado en la definición de una función energía de la red; por otra, su llamada de atención sobre las relaciones de este modelo de red neuronal con ciertos modelos de la física estadística, lo que permitía emplear para su estudio potentes herramientas matemáticas; por último, Hopfield subrayó la facilidad de realización electrónica del modelo, en especial aprovechando la en aquella época recientemente introducida integración VLSI [Hopfield 82].

Hopfield, físico de reputación, proporcionó además a principios de los años ochenta un toque de calidad y credibilidad al trabajo en redes neuronales, del que quizás este campo adoleciera desde hacía tiempo, siendo de este modo uno de los más claros responsables de su renacimiento. Sus dos trabajos seminales [Hopfield 82, 84] pueden considerarse clásicos.

La **arquitectura de la red de Hopfield** consiste básicamente en una única capa de neuronas, donde cada una se conecta con todas las demás (Figuras 4.1 y 4.2). El **modelo de neurona** empleado se describe más detalladamente en la Figura 4.3. Se trata de una neurona similar a la del perceptrón, un sencillo elemento de umbral, con entradas $x_j(t)$ binarias ($\{0, 1\}$ o $\{-1, +1\}$, según convenga), y salidas que en principio llamaremos $y_i(t)$, también binarias. Los pesos w_{ij} son continuos, es decir, números reales. La neurona i calcula el potencial postsináptico h_i (también denominado potencial local o de membrana), como la suma de las entradas ponderada con los pesos sinápticos, menos un cierto umbral de disparo

$$h_i(t) = \sum_j w_{ij} x_j(t) - \theta_i \quad (4.3)$$

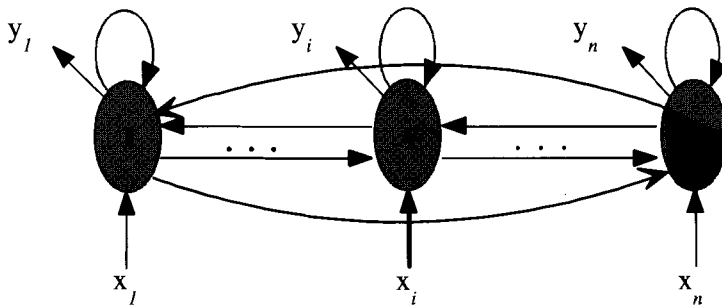


Figura 4.1. Esquema genérico de la red de Hopfield

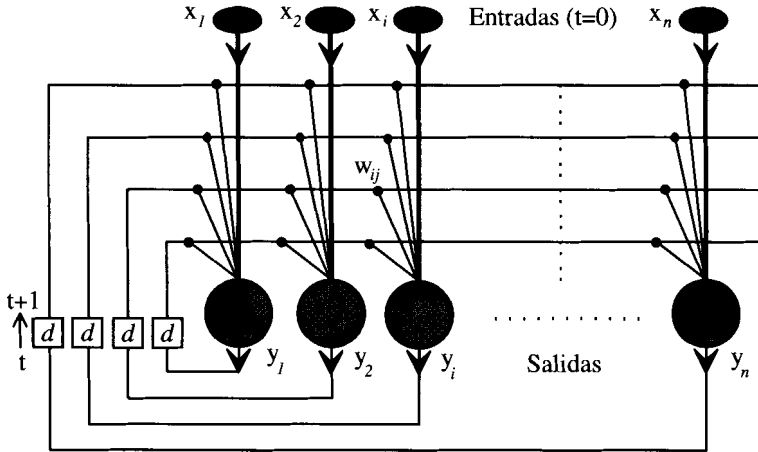


Figura 4.2 Arquitectura de la red de Hopfield. La letra 'd' representa un retraso (delay) de una iteración (una salida y en t se convierte en una entrada x en el instante posterior $t+1$)

Finalmente, al potencial local se aplica una función de tipo escalón $f(\cdot)$ para obtener la salida digital de la neurona

$$y_i(t) = f(h_i(t)) = f\left(\sum_j w_{ij} x_j(t) - \theta_i\right) \quad (4.4)$$

Cuando las neuronas hacen uso de valores $\{-1, +1\}$ se denominan de tipo **Ising** [Müller 90], siendo la función de activación que emplea la *signo-de-equis*, $f(x) = \text{sign}(x)$. En este caso, el estado $+1$ indica una neurona activada, y el -1 desactivada.

En un primer estadio $t=0$, las neuronas reciben las entradas provenientes del exterior $x_j(0)$, calculando las salidas asociadas $y_i(0)$. Debido a las realimentaciones (Figura 4.2), estas salidas se convierten en las nuevas entradas de la red $x_j(1)$, proporcionando ésta una nueva salida $y_i(1)$. En general, ante las entradas $x_i(t)$, la red proporciona una salida $y_i(t)$, que se convierten en las nuevas entradas $x_i(t+1)$, de modo que la operación de la red de Hopfield se puede expresar como

$$x_i(t+1) = f\left(\sum_j w_{ij} x_j(t) - \theta_i\right) \quad (4.5)$$

regla que rige su **dinámica**. En resumen, al comienzo ($t=0$) cada neurona recibe del exterior un vector de entradas $x_j(0)$, de modo que cada neurona i queda situada inicialmente en ese estado, cuyo valor será 0 o 1 (o bien, -1 o $+1$, según nuestra elección). A partir de entonces, la red neuronal interrumpe su comunicación con el exterior y procesa las entradas recibidas; para tiempos $t+1$ sucesivos, las entradas de

cada neurona son el estado en el que se sitúan en el instante anterior t . Habitualmente diremos que el estado $\mathbf{x}$ de la red de Hopfield en t viene determinado por el estado de activación de todas y cada una de sus neuronas

$$\mathbf{x}(t) = (x_1(t) \quad \dots \quad x_i(t) \quad \dots \quad x_n(t))^T \quad (4.6)$$

La dinámica definida según (4.5) es determinista. Más adelante definiremos otra de tipo no determinista.

Por último, debido a la equivalencia de este tipo de red con los modelos de vidrios de espín² (*spin-glasses*) de la mecánica estadística, en ocasiones se emplean en la red de Hopfield términos derivados de aquella, en sustitución de los más convencionales pertenecientes al campo de las redes neuronales. Por ejemplo, una entrada x_i en ocasiones se denomina espín, que al poder tomar los valores -1 o $+1$ (para 0 o 1 basta realizar un cambio de coordenadas), se tiene un espín hacia arriba o un espín hacia abajo (como el espín del electrón, $-1/2$ o $+1/2$). Por lo tanto, la neurona se interpreta como una partícula con espín hacia arriba o hacia abajo. Por otra parte, el potencial postsináptico h_j (4.3) muy a menudo se denomina campo local, pues siguiendo la analogía con la física, representa el campo que un conjunto de neuronas j de espín x_j crearía localmente sobre la neurona i .

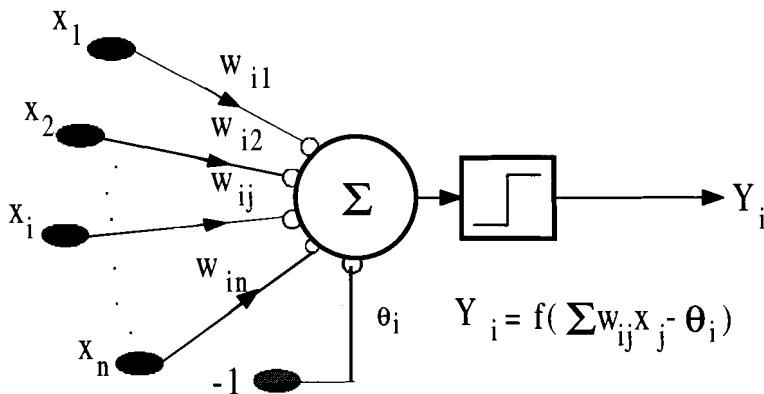


Figura 4.3. Neurona tipo umbral del modelo de Hopfield

²Los vidrios de espín son sistemas magnéticos, compuestos por cierto sustrato no magnético en cuyo seno se distribuyen un conjunto de dipolos magnéticos atómicos o espines, que pueden orientarse en dos direcciones opuestas, es decir, que pueden situarse en dos estados, hacia arriba o hacia abajo (sistema de Ising [Müller 90]). Estas partículas interactúan entre sí mediante interacciones ferromagnéticas y antiferromagnéticas distribuidas aleatoriamente. Se trata de un prototipo de los denominados sistemas *frustrados*, cuyos elementos no pueden satisfacer a la vez todas las restricciones a las que se ven sometidos, motivo por el que pueden situarse en multitud de distintas configuraciones correspondientes a mínimos locales de la energía, en vez de alcanzar la situación de mínima energía a la que todo sistema físico debería tender [Sompolinsky 88].

Dinámicas de la red

Dada la regla que rige la actualización del estado de una neurona individual (ecuación 4.5), podemos plantearnos a continuación los diferentes esquemas de actualización o **dinámicas** de las neuronas de la red (es decir, si todas las neuronas se activan a la vez, o más bien una tras otra, en un determinado orden o aleatoriamente, etc.). Podemos distinguir dos tipos genéricos [Müller 90]:

- a) **Dinámica asíncrona o modo serie** de operación, también denominada de Glauber. En un instante t solamente una neurona de la red actualiza su estado. Ésta puede ser seleccionada aleatoriamente, o bien siguiendo un cierto orden preestablecido.
- b) **Dinámica síncrona o modo paralelo** de operación, también denominada de Little. En un instante t varias neuronas actualizan su estado a la vez. En el caso en que todas las neuronas de la red lo hagan al unísono se tiene el **modo completamente paralelo**. No obstante, cuando hablemos de dinámica síncrona o paralela nos referiremos a la actualización de todas a la vez, por ser el caso más común.

Distintas dinámicas aplicadas sobre una misma arquitectura de red de Hopfield hacen que opere de manera diferente y, por lo tanto, que el estado final de la neuronas sea también distinto [Bruck 90], aspecto este que estudiaremos más adelante.

4.2.2 Memoria asociativa

En los computadores convencionales la mayor parte de la **memoria es de acceso directo**. En ellos, la memoria se encuentra dividida en casillas, numeradas desde 0 al número de casillas presentes menos uno, donde cada una almacena un retazo de información o ítem (un byte, normalmente). El número de orden que se asigna a cada casilla se denomina dirección de memoria. Si queremos recuperar un ítem, basta con direccionar (dar la dirección de) la casilla donde ésta se encuentra almacenada, de ahí que a este tipo de memoria la denominemos de acceso directo, pues accedemos directamente a un contenido sólo con proporcionar su dirección. En resumen, para recuperar un dato de la memoria de un computador se necesita conocer la dirección donde éste se encuentra guardado, y si se produce un error en el almacenamiento, el contenido se verá modificado y sin posibilidad de recuperar el original. Por ello se dice que este esquema no es tolerante a fallos.

En el cerebro la organización de la memoria es totalmente diferente³. La información, al parecer, no se halla en este caso alojada en casillas de memoria definidas, sino más bien aparece dispersa (distribuida). Para recuperar un retazo de información, no buscamos, por ejemplo, en la casilla de memoria número 43.537 de

³Un interesante repaso a esta cuestión puede encontrarse en [Kohonen 89].

nuestro cerebro, sino que la recuperamos a partir de alguna pista, información parcial, o por contexto. Por ejemplo, para recordar el color de pelo de Juanito Pérez, pensamos en la etiqueta *Juanito Pérez*, y casi instantáneamente aparece su imagen en nuestra consciencia. La operación de nuestra memoria se aproxima más a la de las llamadas **memorias asociativas**, también denominadas memorias direccionables por contenido o **CAM** (*Content Addressable Memory*), debido a que asocian ideas con ideas, ideas con imágenes, imágenes con etiquetas, etc.

Las redes neuronales operan siguiendo este mismo esquema. Por ejemplo, un MLP implementa una memoria asociativa de tipo **heteroasociativo**, al asociar un patrón de entrada con cierto patrón de salida diferente, en general. Por el contrario, vamos a ver a continuación cómo el modelo de Hopfield toma el papel de una memoria de tipo **autoasociativo**, pues, en definitiva, está ideada para asociar un patrón de entradas consigo mismo.

La red de Hopfield como memoria asociativa. Estados estables

Hemos visto en la sección anterior que, dado un conjunto de entradas binarias $\mathbf{x}(0)$ procedentes del exterior, la red de Hopfield proporciona una salida $\mathbf{y}(0)$, que es a su vez empleada como nueva entrada de la red en un instante posterior $\mathbf{x}(1)$. Podemos interpretar este proceso como que la red neuronal, inicialmente en el estado $\mathbf{x}(0)$ pasa a estar en el $\mathbf{x}(1)$. En general, una red de Hopfield, en cada iteración t , pasa de un estado $\mathbf{x}(t)$ otro estado $\mathbf{x}(t+1)$. El proceso finalizará cuando se alcance un **estado estable** o punto fijo de la red $\mathbf{x}^*$, es decir, un estado que cumpla la siguiente condición a partir de un cierto t

$$\mathbf{x}(t+1) = \mathbf{x}(t) \equiv \mathbf{x}^* \quad (4.7)$$

pues ello supondrá que la salida de la red ya no cambia, es decir, que la red se ha estabilizado. En ese momento podemos suponer que la red neuronal ha acabado de procesar el patrón original procedente del exterior $\mathbf{x}(0)$, siendo $\mathbf{x}^*$ la respuesta final que proporciona.

Alternativamente, y siguiendo la analogía con la física, puede demostrarse que **en un estado estable el espín de cada neurona se encuentra alineado con su campo local**, es decir

$$x_i^* \cdot h_i(t) > 0, \quad \forall i \quad (4.8)$$

Para verlo, supongamos $x_i(t) \cdot h_i(t) > 0$, entonces

$$x_i \cdot h_i(t) > 0 \Rightarrow \begin{cases} \text{si } h_i(t) > 0 \Rightarrow x_i(t+1) > 0 \\ \text{si } h_i(t) < 0 \Rightarrow x_i(t+1) < 0 \end{cases} \Rightarrow x_i(t) = \text{sign}(h_i(t)) \quad (4.9)$$

por lo tanto, se tiene

$$x_i(t) = \text{sign}(h_i(t)) = x_i(t+1) \Rightarrow x_i(t) = x_i(t+1) \quad (4.10)$$

que es la expresión (4.7) (condición de alineamiento) que define la estabilidad del estado $x_i(t)$, con lo que queda demostrado.

Más adelante veremos que un estado estable es un mínimo local de una cierta función energía de Lyapunov que se definirá para la red, de forma que de cualquiera de las tres maneras expuestas podremos definir un estado estable.

A continuación procederemos a la interpretación de la operación de la red de Hopfield. El objetivo del entrenamiento en este modelo es almacenar un determinado conjunto de patrones $\mathbf{x}^\mu$ en la forma de estados estables de la red. De este modo, ante un cierto patrón de entradas $\mathbf{x}$ se espera que la salida sea uno de los estados estables memorizados $\mathbf{x}^\mu$, por lo que la red actúa como una memoria asociativa, asociando cierto patrón de entrada a otro de salida. Por otra parte, si presentamos a la red como entradas cierto estado estable $\mathbf{x}^\mu$ de los memorizados, la respuesta será obviamente el mismo estado $\mathbf{x}^\mu$ por definición de estado estable, de modo que **la red de Hopfield opera como una memoria autoasociativa**. El interés de una memoria autoasociativa es el siguiente: si en vez de presentar inicialmente uno de los patrones memorizados $\mathbf{x}^\mu$ presentamos uno de ellos, pero que incorpore cierto nivel de ruido, $\mathbf{x}^\mu + \text{ruido}$, se espera que la red neuronal evolucione hasta alcanzar el estado estable más próximo, que deberá ser precisamente el estado memorizado (sin ruido) $\mathbf{x}^\mu$, recuperando el patrón original (que podría ser, por ejemplo, una fotografía o la imagen de un carácter). Así, mediante esta red autoasociativa podemos eliminar ruido en patrones (por ejemplo, imágenes). La red se dice entonces que es tolerante al ruido o que filtra el ruido presente en las señales de entrada.

4.2.3 Función energía de la red

Supongamos el **modelo original de Hopfield discreto** [Hopfield 82], con neuronas tipo umbral de salida $\{0, 1\}$ y dinámica asíncrona, por la cual en cada iteración la neurona a la que le corresponde actualizar su estado según (4.5) es elegida aleatoriamente. Supongamos que no existe realimentación de una neurona consigo misma, es decir, $w_{ii}=0$, o lo que es lo mismo, la matriz sináptica es de diagonal nula.

En vez de proponer directamente la función energía de este sistema, obtendremos su expresión siguiendo el razonamiento de [Aleksandridis 90]. De la regla de activación (4.5), se tiene

$$x_i(t+1) = f(h_i(t)) \Rightarrow \begin{cases} \text{si } h_i(t) > 0 \Rightarrow x_i(t+1) = 1 \Rightarrow \Delta x_i(t) = x_i(t+1) - x_i(t) \geq 0 \\ \text{si } h_i(t) < 0 \Rightarrow x_i(t+1) = 0 \Rightarrow \Delta x_i(t) = x_i(t+1) - x_i(t) \leq 0 \end{cases} \quad (4.11)$$

y por lo tanto,

$$h_i(t) \cdot \Delta x_i(t) \geq 0, \text{ ó bien, } \left(\sum_j w_{ij} x_j(t) - \theta_i \right) \Delta x_i(t) \geq 0 \quad (4.12)$$

Definimos el incremento de energía de la red cuando la neurona i se actualiza como

$$\Delta E_i = - \left(\sum_{j=1}^n w_{ij} x_j(t) - \theta_i \right) \Delta x_i(t) \quad (4.13)$$

que por construcción es siempre menor o igual a cero. A partir de (4.13), podemos definir la energía de una neurona individual en la red como

$$E_i = -x_i(t) \cdot \left(\sum_{j=1}^n w_{ij} x_j(t) - \theta_i \right) \quad (4.14)$$

Finalmente, sumando para todas las neuronas, suponiendo que la **matriz de pesos es de diagonal nula** $w_{ii}=0$, y **además simétrica** $w_{ij}=w_{ji}$, la expresión que se obtiene es

$$E = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i x_j + \sum_{i=1}^n \theta_i \cdot x_i \quad (4.15)$$

que es la **energía de la red**. Por construcción, la variación de esta energía es siempre menor o igual a cero ($\Delta E \leq 0$), o lo que es lo mismo, en la operación de la red (ecuación 4.5) su energía nunca crece. Como la energía está limitada, la red siempre llegará a un estado de mínima energía, que será un mínimo local de la energía de la red. Este mínimo local se corresponderá con un estado estable, punto fijo o atractor.

Realizando un estudio riguroso se llega a la conclusión de que para que el funcional (4.15) sea una verdadera función energía (es decir, que a partir de una dinámica asíncrona por la acción de 4.5 sea no creciente), bastan las siguientes dos condiciones

$$w_{ij} = w_{ji} \quad \text{y} \quad w_{ii} \geq 0 \quad (4.16)$$

Es decir, basta que la matriz de pesos sea simétrica y de diagonal no negativa⁴.

Hasta el momento hemos supuesto una dinámica asíncrona o serie, y unas determinadas condiciones para la matriz de pesos, pero existen otras posibilidades. En este sentido, el teorema de convergencia para la red de Hopfield probado por Bruck [Bruck 90] establece todos los posibles casos que pueden darse:

⁴No obstante, el modelo original de Hopfield considera $w_{ii}=0$, condición que muy a menudo se mantiene pese a que se demuestra que podrían considerarse valores $w_{ii} \geq 0$.

Teorema de convergencia. Sea $RH=(W, \Theta)$, una red de Hopfield, de matriz sináptica W y conjunto de umbrales Θ .

- 1) Si RH opera en **modo serie** y W es simétrica de diagonal nula ($w_{ij}=w_{ji}$, $w_{ii}=0$), entonces la red siempre converge a un estado estable (éste es el modelo de Hopfield original).
- 2) Idem, pero $w_{ii} \geq 0$.
- 3) Si RH opera en **modo completamente paralelo**, para una W simétrica ($w_{ij}=w_{ji}$), la red convergerá siempre a un estado estable o a un ciclo de longitud 2 (o, de otro modo, los ciclos límite son siempre de longitud menor o igual a 2).
- 4) Si RH opera en modo completamente paralelo, W es antisimétrica y de diagonal nula y los umbrales son nulos ($w_{ij}=-w_{ji}$, $w_{ii}=0$, $\theta=0$), la red siempre converge a un ciclo de longitud 4.

#

En el resto de los casos que se pueden considerar

- a) Modo serie, W antisimétrica
- b) Modo serie, W arbitraria
- c) Modo completamente paralelo, W arbitraria

En general, pueden aparecer ciclos de longitud exponencial en el espacio de los estados, es decir, no se garantiza la convergencia de la red a un estado estable⁵.

Resumiendo, para una red de Hopfield que cumpla alguna de las condiciones de estabilidad, podemos interpretar su operación del siguiente modo. Cada posible estado tiene asociado una cierta energía, dada por (4.15). Podemos imaginar un paisaje de energías (Figura 4.4), con valles, cuencas y colinas. El estado inicial de la red $x(0)$ será uno de los puntos del paisaje, su estado evolucionará en tiempos sucesivos bajo la acción de (4.5) por la hipersuperficie $E(\cdot)$, de modo que nunca aumente su energía. Cuando el estado de la red x alcanza un mínimo local, es decir, un estado estable, permanecerá en él para siempre. Solamente la red sale de este estado estable cuando sea presentada una nueva entrada, con lo que será colocada en otro punto de la hipersuperficie y el proceso se repetirá.

En la definición de energía de la red hemos hecho uso de neuronas $\{0,1\}$. Para el caso frecuente de utilizar neuronas Ising $\{-1,+1\}$, y además umbrales nulos, la energía de la red se expresa como

⁵Aunque en general no se asegura, en algunos casos concretos sí podría garantizarse. Por ejemplo en el trabajo de [Personnaz 86] se muestra que utilizando como regla de aprendizaje la pseudoinversa, se llega a una matriz de pesos no simétrica y de diagonal no nula, pese a lo cual, los autores demuestran que para esa regla y con dinámica paralela no puede haber ciclos, y la red siempre alcanza un estado estable.

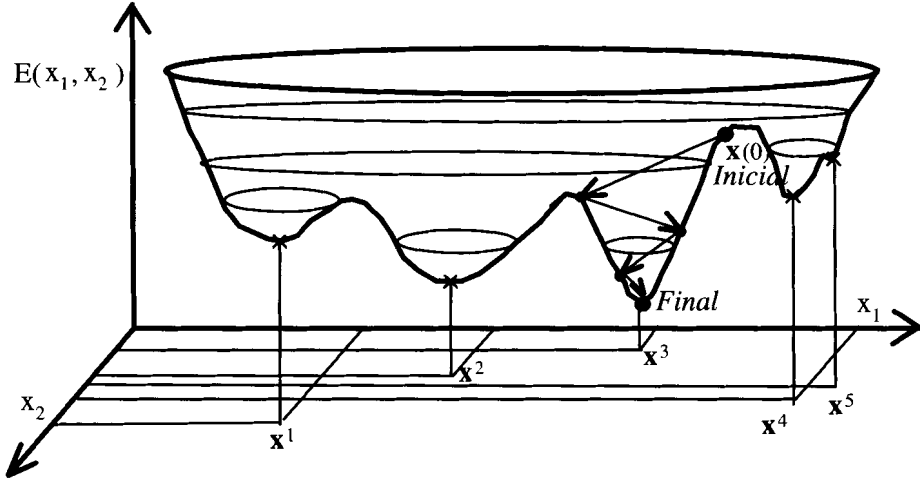


Figura 4.4 Hipersuperficie de energía $E(x)$ de la red en el espacio de los estados (cada mínimo local representa un recuerdo o memoria). Se ilustra la evolución del estado de la red hacia un mínimo local

$$E = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i x_j \quad (4.17)$$

considerando de nuevo $w_{ij} = w_{ji}$ y $w_{ii} = 0$. Podemos apreciar que en este caso si se elimina la segunda condición de diagonal nula no aparece ningún fenómeno extraño, puesto que simplemente estos términos aparecerán como una constante sumada a la definición anterior de energía, ya que $x_i \cdot x_i = +1$, y esta constante sumada a la energía no depende del estado de las neuronas

$$E = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i x_j + \frac{1}{2} \sum_{i=1}^n w_{ii} \quad (4.18)$$

constante adicional que no modifica la dinámica y que puede ser eliminada sin mayor problema [Müller 90].

4.3 APRENDIZAJE EN LA RED DE HOPFIELD

Volvamos a la interpretación de la operación de la red de Hopfield como una memoria asociativa. Un atractor o estado estable de la red es un mínimo local de la función energía, por lo tanto, los recuerdos o memorias que la red almacena son mínimos locales de la red, por lo que el aprendizaje consistirá en conseguir que la red almacene (o memorice) como estados estables un conjunto de recuerdos o memorias dado. Para ello, la regla de aprendizaje que se defina deberá encontrar el conjunto de

pesos sinápticos W que hace que la función energía tenga esos patrones como mínimos locales (estados estables).

4.3.1 Regla de Hebb

La primera regla de aprendizaje propuesta para este modelo de red fue la ya familiar regla de Hebb (capítulo 2), que el propio Hopfield sugirió en su trabajo original [Hopfield 82]. En la tabla 4.1 aparece un resumen de las características de este modelo que podríamos calificar como **modelo de Hopfield discreto clásico**.

Supongamos la neurona tipo Ising $\{-1, +1\}$ y umbrales nulos, y sea un conjunto de p patrones $\mathbf{x}^\mu$, $\mu=1, \dots, p$, que deseamos memorizar. La regla de Hebb en este caso se expresa

$$w_{ij} = \frac{1}{n} \sum_{\mu=1}^p x_i^\mu x_j^\mu \quad (4.19)$$

que puede observarse que cumple las dos condiciones que aseguran la estabilidad de la red

$$w_{ij} = w_{ji} \quad w_{ii} \geq 0 \quad (4.20)$$

Obsérvese que en este caso la regla de Hebb no actúa incrementalmente, sino que de forma directa proporciona la matriz de pesos sinápticos de la red. Se han realizado multitud (y extensos) estudios sobre el modelo de Hopfield con esta regla de aprendizaje, los cuales aparecen recopilados en, por ejemplo, [Sompolinsky 88, Amit 89, Müller 90, Domany 91, Hertz 91].

A continuación estudiaremos la operación de esta regla de aprendizaje. Veremos que solamente en ciertos casos cada patrón $\mathbf{x}^\mu$ así almacenado representará un estado estable de la red, y que pequeñas desviaciones respecto de un patrón $\mathbf{x}^\mu$ harán que el estado final sea precisamente el patrón memorizado.

Comenzaremos con el caso más sencillo. Supongamos que hemos memorizado tan solo un patrón $\mathbf{x}_i$, con lo que la matriz de pesos será

$$w_{ij} = \frac{1}{n} x_i x_j \quad (4.21)$$

Comprobemos que el patrón memorizado es, efectivamente, un estado estable. Para ello comprobemos la condición de estabilidad (4.8), calculando en primer lugar el campo local

$$h_i = \sum_j w_{ij} x_j = \frac{1}{n} \sum_j (x_i x_j) x_j = \frac{1}{n} x_i \sum_j x_j x_j = \frac{1}{n} x_i \sum_j (x_j)^2 = x_i \quad (4.22)$$

Neurona	Tipo umbral, con salidas $\{0,1\}$
Dinámica	Asíncrona, selección aleatoria de la neurona a la que le corresponde actualizar su estado (ec. 4.5)
Arquitectura	Red monocapa, completamente interconectada y realimentada. Con $w_{ii}=0$, y $w_{ij}=w_{ji}$
Aprendizaje	Regla de Hebb
Otras características	Función energía de la red Propiedades emergentes: memoria asociativa y tolerancia a fallos

Tabla 4.1 Resumen de las características del modelo de Hopfield discreto original [Hopfield 82]

donde se ha tenido en cuenta que $(x_j)^2=+1$, por ser una neurona $\{-1, +1\}$. Por lo tanto, la condición de estabilidad queda

$$h_i x_i = x_i \cdot x_i = +1 > 0 \quad (4.23)$$

con lo que se demuestra que el único patrón almacenado por la regla de Hebb es, en efecto, un estado estable, puesto que se cumple la condición (4.8).

A continuación estudiaremos el comportamiento de la red como memoria asociativa. Por medio de (4.19) tenemos que la red memoriza el patrón $\mathbf{x}$. Supongamos que presentamos en $t=0$ a la red entrenada un patrón $\mathbf{x}'(0)$, cuyas primeras m componentes son diferentes a las del patrón memorizado $\mathbf{x}$, y cuyas $n-m$ componentes finales son las mismas, es decir

$$x'_i = \begin{cases} -x_i, & i = 1, \dots, m \\ x_i, & i = m+1, \dots, n \end{cases} \quad (4.24)$$

De modo que el patrón presentado difiere del memorizado en m componentes; estas componentes diferentes podemos interpretarlas como ruido presente en el patrón de entrada. Calculemos la respuesta de la red ante el patrón con ruido en una primera iteración

$$h_i(t=0) = \sum_j w_{ij} x'_j(0) = \frac{1}{n} x_i \sum_j x_j x'_j(0) = \frac{1}{n} x_i [-m + (n-m)] = \left(1 - \frac{2m}{n}\right) x_i \quad (4.25)$$

Por lo tanto, si se cumple la condición

$$\left(1 - \frac{2m}{n}\right) > 0 \Rightarrow m < \frac{n}{2} \Rightarrow x_i(1) = \text{sign}(h_i(0)) = x_i \quad (4.26)$$

recuperamos el patrón puro memorizado. Por lo tanto, si partimos de un estado inicial que contenga menos de la mitad de las componentes diferentes a las del patrón memorizado, la evolución de la red nos lleva a recuperarlo en una sola iteración. En otras palabras, **la red ha eliminado el ruido presente en el patrón de entrada y ha recompuesto el original**. El conjunto de patrones variaciones en torno al memorizado desde los cuales se puede alcanzar un cierto mínimo local se denomina **cuenca de atracción** del estado estable o atractor.

Hasta ahora, por simplicidad, hemos trabajado con un solo patrón memorizado. Si disponemos de p patrones que deseamos memorizar, tendremos por (4.19)

$$w_{ij} = \frac{1}{n} \sum_{\mu=1}^p x_i^{\mu} x_j^{\mu} \quad (4.27)$$

y si presentamos como entrada un patrón v memorizado x_i^v , operando puede obtenerse la siguiente respuesta [Müller 90]

$$h_i^v = \sum_j w_{ij} x_j^v = x_i^v + \frac{1}{n} \sum_{\mu \neq v} x_i^{\mu} \sum_j x_j^{\mu} x_j^v = x_i^v + \text{ruido} \quad (4.28)$$

es decir, en este caso aparece el mismo patrón memorizado pero con cierta cantidad de ruido superpuesto, lo cual se interpreta como que al memorizar varios patrones unos interfieren con otros. El desarrollo (4.28) se denomina **expansión señal-ruido**. Esta interacción entre los patrones supone además la aparición de **estados espurios** en la red neuronal, es decir, estados estables (mínimos locales) que no pretendíamos memorizar, y dado un cierto patrón de entrada, la red en su operación podría estabilizarse en uno de esos estados espurios en vez de en uno de los memorizados. Aunque en la red surgen así muchos mínimos locales, se puede comprobar que los estados espurios directamente derivados de los memorizados mediante la regla de Hebb constituyen mínimos locales de la función energía, y el resto poseen siempre una energía igual o mayor (por lo menos se cumple cuando el número de patrones es muy inferior al de neuronas). Cuestiones de este tipo se estudian en [Müller 90].

Si suponemos patrones no correlacionados y con la misma probabilidad de aparición para los estados -1 y $+1$, la expansión señal-ruido (4.28) queda [Müller 90]

$$h_i^v = \sum_j w_{ij} x_j^v = x_i^v + O\left(\left(\frac{p-1}{n}\right)^{1/2}\right) \quad (4.29)$$

De modo que si $p \ll n$ (pequeño número de patrones almacenados), se tiene $h_i^v \cong x_i^v$, y esencialmente todos los patrones aprendidos son estables.

Calculando la respuesta de la red ante un patrón inicial con m componentes erróneas (de forma similar al caso de un solo patrón memorizado), queda [Müller 90]

$$h_i = \left(1 - \frac{2m}{n}\right) x_i^v + O\left(\left(\frac{p-1}{n}\right)^{1/2}\right) \quad (4.30)$$

con lo que esencialmente, en una iteración alcanzamos el mínimo correcto si se cumple $p \ll n$, y $m \ll n$. Si el número de patrones almacenados p es comparable al número de neuronas n , el segundo término de la expresión es de orden uno, luego comparable con el primero, y los patrones ya no pueden recuperarse. Aplicando el potente (y complicado) aparato matemático de la física estadística se demuestra (véase, por ejemplo, [Müller 90]) que este indeseable caso se da cuando el número de patrones almacenados excede aproximadamente el 14% del número de neuronas. Se suele definir un parámetro α , que representa la relación patrones/neuronas, $\alpha = p/N$, por lo que en la red de Hopfield con aprendizaje hebbiano se debe cumplir

$$\alpha = \frac{p}{n} < 0.14 \quad (4.31)$$

considerando que los patrones son aleatorios, no correlacionados, y con un 50% de probabilidades para que cada neurona esté en +1 o -1. Si imponemos la condición de que los patrones a memorizar sean ortogonales, el término debido al ruido (ecuación 4.28) se anula, por lo que es posible almacenar hasta n patrones ortogonales.

En cuanto a la energía de los estados memorizados y espurios, se puede demostrar [Müller 90] que si α se mantiene pequeño, los estados memorizados, que sabemos son esencialmente estados estables, son los mínimos globales de la función energía, mientras que los estados estables espurios que aparecen son tan sólo mínimos locales, menos profundos que los memorizados.

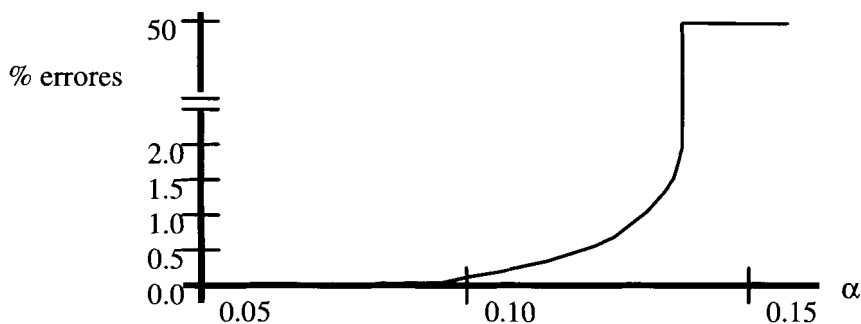


Figura 4.5 Porcentaje de errores en la red de Hopfield entrenada con la regla de Hebb en función del nivel de almacenamiento α . A partir del valor crítico $\alpha=0.14$ se produce una completa pérdida de memoria [Hopfield 82]

En resumen, en el modelo clásico de Hopfield el número de patrones que la regla de Hebb permite almacenar es inferior al número de neuronas de la red (el 14% de n , aproximadamente), lo cual representa un grave problema, pues supone un gran derroche de recursos. No obstante, empleando reglas de aprendizaje más eficientes que la de Hebb (que funcionan incluso en presencia de patrones correlacionados) se demuestra que podrían almacenarse hasta $2 \cdot n$ patrones en una red de n neuronas. Es decir, el α límite teórico para el modelo de Hopfield es 2 [Müller 90].

4.3.2 Reglas de aprendizaje óptimas

Por lo tanto, la regla de aprendizaje de Hebb presenta los siguientes problemas:

- a) No asegura que los patrones de entrenamiento sean mínimos de la energía, es decir, no se garantiza el almacenamiento exacto de dichos patrones.
- b) Las memorias almacenadas pueden tener asociadas cuencas de atracción pequeñas.
- c) La relación patrones/neuronas es pequeña ($\alpha=0.14$), es decir, presenta una muy limitada capacidad de almacenamiento.
- d) Presupone que los patrones son no correlacionados y aleatorios.

Sería interesante disponer de reglas de aprendizaje más eficientes, que aseguren un almacenamiento exacto de los patrones, tengan una amplia capacidad de almacenamiento, maximicen el tamaño de las cuencas de atracción y permitan el almacenamiento de patrones correlacionados y no aleatorios. A continuación estudiaremos algunas reglas superiores a la clásica de Hebb, por mejorar algunas de sus características.

Regla de la pseudoinversa

Una primera regla que mejora el rendimiento es la denominada **regla de la proyección o de la pseudoinversa**, propuesta para el modelo de Hopfield por L. Personnaz y otros [Personnaz 86]. Este algoritmo hace uso del cálculo de la pseudoinversa de una matriz (véase el capítulo 2), es válida para patrones aleatorios y no aleatorios, ortogonales y no ortogonales, asegura el almacenamiento exacto de los patrones, y permite un nivel de almacenamiento de $\alpha=1$, es decir hasta un número de patrones igual al número de neuronas de la red (no obstante, si se almacenan más de $p=n/2$ patrones las cuencas de atracción resultan despreciables, por lo que se suele indicar como cota $\alpha=0.5$). Así, la de la pseudoinversa es una regla más eficiente que la de Hebb, aunque también es computacionalmente más compleja, no iterativa e involucra cálculos no locales. Recordemos que, como vimos en el capítulo 2, la de Hebb es en realidad un caso particular sencillo de la más general regla de la pseudoinversa.

A continuación pasamos a introducir esta regla de aprendizaje. Para ello, sea W la matriz de pesos sinápticos de la red de Hopfield, si $\mathbf{x}^\mu$ es uno de los p vectores a memorizar, se debe cumplir

$$\text{signo}[W \cdot \mathbf{x}^\mu] = \mathbf{x}^\mu \quad (4.32)$$

si trabajamos con neuronas $\{-1, +1\}$. Podemos sustituir la condición genérica anterior por otra particular, más restrictiva

$$W \cdot \mathbf{x}^\mu = \mathbf{x}^\mu \quad (4.33)$$

Si colocamos los p patrones de aprendizaje $\mathbf{x}^\mu$ como columnas de cierta matriz Σ , ésta tendrá por dimensiones $n \times p$, y la condición anterior se puede reescribir como

$$W\Sigma = \Sigma \quad (4.34)$$

El objetivo del aprendizaje será encontrar la matriz W que cumple esta condición. Si Σ fuese una matriz cuadrada de determinante no nulo, sería invertible, con lo que la solución de (4.34) sería

$$W = \Sigma \cdot \Sigma^{-1} \quad (4.35)$$

Pero para el caso general de una matriz no cuadrada, se demuestra que la solución de (4.34) es la siguiente

$$W = \Sigma \cdot \Sigma^+ \quad (4.36)$$

siendo Σ^+ la **matriz pseudoinversa** de Σ , que representa la generalización de la inversión de una matriz cuadrada para el caso de matrices rectangulares (capítulo 2).

Con el algoritmo de aprendizaje basado en la pseudoinversa, la matriz de pesos se obtiene del siguiente modo: en primer lugar se construye la matriz Σ con los patrones como columnas, la matriz de pesos W de la red de Hopfield se obtiene directamente a partir de ella aplicando (4.36); para calcular la pseudoinversa se puede recurrir al método de Greville [Kohonen 89]. Destaquemos de nuevo que de este modo disponemos de una regla más potente que la de Hebb, aunque el precio que hay que pagar es su complejidad de cómputo y su operación no local. No obstante, y en relación a este último punto, en [Diederich 87] se demuestra que la ecuación (4.34) también puede resolverse iterativamente mediante la regla de Widrow-Hoff (la de la adalina), de tipo local, obteniéndose la misma matriz W que la proporcionada aplicando el método de la pseudoinversa (estadísticamente ambas reglas coinciden).

Por último, recordemos que si el conjunto de vectores de aprendizaje es linealmente independiente, la regla de la pseudoinversa da la matriz W que cumple la condición (4.34), es decir, en este caso es capaz de almacenar perfectamente como memorias la totalidad de los patrones de aprendizaje (por lo tanto, puede memorizar hasta n patrones, o $\alpha=1$). Si los vectores son linealmente dependientes, la solución que proporciona es óptima en el sentido del error cuadrático medio (el error cuadrático medio que proporciona la red para estos patrones es el menor posible).

Explorando el espacio de las sinapsis. Reglas óptimas

Se han introducido también una serie de algoritmos de aprendizaje iterativos que tratan de resolver todos los problemas señalados, en particular, tratando de almacenar el mayor número de patrones posibles. Un criterio a partir del cual se pueden deducir reglas de este tipo es el siguiente. Hasta el momento hemos trabajado en el espacio de los estados $\mathbf{x}$, para el cual los pesos W son fijos; la dinámica de recuerdo opera en este espacio modificando las activaciones. Una forma de encontrar reglas de aprendizaje consiste en dar la vuelta al problema, fijando los patrones a memorizar (es decir, los estados $\mathbf{x}$) y operando en el espacio de las interacciones o pesos, de modo que esta dinámica de las interacciones provoque la modificación de los pesos con algún objetivo previamente establecido.

Una de las condiciones u objetivo que se establece en la deducción de estas reglas de aprendizaje óptimas es la siguiente. Como ya sabemos, para que un estado $\mathbf{x}^\mu$ sea estable los espines deben alinearse con los campos locales

$$x_i^\mu h_i > 0 \quad (4.37)$$

Pero si la alineación es pequeña, como, por ejemplo

$$x_i^\mu h_i = 0.01 \quad (4.38)$$

una pequeña perturbación puede hacer que se deje de cumplir la condición de estabilidad. Por lo tanto, para asegurar una alta estabilidad podemos imponer la siguiente condición más severa, denominada condición de embebimiento (*embedding condition*) [Gardner 88]

$$x_i^\mu h_i = x_i^\mu \sum_j w_{ij} x_j^\mu > B > 0 \quad (4.39)$$

es decir, exigimos que la condición de estabilidad se cumpla por encima de un cierto margen, que denotaremos por B ; de este modo *excavamos* unas cuencas de atracción más amplias en torno al patrón $\mathbf{x}^\mu$ a memorizar.

Varios modelos de aprendizaje hacen uso de esta técnica [Diederich 87, Gardner 88]. Así, en el ya citado trabajo de Diederich y Oppen [Diederich 87] se deducen dos reglas de entrenamiento para la red de Hopfield. Una de ellas resulta ser la del perceptrón aplicada a este modelo; la otra presenta el mismo aspecto que la de la adalina, que se demuestra equivale a la regla de la pseudoinversa, como señalamos ya.

En [Gardner 88] se demuestra que la máxima capacidad de almacenamiento de la red de Hopfield es $\alpha=2$, es decir, **en una red de n neuronas podrían almacenarse hasta $2n$ patrones**. Gardner obtiene una regla de aprendizaje iterativa a partir de una condición como la (4.39), demostrando que permite alcanzar la máxima capacidad de almacenamiento teóricamente permitido. Además demuestra la convergencia de su algoritmo, prueba que resulta ser una generalización de la del perceptrón.

4.4 EJEMPLO: RECONOCIMIENTO DE CARACTERES

Un ejemplo ilustrativo de la operación de la red de Hopfield se basa en el reconocimiento de caracteres. Supongamos que disponemos de un conjunto de imágenes formado por diez prototipos correspondientes a los dígitos del 0 al 9, con cada imagen compuesta por 11×7 píxeles. A cada píxel, que puede estar encendido (1) o apagado (0), se le asigna el estado de una neurona ($11 \times 7 = 77$ neuronas en total).

En una primera experiencia entrenamos una red de Hopfield discreta clásica de 77 neuronas haciendo uso de la **regla de Hebb**. Si tratamos de memorizar un único prototipo (por ejemplo, el correspondiente al dígito 0) podremos fácilmente observar que éste se convierte en un estado estable de la red, y que variaciones del anterior (introduciendo ruido en el prototipo original, invirtiendo aleatoriamente un determinado número de píxeles) pueden converger al dígito 0 memorizado, presentando por lo tanto cierta tolerancia al ruido.

Si a continuación entrenamos la red con un número pequeño de prototipos (2, 3 o 4 prototipos), debido a la existencia de interferencias (correlaciones) alguno de ellos podría no llegar a ser un estado estable. En todo caso, las cuencas de atracción que presentan serán en general más pequeñas que en el caso anterior. Finalmente, si tratamos de memorizar los diez patrones del conjunto de entrenamiento podremos apreciar más manifiestamente el insatisfactorio comportamiento de la sencilla regla de Hebb, pues observaremos que no es capaz de almacenar perfectamente los diez prototipos.

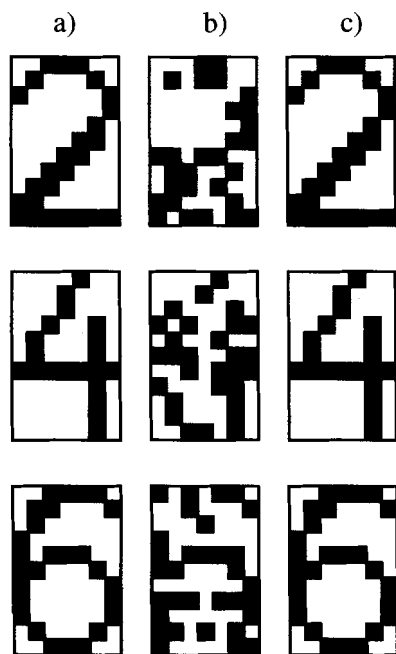


Figura 4.6

Columna a), dígitos originales; columna b), dígitos empleados como entradas a la red de Hopfield, con el 25% de los píxeles invertidos (ruido); columna c), estado final proporcionado por la red tras cierto número de iteraciones

En una segunda experiencia entrenamos la red usando como ejemplos las imágenes correspondientes a los diez dígitos anteriores haciendo uso de la **regla de la pseudoinversa**, que sabemos que asegura el almacenamiento perfecto de los patrones si éstos son linealmente independientes o, en todo caso, su óptimo almacenamiento en el sentido del error cuadrático medio. En la experiencia desarrollada comprobamos que, efectivamente, los diez prototipos se convierten en estados estables de la red. Posteriormente introducimos un cierto nivel de ruido en los patrones originales, y presentamos los nuevos prototipos *ruidosos* a la red, observando que si el nivel de ruido es pequeño, la red converge en todos los casos al patrón original. Si se introduce un exceso de ruido, la red puede acabar en un estado espurio, que se identifica porque no se corresponde con ninguno de los patrones almacenados en el entrenamiento, aunque en ocasiones su imagen puede recordar la de alguno de ellos.

El nivel de ruido tolerado por la red de Hopfield puede ser relativamente alto si se hace uso de un algoritmo de aprendizaje como el de la pseudoinversa, pues puede invertirse una fracción relativamente elevada de los píxeles, de modo que la imagen del dígito original resulta prácticamente irreconocible a nuestros ojos, y no por ello la red neuronal deja de converger al estado original correcto (Figura 4.6). En la experiencia que describimos encontramos que invirtiendo hasta un 25% de los píxeles (seleccionados aleatoriamente) el número de aciertos seguía siendo del 100% para todos los dígitos.

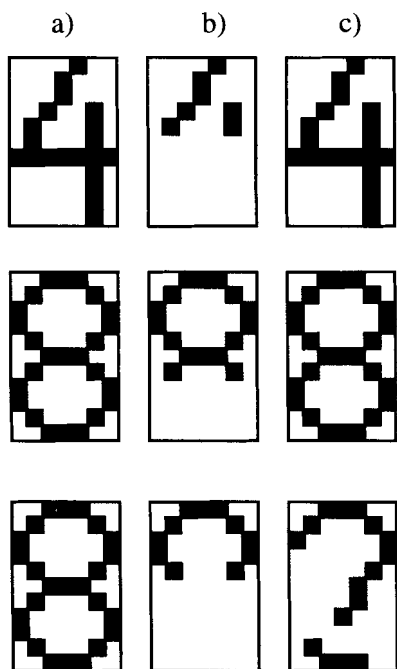


Figura 4.7. Columna a), dígito original; columna b), dígito con la parte inferior tapada, empleado como entrada a la red de Hopfield; columna c), estado final de la red tras varias iteraciones. En el último caso la red no tiene suficiente información y no reconoce el 8 (podría ser también un 2 o un 9)

Otra experiencia interesante a realizar con la red entrenada mediante la pseudoinversa consiste en *tapar* una zona del dígito, por ejemplo la parte inferior. En nuestro caso tapar significa poner a cero todos los píxeles correspondientes a varias líneas inferiores del dígito. Podemos apreciar diversos casos en la Figura 4.7. Obsérvese que si ocultamos demasiados píxeles, la red no es capaz de distinguir correctamente el patrón original, acabando estancada en un estado espurio.

También merece la pena observar que si tratamos con neuronas $\{-1, +1\}$, los patrones construidos a partir de los de aprendizaje mediante la inversión de todos sus píxeles (es decir, sus imágenes en negativo) también resultan ser estados estables. Ello no es casualidad, pues en este caso la condición que deben cumplir los patrones a memorizar es

$$\text{signo}[\mathbf{W} \cdot \mathbf{x}^\mu] = \mathbf{x}^\mu \quad (4.40)$$

cumpléndose también que

$$\text{signo}[\mathbf{W} \cdot (-\mathbf{x}^\mu)] = (-\mathbf{x}^\mu) \quad (4.41)$$

por lo que, efectivamente, tanto $\mathbf{x}^\mu$ como su imagen invertida, $-\mathbf{x}^\mu$, son estados estables.

Para concluir, debe tenerse presente que el ejemplo del reconocimiento de caracteres tal y como se ha presentado resulta de extrema simplicidad. La sencilla experiencia descrita trata de ilustrar la operación de la red de Hopfield, resultando fácil de realizar en un ordenador convencional para una persona con una pequeña experiencia en programación (invitamos al lector a llevarlo a cabo, utilizando como algoritmo de aprendizaje el de la adalina, muy fácil de programar; en este caso las entradas $\mathbf{x}^\mu$ serán los dígitos del 0 al 9, y las salidas objetivo los mismos $\mathbf{x}^\mu$, puesto que la red es autoasociativa). A la hora de llevar a cabo una aplicación real habría que tener en cuenta numerosos aspectos que no se han contemplado, por ejemplo, introducir invariancias frente a traslaciones, escalas o pequeños giros [Haykin 99]. Reiteramos que este ejemplo se ha expuesto únicamente a título de demostración de la operación y posibilidades del modelo de Hopfield discreto.

4.5 NEURONAS ESTOCÁSTICAS: MÁQUINA DE BOLTZMANN

Hasta ahora hemos supuesto un modelo de Hopfield, cuyas neuronas de operación determinista se actualizan según la dinámica (4.5). Podría introducirse una cierta estocasticidad en la operación de la neurona, incorporando en ella probabilidades de activación, que serán función de un nuevo parámetro que, por analogía con la física, denominaremos temperatura de la red T . La probabilidad de

activación que consideraremos en la **neurona estocástica** viene dada por la función de Fermi [Müller 90]

$$P[x_i(t+1) = x'] = \frac{1}{1 + e^{-2\beta h_i \cdot x'}} \quad , \quad \text{con } \beta = 1/KT \text{ y } x' = \pm 1 \quad (4.42)$$

que da la probabilidad de que la neurona i en t pase a cierto estado x' (+1 o -1) en $t+1$; K es un parámetro constante (la constante de Boltzmann de la física estadística). De este modo, la neurona posee una cierta probabilidad de activación; así, a mayor campo h_i tendrá una mayor probabilidad de activarse ($x'=+1$), pero su activación no queda en ningún momento garantizada (Figura 4.8). Si en (4.42) hacemos que la temperatura tienda a cero ($T \rightarrow 0$, o alternativamente, $\beta \rightarrow \infty$), puede verse que se retoma la dinámica determinista conocida. Así, a temperaturas bajas, la neurona posee un comportamiento más determinista, mientras que a temperaturas altas éste es más estocástico.

Para el estudio de este modelo se emplea todo el utillaje matemático de la física estadística (que comienza con la construcción de una función de partición Z), complejo tratamiento perfectamente expuesto en los excelentes libros de [Amit 89, Müller 90, Hertz 91, Domany 91, Haykin 94], y en el que apenas entraremos, simplemente destacaremos dos cuestiones. En primer lugar, señalaremos los interesantes diagramas de fases que surgen en el modelo a temperatura finita, con fases paramagnéticas, ferromagnética y de vidrio de espín.

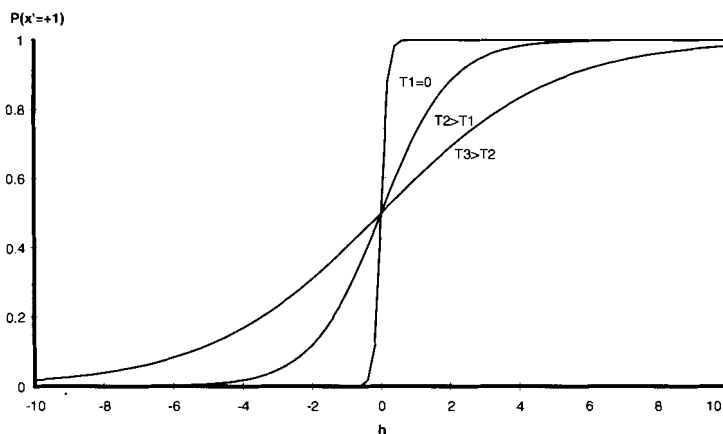


Figura 4.8. Probabilidad de activación $P(x'=+1)$ en función del campo local h , a diferentes temperaturas. En $T=0$ la neurona es determinista

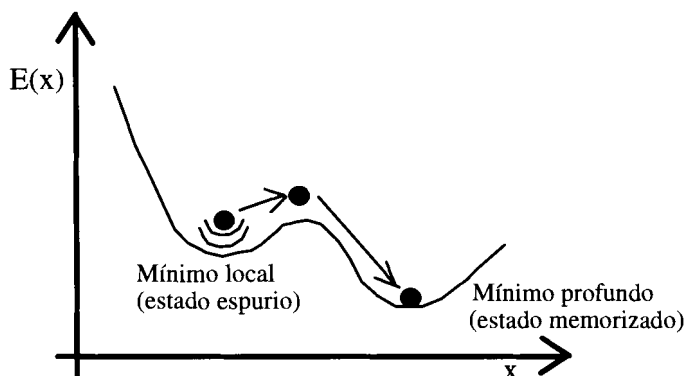


Figura 4.9. El ruido térmico puede ocasionar que eventualmente la energía de la red aumente levemente, lo suficiente para escapar de un estado espurio o mínimo local

En segundo lugar, hay que resaltar el **papel beneficioso que la presencia de un pequeño nivel de ruido térmico origina en la operación del modelo**. Si la temperatura aumenta levemente (en torno a un valor de 0.25), el ruido causado por la temperatura hace que desaparezcan algunos de los estados espurios de la red, aumentando la probabilidad de convergencia hacia uno de los estados memorizados [Pérez 89]. Por otra parte, podemos interpretar también el papel del ruido térmico como una posibilidad de salvar mínimos locales correspondientes a energías más altas que las de los estados memorizados (Figura 4.9). Por último, subrayaremos que un exceso de ruido térmico tendrá un efecto catastrófico sobre la operación de la red.

El modelo neuronal denominado **máquina de Boltzmann** [Hinton 86] hace uso del tipo de neurona probabilística descrito. El nombre hace referencia a la computación que se efectúa mediante procesos estocásticos, en oposición a, por ejemplo, las clásicas máquinas von Neumann, basadas en procesos deterministas.

La operación de una red de este tipo comienza con una temperatura alta, que muy lentamente va reduciéndose hasta cero, de manera que al final la red se situará en un estado de mínima energía. Intuitivamente, a temperatura alta la red posee una gran perspectiva de los mínimos del paisaje de energía, puesto que describe una trayectoria errática abarcando buena parte del espacio de los estados. Conforme la temperatura disminuye (lentamente), la red pierde *movilidad* y se va centrando en las zonas del espacio de menor energía. Por medio de este **templado simulado** (*simulated annealing*), denominación tomada del proceso de templado de los metales, se espera que la red alcance al final un mínimo profundo o el global.

Así, la máquina de Boltzmann es esencialmente una red de Hopfield de neuronas binarias, conexiones simétricas y sin autorrealimentación, que incluye la presencia de nodos ocultos y sus neuronas operan estocásticamente. Su entrenamiento puede ser supervisado, haciendo uso de un algoritmo probabilístico. Una interesante introducción al modelo aparece en [Haykin 99]. Por otra parte, debido a la lentitud de

simulación de este modelo probabilístico se ha introducido la denominada **máquina del campo medio** [Haykin 99] (*mean field machine*), que mediante una aproximación denominada de campo medio transforma las neuronas binarias probabilísticas de la máquina de Boltzmann en neuronas deterministas analógicas. Hay que remarcar que al introducir nodos ocultos, estos modelos presentan características de cómputo superiores a la de la red de Hopfield, el problema fundamental reside en los elevadísimos tiempos de simulación requeridos (muy superiores a los del BP).

4.6 MODELO DE HOPFIELD ANALÓGICO (CONTINUO)

4.6.1 Modelo de Hopfield de neuronas continuas

El modelo de Hopfield discreto presenta ciertos rasgos que lo apartan de la realidad biológica. Uno de ellos es su simple modelo de neurona de salida binaria, por el que solamente tienen la posibilidad de permanecer en dos estados, activada o desactivada (0 o 1). Sin embargo, las neuronas reales pueden situarse en todo un continuo de estados entre el de reposo (0) y el de máxima activación (1), codificando su estado de excitación como frecuencia de pulsos. Por ello, poco tiempo después de introducir el modelo discreto de red recurrente, Hopfield lo modificó [Hopfield 84], permitiendo que, como las biológicas, las neuronas de la red presentasen una respuesta analógica (continua o gradual), como pueda ser la conocida sigmoidea, con salidas en el intervalo [0,+1]

$$y_i(t) = f(x) = \frac{1}{1 + e^{-\beta x}} \quad (4.43)$$

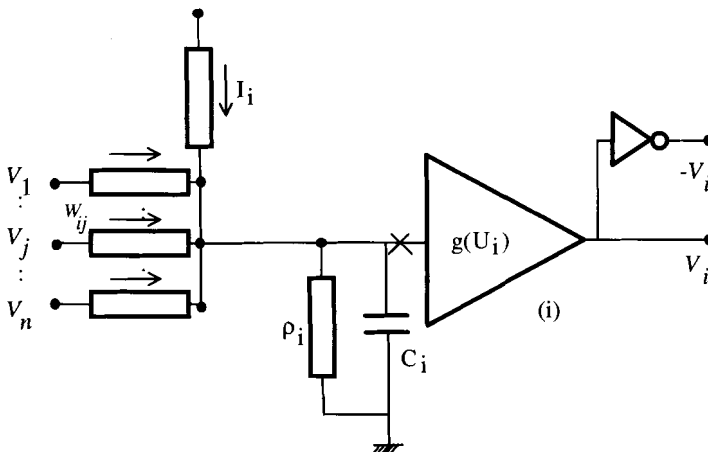


Figura 4.10. Neurona de Hopfield analógica [Hopfield 84]

Hopfield demostró que este modelo, más plausible desde un punto de vista biológico, y muy fácil de implementar mediante **electrónica analógica**, presenta propiedades similares al discreto. Por ejemplo, también puede definirse para él una función energía de la red, y también se trata de un sistema con dinámica de relajación, cuando $w_{ij}=w_{ji}$ y $w_{ii}=0$. Hopfield demostró además la utilidad de este modelo para la resolución de problemas de optimización.

Una neurona de Hopfield continua puede implementarse fácilmente con el circuito de la Figura 4.10, basado en un amplificador que hace el papel de cuerpo neuronal, y de un conjunto de resistencias de entrada como sinapsis, por las que la información entra codificada en la forma de tensiones analógicas V_j (es decir, la frecuencia de disparo de las neuronas biológicas, que puede tomar un continuo de valores, se representa como un voltaje analógico). En esta figura ρ_i y C_i son la resistencia y capacidad eléctrica de entrada del amplificador, que se corresponderían con la resistencia y capacidad de la membrana celular. Los w_{ij} denotan las conductancias asociadas a las resistencias R_{ij} ($R_{ij}=w_{ij}^{-1}$) de la figura, que representan la eficacia de las sinapsis (pesos sinápticos). Además se introduce un parámetro I_i , corriente adicional que denominaremos intensidades externas (*bias*), que juegan el papel del umbral de la neurona (como *bias*, no como umbral de disparo).

La salida de la neurona es una tensión continua positiva V_i , que viene dada por la función de transferencia del amplificador

$$V_i = g(U_i) \quad (4.44)$$

siendo U_i su tensión de entrada, que se correspondería con el potencial de membrana de una neurona biológica. La salida del amplificador es aproximadamente sigmoidea, con una tensión mínima (0 voltios) y otra máxima (tensión de saturación, $+V^{\text{sat}}$), pudiendo situarse en todo un continuo de valores intermedios. Para valores elevados de la ganancia del amplificador (Figura 4.11) la transición entre 0 y $+V^{\text{sat}}$ será muy abrupta (casi una función escalón como la del modelo de Hopfield discreto), y más suave para ganancias más reducidas (implementando una neurona cuya salida es casi una función lineal a tramos). Debido a que los pesos sinápticos se implementan mediante resistencias, solamente podremos hacer uso de pesos positivos. Para poder emular la existencia de pesos negativos dotamos al amplificador con una salida inversora adicional $[-V^{\text{sat}}, 0]$.

A continuación analizaremos la operación del circuito. Contabilizando las intensidades que entran y salen del nodo señalado en la Figura 4.10, se tiene

$$\sum_j \frac{V_j - U_i}{R_{ij}} + I_i = C_i \frac{dU_i}{dt} + \frac{U_i}{\rho_i} \quad (4.45)$$

que reordenando se transforma en

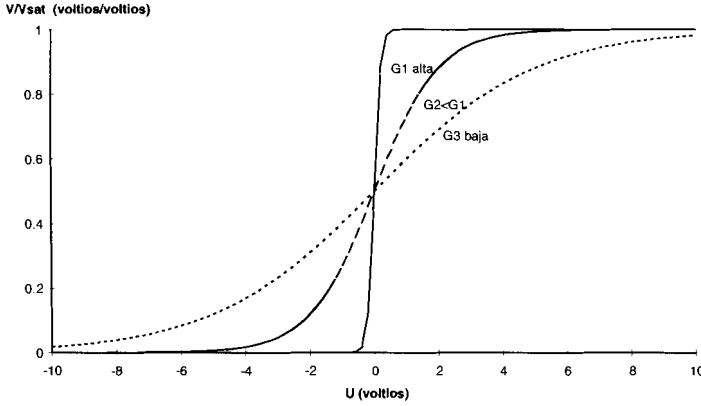


Figura 4.11. Función de transferencia del amplificador que representa el cuerpo neuronal, para diversas ganancias

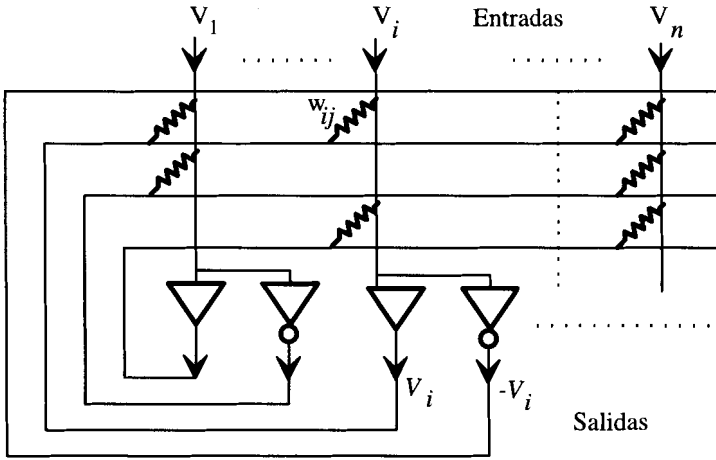


Figura 4.12 Modelo de red de Hopfield analógica

$$C_i \frac{dU_i}{dt} = \sum_j \frac{V_j}{R_{ij}} - U_i \sum_j \left[\frac{1}{R_{ij}} + \frac{1}{\rho_i} \right] + I_i \quad (4.46)$$

y llamando

$$\frac{1}{R_i} \equiv \frac{1}{\rho_i} + \sum_j \frac{1}{R_{ij}}, \text{ y } w_{ij} = 1/R_{ij} \quad (4.47)$$

la ecuación que describe la operación de la neurona analógica (Figura 4.10) queda finalmente

$$C_i \frac{dU_i}{dt} = \sum_j w_{ij} V_j - \frac{U_i}{R_i} + I_i \quad (4.48)$$

En resumen, en este nuevo modelo se tienen en cuenta los siguientes aspectos de la realidad biológica: neurona de respuesta continua, presencia de retardos, efectos RC de la célula y asincronismo de la operación. Este modelo analógico de red de Hopfield resulta más difícil de simular que el discreto, pero resulta mucho más sencillo de realizar mediante electrónica analógica (aunque se presentan problemas a resolver, como la forma de variar las resistencias).

Energía de la red analógica

De la misma manera que en el caso discreto, puede definirse una función energía de Lyapunov para el caso continuo [Hopfield 84, Zurada 92]

$$E = -(1/2) \sum_{i,j} w_{ij} V_i V_j + \sum_i (1/R_i) \int_0^{V_i} g^{-1}(V) dV - \sum_i I_i V_i \quad (4.49)$$

y se puede demostrar que su variación con el tiempo es no positiva, si la función g^{-1} es monótona creciente y la matriz de pesos simétrica [Hopfield 84]. Como además su valor está limitado, transcurrido un tiempo la red alcanzará un mínimo de E , y el sistema permanecerá allí, en un estado estable.

En el límite de ganancias elevadas (función $g(\cdot)$ abrupta), los estados estables del sistema continuo coinciden con los del discreto si se cumple $w_{ij}=w_{ji}$ y $w_{ii}=0$. Es decir, **en el límite de alta ganancia el modelo discreto y el continuo coinciden**. Se puede ver que, en ese caso, la función energía para la red de la Figura 4.12, con las neuronas de la Figura 4.10, es similar a la del caso discreto (4.15), sólo que el término de los umbrales queda sustituido por el de las intensidades externas

$$E = -(1/2) \sum_i \sum_{j, i \neq j} w_{ij} V_i V_j - \sum_i I_i V_i \quad (4.50)$$

Por otra parte, se comprueba que para ganancias reducidas (función $g(\cdot)$ suave), el modelo continuo presenta menos estados espurios que el discreto. Finalmente, hay que resaltar que aunque en el análisis matemático se exige simetría de los pesos, el sistema tolera pequeñas asimetrías [Hopfield 84].

4.6.2 Aplicaciones del modelo de Hopfield analógico. Optimización

El modelo de Hopfield analógico se aplica a la resolución de problemas de optimización [Hopfield 85, 86, Wassermann 89, Haykin 94], en los cuales se trata de encontrar la mejor solución a un determinado problema, cumpliendo ciertas

condiciones restrictivas. Ejemplos de su aplicación que pueden encontrarse en la literatura son los siguientes: problema del viajante de comercio, construcción de conversores A/D, resolución de problemas de programación lineal, control adaptativo predictivo, entrenamiento de otra red neuronal, etc.

La idea de partida consiste en que la salida de cada neurona (una tensión) representa una de las variables del problema a resolver. Las condiciones o **restricciones** que deben cumplir las posibles soluciones del problema se incorporarán explícitamente **como términos de la función energía de la red**, conformando así un paisaje de energías, con mínimos locales (y quizás globales), donde cada mínimo de la energía representará una solución localmente óptima del problema. Si logramos construir una red de Hopfield cuyos parámetros (pesos y corrientes) conformen la función energía dada, dejando a la red evolucionar alcanzará en su operación finalmente un mínimo local, que representará una solución localmente óptima del problema. Los parámetros de la red que resuelve el problema se calculan comparando la función energía propuesta con la genérica (4.50), y despejando los valores de los pesos y las corrientes externas.

A modo de ejemplo, describiremos el procedimiento que Hopfield propone para resolver el **problema del viajante** o **TSP** (*Traveling Salesman Problem*) [Hopfield 85]. Éste consiste en encontrar el orden en el que un viajante de comercio debería visitar varias ciudades para que la distancia recorrida sea mínima. Se trata éste de un problema denominado NP completo, en el que la única alternativa para su resolución consiste en ensayar todas las posibles soluciones para encontrar cuál es la óptima. Aunque ello puede resultar viable para un número muy pequeño de ciudades, resulta impracticable en cuanto el número de ciudades es moderadamente elevado, ya que hay que tener en cuenta que si el número de ciudades es n , el número de posibles recorridos a ensayar resulta ser de $n!/(2n)$; así, para el caso de 10 ciudades habría que ensayar con 181.440 posibles soluciones, y para 60 ciudades, habría que ensayar con nada menos que 69×10^{78} soluciones! (para comprender la magnitud de este número, recordemos que el número de átomos del universo se estima en 10^{40}).

Veamos cómo mediante una red de Hopfield continua podemos resolver el TSP con un coste de cómputo muy inferior, al no tener que recurrir al cálculo de todas las posibilidades. Nuestro objetivo, en principio, no será obtener la mejor solución, sino **tan sólo una buena solución** (mínimo local). Consideremos n ciudades y n^2 neuronas. La salida de cada neurona vendrá representada por una tensión $V_{xj} \in [0, 1]$, donde el índice x indica la ciudad y el j el orden de visita. Una neurona activada (con $V_{xj}=1$) indicará que la ciudad x es visitada en j lugar (Figura 4.13)

La función energía deberá ser tal que resulte pequeña cuando los estados V_{xj} de las neuronas señalen una solución (camino) válida, y además ésta sea óptima (de pequeña longitud). Para ello deberá incorporar las siguientes condiciones: a) la energía deberá ser pequeña (cero) solamente cuando una neurona de una fila esté activada, y ser elevada cuando haya varias; b) ídem con las neuronas de las columnas (ambas condiciones aseguran que el viajante visitará cada ciudad una sola vez); c) la

energía será pequeña (cero) si ha visitado todas las ciudades, y grande en caso contrario (esta condición asegura que el viajante no se deja ninguna ciudad por visitar); d) por último, si la trayectoria es larga, la energía debería ser alta (condición de camino óptimo). Las tres primeras (a-c) son denominadas restricciones fuertes, puesto que son las que la solución debe cumplir para que sea considerada válida (por ejemplo, que el viajante no se olvide de ninguna ciudad o que no visite dos veces la misma). La última de ellas es una restricción suave, puesto que no tiene por qué cumplirse de forma exacta, indicando tan sólo la bondad de la solución alcanzada.

La siguiente función energía que cumple todas estas condiciones [Hopfield 85]

$$E = (A/2) \cdot \sum_x \sum_i \sum_{j, j \neq i} V_{xi} V_{xj} + (B/2) \cdot \sum_i \sum_x \sum_{y, y \neq x} V_{xi} V_{yj} + (C/2) \cdot \left[\left(\sum_x \sum_i V_{xi} \right) - n \right]^2 + (D/2) \cdot \sum_x \sum_{y, y \neq x} \sum_i d_{xy} V_{xi} (V_{y, i+1} + V_{y, i-1}) \quad (4.51)$$

Recordemos que nuestro objetivo es encontrar una expresión para la energía que sea mínima cuando se cumplan las cuatro condiciones descritas, cada una de ellas la asociamos a uno de los términos de la energía: el primer término se anula cuando cada fila x de la tabla de la Figura 4.13 contiene un único 1; el segundo cumple el mismo papel para las columnas; el tercero será cero cuando haya n 1's en la matriz (visita de las n ciudades); por último, el cuarto término da la longitud de cualquier recorrido válido, que debe ser mínimo para E mínima (los índices se han definido módulo n , es decir, $V_{n+j}=V_j$). Valores suficientemente grandes para A , B y C aseguran que la energía será pequeña para recorridos válidos; un valor alto de D asegura que el recorrido elegido será corto.

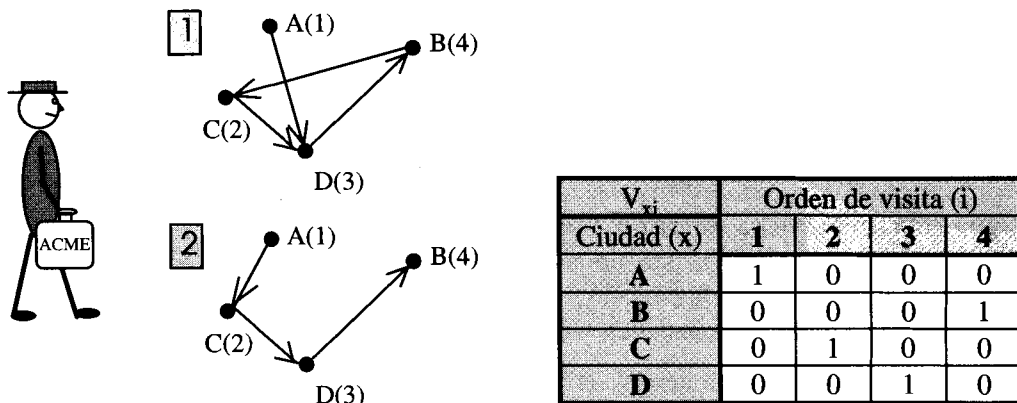


Figura 4.13. Problema del viajante. Ejemplo de cuatro ciudades. El viajante duda entre dos recorridos, aunque uno de ellos es óptimo y el otro no

La cuestión más inmediata a resolver es cómo lograr que la red de Hopfield (Figura 4.12) que resuelva el problema posea precisamente la función energía (4.51). En otras palabras, deberemos encontrar los pesos de la red de Hopfield para que su función energía sea la dada. Para ello deberemos igualar los términos de (4.51) con los de la función energía genérica (4.50), con lo que determinaremos los pesos y corrientes externas de las neuronas de la red $n \times n$ de Hopfield. Operando se obtiene [Hopfield 85, Wassermann 89]:

$$w_{xi,yj} = -A\delta_{xy}(1 - \delta_{ij}) - B\delta_{ij}(1 - \delta_{xy}) - C - Dd_{xy}(\delta_{j,i+1} + \delta_{j,i-1})$$

$$I_i = C.n \quad (4.52)$$

Donde δ_{ij} es la función delta de Kronecker ($\delta_{ij}=1$, si $i=j$; $\delta_{ij}=0$, si $i \neq j$).

Por lo tanto, una red con los parámetros (4.52) posee la función energía (4.51) y en su relajación alcanzará un mínimo local, que representará una solución al problema, que se espera sea satisfactorio. Los primeros estudios así parecían indicarlo [Hopfield 85, 86]: en las pruebas llevadas a cabo por Hopfield y Tank 16 de los 20 intentos convergieron a soluciones válidas, y el 50% fueron soluciones buenas. No obstante, la solución de Hopfield y Tank ha sido criticada (véase, por ejemplo, [Müller 90, Wilson 88]), pues estudios más extensos parecen mostrar que en bastantes ocasiones no se alcanza una solución satisfactoria. Ello es debido a que la operación de la red de Hopfield es muy sensible a las constantes A , B , C y D seleccionadas, de manera que pequeñas variaciones en ellas pueden conducir al éxito o a un completo fracaso (en [Haykin 99] aparece una síntesis de los trabajos realizados al respecto en los últimos años).

A modo de resumen podemos decir que esta metodología representa un procedimiento original e interesante para resolver un conjunto de problemas importantes y difíciles de solucionar por otros métodos (quizás NP completos), como el de la partición de grafos, emplazamiento óptimo o encontrar trayectorias mínimas sujetas a ciertas restricciones, que tienen importantes aplicaciones en campos como el diseño electrónico o el guiado de robots, por ejemplo. Su resolución se puede llevar a cabo a nivel algorítmico, mediante la simulación por ordenador de la red en cuestión, aunque un aspecto muy interesante es que puede realizarse en un circuito electrónico específico que operará a una velocidad muy superior, lo que podría permitir la resolución de problemas importantes de optimización en tiempo real. No obstante, queda todavía mucho trabajo que realizar para asegurar la fiabilidad de la operación del procedimiento. Finalmente, otras técnicas de resolver este tipo de problemas, como por ejemplo, la del templado simulado, y su comparación con la red de Hopfield analógica, se muestran en [Müller 90].

4.7 FUNCIONES DE BASE RADIAL (RBF)

Trataremos a continuación el importante modelo de **funciones de base radial o RBF** (*Radial Basis Functions*) [Moody 89, Poggio 90] que, aunque de reciente introducción, cada vez cuenta con más aplicaciones prácticas gracias a su simplicidad, generalidad y rapidez de aprendizaje. Se trata de un modelo que a menudo se estudia junto al MLP por ser una red unidireccional para aproximación funcional, pero que puede considerarse **de tipo híbrido** por incorporar aprendizaje supervisado y no supervisado.

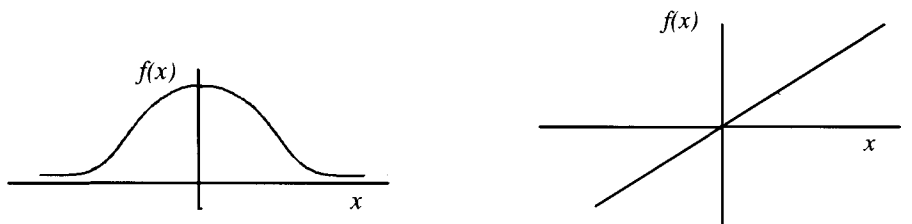
Como sucede en el caso del MLP, las RBF permiten modelar con relativa facilidad sistemas no lineales arbitrarios, con la particularidad de que el tiempo requerido para su entrenamiento suele ser mucho más reducido que el del BP clásico. Buena parte de lo que exponremos en la presente sección está basado en la referencia [Warwick 95], así como en [Wassermann 93, Hush 92, 93]. En [Haykin 99, Principe 00] pueden consultarse resultados recientes sobre las RBF.

La arquitectura de una red RBF cuenta con tres capas de neuronas, de entradas, oculta y capa de salida (similar a un MLP de una sola capa oculta). Las neuronas de entrada, como suele ser habitual, simplemente envían la información del exterior hacia las neuronas de la capa oculta. Las neuronas de la capa de salida son lineales, esencialmente calculan la suma ponderada de las salidas que proporciona la capa oculta.

La diferencia fundamental entre la arquitectura de este modelo y la del MLP se centra en la operación de las neuronas ocultas. Éstas, en vez de computar la suma ponderada de las entradas y aplicarle una función de tipo sigmoideo, como en el modelo de Kohonen, operan en base a la distancia que separa el vector de entradas respecto del vector sináptico que cada una almacena (denominado **centroide**), cantidad a la que aplican una función radial con forma gaussiana (Figura 4.14). Es decir, así como en el MLP las neuronas ocultas poseen una respuesta de rango infinito (cualquier vector de entrada, con independencia del lugar del espacio de entrada de donde proceda puede causar que la neurona se active), en el RBF **las neuronas son de respuesta localizada**, pues sólo responden con una intensidad apreciable cuando el vector de entradas presentado y el centroide de la neurona pertenecen a una zona próxima en el espacio de las entradas.

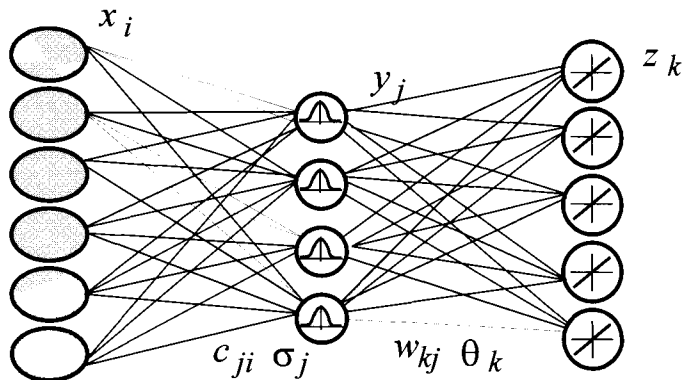
A continuación describiremos matemáticamente el modelo. Denominaremos x_i a las entradas de la red, y_j serán las salidas de la capa oculta, y z_k las salidas de la capa final (y globales de la red). Cada neurona j de la capa oculta almacena un vector c_{ji} , el centroide; como en la red de Kohonen cada una de estas neuronas calcula la distancia euclídea r_j que separa el vector de entradas x_i de su centroide

$$r_j^2 = \|\mathbf{x} - \mathbf{c}_j\|^2 = \sum_i (x_i - c_{ji})^2 \quad (4.53)$$



Capa oculta: neurona gaussiana

Capa de salida: neurona lineal

**Figura 4.14.** Arquitectura del RBF y forma de las funciones de activación

La salida de la neurona y_j se calcula a partir de una función de activación denominada **función radial** $\phi(r)$. Una de las más típicas es la función gaussiana

$$\phi(r) = e^{-r^2/2\sigma^2} \quad (4.54)$$

En ocasiones se emplean funciones diferentes, aunque de similar dependencia radial, como la que presentamos seguidamente

$$\phi(r) = r^2 \ln(r) \quad (4.55)$$

El término *función de base radial* procede precisamente de la simetría radial de estas funciones (el nodo da una salida idéntica para aquellos patrones que distan lo mismo del centroide). En principio, trabajaremos con la función gaussiana (ecuación 4.54). El parámetro de normalización σ (o factor de escala) mide la anchura de la gaussiana, y equivaldría al radio de influencia de la neurona en el espacio de las entradas; a mayor σ la región que la neurona *domina* en torno al centroide es más amplia (Figura 4.15).

Recopilando las expresiones introducidas, la salida de la neurona oculta j se escribirá

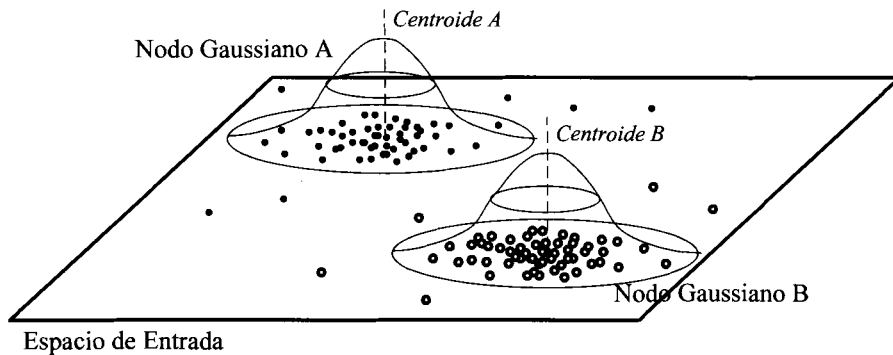


Figura 4.15. Respuesta localizada de las neuronas ocultas en el RBF (nodos gaussianos). Los puntos representan patrones en el espacio de las entradas, que en su mayoría se agrupan en torno a dos centros

$$y_j = e^{-r_j^2 / 2\sigma_j^2} = e^{-\sum_i (x_i - c_{ji})^2 / 2\sigma_j^2} \quad (4.56)$$

Así, si el vector de entradas coincide con el centroide de la neurona j ($\mathbf{x} = \mathbf{c}_j$), ésta responde con máxima salida (la unidad). Es decir, cuando el vector de entradas se sitúa en una región próxima al centroide de una neurona, ésta se activa, indicando que *reconoce* el patrón de entrada; si el patrón de entrada es muy diferente del centroide, la respuesta tiende a cero. Hemos considerado funciones $\phi(\cdot)$ simétricas, en algunos casos se elige una σ diferente para cada dirección, σ_{ji} , adquiriendo formas elipsoidales.

Las salidas de las neuronas ocultas son a su vez las entradas de las neuronas de salida, las cuales calculan su respuesta z_k de la forma

$$z_k = \sum_j w_{kj} y_j + \theta_k = \sum_j w_{kj} \phi(r_j) + \theta_k \quad (4.57)$$

siendo w_{kj} el peso que conecta la neurona oculta j con la de salida k , y θ_k un parámetro adicional de la neurona k , que por similitud con el perceptrón llamaremos umbrales (*bias*). Puede observarse que esta expresión es completamente similar a la (3.50), que define la operación de un MLP con una capa oculta y neuronas de salida lineales, sólo que en aquel caso la función de activación de las neuronas ocultas era de tipo sigmoideo. Como ya se comentó en su momento, esta arquitectura de MLP basada en funciones sigmoideas $f(\cdot)$ constituye un aproximador universal de funciones; paralelamente se demuestra que la arquitectura RBF basada en funciones de respuesta localizada $\phi(\cdot)$ (como las gaussianas), también lo es [Hartman 90]. En el caso de la RBF, cada nodo gaussiano se ocupa de una zona del espacio, y el conjunto de nodos debe cubrir totalmente la zona de interés (Figura 4.16). Este cubrimiento debe llevarse a cabo de la forma más suave posible, lo cual se controla con el número de nodos de la capa oculta y con la anchura σ .

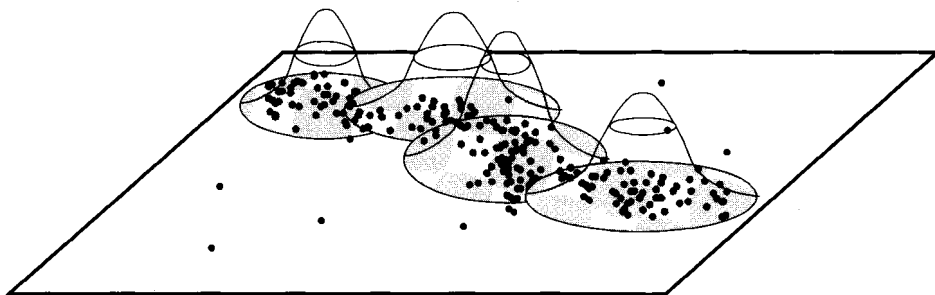


Figura 4.16. Cuatro nodos gaussianos cubren el espacio de trabajo

Aprendizaje en las RBF

Existen distintas cuestiones a abordar en el trabajo con las RBF. La primera, como en el caso del MLP-BP, es cómo elegir el número de nodos radiales (ocultos). Cada nodo radial cubre una parte del espacio de entrada (Figura 4.16), de modo que habrá que elegir el número de nodos adecuados que cubra suficientemente (para una aplicación dada) dicho espacio. El problema surge al tratar con espacios de entrada de muchas variables, pues el número de nodos necesarios crece exponencialmente al hacerlo la dimensión (maldición de la dimensionalidad). De este modo, debe llegarse a un compromiso entre el número de nodos radiales seleccionado, el error que se alcanza en el ajuste de los patrones de aprendizaje, y la capacidad de generalización (puede aparecer sobreajuste al tomar demasiadas neuronas ocultas). A todos estos aspectos se aplican las consideraciones ya discutidas para el caso del BP (capítulo 2).

Además de hacer uso del socorrido método de prueba y error, ampliamente empleado ya en el algoritmo BP, pueden utilizarse otros procedimientos más o menos automáticos para determinar el número de neuronas ocultas en una RBF. Por ejemplo, puede comenzarse con un determinado (reducido) número de nodos gaussianos, y si algún patrón de entrada no activa en suficiente medida ninguna de estas neuronas ocultas, consideramos que es necesario introducir una nueva que dé cuenta de la presencia de una nueva clase de patrones. Un procedimiento así se apunta en el modelo denominado **algoritmo autoorganizado jerárquico** (*Hierarchically Self-organizing Learning Algorithm*) de S. Lee [Lee 92].

Determinado de una u otra manera el número de nodos radiales, se trata a continuación de encontrar los parámetros de la arquitectura. En principio podría tomarse una aproximación global al problema, aplicando a la arquitectura particular de una red RBF el método de descenso por el gradiente, de similar forma a como hicimos para determinar el algoritmo BP.

No obstante, suele emplearse un **aprendizaje por etapas**, en el que en primer lugar se realiza el entrenamiento de las neuronas ocultas gaussianas, para finalmente proceder al entrenamiento de las neuronas de salida.

En el entrenamiento de los nodos gaussianos debe determinarse en primer lugar **el valor de los centroides** c_{ji} , pudiéndose aplicar para ello el conocido algoritmo de las k -medias (*k-means*, véase, por ejemplo, [Tou 74]), o cualquier otro algoritmo no supervisado para agrupamiento (*clustering*). En el caso del algoritmo de las k -medias, k hace referencia al número de grupos (*clusters*) que se trata de encontrar, que en nuestro caso se corresponderá con el número de nodos gaussianos propuesto de partida. Así, en este procedimiento hay que proponer en primer lugar un número k de grupos, es decir, de neuronas, a continuación se sigue el siguiente proceso

- 1) Se eligen los valores de los k centroides c_j de partida. Suelen tomarse como centroides los primeros k patrones de aprendizaje (la elección concreta no es relevante para al resultado final).
- 2) En cada iteración t se reparten los patrones de aprendizaje $\mathbf{x}$ entre las k neuronas. Cada patrón se asigna a la neurona de cuyo centroide dista menos.
- 3) Se calculan los nuevos centroides de cada neurona como promedio de los patrones de aprendizaje asignados en el paso (2). Así, llamando N_j al número de patrones que han correspondido a la neurona j en el reparto, se tiene

$$c_{ji} = \frac{1}{N_j} \sum_{\mathbf{x} \in \text{neurona } j} \mathbf{x} \quad (4.58)$$

- 4) Si los valores de los centroides no han variado respecto de la iteración anterior, el algoritmo ya ha convergido. Si no es así, volver a (2).

Existen otros procedimientos para calcular el valor de los centroides, por ejemplo, pueden emplearse reglas de aprendizaje pertenecientes al campo de las redes neuronales, como el algoritmo de Kohonen simple (sin tener en cuenta la función de vecindad), que estudiamos en el capítulo 4, y cuyo paralelismo con el algoritmo de k -medias ha sido señalado con frecuencia [Kohonen 90].

Obtenidos los centroides, se procederá al **cálculo de los parámetros de escala** σ_j de cada neurona, para lo cual suele hacerse uso de criterios heurísticos. Un procedimiento se basa en calcular de forma aproximada el radio de influencia en el espacio de las entradas de cada neurona en relación a las demás, para lo cual se procede de la siguiente forma: para calcular el σ_j de la neurona j seleccionaremos los centroides de las N neuronas que estén más próximos al del nodo j , y a continuación calcularemos el promedio de las distancias cuadráticas entre ellos [Warwick 95, Wassermann 93], es decir

$$\sigma_j^2 = \frac{1}{N} \sum_{l=1}^N \|\mathbf{c}_l - \mathbf{c}_j\|^2 = \frac{1}{N} \sum_{l=1}^N \sum_k (c_{lk} - c_{jk})^2 \quad (4.59)$$

El caso extremo $N=1$, que consiste en tener en cuenta únicamente el nodo más cercano, obviamente resulta el más rápido de calcular, y sin embargo proporciona buenos resultados en muchos casos [Wassermann 93].

Otro procedimiento es el indicado en [Hush 93], consistente en el cálculo del promedio de la distancia de diversos patrones representativos al centroide

$$\sigma_j^2 = \frac{1}{N_j} \sum_{\mathbf{x} \in \text{nodo } j}^p \|\mathbf{x} - \mathbf{c}_j\|^2 \quad (4.60)$$

donde se ha denominado N_j al número de patrones tomados para el cálculo del factor de escala del nodo j . Calculados los factores de escala de una forma u otra, con ello finaliza el entrenamiento de las neuronas de la capa oculta.

Por último, se procederá al **entrenamiento de las neuronas de salida**. Haciendo notar que las cantidades $\phi(r_j)$ son valores numéricos conocidos, puesto que son función de los valores de las entradas, centroides y factores de escala (cantidades todas ya conocidas), los pesos w_{kj} y umbrales θ_k se calculan simplemente aplicando el **algoritmo LMS** (el de la adalina) a la expresión de la salida de la capa final

$$z_k = \sum_j w_{kj} \phi(r_j) + \theta_k \quad (4.61)$$

que quedará de la forma

$$w_{kj}(t+1) = w_{kj}(t) + \varepsilon(t_k - z_k) \phi(r_j) \quad (4.62)$$

siendo t_k los valores objetivo (*target*). Los umbrales se actualizan siguiendo el mismo esquema, considerando que se trata de pesos con entradas de valor -1.

Al romper el aprendizaje en dos etapas (cálculo de los parámetros de la capa oculta, cálculo de los de la capa de salida), **se consigue acelerar notablemente el proceso de aprendizaje respecto del BP**, el cual opera globalmente (se suele indicar que el BP es unas tres veces más lento). Es muy importante tener en cuenta que de las dos etapas que conforman el aprendizaje, la primera hace uso de un algoritmo no supervisado, sea el de Kohonen o el de k -medias, mientras que en la segunda se emplea el algoritmo de la adalina, de tipo supervisado. Al conjugar un tipo de aprendizaje no supervisado en la capa oculta, con otro supervisado en la de salida, en ocasiones se dice que este modelo es una **red híbrida**.

Relaciones de las RBF con otras técnicas

Las RBF y el MLP son redes de ajuste funcional muy relacionadas: ambas son arquitecturas en capas y unidireccionales, las dos son también aproximadores universales y se aplican al mismo tipo de tareas. **El aprendizaje de las RBF es más rápido que el del BP**, aunque cuando se simulan sobre sistemas serie (computadores

convencionales) **en fase de ejecución las RBF son más lentas**, debido a que normalmente precisan de un elevado número de nodos ocultos. La razón es que las neuronas intermedias de las RBF actúan localmente, mientras que las del MLP lo hacen globalmente, por lo que un nodo MLP cubre una mayor parte del espacio de entradas, requiriéndose por lo tanto menos nodos (hablando en términos generales). Como siempre, el empleo de uno u otro modelo con mayor o menor éxito dependerá del problema concreto.

Por otro lado, recientemente se ha demostrado la **equivalencia de las RBF con un cierto tipo de sistema borroso** [Jang 93a, Jang 95, Jang 97], y en [Li 00] se demuestra la equivalencia en general de las redes unidireccionales y los sistemas borrosos; este asunto se tratará más ampliamente en el capítulo 9, ya dentro de la segunda parte del libro dedicada a los sistemas borrosos. Finalmente, indicaremos que en [Rayneri 99] se propone la unificación de redes neuronales, sistemas borrosos y wavelets, haciendo uso del modelo **WRBF** (*weighted RBF*). Parece ser que todos estos modelos, que en principio surgen de áreas diversas, pueden considerarse casos particulares de un modelo de procesamiento distribuido y adaptativo general.

Remitimos al lector interesado en profundizar en el modelo RBF a las referencias citadas a lo largo de la presente sección. Nos resulta especialmente destacable el capítulo dedicado a las RBF del libro [Wassermann 93], donde se discuten ampliamente los conceptos aquí expuestos, y se muestran algunos otros modelos relacionados, como las **Redes de Regresión Generalizada o GRNN** (*General Regression Neural Networks*) de D. Specht [Specht 90], y la **Red de Potenciales Gaussianos o GPFN** (*Gaussian Potential Function Network*), de S. Lee [Lee 91, 92], que pueden estudiarse dentro de un marco común. En [Haykin 99, Principe 00] se realiza un estudio detallado y actualizado de las RBF.

4.8 LVQ

El **LVQ** (*Learning Vector Quantization*) es un modelo supervisado introducido por Kohonen a finales de los años ochenta [Kohonen 90] para llevar a cabo tareas de **clasificación** de patrones. Esta red se compone de una capa simple de neuronas de Kohonen (sin relaciones de vecindad, véase el capítulo 3); supondremos de entrada que cada una de las n neuronas representa una clase, y almacena un vector de referencia $\mathbf{w}_j$ ($j=1,2,\dots, n$) que representa el **prototipo de una clase**. Cuando se presenta a la red un patrón de entrada $\mathbf{x}$, compuesto por m entradas, cada neurona **calcula la distancia** que separa su vector de referencia del de entrada

$$d^2(\mathbf{w}_j, \mathbf{x}) = \sum_{k=1}^m (w_{jk} - x_k)^2 \quad (4.63)$$

la neurona c de menor distancia, con vector de referencia $\mathbf{w}_c$, indica la clase C en la que el patrón $\mathbf{x}$ queda clasificado.

El **aprendizaje** se basa en premiar a la neurona que clasifica correctamente un patrón, actualizando sus pesos con la clásica regla de Kohonen (capítulo 3), y castigar a la que realiza una clasificación errónea, modificando sus pesos en sentido contrario.

Supongamos que disponemos de p patrones de aprendizaje $\mathbf{x}^\mu$ ($\mu=1,\dots,p$) perfectamente etiquetados, es decir, cada patrón pertenece a una clase $C_{x\mu}$ conocida, habiendo en total n clases (tantas como neuronas); asimismo, cada neurona de pesos $\mathbf{w}_j$ es etiquetada con una clase C_{w_j} . Si $\mathbf{w}_c$ es el vector de referencia más próximo al patrón de entrada $\mathbf{x}^\mu$ presentado (según 4.63), y denominamos C_{w_c} a la clase definida por dicho vector de referencia, se tendrá la siguiente actualización para la neurona c :

- 1) Si $C_{w_c}=C_{x\mu}$ (clasificación correcta), el vector de referencia se modifica de la forma

$$\mathbf{w}_c(t+1) = \mathbf{w}_c(t) + \alpha_t [\mathbf{x}^\mu - \mathbf{w}_c(t)] \quad (4.64)$$

con $0 < \alpha_t < 1$, y normalmente decreciente con el tiempo t (por ejemplo, partiendo de 0.1). En este caso, los pesos rotan hacia las entradas, aproximándose (se puede considerar equivalente a un *premio*).

- 2) Si por el contrario $C_{w_c} \neq C_{x\mu}$ (clasificación incorrecta), la modificación se realiza justo en sentido contrario, viniendo dada por

$$\mathbf{w}_c(t+1) = \mathbf{w}_c(t) - \alpha_t [\mathbf{x}^\mu - \mathbf{w}_c(t)] \quad (4.65)$$

de forma que los pesos tienden a alejarse de las entradas (*castigo*).

El resto de los vectores de referencia no se modifican. El proceso se repetirá iterativamente con todos los ejemplos de entrenamiento, una y otra vez. En [Kohonen 97] se indica que para evitar sobreentrenamiento (como en el MLP) debe realizarse la parada temprana del proceso de entrenamiento, tras, por ejemplo, un número de iteraciones comprendido entre 50 y 200 veces el número de neuronas de la red.

Por otro lado, para establecer los **vectores de referencia iniciales** $\mathbf{w}_j$ pueden emplearse directamente vectores ejemplo $\mathbf{x}^\mu$ pertenecientes a la clase que representa la neurona, aunque también puede emplearse el algoritmo de k -medios o incluso un mapa autoorganizado [Kohonen 97]. Finalmente, indicaremos que hasta ahora hemos supuesto que cada neurona representa una clase, sin embargo suelen obtenerse mejores resultados **cuando varias neuronas representan una misma clase**; el número de neuronas por clase será dependiente del problema.

El LVQ es un modelo simple pero sumamente eficaz, por lo que muy pronto encontró una amplia difusión, siendo hoy en día muy empleado en aplicaciones reales. En [Kohonen 97] pueden estudiarse dos modificaciones al LVQ, denominadas LVQ2 y LVQ3, que pueden ofrecer alguna mejora en ciertos casos, como cuando los datos se distribuyen multimodalmente. Finalmente, en el manual del programa LVQ_PACK, desarrollado por el grupo de Kohonen [LVQ_PACK 95] puede encontrarse interesante información adicional.

4.9 OTROS MODELOS DE REDES NEURONALES

Por motivos pedagógicos (para no añadir mayor confusión al lector que se inicia en redes neuronales), en este texto decidimos tratar tan sólo los modelos más ilustrativos y utilizados, de modo que hemos dejado de lado intencionadamente una serie de modelos clásicos, como puedan ser el **Neocognitrón** [Fukushima 80, 83], **ART** (*Adaptive Resonance Theory*) [Carpenter 88], **BSB** (*Brain State in a Box*), **CMAC** (*Cerebellum Model Articulation Controller*), **contra-propagación** [Hecht-Nielsen 90] o **BAM** (*Bidirectional Associative Memory*). No obstante, el lector interesado en alguno de estos modelos puede consultar libros bien conocidos, como [Hecht-Nielsen 90, Hertz 91, Wasserman 89, 93, Zurada 92a].

Por otro lado, en la actualidad siguen apareciendo modelos nuevos y obteniéndose nuevos resultados sobre los tradicionales (MLP, SOFM, RBF, etc.), por lo que hay que permanecer atento a las revistas especializadas en el tema, como *IEEE Transactions on Neural Networks* o *Neural Networks*, por citar dos de las más relevantes. No obstante, los recientes textos [Bishop 95, Haykin 99, Principe 00] son muy interesantes en este sentido.

De entre los nuevos modelos destacaríamos especialmente dos. Por un lado, las citadas **GRNN** (redes de regresión generalizada) [Specht 90], como modelo que implementa aprendizaje *en un paso* (es decir, no iterativo), que presenta la gran ventaja frente a modelos como las RBF (con las que está emparentado) y, sobre todo, BP, de su rapidez de aprendizaje.

El otro modelo reciente que se está extendiendo es el de las **máquinas de vectores soporte o SVM** (*Support Vector Machines*), empleadas tanto en reconocimiento de patrones como en ajuste funcional, que parten del trabajo de Vapnik, Chervonenkis y otros autores [Vapnik 99b] sobre teoría estadística del aprendizaje (*statistical learning theory*). Las SVM, introducidas por Vapnik, permiten el uso de diversos *kernels* como *neuronas intermedias* (polinomios, *splines*, RBF, MLP), y formula el proceso de aprendizaje como un problema de optimización convexa, del que se deduce también la complejidad del modelo (arquitectura); a partir de la dimensión VC pueden establecerse límites sobre su capacidad de generalización. En [IEEE 99b, Haykin 99, Principe 00] puede encontrarse información sobre este importante modelo.

IMPLEMENTACIÓN DE REDES NEURONALES¹

En los dos capítulos que restan expondremos los aspectos relacionados con la puesta en práctica de sistemas neuronales artificiales, comenzando con las distintas formas de realizar redes neuronales, dejando para el capítulo siguiente distintos asuntos relativos al desarrollo de aplicaciones prácticas.

Un ANS puede **simularse** mediante programas ejecutados en computadores convencionales, lo que constituye siempre el primer paso en el desarrollo. No obstante, deberá **realizarse en hardware** si se desea aprovechar su capacidad de cálculo masivamente paralelo, permitiendo así su aplicación a problemas donde deban tratarse cantidades masivas de datos en tiempo real. La tecnología de implementación más habitual es la **microelectrónica VLSI** (*Very Large Scale Integration*).

Expondremos en este capítulo una panorámica del estado de desarrollo actual de la realización de ANS, haciendo énfasis en la implementación electrónica y en los **sistemas disponibles comercialmente**. No es nuestra intención ofrecer una revisión exhaustiva, sino presentar los aspectos generales del problema, junto a unos cuantos ejemplos de **chips neuronales y neurocomputadores** interesantes, pues hay numerosas revisiones de este tipo en la literatura [Kolinummi 97, Lindsey 98a, 98b, Przytula 93, Ienne 93, Cabestany 93, Atlas 89, Graff 89, Farhat 89, Mead 89, Ramacher 91b, Treleaven 89].

La lectura de este capítulo se recomienda incluso al no especialista en electrónica, puesto que es en buena parte descriptivo, resultando fundamental para conocer el estado actual del hardware neuronal. El lector no especialista o no interesado puede saltarse las secciones 5.5.1, 5.5.2, 5.6, 5.7, 5.8 y 5.11, **pero conviene que lea las secciones 5.1, 5.2, 5.3, 5.9, 5.10 y 5.12.**

¹ Este capítulo ha sido realizado por Nicolás Medrano Marqués y Bonifacio Martín del Brío. Nicolás Medrano pertenece al Departamento de Ingeniería Electrónica y Comunicaciones de la Universidad de Zaragoza.

5.1 INTRODUCCIÓN

Uno de los motivos más importantes del resurgir de las redes neuronales en la década de los ochenta fue el desarrollo de la **tecnología microelectrónica de alta escala de integración o VLSI** (*Very Large Scale Integration*), debido a dos circunstancias. Por una parte, posibilitó el desarrollo de computadores potentes y baratos, lo que facilitó la simulación de modelos de redes neuronales artificiales de un relativamente alto nivel de complejidad, permitiendo su aplicación a numerosos problemas prácticos en los que demostraron un excelente comportamiento. Por otra parte, la integración VLSI posibilitó la realización hardware directa de ANS, como dispositivos de cálculo paralelo aplicables a problemas computacionalmente costosos, como visión o reconocimiento de patrones.

Aunque en el presente capítulo nos centraremos en la implementación hardware de redes neuronales, describiremos en primer lugar las diferentes **alternativas para su realización** [Kröse 93, Toxa 91]:

- a) **Simulación software.** Consiste en modelar el sistema neuronal mediante un programa (software), que puede ejecutarse en un computador secuencial convencional tipo Von Neumann, por ejemplo, **un PC compatible**, o bien sobre un computador de propósito general con capacidad de cálculo paralelo (sistemas multiprocesador, máquinas vectoriales, paralelas...). Es la realización más usual y económica.
- b) **Emulación hardware (propósito general).** Se utilizan sistemas expresamente diseñados para emular ANS, basados en CPU de altas prestaciones (RISC, DSP,...), o en otros microprocesadores diseñados específicamente para ANS. Son los neuroprocesadores de propósito general, que dan lugar a **aceleradores (neuroemuladores) y neurocomputadores**
- c) **Implementación hardware específica.** Se trata de realizar físicamente la red neuronal en hardware, es decir, mediante estructuras específicas que reflejen con mayor o menor fidelidad la arquitectura de la red, lo cual da lugar a los neuroprocesadores de propósito específico, muchos de los cuales se emplean en aplicaciones específicas (por ejemplo, en bolígrafos-escáner OCR o reconocedores de comandos hablados). La tecnología más habitualmente utilizada es la electrónica, bien utilizando dispositivos de lógica programable (como FPGA), bien mediante integración microelectrónica, lo que da lugar a los denominados **chips neuronales**.

Tecnologías para la realización de ANS

Los soportes o tecnologías más utilizadas en la realización de redes neuronales artificiales son el lógico (software) y el microelectrónico (tabla 5.1). El más extendido de todos es el **soporte software**, que supone la oportunidad más inmediata para experimentar con un sistema neuronal. Muchas veces el desarrollo de un ANS queda

reducido a su simulación en un computador convencional y, dada la potencia de los ordenadores actuales (incluso los PC) suele resultar suficiente en muchas ocasiones.

La otra tecnología extendida es la **microelectrónica**, gracias al extraordinario avance de la integración VLSI y a la disponibilidad de potentes herramientas CAD para diseño. Es ésta la que se tratará ampliamente en este capítulo.

Desde hace unos años se está experimentando también con otras **técnicas novedosas**, como las basadas en **procesamiento óptico** [Kosko 92b, Lange 94], la cual cuenta con las siguientes ventajas: velocidad de proceso (el fotón es más rápido que el electrón), posibilidad de trabajar en las tres dimensiones del espacio (frente a las tecnologías microelectrónicas, esencialmente planares) y permitir el cruce de haces de fotones sin interferencia. **Nanotecnología y nanoelectrónica** representan un paso más allá de la microelectrónica tradicional [ISCAS 00]; en [Roermund 00, Gerousis 00] se presentan unas primeras realizaciones de redes neuronales mediante estas técnicas. Por último, señalaremos que las realizaciones basadas en el **soporte biológico-químico** están en un estadio inicial de experimentación, aunque la computación molecular puede adquirir importancia en un futuro² [IEEE 92].

Realización física de ANS	Soportes (tecnologías)
Simulación por computador	Lógico
Emulación (hardware)	Microelectrónico Optoelectrónico
Realización hardware	Nanotecnología Biológico-químico

Tabla 5.1 Realización física de redes neuronales artificiales y soportes utilizados

5.2 SIMULACIÓN (SOFTWARE) DE ANS

Denominaremos simulación software de un ANS a su realización en forma de un programa ejecutable sobre un computador de propósito general. Por ser el procedimiento más simple, rápido y económico, constituye siempre la primera etapa en el desarrollo de un ANS, incluso cuando el objetivo último sea la creación de un chip de propósito específico. Además, es una solución muy versátil que permite el ensayo con cualquier tipo de arquitectura.

² No olvidemos que el dispositivo de cómputo más potente sigue siendo el cerebro humano, cuyo hardware (*wetware*) está compuesto por proteínas, sales, grasas, ácidos nucleicos, hidratos de carbono y agua.

El proceso de simulación de ANS comienza con su modelado mediante programas de ordenador escritos en **lenguajes de alto nivel** como C o Pascal (o incluso Visual Basic), que se ejecutarán en computadoras convencionales, como **PC compatibles**. Aunque de esta manera se pierde su capacidad de cálculo en paralelo, las prestaciones que ofrecen los ordenadores actuales resultan suficientes para resolver numerosos problemas, al permitir simular redes de tamaño medio a una velocidad razonable. En este caso, la actividad de cada neurona se suele representar mediante una variable del programa. Como las neuronas se agrupan en capas, las entradas de las neuronas de una capa se suelen reunir en vectores. Lo mismo sucede con las salidas.

```
float x[N_INPUTS], y[N_OUTPUTS]; /* Entradas x y salidas y de una capa */
```

Cada neurona posee un vector de pesos sinápticos, por lo tanto, los pesos correspondientes a las neuronas de una capa se agrupan en una matriz *W*

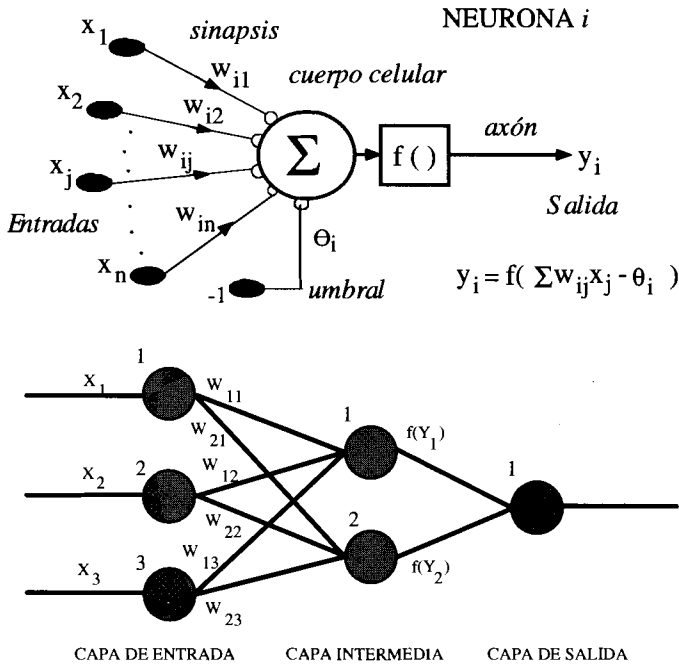
```
float w[N_OUTPUTS][N_INPUTS]; /* Matriz de pesos sinápticos */
```

De este modo, las salidas *y* de las neuronas de la capa en cuestión se calcularán a partir del producto de sus entradas *x* por la matriz de pesos *W*, resultado al que se aplica la función de transferencia *f(.)* (Figura 5.1).

Con el fin de ahorrar esfuerzos de programación **se comercializan simuladores de redes neuronales**, como puedan ser *BrainMaker* (<http://www.brainmaker.com/>), *NeuralWorks* (<http://www.neuralware.com/>) o *NeuroSolutions* (<http://www.nd.com/>) (consúltese el capítulo 6). Estos programas están dotados de entornos gráficos de manejo intuitivo y simulan muchos modelos neuronales clásicos (en <ftp://ftp.sas.com/pub/neural/FAQ.html> se realiza una revisión actualizada del software neuronal disponible). A modo de ilustración, en la Figura 5.2 se muestra el aspecto externo de un programa de simulación de ANS. En conclusión, la simulación software se presenta como un método ideal para el entrenamiento y evaluación de un ANS.

Los principales problemas de la simulación en computadores convencionales surgen del carácter esencialmente secuencial de éstos, en contrapartida al inherente paralelismo de los ANS. Si tenemos en cuenta que muchos de los modelos neuronales son computacionalmente muy costosos (especialmente en la fase de aprendizaje, donde para modelos como el BP aplicado sobre redes de gran tamaño podemos encontrarnos con tiempos de cálculo de varios días de CPU), es fácil comprender la dificultad de su aplicación en tiempo real en ciertos casos.

Por ejemplo, en [Hammerstrom 93a] se plantea que un problema que precise de una red de 100 neuronas (lo que constituye un sistema relativamente pequeño), incluirá del orden de 10.000 conexiones o pesos sinápticos. Si suponemos un microprocesador estándar que realice alrededor de 10 millones de multiplicaciones y acumulaciones por segundo, podría ejecutar aproximadamente 1.000 fases de recuerdo por segundo. Si el tiempo de respuesta necesario es del orden del milisegundo, la simulación en un ordenador convencional resultará suficiente en este caso.



```

for (i = 0; i < N_OUTPUTS; i++)
{
    /* 'actividad' es un 'float' que se usa como variable muda */
    actividad = 0.0;

    /* suma ponderada */
    for (j = 0; j < N_INPUTS; j++) actividad = actividad + w[i][j]*x[j];

    /* f() es la función de activación */
    y[i] = f(actividad);
}

```

Figura 5.1 Variables involucradas en el cálculo de la salida de una neurona, y pseudocódigo de alto nivel que reproduce la operación hacia adelante de una red neuronal; los datos son almacenados como vectores y los pesos como matrices (donde cada columna se corresponde con el vector de pesos asociado a una neurona)

Sin embargo, si la aplicación requiere 300 neuronas (tres veces más), la red contendrá 90.000 conexiones, con lo que sólo podrá realizar unas 100 fases de recuerdo por segundo (10 veces menos). Así, para un problema cuya resolución precise de miles o millones de neuronas (como puede ser una tarea de visión), **la simulación sobre un ordenador convencional puede ser inoperante**, y se hace necesaria una solución distinta, en la que el paralelismo de cálculo sea manifiesto.

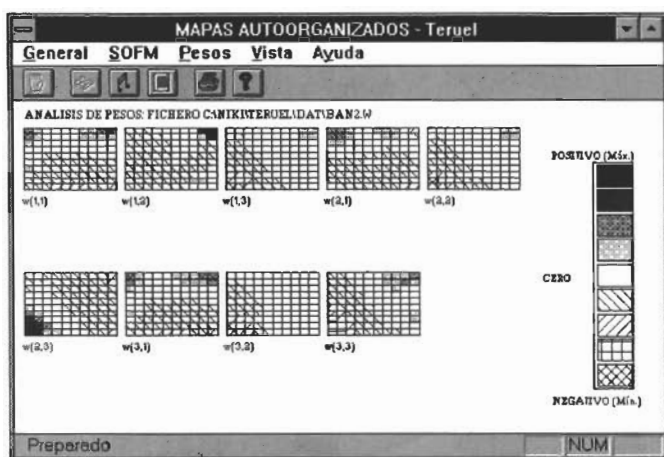


Figura 5.2 Interfase de usuario de un programa de simulación de redes neuronales

Una solución puede ser la utilización de máquinas vectoriales, multiprocesador o supercomputadores; el inconveniente obvio es su precio y el difícil acceso a este tipo de máquinas. Además, en muchas ocasiones se requiere un sistema económico y de tamaño y consumo reducidos, especialmente diseñado para una aplicación concreta. A modo de ejemplo, no podemos utilizar un supercomputador Cray para realizar el control de un sencillo brazo robot o para traducir voz a texto.

En conclusión, debido a los considerables recursos computacionales necesarios para entrenar un ANS de tamaño medio o alto y a la necesidad de respuesta en tiempo real en diversas aplicaciones, **en ocasiones se debe superar la simple simulación en ordenador y recurrir a estructuras hardware especialmente diseñadas para la emulación o realización de redes neuronales.**

5.3 EMULACIÓN (HARDWARE) DE ANS

Con el término **emulación** hardware nos referiremos a la utilización de una estructura de procesamiento con cierta capacidad de cálculo paralelo, especialmente diseñada para el trabajo con ANS. Se trata de una solución intermedia entre la simulación y la verdadera realización hardware, que resulta perfectamente válida en numerosos casos. Normalmente, se emplea una tarjeta coprocesadora dependiente de un computador central o *host* (Figura 5.3), construida tomando como base microprocesadores de altas prestaciones (RISC, DSP) o procesadores neuronales (sección 5.8); normalmente incluye varias unidades de procesamiento para permitir cierto paralelismo en los cálculos. Mediante un software de simulación adecuado, se pueden aprovechar las características de cálculo limitadamente paralelo de la tarjeta utilizada para acelerar la simulación de un ANS, por lo que suelen denominarse

tarjetas aceleradoras, y al conjunto PC más placa, **neurocomputador** (Figura 5.3). Constituye ésta una solución tanto para el desarrollo y evaluación de un sistema neuronal como para aplicaciones prácticas complejas. Aunque permiten el trabajo con prácticamente cualquier tipo de modelo neuronal, estas tarjetas suponen un importante desembolso económico, y, lo que es peor, **dado el rápido incremento de las prestaciones de los computadores actuales, puede quedarse anticuada en poco tiempo** [Granado 99, Lindsey 98].

Desde hace años se comercializan tarjetas aceleradoras o emuladores, pero este área evoluciona muy deprisa y productos que hasta hace poco tiempo lideraban el mercado en la actualidad ya no se venden, como sucede con los neurocomputadores Balboa 860 y SNAP de la Hecht-Nielsen Company (HNC).

A modo de ejemplo, vamos a comentar dos productos que sí están disponibles actualmente. La empresa alemana MediaInterface (<http://www.mediainterface.de/>) comercializa **Synapse3-PC**, una placa neurocomputadora para ordenador PC compatible realizada por Siemens-Nixdorf, que incluye dos procesadores matriciales (*array processors*) MA-16 diseñados por Siemens para trabajar con ANS [Kolinummi 97]. Una placa Synapse3 puede alcanzar unas prestaciones de más de mil millones de MACS (multiplicaciones y acumulaciones por segundo) y un PC puede equiparse con hasta tres placas. Otro ejemplo es el de la empresa italiana NeuriCam (<http://www.neuricam.com>), que comercializa los chips neuronales NC300x TOTEM, base de la serie de placas aceleradoras **TOTEM PCI** para ordenadores PC. Una placa con dos procesadores NC300x alcanza los 800 millones de MACS, por un precio de unos 2000 euros (mucho más reducido que el correspondiente a un supercomputador).

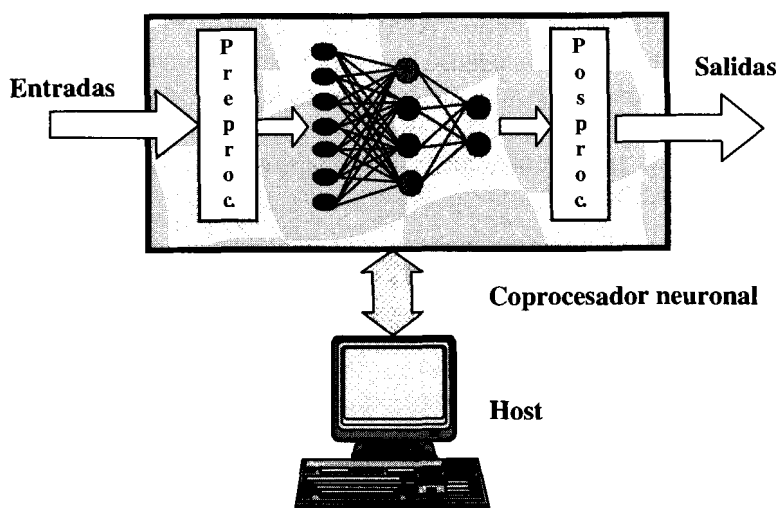


Figura 5.3 Estructura host-coprocesador neuronal, constituyendo un neurocomputador

5.4 REALIZACIÓN HARDWARE DE ANS

Consiste en construir físicamente un sistema cuya arquitectura refleje en mayor o menor medida la estructura de la red neuronal. Debido a ello, el sistema hardware debe soportar un gran número de conexiones, en ocasiones globales, y un flujo elevado de comunicaciones de datos. Estas circunstancias limitan considerablemente el tamaño de la red neuronal que puede ser implementada.

Este tipo de realización está especialmente indicado para aplicaciones especiales donde se precisa un **alto rendimiento**, ya que el diseño se realiza para un cierto modelo de red neuronal y para una aplicación concreta [Lindsey 98a]. También es la solución en las **aplicaciones específicas (*embedded systems*)**, donde se requiere un circuito electrónico empotrado en la aplicación; un ejemplo es el *Quicktionary*TM, de la casa Wizcom (<http://wizcomtech.com>), escáner con forma de bolígrafo, traductor y lector, que integra el chip neuronal OCR-on-a-chip de reconocimiento de caracteres (OCR) de la empresa Ligature (<http://www.ligatureltd.com>).

La realización hardware se puede basar en tecnologías electrónicas, ópticas o bioquímicas. Aunque se experimenta con realizaciones de ANS basadas en tecnologías optoelectrónicas [Farhat 89, Lange 94], y comienzan a ensayarse implementaciones biológico-químicas, son las tecnologías electrónicas las más empleadas en la actualidad, debido a la disponibilidad de los procesos tecnológicos microelectrónicos y el elevado grado de automatización del diseño [Carrabina 91].

La realización electrónica se lleva a cabo mediante tecnología microelectrónica VLSI, en forma de ASIC (*Application Specific Integrated Circuit*) o mediante FPGA (*Field Programmable Gate Arrays*). Aunque el diseño de ASIC es lo más habitual, en la actualidad la realización mediante FPGA [Kung 93a, Lindh 93] presenta la ventaja de un período de desarrollo más corto. Aunque su densidad de puertas es menor³, son fácilmente programables por el usuario mediante herramientas CAD que permiten captura esquemática, síntesis y simulación a partir de una descripción en un HDL (*Hardware Description Language*). Las FPGA resultan interesantes para el desarrollo rápido y económico de prototipos o series pequeñas, aunque no permiten diseños tan complejos como los ASIC.

Sin duda, la microelectrónica constituye el tipo de implementación hardware de redes neuronales más habitual. Su diseño se puede estructurar de forma jerárquica, de modo que la realización de un sistema neuronal puede llevarse a cabo desde distintos **niveles de abstracción** [Toxa 91]:

- a) Nivel de sistema. Utilización de transputers, coprocesadores, DSP, etc.
- b) Nivel de estructura funcional. Arquitecturas sistólicas, ALU específicas.
- c) Nivel de puerta lógica. Lógica de pulsos, lógica estocástica.

³ Si bien las actuales FPGA pueden incluir ya hasta millones de puertas equivalentes.

- d) Nivel de puerta analógica. Redes de Hopfield, Kohonen, Maxnet, etc.
- e) Nivel de transistor. Realización de chips de muy alta densidad de integración y elevado número de neuronas, como redes de Hopfield.
- f) Nivel de modelo. Modelo del transistor MOS como resistencia, MOS en región sub-umbral (multiplicadores analógicos).
- g) Nivel de *layout*. Diseño de resistencias, modelos de interconexión.
- h) Nivel de proceso. Modificaciones del proceso CMOS (fusibles, EEPROM, EPROM), CCD, optoelectrónica, bipolares.

En [Toxa 91] se muestran ejemplos clásicos de realizaciones para cada uno de estos niveles.

5.5 NEUROCOMPUTADORES Y CHIPS NEURONALES

Describiremos a continuación diferentes formas de realizar redes neuronales artificiales haciendo uso de las tecnologías microelectrónicas, desde los emuladores (aceleradores) hardware, especialmente concebidos para la emulación de los ANS, hasta los chips neuronales, más próximos a la realización fiel de la arquitectura de la red. La implementación hardware de ANS no resulta una cuestión anecdótica, pues el impacto tecnológico (y económico) de las redes neuronales está siendo importante [SIENA 96, COTEC 98], y en muchas aplicaciones, debido a los grandes recursos computacionales necesarios, hay que recurrir a realizaciones electrónicas.

En la Figura 5.4 se muestra un gráfico procedente de un famoso estudio sobre redes neuronales del DARPA⁴ [DARPA 88], donde se representan los requisitos computacionales (capacidad de almacenamiento o sinapsis, y velocidad de procesamiento o conexiones por segundo) asociados a varios problemas importantes abordados mediante ANS, como los de visión, habla o procesamiento de señal, y las características del hardware disponible al efecto a mediados de los ochenta (incluyendo placas neurocomputadoras de la época, como ANZA, Mark IV y otras). Se puede apreciar que en casi todos los casos **las necesidades de hardware se situaban entonces por encima de los recursos disponibles**, lo que constituía un freno importante en la investigación y aplicabilidad de las redes neuronales a importantes problemas tecnológicos [Ramacher 91b], lo que llevó a la conclusión de que **se hacía necesario el desarrollo de hardware específico para redes neuronales**.

⁴ Defence Advanced Research Projects Agency. Se trata de un estudio llevado a cabo por el gobierno norteamericano, alertado por el interés que las técnicas neuronales estaban despertando y por su potencial aplicación a la resolución de problemas estratégicos.

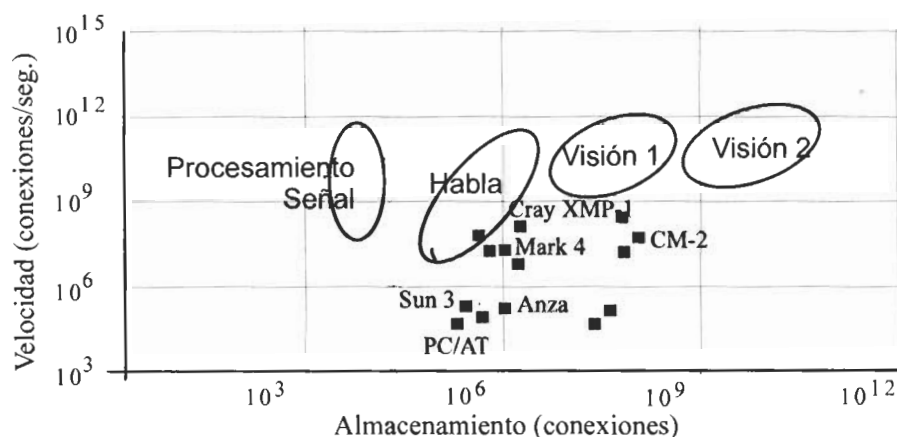


Figura 5.4 Capacidad del hardware disponible a mediados de los ochenta en relación a los recursos necesarios para diversas aplicaciones [DARPA 88]

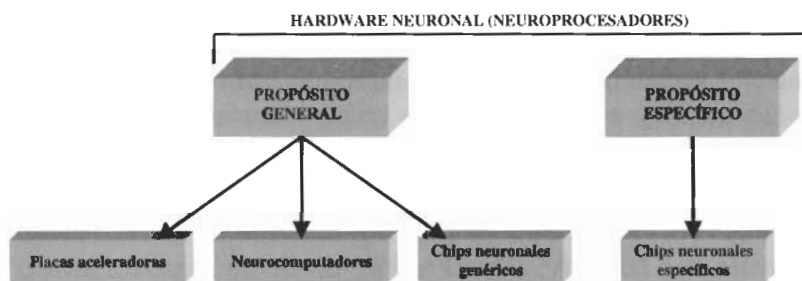


Figura 5.5 Neurocomputador, placas aceleradoras y neurochips

Debido a ello, multitud de **grupos universitarios y empresas** comenzaron a trabajar en el desarrollo de neurochips y neurocomputadores; algunas de estas empresas eran multinacionales del sector de la electrónica e informática bien conocidas, como IBM, Siemens, Hitachi, Intel, AT&T o Philips, y otras se crearon viendo las nuevas posibilidades de este campo, como Hecht-Nielsen Company, Adaptive Solutions, Synaptics, Sensory, NeuroCam, Nestor, Innovative Computing o Ligature. Actualmente podemos señalar que por diversas circunstancias parte de las multinacionales y pequeñas empresas citadas parecen haber abandonado sus programas de desarrollo y comercialización de hardware neuronal, pero bastantes empresas de reciente creación continúan con esta área de negocio, a veces compaginándolo con su trabajo como consultoras para el desarrollo de aplicaciones basadas en ANS. Trataremos un poco más estos aspectos comerciales al final del capítulo.

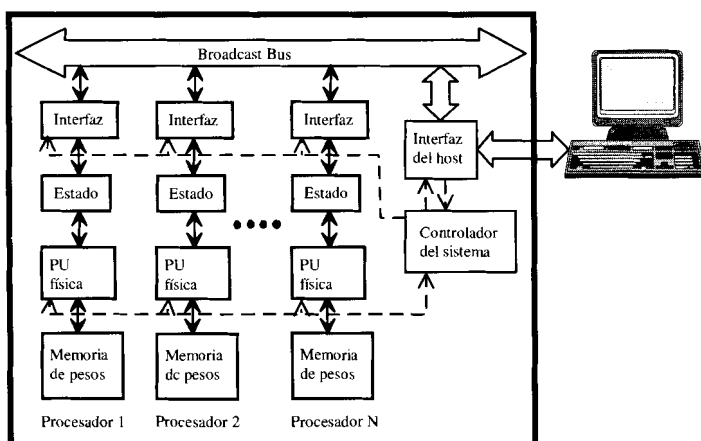


Figura 5.6 Neurocomputador de propósito general [Treleaven 89]

Hasta el momento hemos hablado de conceptos generales relacionados con la simulación, emulación e implementación de ANS, a partir de ahora nos centraremos en la realización electrónica de chips neuronales y neurocomputadores (Figura 5.5).

5.5.1 Especificaciones de un neuroprocesador

Por **neuroprocesador** entendemos un dispositivo con capacidad de cálculo paralelo, diseñado para la implementación de redes neuronales artificiales. Puede ser de propósito general o específico, y puede realizarse como un chip neuronal o como una placa aceleradora dependiente de un computador *host* (Figura 5.5). El término **neurocomputador** lo reservaremos para el sistema *host* más placa aceleradora (Figura 5.6); el papel del *host* (un computador convencional) es el de proporcionar al usuario un medio de comunicación con la placa procesadora, mediante su teclado, pantalla y unidades de almacenamiento masivo. Además, el *host* gestiona el sistema completo por medio de un programa.

Un ejemplo de neuroprocesador de propósito general se muestra en la Figura 5.6, con arquitectura clásica orientada a bus, compuesta por un conjunto de unidades procesadoras o PU (*Processing Units*), dotadas de memoria local para el almacenamiento de los pesos de las neuronas virtuales que emulan, y conectadas a un bus global que las comunica con el host.

En términos generales, en el análisis de la arquitectura de un neuroprocesador se deben tener en cuenta los siguientes aspectos relevantes [Treleaven 89]:

- **Grado de programabilidad del sistema** (flexibilidad). Indica la versatilidad para la emulación de diferentes modelos neuronales.

- **Número de unidades de procesamiento** (paralelismo). Indica el paralelismo del neuroprocesador (disponer de un procesador por neurona suele resultar costoso, por lo que habitualmente hay que distribuir las neuronas a emular entre los procesadores realmente disponibles).
- **Complejidad de cada procesador individual**. Es un índice de la potencia de cálculo y versatilidad de una unidad procesadora.
- **Rendimiento el sistema**. Para medir la velocidad de procesamiento, en la fase de recuerdo se suele proporcionar el número de conexiones procesadas por segundo o CPS (*connections per second*), y en la de aprendizaje las conexiones actualizadas o CUPS (*connections updated per second*). Rigurosamente hablando, CPS y CUPS miden una única característica del procesamiento (en esencia, el tiempo de una operación MAC⁵), sin contemplar otros factores, como el tiempo de carga de datos desde la memoria principal o el de intercambio de datos entre procesadores. Además, la velocidad dependerá del modelo neuronal concreto que se emule, pues cada uno requiere la realización de operaciones diferentes (sigmoideas, normas, etc.). En definitiva, la medida del rendimiento real de un neuroprocesador es en realidad algo más complejo que el simple dato de CPS y CUPS.

Un aumento en las prestaciones (un sistema con mayor paralelismo, más flexible, o más veloz) supone una mayor utilización de área de silicio, lo que conlleva el incremento en el precio del sistema. Como en toda cuestión de desarrollo tecnológico, se debe alcanzar un compromiso entre prestaciones y coste. Conjugar ambas consideraciones conduce a desarrollos totalmente diferentes, como se muestra en la Figura 5.7, donde se representan desde sistemas de simplicidad comparable a las RAM tradicionales, hasta otros programables de extrema complejidad.

Neuroprocesadores de propósito general y específico

En relación a los modelos que un neuroprocesador puede emular, se distinguen dos grandes tipos [Cabestany 93, Toxa 91]:

- a) **Neuroprocesadores de propósito general**. Son los diseñados para emular un gran número de modelos de redes neuronales; constituyen por tanto máquinas flexibles y versátiles.
- b) **Neuroprocesadores de propósito específico**. Se trata de sistemas limitados a la emulación de un único modelo neuronal. Optimizan la ocupación de área de silicio (y el precio, por lo tanto) y alcanzan mayores velocidades de procesamiento para ese modelo concreto. Como contrapartida, son menos flexibles y limitadamente programables.

⁵ Multiplicación y acumulación, es el tipo de operación básica en los DSP.

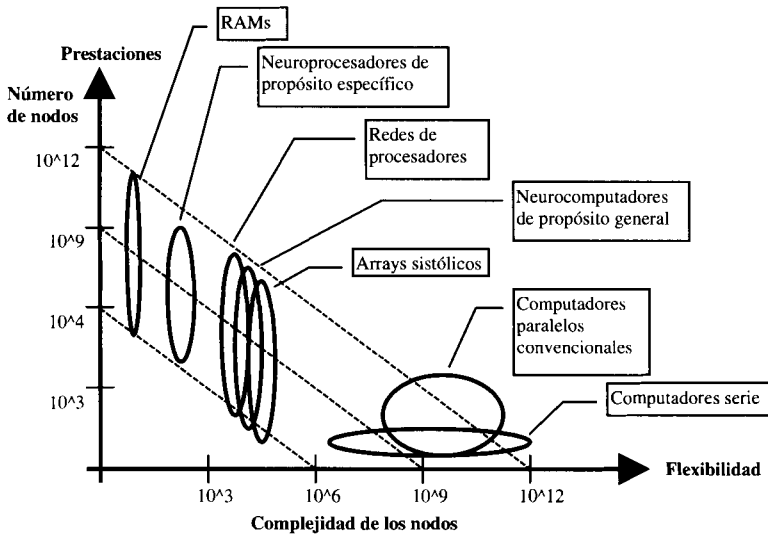


Figura 5.7. Espectro de neuroprocesadores [Treleaven 89]

Un **neuroprocesador de propósito general** tendrá un alto grado de programabilidad y sus unidades procesadoras o PU serán más complejas; a partir de él se construyen **neurocomputadores**, por lo que a menudo se utiliza directamente este último término. Dispone, por tanto, de un número de PU no muy elevado, por lo que cada una deberá representar varias neuronas de la red. En este sentido, existe un paralelismo entre el concepto habitual de memoria virtual y el de **red virtual**: un neuroprocesador dotado de capacidad de red virtual podrá emular un ANS de mayor número de neuronas que PU disponibles, de modo que cada procesador realiza, en tiempos diferentes, distintas neuronas de la red. Para ello, cada PU deberá disponer de una zona de memoria para almacenar el estado de aquellas neuronas que le hayan correspondido en la asignación, y que en algún momento tendrá que emular. Para gestionar el sistema existirá un software adecuado, similar a un sistema operativo. Dentro de esta categoría de neurocomputadores se incluyen (Figura 5.5) tanto los contruidos alrededor de chips neuronales de propósito general (como CNAPS o la serie SYNAPSE, sección 5.10), como aquellos que toman como base microprocesadores estándar no neuronales (de tipo RISC, como el i860 de Intel, o DSP, como los TMS320 de Texas Instruments). Un ejemplo de este último caso es MUSIC, que puede alcanzar una velocidad punta de 2.7 GFLOPS [Müller 92].

En cambio, un **neuroprocesador de propósito específico**, o simplemente chip neuronal específico (Figura 5.5), estará constituido por PU más simples, pero menos flexibles, aunque podrá incluir un mayor número de ellas. Dentro de esta categoría estarían incluidos los desarrollados en torno a chips neuronales de propósito específico (secciones 5.9 y 5.10).

5.5.2 Aspectos generales de la realización VLSI

El aumento continuo de la densidad de integración hace que la **tecnología VLSI** se adapte muy bien a la implementación de sistemas paralelos, como es el caso de los neuroprocesadores. Algunos de los mayores **problemas de diseño** a la hora de implementar estos sistemas (sean analógicos o digitales) en silicio, son los siguientes [Treleaven 89]:

- a) Los modelos neuronales precisan de altos niveles de conectividad.
- b) Se requiere alta velocidad de procesamiento y de comunicación de datos.
- c) Debe optimizarse la representación de los datos y el almacenamiento de los pesos.

Respecto del primer punto, y debido al problema que representa el emplazamiento y conexonado (*placement and routing*) sobre la superficie plana de silicio, el **alto grado de interconectividad existente entre las neuronas** (en ocasiones globales) limita severamente el número de PU que pueden ser integradas en un único chip. El espacio ocupado por las conexiones o a los problemas asociados a ellas (por ejemplo, retardos), pueden impedir la realización de un determinado circuito. Una solución consiste en optimizar la arquitectura, buscando alternativas al procesamiento masivamente paralelo ideal.

En cuanto a los **pesos sinápticos**, hay que decidir en primer lugar si se almacenarán en el chip o fuera de él, lo que dependerá en buena medida de los requisitos de velocidad y área. La forma de almacenar los pesos depende fundamentalmente de si la implementación es analógica o digital, así como de la resolución en bits requerida por los diferentes modelos neuronales. Muchos de los algoritmos (como el BP) requieren de 10 a 14 bits en la fase de aprendizaje [Baker 89, Castillo 91], y de 8 en la de recuerdo; consúltase [Medrano 98]. El tipo de memoria a utilizar depende de la implementación [Cabestany 93, Pino 93]. En una realización digital, los pesos se almacenan en memorias o registros, con un alto coste en silicio, aunque como contrapartida puede lograrse la precisión deseada. Como veremos, hay varias alternativas en las realizaciones analógicas para el almacenamiento de los pesos; en general, haciendo uso de almacenamiento puramente analógico se tiene una baja ocupación de silicio, pero también la precisión alcanzable es reducida [Pino 93].

Podemos observar que desde el principio surge el dilema clásico de **realización analógica frente a digital**. Si bien los sistemas analógicos pueden lograr mayores densidades de empaquetamiento de PU en un único chip, al ser sistemas cableados, resultan mucho menos flexibles y difíciles de reconfigurar. Por su importancia, este tema será abordado con mayor extensión en la sección siguiente.

El diseño de neuroprocesadores que optimicen paralelismo de cómputo, rendimiento y flexibilidad, precisa **tener en cuenta las siguientes características de la integración VLSI** [Treleaven 89]:

- La complejidad de los diseños favorece el desarrollo de estructuras simples, regulares y replicables.
- El cableado ocupa gran parte del espacio en un circuito integrado.
- Los retrasos en las comunicaciones degradan el rendimiento.
- Las tecnologías MOS, de bajo consumo, requieren concurrencia para lograr muy altos rendimientos.
- Debe haber una perfecta coordinación entre la entrada/salida con los cálculos internos, de modo que el sistema no vea limitado su rendimiento a causa de una entrada/salida lenta.

Teniendo en cuenta las anteriores circunstancias, **un buen diseño VLSI de un neuroprocesador** debería incluir las siguientes propiedades arquitecturales:

- a) Simplicidad en el diseño, que debe conducir a una arquitectura basada en reproducciones de unos pocas celdas simples.
- b) Modularidad que limite los requerimientos de comunicaciones, reduciendo el coste en conexiones. Los dos puntos citados conducen a la realización de una arquitectura compuesta por PU replicables, incluyendo un procesador, comunicaciones locales y también memoria local.
- c) Diseño expandible y escalable, que además de incluir muchas PU en un chip, permita la construcción de un sistema complejo mediante la interconexión de varios.

En la actualidad existen numerosos diseños de neuroprocesador, que varían en cuanto a densidad de empaquetamiento (PU integradas), rendimiento y flexibilidad, dependiendo algunos de estos aspectos de si la implementación se ha realizado de forma analógica o digital. **Los neuroprocesadores de propósito específico sacrifican la generalidad en beneficio de la velocidad, mientras que los de propósito general tienden a ser más lentos, pero permiten trabajar con muchos más modelos.** Por tanto, la orientación del diseño **dependerá del propósito final** del neuroprocesador. Así, si necesitamos un sistema de desarrollo para un amplio número de aplicaciones, lo más adecuado es un neuroprocesador de propósito general. Sin embargo, si se requiere un sistema de tratamiento en bajo nivel de datos sensoriales, tendrá que hacerse uso de un neuroprocesador de propósito específico, probablemente en forma de un ASIC, que reciba la información de los sensores, realice un primer procesamiento y la envíe al resto del sistema.

En cuanto a densidad de empaquetamiento, es decir, número de PU integrables por área, en la actualidad existen realizaciones de hasta algunos cientos de neuronas incluyendo miles de sinapsis (haciendo uso de tecnologías analógicas).

La velocidad de procesamiento, CPS para la fase de ejecución y CUPS para la de aprendizaje, depende de la precisión en los cálculos, de la resolución de los pesos y

de las características de programabilidad del PU concreto. Centrándonos en la fase de ejecución, con neurochips analógicos se llega hasta los GCPS (miles de millones de conexiones por segundo), con resoluciones equivalentes de 8 bits. Con realizaciones digitales se trabaja en el orden de los MCPS (millones de conexiones por segundo), pero con resoluciones más altas, de hasta 32 bits, con mayor programabilidad y con menos dificultades para incluir el aprendizaje.

5.6 BLOQUES BÁSICOS EN LA REALIZACIÓN DE NEUROPROCESADORES DIGITALES

A la hora de abordar la realización electrónica de un modelo neuronal, en primer lugar se debe **escoger la tecnología a emplear**. Si lo que se pretende es realizar un modelo electrónico que se ajuste fielmente al modelo formal, que reproduzca una estructura biológica, o bien un diseño que ocupe la menor área de silicio, la elección se inclinará hacia la **realización analógica**. Por el contrario, si se busca flexibilidad, precisión en los cálculos, o el uso de herramientas de desarrollo mucho más elaboradas, se hará uso de **tecnologías digitales**.

Una vez escogida la implementación electrónica que más se adapte a nuestras especificaciones, debemos seleccionar la estructura de los diferentes bloques básicos que constituyen toda realización electrónica de una red neuronal. Estos componentes básicos o unidades funcionales, adecuadamente ensamblados, conformarán el diseño electrónico.

Los **bloques básicos** de una implementación electrónica, tanto analógica como digital, de una red neuronal son [Avellana 95]:

- 1) Sistema de Control
- 2) Unidad de Proceso
- 3) Unidad de Comunicación
- 4) Unidad de Almacenamiento

En los siguientes apartados profundizaremos en el análisis de estos cuatro bloques básicos, considerados desde la perspectiva del diseño de un neurocomputador digital, dejando la analógica para la sección 5.7.

5.6.1 Sistema de control

El **sistema de control** es el responsable de controlar el flujo de instrucciones y datos que llegan a las unidades de proceso, verdaderos elementos de cálculo.

		FLUJO DE DATOS	
		Simple	Múltiple
FLUJO DE INSTRUCCIONES	Simple	SISD (von Neumann)	SIMD (vector, array)
	Múltiple	MISD (¿pipeline?)	MIMD (múltiples CPUs)

Tabla 5.2 Clasificación de Flynn de las máquinas, atendiendo al tipo de control

Atendiendo al modo de operación del sistema de control, Flynn clasifica las arquitecturas de cómputo en cuatro grandes grupos [Tanenbaum 90] (tabla 5.2):

- 1) **SISD (Single Instruction, Single Data)**. Máquinas en las que una instrucción opera sobre un único dato. Son las clásicas von Neumann secuenciales en la que se basan la mayor parte de los computadores convencionales, que constan de una única CPU.
- 2) **SIMD (Single Instruction, Multiple Data)**. En ellas, una única instrucción se aplica sobre múltiples datos a la vez. El sistema de control está constituido por una única unidad de control que envía la misma instrucción a múltiples procesadores, que la ejecutarán sobre diferentes datos. Un ejemplo son las máquinas vectoriales, en las que una determinada instrucción opera sobre todo un vector de datos.
- 3) **MISD (Multiple Instruction, Single Data)**. En estas estructuras un conjunto de instrucciones operan a la vez sobre un único dato. Surgen de manera natural dentro de la taxonomía de Flynn, pero no está claro si tales máquinas existen, aunque se suele citar como ejemplo la técnica de segmentación (*pipeline*).
- 4) **MIMD (Multiple Instruction, Multiple Data)**. Máquinas en las que múltiples instrucciones actúan sobre múltiples datos. En ellas varias unidades de control envían diferentes instrucciones a distintos elementos de proceso, ejecutándolas con datos locales. Son las arquitecturas masivamente paralelas. Un ejemplo son las matrices de transputers.

5.6.2 Unidad de proceso

La **unidad de proceso o PU (Processing Unit)**, también denominada elemento de proceso, **procesador elemental o PE (Processing Element)**, es el elemento de la arquitectura encargado de efectuar las operaciones aritméticas con los datos que se le suministran. Su estructura depende esencialmente de las tareas que deba ejecutar.

Así, en relación a su **funcionalidad**, la UP puede tener asignadas, además de las tareas de cómputo matemático, las relacionadas con el proceso de comunicación (transferencia de datos entre procesadores) o con el de control (en ocasiones la UP consta de una pequeña máquina de estados que es capaz de ejecutar pequeñas tareas propias del proceso de control).

Por otro lado, y en cuanto al **tipo de operaciones** que puede ejecutar, las operaciones aritméticas involucradas en el proceso de recuerdo o aprendizaje de una red neuronal pueden ser clasificadas [Viredaz 94] como operaciones de frecuencia $O(n)$ (léase “de orden n ”) y de frecuencia $O(n^2)$, siendo n el número de neuronas que constituyen la red neuronal que se pretende emular. Son operaciones de orden n^2 aquellas cuyo número aumenta cuadráticamente con el número de neuronas del modelo. Entre este tipo de operaciones se encuentran las sumas, restas, productos o acumulaciones⁶. Debido a su frecuencia, resultan críticas a la hora de evaluar el tiempo de proceso, siendo relativamente sencillas de implementar sobre tecnología digital, haciendo uso de algoritmos ampliamente desarrollados [Omondi 94].

Por otro lado, se denominan operaciones de orden n aquellas que, no siendo necesario realizarlas con tanta frecuencia, resultan computacionalmente complejas. Es el caso de las funciones de activación neuronales (sigmoides y tangentes hiperbólicas, entre otras), divisiones o raíces cuadradas. La menor frecuencia con que estas operaciones son realizadas en los modelos neuronales hacen que su influencia pueda resultar muy inferior a las primeras, en orden a estimar los tiempos de cálculo de una implementación. Sin embargo, la complejidad de cálculo que implican obliga al empleo de estructuras hardware mucho más sofisticadas, ya que el uso de unidades aritméticas sencillas para elaborar este tipo de resultados implicaría el consumo desmesurado de tiempo de cálculo⁷. Para ello, basta con pensar en el número de operaciones del tipo antes denominado $O(n^2)$ que serían precisas para calcular el valor de salida de una función de activación.

Así, en función de qué tareas deseemos encomendar a la PU, podremos seleccionar entre los **dos tipos básicos de unidades de procesamiento**:

- a) Unidades **CISC** (*Complex Instruction Set Computer*). Son procesadores con un elevado conjunto de instrucciones que incluyen operaciones complejas. Su inconveniente fundamental es la complejidad de su diseño, así como la necesidad de un sofisticado microprograma que traduzca las instrucciones complejas a conjuntos de instrucciones más sencillas, disminuyendo su velocidad de cálculo. Normalmente se escogen unidades CISC cuando se

⁶ En un modelo totalmente interconectado, como el de Hopfield, todas las neuronas que forman la red están conectadas entre sí. Suponiendo n neuronas en la red, el número de conexiones entre ellas es $n \times n$, es decir, n^2 . Cada una de estas conexiones implica un producto peso $\times$ entrada y una suma, operaciones que precisamente hemos definido como $O(n^2)$.

⁷ En el caso de la red de Hopfield de n neuronas, cada una de ellas proporciona una salida procedente de su activación, es decir, la red efectúa n operaciones complejas, de ahí denominarlas de tipo $O(n)$.

desea que la PU realice tareas de control o comunicaciones, o del tipo $O(n)$. Por ejemplo, son de tipo CISC los microprocesadores Intel de la familia 80X86 o los transputers.

- b) Unidades **RISC** (*Reduced Instruction Set Computer*). Estos procesadores se caracterizan por contener un reducido conjunto de instrucciones sencillas, que, como norma general, no consumen más de un ciclo de reloj para su ejecución. Esta simplicidad facilita la incorporación de “facilidades” hardware que aumentan la velocidad de proceso, como la segmentación (*pipeline*). Como inconveniente, la simplicidad de su conjunto de instrucciones obliga a disponer de un compilador relativamente más sofisticado que sea capaz de traducir a instrucciones comprensibles por el procesador las tareas más complejas. Como ejemplo de microprocesadores RISC convencionales, podemos citar los incluidos en las estaciones de trabajo, como SPARC, ALPHA o PowerPC. En la realización de ANS habitualmente se escoge este tipo de procesador para PU que realizan exclusivamente tareas de tipo $O(n^2)$. En general, las PU diseñadas expresamente para su aplicación a la emulación de ANS son de tipo RISC.

5.6.3 Unidad de almacenamiento

Es la unidad funcional responsable de alojar los valores de pesos, resultados intermedios o errores, para ser suministrados en el momento adecuado a la unidad de proceso. La mayor parte se encuentra ocupada en almacenar los valores de los pesos asociados a las neuronas, por lo que se la suele llamar memoria de pesos. La unidad de almacenamiento, o memoria de pesos, puede adoptar tres estructuras básicas, que pasamos a describir.

En primer lugar, en una **memoria de pesos distribuida** cada PU dispone de su propia unidad de memoria local, donde se encuentran almacenados los pesos propios de las neuronas virtuales que se encuentran proyectadas sobre él. Si bien es la configuración ideal cuando la información que cada PU debe elaborar es de origen completamente local, presenta serias restricciones cuando las operaciones que la PU debe realizar implican el uso de datos procedentes de otros procesadores.

Este último es precisamente el caso del modelo neuronal más habitual, el perceptrón con aprendizaje BP: cuando la red se encuentra realizando una operación de recuerdo, cada procesador hace uso exclusivamente de los datos alojados en su memoria local, pero cuando en fase de aprendizaje propaga hacia atrás el error, cada PU requiere el uso de datos pertenecientes a la memoria local de otras PU, lo que presenta una dificultad para este tipo de almacenamiento de información.

Este problema puede resolverse si cada memoria local dispone de una copia de los pesos que necesita tanto en la fase de recuerdo como en la de aprendizaje. Ello implica que en cada paso del aprendizaje, en los cuales los pesos son modificados y

actualizados, cada procesador debe suministrar una copia de sus pesos a las unidades que lo requieran para seguir propagando el error, consumiendo de esta forma recursos de comunicaciones, que pueden llegar a ser el cuello de botella de todo el sistema. Una solución a esto último consiste en que cada neurona suministre directamente la parte de corrección debida a los pesos que contiene en relación con el resto de las neuronas, debiendo transmitir así un menor número de datos, de forma que cada procesador emplee el menor tiempo posible el sistema de comunicaciones, reduciendo los retardos asociados a este proceso.

Un segundo esquema denominado **memoria de pesos duplicada**, consiste en que cada unidad de memoria asociada a cada unidad de proceso dispone de una copia completa de todo el mapa de pesos de la red. Esta estructura es poco empleada debido al elevado consumo de tiempo que las tareas de comunicaciones implican.

Un tercer y último esquema, la **memoria de pesos centralizada**, consiste en que una única unidad de almacenamiento proporciona los datos necesarios a todas las unidades de procesamiento, de forma que cada una de ellas tiene acceso a todos los pesos. Debido a que el acceso a los datos contenidos en la memoria se produce de forma secuencial, se produce un cuello de botella asociado a las comunicaciones que ralentiza sustancialmente la velocidad de proceso del emulador.

5.6.4 Unidad de comunicación

Se encarga de la transmisión de datos entre las unidades de proceso (conexiones entre las salidas de una neurona y las entradas de otras, transmisión de correcciones de pesos, etc.), entre PU y memorias de almacenamiento, o entre las PU y las unidades de control. La unidad de comunicación viene **caracterizada por la topología de interconexión de los buses** de comunicaciones, pudiéndose distinguir **tres arquitecturas diferentes**: orientadas a bus, sistólicas y fractales.

1) Arquitecturas orientadas a bus o BBA (*Broadcast Bus Architecture*, Figura 5.8 a). Las PU se conectan entre sí por medio de buses en configuración *broadcast*, es decir, todas ellas se encuentran conectadas a uno o varios buses generales que las comunican entre sí, y a través de los cuales se transfieren todos los datos, direcciones y operaciones. Habitualmente una configuración del módulo de comunicaciones de este tipo implica el empleo de PU que incluyen, a diferencia del resto de las arquitecturas, un banco interno de registros para el almacenamiento de los pesos. Es ésta la estructura de neuroprocesador más general y una de las más interesantes, debido a su modularidad, versatilidad y fácil escalabilidad, aunque se ve penalizada cuando los pesos sinápticos no caben en la memoria local de la PU, lo que obliga a realizar frecuentes intercambios de datos con la memoria externa.

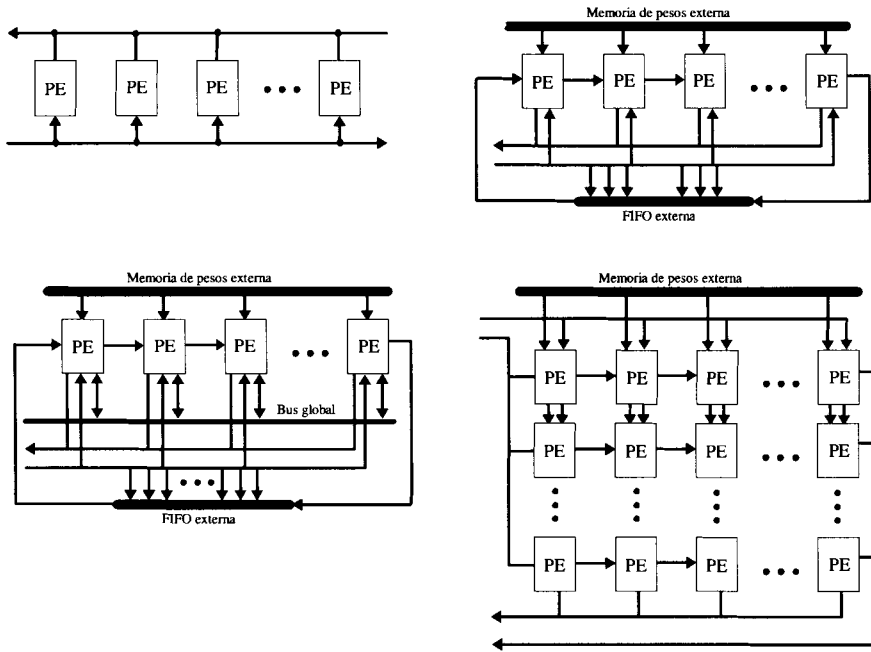


Figura 5.8. Diversas arquitecturas de Unidad de Comunicaciones: a) arquitectura orientada a bus, b) sistólica SRA, c) sistólica SRAGB y d) sistólica SSAA [Ienne 93]

2) Arquitecturas sistólicas. Son diseños más o menos sofisticados, derivados de las implementaciones sistólicas de multiplicaciones matriz-vector. Se caracterizan porque cada procesador se encuentra, en general conectado exclusivamente a procesadores físicamente próximos a él (Figuras 5.8 b-d), de forma que la transmisión de datos a PU alejadas implica su transferencia a través de procesadores vecinos. Entre otras cosas, las PU tienden a ser más simples que en el caso anterior y no incluyen los pesos sinápticos. Se denominan arquitecturas sistólicas porque tras un período de latencia inicial los resultados de las operaciones (del tipo matriz $\times$ vector) van surgiendo de los procesadores en cada ciclo de reloj. Dentro de esta clasificación podemos encontrar:

- Arquitecturas en anillo sistólico o SRA** (*Systolic Ring Architecture*, Figura 5.8 b). Cada PU se encuentra conectada con otras dos, formando un anillo de procesadores. Los datos con los que opera proceden de dos fuentes: una el procesador anterior en el anillo y la otra un bus que conecta los procesadores a la unidad de memoria donde se almacenan los pesos.
- Arquitecturas en anillo sistólico con bus global o SRAGB** (*Systolic Ring Architecture with Global Bus*, Figura 5.8 c). Se obtiene de la SRA añadiendo un bus global que permite el envío de datos de un procesador a cualquier otro en un único ciclo de reloj. Así se facilita, por ejemplo, la realización de una suma global a partir de las sumas parciales realizadas por diferentes PU.

- c) **Arquitecturas en anillo sistólico múltiple o MSRA** (*Multiple Systolic Ring Architecture*). Se compone de varias SRA en paralelo, alimentadas con la misma memoria de pesos. Su propósito es obtener una mejora en el rendimiento mediante el procesamiento de varios patrones a la vez.
- d) **Arquitecturas de matriz sistólica cuadrada o SSAA** (*Systolic Square Array Architecture*, Figura 5.8 d). Se trata también de una estructura sistólica cuyas PU se ordenan en forma de matriz cuadrada. Procede también de una arquitectura clásica de multiplicación de matriz por vector, considerándose como una de las más interesantes al aprovechar muy eficientemente el paralelismo de cómputo.
- e) **Arquitecturas de anillo sistólico de matrices de procesadores o SRMPA** (*Systolic Ring of Matrix Processors Architecture*). Es similar al SRA, pero el flujo y manipulación de datos se realiza sobre matrices, en vez de sobre escalares, que se construyen concatenando diversos vectores de entrada.

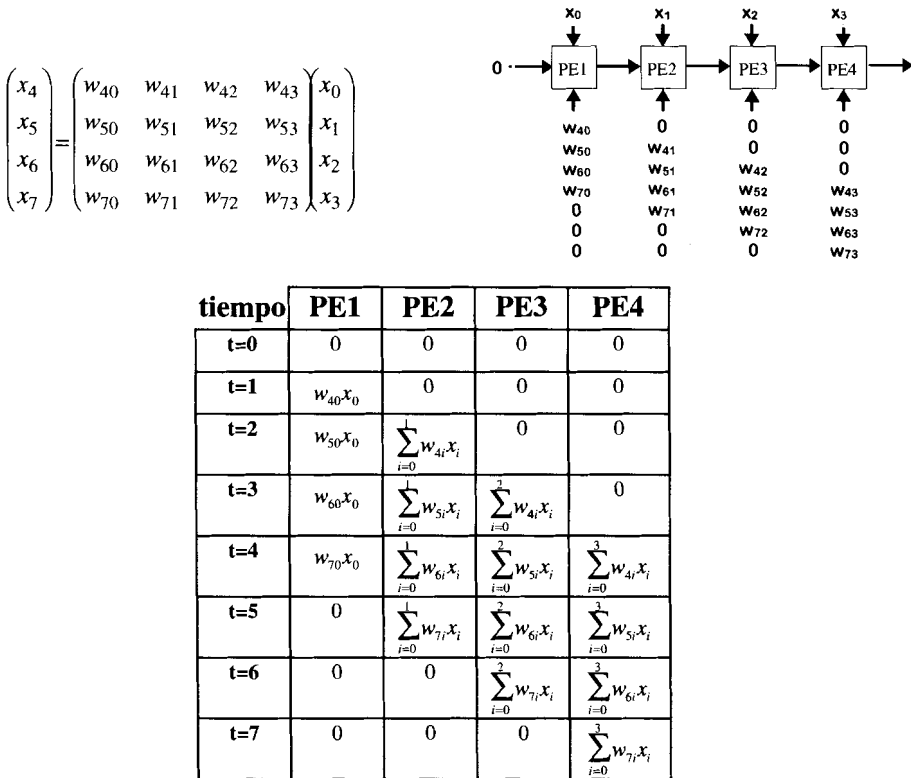


Figura 5.9. Dispositivo sistólico, donde cada procesador acumula la salida del anterior en la cadena con el resultado de su operación. Representación matricial y del flujo de datos entre las PU (arriba), y tabla temporal de sus estados (abajo)

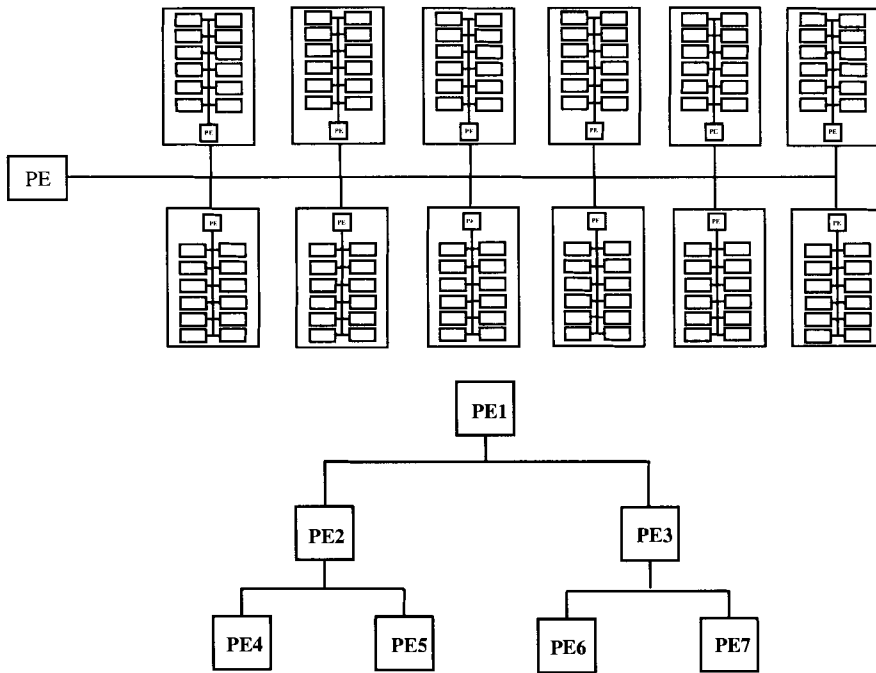


Figura 5.10. Arquitectura de comunicaciones fractales (la estructura se repite hacia abajo en topología fractal), y esquema simplificado (árbol). Una tarjeta neuro-emuladora conectada a un host respondería a esta estructura: PE1 es el host, PE2 y PE3 las unidades de control, que gestionan los procesadores PE4, PE5, PE6 y PE7

Este tipo de arquitecturas de comunicaciones se caracteriza porque todos los procesadores se ven implicados en el cálculo de un producto matriz-vector (Figura 5.9), por lo que en ocasiones es preciso esperar a que todos concluyan el proceso que están realizando para emprender otros cálculos. Debido a que el último procesador de una estructura sistólica necesita los cálculos efectuados por las PU situadas en posiciones anteriores en la cadena de comunicaciones, éstos permanecerán inactivos mientras el último no concluya sus operaciones, lo que supone una pérdida de rendimiento de la arquitectura. En [Kung 93a, 93b] se realiza un amplio estudio sobre formas de simular eficientemente redes neuronales en arquitecturas sistólicas (cómo asignar las neuronas a las PU, maneras eficientes de efectuar los intercambios de datos, etc.), proponiéndose para ello una metodología sistemática.

3) Arquitecturas fractales o de jerarquía en árbol (Figura 5.10). En ellas, los procesadores se conectan entre sí adoptando configuraciones en forma de árbol: un conjunto de procesadores, pertenecientes a un mismo nivel, están a su vez conectados a un conjunto de procesadores de un nivel inferior (nodos hijos), conectados exclusivamente a ese procesador padre, de forma que los datos procesados por las PU

de un nivel pasan siempre por su nodo padre, de un nivel superior en la jerarquía, para comunicarse con otras ramas del sistema. Si bien es una arquitectura de cierto atractivo teórico (permite una gran escalabilidad, comunicaciones de tipo local, etc.), no tenemos noticia de neuroprocesadores que hayan sido diseñados expresamente con esta topología de interconexión, aunque la mayoría de los que efectúan sus procesos dependiendo de un host (un PC o una estación de trabajo) son, de alguna manera, versiones simples de una arquitectura de estas características.

5.6.5 Arquitecturas reconfigurables

Mención aparte merecen las realizaciones digitales de neuroprocesadores basadas en la aplicación de dispositivos con la capacidad de modificar sus estructura hardware, adaptándola a los requisitos de la aplicación a la que se desea realizar.

Pertenecientes a la misma familia de dispositivos que las PLD, PROM, PAL o EPLD, las **FPGA (Field Programmable Gate Array)** son elementos programables de elevado número de puertas (pueden integrar millones), constituidos por una matriz de celdas lógicas o **CLB (Configurable Logic Block)**, altamente interconectables a través de una malla de conexiones. Cada una de estas celdas contiene tanto lógica combinacional como biestables (Figura 5.11), proporcionando así la capacidad de generar lógica secuencial. El interés que presentan estos circuitos lógicos reside en la posibilidad de configurarlos adecuándolos a los requisitos del usuario. Debido al elevado número de puertas integradas en un dispositivo de este tipo, y a su reducido coste económico y de tiempo de desarrollo, podemos encontrar gran variedad de aplicaciones en las que las FPGA cumplen un papel importante, desde el control de un bus PCI, hasta el control de navegación reconfigurable en misiles [Tavernier 94].

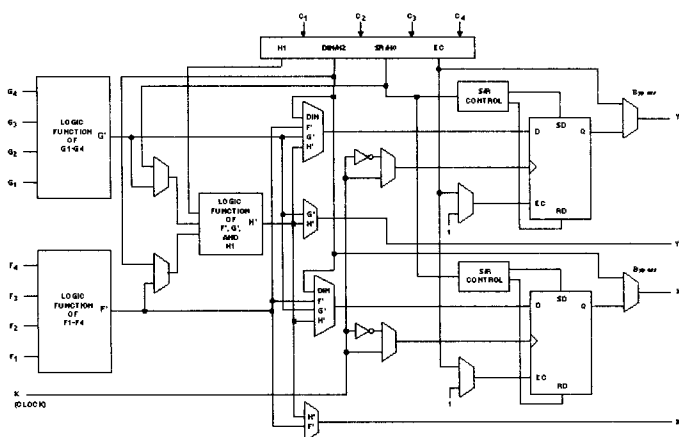


Figura 5.11. Estructura básica de un CLB de las FPGA de la compañía Xilinx

Dentro de este tipo de dispositivos lógicos programables, podemos encontrar a su vez dos variantes distintas: las FPGA de antifusibles y las FPGA de tipo RAM, comúnmente llamadas **LCA (Logic Cell Array)**. Mientras que en las primeras la estructura de operación interna se programa mediante la eliminación de fusibles, resultando un proceso irreversible, la interconexión entre los componentes lógicos de una LCA viene determinada por una memoria RAM interna, reprogramable tantas veces como sea necesario.

Esta reprogramabilidad de las LCA hace de ellas un elemento atractivo a la hora de implementar dispositivos cuya configuración requiera ser modificada en algún momento, como es el caso de los **computadores reconfigurables**. Estos dispositivos son sistemas de procesamiento cuyo hardware puede ser modificado, adaptándolo a los problemas que debe resolver. Por ejemplo, los denominados procesadores digitales de señal reconfigurables [Klein 95] son capaces de modificar su hardware, acondicionándolo a los requisitos de la señal que se debe tratar. Este tipo de arquitecturas permiten obtener rendimientos típicos de supercomputadores a un costo considerablemente inferior, ya que la estructura interna del sistema de procesamiento se ajusta en cada momento al problema. Tal es el caso del sistema de computación reconfigurable presentado en [Xilinx 95a]. Una muestra de diferentes realizaciones de elementos de computación reconfigurables basados en FPGA puede encontrarse en [Xilinx95b].

Existen ya en la bibliografía referencias relativas a la aplicación de estos dispositivos programables a la implementación de neuroprocesadores [Cox 92, Botros 94, MicroNeuro 96, Medrano 98]. Su reconfigurabilidad es aprovechada en la mayoría de las ocasiones para optimizar operaciones de multiplicación, creando tablas internas (*look-up tables*) que contienen los resultados de operar cualquier vector de entrada (patrón) por un conjunto limitado de valores de pesos, obtenidos previamente en un proceso de entrenamiento realizado exteriormente (en un simulador). La rapidez de acceso a las tablas hace que este tipo de multiplicadores sea sensiblemente más rápido que los tradicionales, aun a costa de perder parte de su flexibilidad. Sin embargo, tales desarrollos no aprovechan a fondo la capacidad de reconfigurabilidad de las FPGA tipo RAM. Una alternativa ya planteada sería el aprovechamiento de las posibilidades que las LCA ofrecen, adaptando el hardware a diferentes precisiones (8, 12, 16, 32 bits) según los requisitos del problema, diferentes tipos de operación, e incluso diferentes arquitecturas de la unidad de comunicaciones, aprovechando los recursos de conectividad con que estos dispositivos cuentan. Propuestas orientadas en esta línea han sido planteadas en [Hernández 96, Martín del Brío 98, Medrano 98, MicroNeuro 96].

5.6.6 Realizaciones especiales: lógica de frecuencia de pulsos

La gran mayoría de las implementaciones digitales hacen uso de diseños de unidades aritmético-lógicas más o menos sofisticadas, que realizan las operaciones

propias de la unidad de proceso. Sin embargo, algunos diseños digitales se hallan inspirados en el modo en el que el sistema nervioso codifica y procesa los pulsos eléctricos que llegan a las neuronas a través de sus axones (capítulo 1). Éste es el caso de los basados en lógica de pulsos, cuyo fundamento es la **codificación de la información en la frecuencia de los trenes de pulsos** que viajan por las líneas de datos, en lugar de codificarla como números binarios. Éste tipo de procesamiento presenta determinadas características de las que carecen los sistemas digitales convencionales, como la simplicidad y rapidez en la realización de multiplicaciones, sumas y funciones de activación de tipo sigmoide. A modo de ejemplo, en la Figura 5.12 se muestra el modo en que la lógica de frecuencia de pulsos es capaz de efectuar productos de trenes de señales, así como operaciones de tipo no lineal.

Este tipo de realizaciones presentan un gran interés desde el punto de vista de realizar sistemas artificiales que emulan con mayor fidelidad la estructura del sistema nervioso de los seres vivos; además, se trata de diseños expandibles. Un trabajo pionero en cuanto a implementación hardware es el de [Murray 89], Hitachi desarrolló el neurocomputador **Neuro-WSI** [Masaki 90]; finalmente, en [IEEE 99c] pueden encontrarse trabajos recientes y diversas implementaciones hardware modernas.

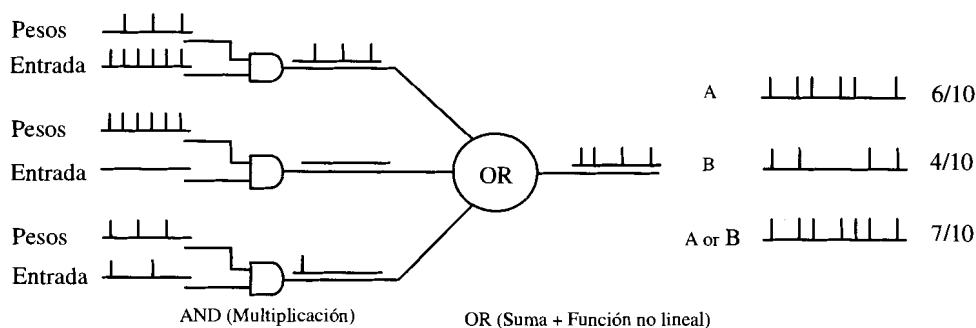


Figura 5.12. Neurona de lógica de pulso y ejemplo de la función no lineal realizada por una simple puerta OR

5.7 BLOQUES BÁSICOS EN LA REALIZACIÓN DE NEUROPROCESADORES ANALÓGICOS

Como hemos dicho anteriormente, el diseño electrónico de neuroprocesadores basados en tecnología analógica está especialmente indicado cuando lo que se pretende es realizar una implementación que no sólo reproduzca las capacidades computacionales del modelo neuronal, sino también **la arquitectura física de la red**. Por las características propias de la tecnología, si bien se puede hablar de los cuatro

bloques básicos ya descritos (unidad de control, de comunicaciones, procesador elemental y almacenamiento de datos), no existe una clasificación para los dos primeros. La tendencia a realizar físicamente la estructura del modelo neuronal **mediante transistores** hace que no se requiera una clasificación definida para las **comunicaciones**: si lo que se diseña es un emulador de red de Hopfield, cada PU estará directamente comunicada con todos los demás procesadores; si es una realización del perceptrón, cada unidad de proceso conectará directamente con todos los procesadores de la capa siguiente, y sólo con ellos. El hecho de que la transmisión de los datos entre procesadores se realice a través de una o dos líneas de datos facilita enormemente la tarea, pues el consumo de área de silicio para estos fines es sensiblemente inferior al caso de la realización digital⁹.

De igual forma, la manera de operar de la **unidad de control** dependerá del modo en que las neuronas del modelo formal se conecten y efectúen sus tareas; muy a menudo su operación será de tipo asíncrono, como sucede en las neuronas biológicas o como estudiamos en el capítulo 4 en el caso del modelo de Hopfield analógico.

A continuación analizaremos las características esenciales de las dos unidades restantes, procesamiento y almacenamiento.

5.7.1 Unidad de proceso

En implementaciones analógicas, el estudio de la unidad de proceso se centra en especial en el diseño de estructuras que realicen las operaciones matemáticas propias del modelo, en general, productos, sumas y funciones de activación. Como es bien sabido, las operaciones de adición o producto son sencillas de realizar. En el caso de la **suma**, la convergencia de dos corrientes que circulen por sendos hilos hacia un tercero hace que por éste circule la suma de ambas corrientes. Existen también sumadores especiales, como el de la Figura 5.13, llamado sumador de Kirchhoff.

En el caso de **productos**, la intensidad que circula por una resistencia no es más que el producto del voltaje entre sus extremos por su conductancia. Sin embargo, este método de realizar multiplicaciones adolece de flexibilidad, pues el valor de uno de los multiplicandos, dado por una resistencia, es fijo.

Existen diversos dispositivos diseñados para la realización del producto de dos valores representados como magnitudes eléctricas, bien tensión $\times$ tensión, intensidad $\times$ intensidad o tensión $\times$ intensidad. Uno de los más conocidos y empleados es el multiplicador de transconductancia de Gilbert (Figura 5.14), que es capaz de proporcionar el resultado de multiplicar dos valores de tensiones de cualquier signo, en la forma de una intensidad de salida.

⁹ En el caso de una implementación digital, la transmisión de un dato debe realizarse bien por un bus, de tantas líneas como bits de precisión se estén empleando, bien en serie, un bit tras otro. Desde luego, este sistema es mucho más lento, ya que precisa de diseños que gestionen con efectividad los recursos de comunicaciones de la implementación.

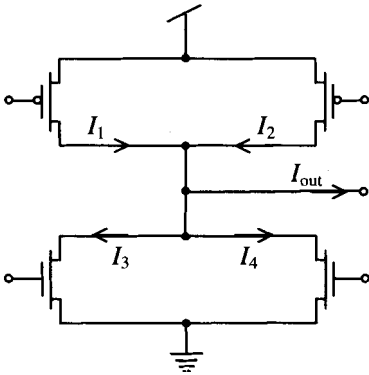


Figura 5.13. Sumador de Kirchhoff. Las corrientes que circulan por los transistores se suman $I_{out}=(I_1+I_2)-(I_3+I_4)$

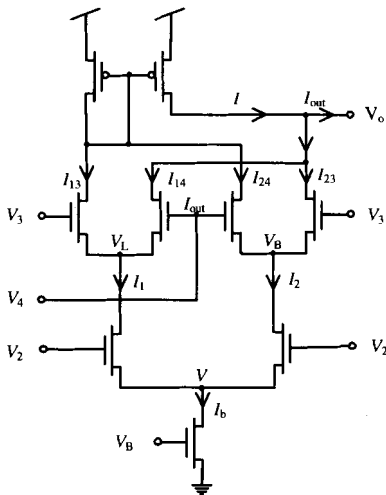


Figura 5.14. Multiplicador de transconductancia de Gilbert. Arriba, modelo de transistores; abajo, expresión de la intensidad de salida como producto de tensiones. En el rango lineal (la zona propia del multiplicador) $\tanh(x)=x$

$$I_{out} = I_b \cdot \tanh \frac{\kappa(V_1 - V_2)}{2} \cdot \tanh \frac{\kappa(V_3 - V_4)}{2}$$

Otros dispositivos analógicos, con el fin de reducir la complejidad del elemento multiplicador, realizan el producto de cuatro cuadrantes en dos partes: primero, proporcionan dos magnitudes de salida (que casi siempre son intensidades), de forma que una corresponde a valor positivo y otra a negativo. En una segunda parte un bloque sumador se encarga de sumar las dos intensidades, dando como resultado una intensidad positiva o negativa, según el caso. En [Lehman 93] se presenta uno de estos modelos de multiplicador (llamado multiplicador sináptico), que, compuesto por cuatro transistores (Figura 5.15 a), y conectado a las entradas de un convector de intensidad, es capaz de proporcionar una intensidad de salida proporcional al producto de las dos tensiones presentes en sus entradas (Figura 5.15 b).

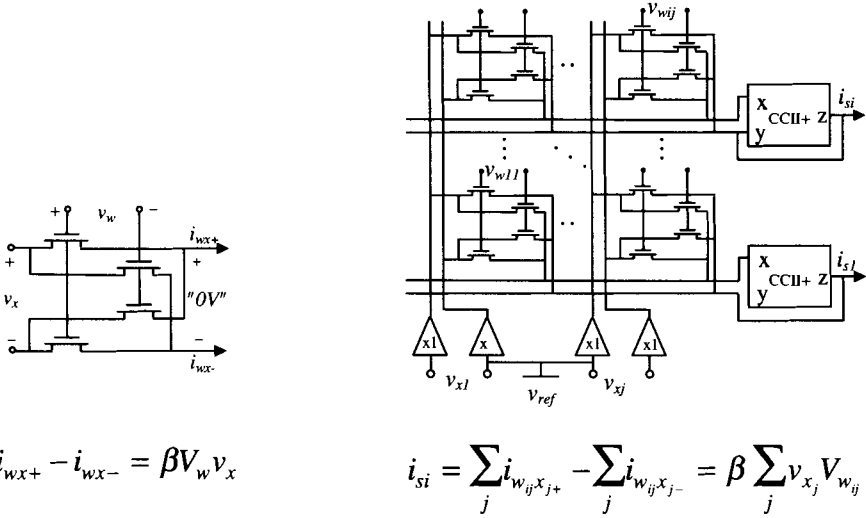


Figura 5.15. Multiplicador sináptico de cuatro cuadrantes. a) Modelo de cuatro transistores del multiplicador (las intensidades i_{wx+} e i_{wx-} son proporcionales al producto de las dos tensiones). b) La segunda etapa, constituida por convectores de corriente, suma las intensidades dando el valor final con su signo

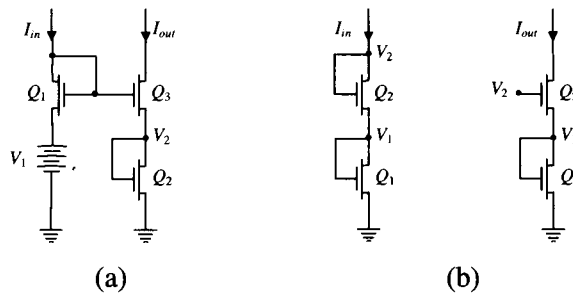


Figura 5.16. Diversos dispositivos electrónicos para la realización de operaciones no lineales. a) Generador de la raíz cuadrada del valor dado por una intensidad. b) Generador de una tensión proporcional al logaritmo de una intensidad de entrada (izda.), y de una intensidad proporcional a la exponencial de una tensión (decha.)

La **función no lineal** necesaria en la implementación de la activación neuronal puede realizarse empleando un amplificador operacional que suma los productos realizados por una etapa previa. Sin embargo, la construcción de circuitos con transistores capaces de efectuar no-linealidades es relativamente simple, con tal de aprovechar precisamente esa característica consustancial a los transistores. En la Figura 5.16 se muestran diferentes realizaciones de dispositivos con salida logarítmica, exponencial y raíz n -ésima de la señal de entrada al circuito.

5.7.2 Unidad de almacenamiento

Otro aspecto importante a la hora de diseñar un neuroprocesador analógico es la manera de **almacenar los parámetros de la red**, en concreto los **pesos** asociados a cada neurona. Si en el caso del diseño digital la preocupación estriba en cómo acceden los procesadores a la memoria de almacenamiento, clasificando en función de este acceso los distintos tipos de unidad de memoria, en el caso analógico la cuestión reside en el **soporte físico** en el que se almacenarán los datos. El método más simple es la implementación en silicio de **resistencias**, cuyo valor esté relacionado con los pesos sinápticos de cada neurona. Como se ha comentado en el apartado anterior, una tensión aplicada entre sus extremos producirá una corriente proporcional al producto del valor de su admitancia por la tensión aplicada. Sin embargo, este método presenta serias limitaciones: las resistencias ocupan una gran área de silicio, incrementando considerablemente el tamaño del circuito, y no ofrecen demasiada precisión. Además, la dificultad de modificar su valor hace que carezcan de la flexibilidad necesaria para modificar la respuesta del circuito ante cambios en el problema para el cual se ha diseñado, precisando la creación de un nuevo circuito integrado, con diferentes valores de resistencias.

Para evitar estas limitaciones se emplean diferentes técnicas de almacenamiento de pesos que permiten su modificación, dotando al circuito de mayor **flexibilidad**. Los tres métodos más empleados para almacenar datos son dispositivos CCD, condensadores, y el empleo de memorias PROM.

Los **dispositivos CCD** (*Charge Coupled Devices*) son, en esencia, líneas de retardo o registros de desplazamiento dinámicos analógicos, en los que los datos son representados por paquetes de carga eléctrica. Las cargas son inyectadas sobre un sustrato semiconductor por métodos eléctricos u ópticos, pudiendo ser almacenadas o transferidas como en un registro. Mediante la adecuada aplicación de unos pulsos de reloj (Figura 5.17), los pozos de potencial que almacenan dentro del semiconductor la magnitud analógica son desplazados a lo largo del sustrato, pudiendo ser mantenidos en zonas cerradas en forma de anillos (a la manera de un registro), o bien pudiendo ser transferidos a otros dispositivos para su procesamiento.

En la Figura 5.17 se muestra la operación de un CCD. Inicialmente la magnitud que se pretende almacenar es introducida en el semiconductor por la aplicación de un potencial sobre el terminal *in* (izquierda). Para mantener el valor almacenado son aplicados secuencialmente sobre sectores adyacentes del canal de almacenamiento del semiconductor una serie de pulsos (en el ejemplo, los transmitidos a través de las líneas V_1 , V_2 y V_3) parcialmente superpuestos en el tiempo. La aplicación de estas tensiones hace que el pozo de potencial que almacena la carga se desplace, de forma que en la salida de esa línea (derecha) aparecerá el valor de la carga un cierto tiempo después. Si se desea mantener ese dato, la salida será conectada a la entrada a través de un circuito de refresco, formando un bucle cerrado que mantiene el valor contenido en el dispositivo hasta que sea necesario. Los CCD no sólo son empleados para

almacenamiento de datos, sino que pueden emplearse como elementos de proceso. En [Ramacher 91b] se muestra cómo realizar con CCD puertas lógicas, líneas de retardo y memorias de paquetes de carga.

Otro método de almacenar de forma analógica datos es como la tensión en los extremos de un **condensador**. En la Figura 5.18 a se muestra un sistema de muestreo y almacenamiento por condensador. Durante el proceso de muestreo ($W=1$, V_G es positivo) cualquier diferencia entre V (derecha) y V_{in} (izquierda) es corregida mediante la corriente que circula hacia el condensador a través del transistor T_S , resultando de ello que la nueva tensión entre los bornes del condensador sea V_{in} . En el periodo de almacenamiento el transistor se mantiene abierto ($W=0$). La unión D_2 del transistor es mantenida prácticamente a 0V gracias a la tierra virtual debida a G_{m2} , reduciendo considerablemente las posibles fugas. Además, durante este período el interruptor S_2 se mantendrá cerrado para desviar a tierra posibles corrientes generadas por G_{m1} que podrían activar la unión D_1 del transistor, desvirtuando el valor almacenado. La precisión de este sistema viene limitada por el valor de la tensión de *offset* ΔV_1 , y por las pérdidas del condensador.

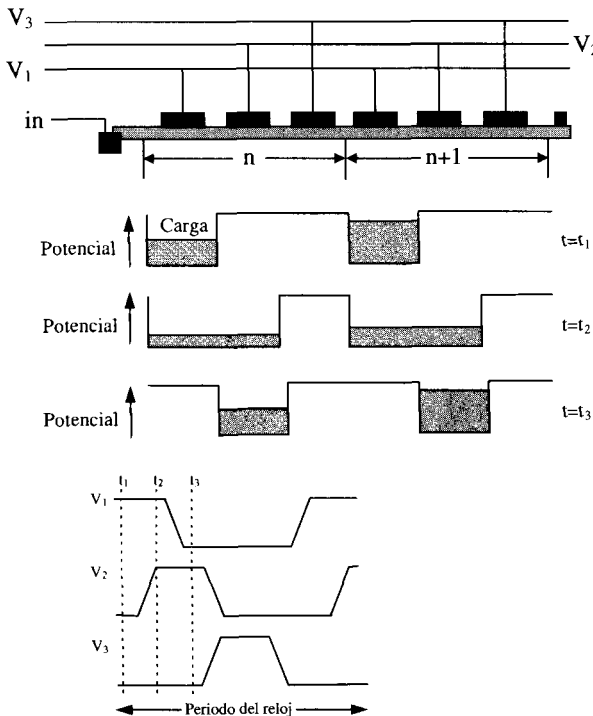


Figura 5.17. Dispositivos CCD. Tres líneas envían pulsos desfasados que transportan la carga por el sustrato (previamente inyectada en él) manteniéndola en su interior. Cerrando el circuito se tiene un registro que almacena dinámicamente la carga

Una versión más sofisticada de este método de almacenamiento de datos, ideado para períodos de tiempo superiores, es el mostrado en la Figura 5.18 b. Similar al presentado anteriormente, consta además de un circuito local de refresco que regenera el valor almacenado en el condensador mediante comparaciones periódicas con unos valores cuantificados. La señal de comparación es suministrada a todos los circuitos de refresco a través de una única línea (V_q). El valor almacenado en el condensador C , representado por la tensión V , es constantemente comparado con los valores escalonados de la línea V_q . En el momento en que V es menor que el valor de comparación (en la figura, a la derecha), se cierra el interruptor S , cargando el valor en el condensador. En el momento en que la tensión del condensador es igual a la de comparación, el interruptor S se abre, hasta el próximo muestreo.

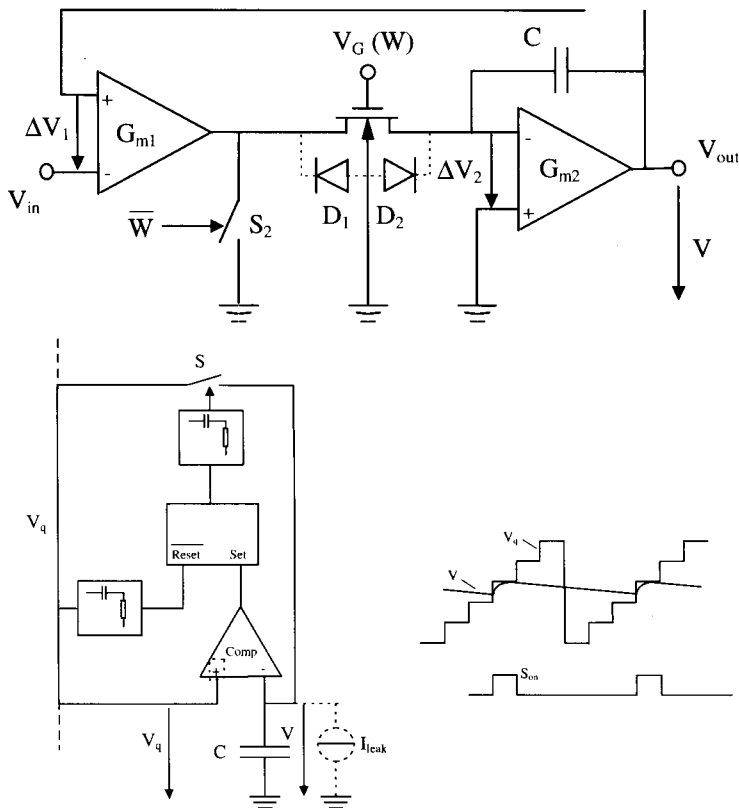


Figura 5.18. Almacenamiento de pesos por carga en condensadores. Arriba, sistema de almacenamiento de medio plazo; el valor almacenado degenera con el tiempo, como consecuencia de las pérdidas del condensador y las tensiones de offset de los elementos activos. Abajo, más sofisticado, el sistema de muestreo y almacenamiento refresca de forma dinámica el valor de la tensión en bornes del condensador

Por último, hablaremos del **almacenamiento en memorias PROM**, que estrictamente no es una realización analógica, perteneciendo más bien al diseño de dispositivos mixtos analógico-digitales. En este caso los valores de los pesos son almacenados como valores binarios en memorias PROM, siendo convertidos posteriormente mediante conversores D/A en valores analógicos (corrientes o tensiones) para su utilización en la red neuronal. En este tipo de dispositivos de almacenamiento de pesos se garantiza especialmente la conservación de los valores que se guardan. Por otro lado, la superficie en silicio ocupada por la unidad de almacenamiento se hace considerablemente grande frente a la requerida por cualquiera de las otras dos técnicas mencionadas, siendo además necesaria la inclusión del sistema de conversión de magnitud digital a analógica.

5.8 ¿REALIZACIÓN ANALÓGICA O DIGITAL?

Desde mediados de los años ochenta se han desarrollado multitud de chips neuronales, tanto analógicos como digitales, y cada estilo parece ser interesante en un campo de aplicación particular. Para seleccionar un tipo de realización se deben considerar diferentes aspectos técnicos, como son el del almacenamiento y transferencia de datos, la velocidad y precisión en bits alcanzables, la adaptabilidad o capacidad de aprendizaje, programabilidad, etc. [Kung 93a].

Pros y contras de la realización analógica

Una neurona puede implementarse fácilmente utilizando **dispositivos analógicos, aprovechando las leyes físicas para la realización de cálculos** [Mead 89a], como sucedía en las antiguas computadoras analógicas. Así, la suma ponderada de la neurona surge de manera directa de la aplicación de las **leyes de Kirchhoff** como suma de corrientes. Además, las características no lineales de los dispositivos facilitan la realización de funciones de activación de tipo sigmoideo. Por tanto, inicialmente la realización analógica parece presentar importantes ventajas.

Un posible esquema de una neurona analógica simple se muestra en la Figura 5.19, donde el soma se representa mediante un amplificador operacional, y los pesos sinápticos mediante resistencias. La realización de la **suma ponderada** es muy simple, pues la tensión de salida V_i de la neurona analógica i en la zona de operación lineal del dispositivo es la siguiente

$$V_i = \sum_k [-G_{ik} R_f] x_k - [-T_i R_f] \quad (5.1)$$

siendo x_k tensiones (entradas de la neurona), G_{ik} conductancias ($-G_{ik} R_f$ representan los pesos), y T_i es una conductancia adicional que se asocia al umbral. La función de activación es aproximadamente sigmoidea, debido al efecto de saturación que aparece en los extremos de la zona lineal de operación del amplificador. Este esquema tan

simple permite trabajar solamente con pesos negativos; para utilizar pesos positivos basta con emplear configuraciones no inversoras [Zurada 92 a].

Aunque este circuito ilustra la forma de realizar una neurona analógica, presenta notables limitaciones, como operar con una única función de activación, excesivo consumo de potencia, y la gran cantidad de área ocupada por el dispositivo y su red de resistencias, que además no son ajustables. En muchas ocasiones, como ya hemos visto, no es necesario recurrir a una estructura tan compleja como un amplificador operacional, sino que bastan estructuras más sencillas, como amplificadores de transconductancia [Tank 91], lo que permite incluir un elevado número de neuronas en un chip. En [Zurada 92 a, b] se indica una forma de implementar resistencias que limiten el área consumida y posean una cierta programabilidad; por ejemplo, se pueden obtener resistencias ajustables, controlables por una tensión mediante un transistor MOS operando en la región resistiva.

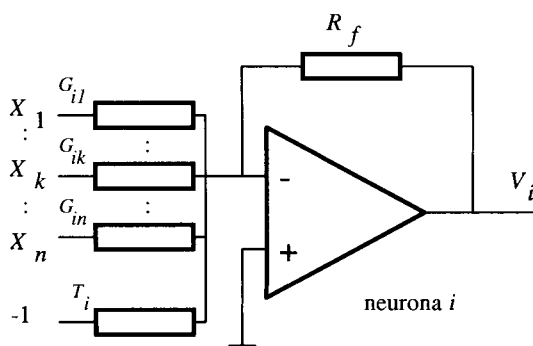


Figura 5.19. Modelo sencillo de neurona analógica basada en un amplificador operacional

Por otra parte, merece la pena señalar que el procesamiento de señal en **modo corriente** presenta ventajas interesantes para la realización de circuitos neuronales analógicos, como la facilidad para realizar la suma de corrientes, el aumento en el rango dinámico, incremento del ancho de banda, etc. [Bibyk 90].

Los circuitos analógicos, en general, presentan la interesante característica de permitir el procesamiento de más de 1 bit por transistor [Pino 93], así como proporcionar una velocidad de proceso mayor que la de los circuitos digitales. Además, en algunos modelos neuronales la operación asíncrona de los circuitos analógicos conduce a dinámicas cualitativamente diferentes a los proporcionados por los digitales síncronos [Hopfield 85].

Por otra parte, comentaremos también algunos **aspectos negativos** de la realización analógica. Aunque ésta sea muy adecuada para la implementación de los modelos de redes neuronales más próximos a la biología, su adecuación a la realización de modelos conexionistas más formales es bastante más cuestionable. Por ejemplo, y comparándolos con los circuitos digitales, los analógicos son más

susceptibles al ruido en la señal, ruido cruzado, efectos térmicos, variaciones en la tensión de alimentación, etc. Además, aunque el almacenamiento de pesos sinápticos no volátiles proporciona en el campo analógico una alta densidad, éstos no resultan fácilmente programables, y se necesita una mayor área de silicio a medida que se requiere una mayor precisión. A este respecto, la precisión proporcionada por el procesamiento analógico está limitada normalmente a 8 bits, lo que resulta inadmisiblemente para muchos algoritmos de aprendizaje (como el BP). En cuanto a la potencia disipada, un bajo consumo está asociado siempre a resistencias elevadas y, por tanto, a una mayor ocupación de área de silicio. Por otra parte, si se utilizan capacidades conmutadas o, en general, en circuitos con resistencias y condensadores, limitar el ruido supone reducir la superficie de transistores y capacidades. En resumen, en las realizaciones analógicas la combinación de los factores de precisión, ruido y consumo lleva a una ocupación de área mayor de la que se supone en principio [Kung 93b].

Con todo, el mayor problema atribuible a la integración analógica es la **no escalabilidad con la tecnología**, lo que significa que cuando la tecnología de integración cambia, se debe rehacer la mayor parte del diseño. Esta circunstancia no sucede en las realizaciones digitales, gracias al nivel de desarrollo que han alcanzado las herramientas disponibles [Ramacher 91b].

Concluyendo, existen un buen número de realizaciones de circuitos neuronales analógicos de propósito específico, que resultarán muy útiles debido al auge de la integración de sensores y la necesidad de etapas de preprocesamiento y postprocesamiento, vitales en la mayor parte de las aplicaciones del mundo real. Para llevar a cabo el **procesamiento en bajo nivel y tiempo real** (por ejemplo, en las primeras etapas de un sistema de visión), las realizaciones analógicas parecen ser la alternativa más interesante, al proporcionar alta velocidad y tamaño reducido, siendo en estas tareas suficientes los 8 bits de precisión que proporcionan.

Pros y contras de la realización digital

La realización digital de redes neuronales se basa en el almacenamiento de los pesos y activaciones neuronales en registros, así como en la utilización de estructuras de cálculo clásicas, como sumadores, multiplicadores o ALU, para la realización de las operaciones que requieren los ANS. Es decir, las estructuras en las que se basa la realización digital son **complejas y costosas** y, en principio, no permiten un nivel de paralelismo tan alto como las analógicas.

Un ejemplo de realización digital se muestra en la Figura 5.20, que podemos comparar con la simplicidad de la neurona analógica de la Figura 5.19. Constituye una unidad de procesamiento digital que puede implementar varias neuronas virtuales, y que incluye los siguientes bloques fundamentales: ALU optimizada para la realización de operaciones MAC; memoria local para el almacenamiento de los pesos sinápticos, estados de las neuronas virtuales y tabla de la función de activación no lineal; banco de registros, unidad de control e interfaz con el bus del sistema.

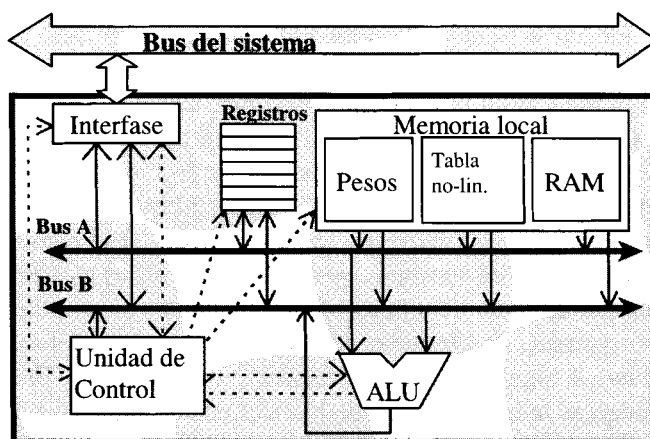


Figura 5.20. Unidad de procesamiento digital genérica, que implementa un conjunto de neuronas virtuales

Para las **redes neuronales menos inspiradas en la biología**, es decir, casi todos los modelos convencionales de ANS (MLP, SOFM, RBF, etc.), la integración digital en un chip neuronal o neurochip proporciona características muy interesantes, como flexibilidad en el diseño, posibilidad de integrar la fase de aprendizaje, realización de diseños expandibles y precisión elevada. Los diseños digitales proporcionan mayores ventajas globales, pues hay que tener en cuenta que rango dinámico y precisión son aspectos críticos en muchos modelos neuronales (como el BP), y con la tecnología digital es posible alcanzar el nivel requerido.

Con todo, **una de las grandes ventajas** del diseño de circuitos digitales VLSI es la disponibilidad de potentes paquetes CAD y la posibilidad de abordar los diseños con una metodología modular. Además, otra importante ventaja es la rápida adecuación de los diseños digitales a las nuevas tecnologías disponibles con mayor nivel de integración (escalabilidad con la tecnología), junto con la disponibilidad actual de herramientas de síntesis automática a partir de descripciones HDL del circuito a construir, disminuyendo notablemente los tiempos de desarrollo y abaratando costes.

Obviamente, las desventajas de la implementación digital son el gran consumo de área de silicio y la velocidad de ejecución relativamente lenta (en comparación con las realizaciones analógicas).

En la tabla 5.3 se muestra un resumen de las ventajas e inconvenientes de ambos estilos de implementación; en parte está inspirada en [Cabestany 93], aunque hemos incorporado modificaciones significativas, completándola en algunos aspectos adicionales.

Característica	Realización analógica	Realización digital
Velocidad	GCPS (10^9 CPS)	MCPS (10^6 CPS)
Consumo	Moderado	Alto (depende del reloj)
Área de silicio	Menos ocupación	Más ocupación
Inmunidad al ruido	Pobre	Buena
Precisión	1 a 8 bits, insuficiente para muchos tipos de aprendizaje	8 a 32 bits, controlable
Memoria	Dificultades, especialmente en cuanto a adaptabilidad	Muy fácil, modificable
Conectividad	Más fácil (señal por línea)	Más difícil (un bit por línea)
Flexibilidad (programabilidad)	Nula (propósito específico)	Alta (propósito general o específico)
Adaptabilidad (aprendizaje)	Pobre, difícil de lograr	Muy Buena
Herramientas CAD	Pobres	Muy buenas, madurez
Escalabilidad con la tecnología	Mala	Buena

Tabla 5.3 Realización analógica frente a digital

Realizaciones mixtas

Dado que las operaciones típicas en un ANS (sumas, productos y no linealidades) son fáciles de realizar mediante dispositivos analógicos, mientras que el almacenamiento de pesos y el control del sistema se resuelven más eficientemente en digital, una solución de tipo mixto parece interesante.

De esta manera, mediante circuitos neuronales mixtos analógico-digitales puede conjugarse la alta velocidad de procesamiento y densidad de integración de los circuitos analógicos, con la precisión que proporciona el almacenamiento digital de los pesos. No dedicaremos una sección especial a este tipo de circuitos, sino que los describiremos junto a los puramente analógicos o digitales, dependiendo de su mayor componente de uno u otro tipo. Un ejemplo clásico que se tratará es el chip neuronal ETANN de Intel.

5.9 REALIZACIONES ANALÓGICAS DE ANS

Veremos a continuación algunas realizaciones esencialmente analógicas, que emulan con mayor o menor fidelidad la arquitectura de la red. Aunque los diseños analógicos trabajan con parámetros eléctricos continuos, pueden ser implementados mediante circuitos en tiempo continuo (redes RC) o discreto (capacidades conmutadas y CCD). Una referencia imprescindible en este campo es el famoso clásico libro de Carver Mead *Analog VLSI and Neural Systems* [Mead 89a]. Un buen complemento es una recopilación realizada por el mismo autor junto a M. Ismail [Mead 89b]. Por otra parte, en los libros de Kosko [Kosko 92b] y Zurada [Zurada 92a] se exponen amplias introducciones a la realización analógica de redes neuronales. En [Jabri 96, ISCAS 00] pueden encontrarse algunas realizaciones recientes.

Circuitos neuromorfos: retina y cóclea

La **retina de silicio** fue desarrollada a finales de los años ochenta por Carver Mead y sus colaboradores [Mead 89a, 89b], e implementada en un chip VLSI con tecnología CMOS de 3 micras. La retina es una estructura neuronal existente en el ojo encargada de recoger los fotones y de realizar un preprocesamiento de la imagen percibida, para ser posteriormente enviada al área visual del córtex cerebral.

Esta retina de silicio se construyó imitando la estructura neuronal de la retina de los primates (Figura 5.21 a). Un chip individual contiene una matriz de celdas (48×48 o bien 88×88) que implementan los píxeles; cada celda incluye un fotorreceptor, un circuito de procesamiento y un circuito para transmisión de datos al exterior del chip (Figura 5.21b). Estos dispositivos imitan los diferentes tipos de neuronas existentes en la retina: conos y bastones, células bipolares, horizontales y ganglionares. Los circuitos analógicos encargados del procesamiento de datos proporcionan la derivada espacial y temporal de la imagen, realizándose la primera mediante una red resistiva hexagonal (Figura 5.21 b). La retina de silicio construida presenta funcionalidades similares a la biológica, como acentuación de contrastes, detección de contornos, respuesta a movimientos o habituación a estímulos persistentes, incluso se han observado algunas de las ilusiones ópticas conocidas [Mahowald 91 b].

Aunque se han realizado bastantes chips neuromorfos para visión [Koch 96], también se han construido algunos para audio emulando la estructura de la **cóclea** [Mead 89 a, Lande 00], mecanismo en forma de caracol alojado en el oído interno, relleno con un fluido por el que se propagan las ondas sonoras, las cuales hacen vibrar los cilios de las células de la superficie, transductores de las vibraciones [Mead 89a]. La cóclea, primera etapa de procesamiento en el sistema auditivo, convierte así la información del dominio temporal en información codificada espacialmente, gracias al papel de filtros pasabanda de las células pilosas. Por ejemplo, en [Mead 89a] se describe una cóclea electrónica implementada en un chip CMOS mediante OTA y condensadores, realizados mediante transistores MOS. Cada chip incluye 100 etapas de filtrado para separar las componentes frecuenciales.

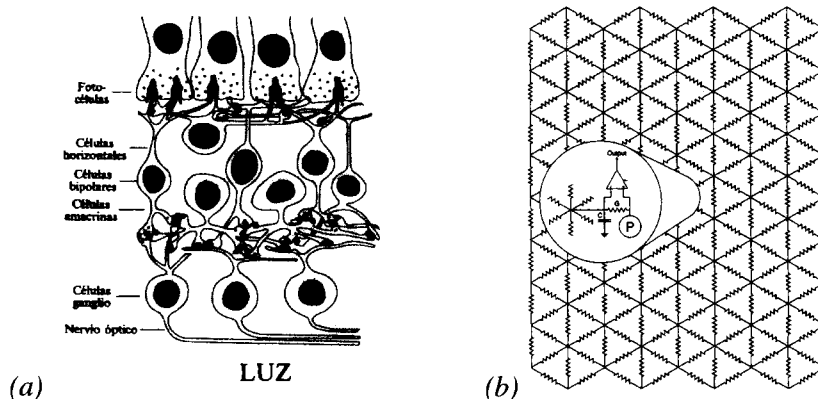


Figura 5.21. Retina electrónica [Mead 89a]. a) Estructura celular de la retina de un primate. b) Retina de silicio: red resistiva y ampliación de un píxel

ETANN 80170NX (Intel)

ETANN (*Electrically Trainable Analog Neural Network*) [Zurada 92 a, Cabestany 93] es un chip neuronal creado por Intel hacia 1991, que durante unos años estuvo comercialmente disponible (Intel abandonó esta área, igual que ha abandonado otras, debido al mayor negocio que representan para ellos los microprocesadores). En las siglas ETANN, el término entrenable hace referencia únicamente a la posibilidad de reprogramar los pesos. Se trata de una realización analógica, aunque hace uso de celdas EEPROM de una micra para almacenamiento (no volátil) de pesos. Cada chip de 208 pines contiene 64 neuronas analógicas y 10.240 pesos sinápticos programables, alcanzando velocidades del orden de los 2 GCPS (2×10^9 CPS); pueden realizarse conexiones de hasta 8 chips ETANN para configurar arquitecturas complejas.

El aprendizaje de ETANN no está incluido en el chip, sino que se realiza en un ordenador convencional, pudiéndose utilizar para ello software desarrollado por otras casas comerciales, como *BrainMaker*. Los pesos obtenidos se incorporan al chip mediante un grabador proporcionado por Intel. ETANN permite implementar varios algoritmos, como BP, Madalina y BP recurrente. La precisión equivale a 7 bits, y la superficie total del chip es de 0.8 cm^2 , ocupando cada neurona un área de 1.25 mm^2 .

Circuitos comerciales actuales

En este punto trataremos algunas realizaciones analógicas de ANS comercialmente disponibles en la actualidad. Muchas de estas realizaciones se basan en redes neuronales celulares o CNN (*Cellular Neural Networks*), modelo utilizado en tareas de visión. Por ejemplo, la empresa americana **IC Tech** (<http://www.ic-tech.com>) comercializa la serie de circuitos neuronales para visión **CISP** (*Cellular Image Sensor Processor*). Por otro lado, la empresa sevillana **AnaFocus** (*spin-off* del

Centro Nacional de Microelectrónica de Sevilla, <http://www.anafocus.com>) comercializa el chip **ACE4K**, máquina CNN universal para visión en tiempo real [Roska 00, Liñán 99], que integra 64x64 celdas en un chip CMOS de 0.5 μm , 10^6 transistores, 3.3V y 1W. A partir de él se ha desarrollado la placa **ALADDIN** [Roska 00, Szátmari 00] de la empresa húngara **Analogic Computers Ltd.**

Es de destacar la empresa **Synaptics** (<http://www.synaptics.com>), fundada por Carver Mead (*padre* de la integración VLSI y de la realización de circuitos neuromorfos) y Federico Fagin (quien cuando estaba en Intel desarrolló el primer microprocesador, el 4004, también el 8080, y más tarde fundó Zilog, donde diseñó el Z80). Esta empresa comercializó a principios de los años noventa un chip basado en la retina de silicio empleado en OCR (*Optical Character Recognition*, reconocimiento óptico de caracteres), el cual se empleó en lectura de cheques y direcciones en sobres postales. En la actualidad esta empresa lidera el mercado de los *touchpads* para ordenadores portátiles, cuya señal es procesada mediante un circuito neuomorfo de similitudes con la retina, y comercializa un OCR para caracteres chinos (en software).

Finalmente, en [Fang 00] se describe el desarrollo de un **sistema de visión en tiempo real completo** en un módulo multichip, que incluye sensores y procesamiento mediante una red neuronal CNN, que será empleado por la NASA en sondas espaciales de exploración planetaria.

5.10 REALIZACIONES DIGITALES DE ANS

Mediante integración digital VLSI se construyen chips neuronales de propósito general y específico; los primeros se utilizan en el desarrollo de neurocomputadores. Chips neuronales de propósito específico se utilizan en la actualidad en el desarrollo de reconocedores de voz y de caracteres escritos en productos comerciales.

Lneuro (Philips)

Philips desarrolló Lneuro 1.0 a principios de los años noventa [Mauduit 92]; la última versión de la que tenemos constancia es L-neuro 2.3 [Duranton 96]. Se trata de un circuito totalmente digital para redes neuronales y procesamiento de señal en general, que integra en un chip, entre otros, los bloques siguientes: unidad vectorial SIMD con 12 bloques DSP (cada uno con una ALU de 32 bits y multiplicador entero 16×16), unidad escalar de 32 bits, módulo de direccionamiento de imágenes (1024×1024) y unidad de conversión vector a escalar. Este chip se construyó en colaboración con el Departamento de Altas Energías de la Escuela Politécnica de París con tecnología de 0.6 μm , integra 1.8 millones de transistores y opera a 60MHz. En [Duranton 96] se dan cifras de rendimiento y se explica su aplicación a tareas como el procesamiento de imagen para clasificación de naranjas y la detección de eventos en física de altas energías. Desconocemos su disponibilidad comercial actual.

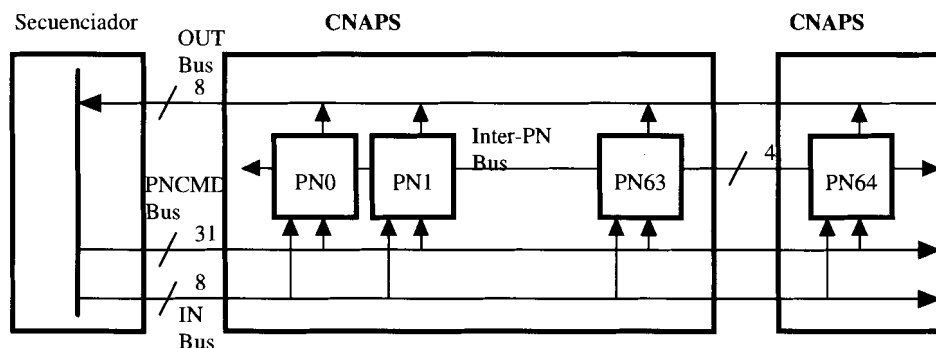


Figura 5.22. CNAPS [Hammerstrom 90]. Estructura del neurochip N64000

CNAPS (Adaptive Solutions Inc.)

CNAPS (*Connected Network of Adaptive ProcessorS*) [Hammerstrom 90, 91, Ienne 93] es una arquitectura neuronal de propósito general, construida a finales de los años ochenta, que utiliza tecnología ULSI y ha estado disponible comercialmente durante los años noventa. Su arquitectura, mostrada en la Figura 5.22, consiste en una clásica estructura SIMD unidimensional de nodos procesadores o PN (*processing nodes*). Los PN se conectan en topología orientada a bus (*broadcast*) a través de tres buses: uno de entrada de datos, otro de salida (ambos de 8 bits), y un tercero de control. Cada PN es similar a un DSP, incluyendo 4 Kbytes de memoria local.

La arquitectura CNAPS se compone de un secuenciador y cuatro grupos de chips **N64000**. Cada N64000 contiene 80 PN, contruidos con tecnología ULSI redundante, con un 90% de probabilidad de que puedan utilizarse 64 de los 80; en su construcción se ha utilizado un proceso CMOS de 0.8 micras y bimetálico. El tamaño del dado es de una pulgada de lado y contiene 11 millones de transistores. A 25 MHz y con todos los PN operando, consume 4 vatios. Cada PN calcula un MAC por ciclo, por lo que a 25 MHz se estima su velocidad en 1.6 GCPS, con aritmética de precisión 8x8 y 8x16. Para precisión 1x1, la velocidad asciende a 12.8 GCPS.

La versión comercial de CNAPS como servidor para neurocomputación contiene 4 chips N64000 a 20 MHz y ofrece unas prestaciones de 5 GCPS y 944 MCUPS en entrenamiento tipo BP (las mejores cifras de la época). Incluye además un sistema de desarrollo y compilador en C.

CNAPS es uno de los neurocomputadores que han contado con mayor éxito comercial (junto con Synapse de Siemens). No obstante, parece ser que la empresa *Adaptive Solutions* desapareció hace unos años, encargándose por un tiempo *BrainMaker* de la comercialización de CNAPS (<http://www.calsci.com/cnaps.html>), aunque parece que ha sido definitivamente abandonada.

ZISC036 (IBM)

Desarrollado por los laboratorios de IBM en Essonnes, Francia, (<http://www.fr.ibm.com/france/cdlab/zisc.htm>), el ZISC (*Zero Instruction Set Computer*) es un chip digital con 36 neuronas y 64 entradas de 8 bits de precisión. El ZISC, cuya entrada de datos es serie, resulta fácilmente interconectable a otros chips ZISC para obtener arquitecturas de tamaño arbitrario. La función de transferencia de las neuronas tiene forma de función de base radial (RBF), y proporciona salidas de 14 bits de precisión, pudiendo emplear dos funciones distancia distintas: la de Manhattan, y la distancia como máxima diferencia entre componentes del vector de entradas y de pesos. El proceso de aprendizaje se halla incorporado en el chip, permitiéndose el almacenamiento interno de los vectores de aprendizaje y sus salidas deseadas.

En el proceso de ejecución el chip emplea 3.5 μ s en la carga de las 64 entradas, más otros 0.5 μ s adicionales en mostrar la señal de salida. Para el proceso de aprendizaje se emplean 2 μ s más por vector. De este modo, y a 16 MHz de velocidad de reloj, el tiempo de ejecución de un proceso de recuerdo de un vector de 64 entradas es de 4 μ s, mientras que su proceso de aprendizaje emplea 6 μ s. IBM ha desarrollado también una **placa aceleradora ZISC/ISA** para PC con 16 chips ZISC. En la actualidad desconocemos su disponibilidad comercial

SYNAPSE (Siemens)

La serie de neurocomputadores SYNAPSE fueron desarrollados por el grupo de tecnologías avanzadas de Siemens-Nixdorf durante los años noventa [Ramacher 94]. Se basan en el chip neuronal **MA16**, arquitectura sistólica constituida por 4 unidades de cálculo denominadas cadenas sistólicas. Cada cadena consta de 4 multiplicadores de 16 bits, 3 sumadores de 32 bits para aplicaciones con matrices, diversos buffers, un multiplicador específico y un acumulador final, implementando el aprendizaje en el chip. MA16 está pensado para facilitar la interconexión de múltiples unidades, las unidades de cálculo están optimizadas para 16 bits, y está construida con CMOS de 1 micra, ocupando 187 mm² y 610.000 transistores. A 50MHz alcanza los 800 MCPS.

SYNAPSE3-PC es una placa neurocomputadora que incluye dos chips MA16, comercializada por la empresa alemana MediaInterface (<http://www.mediainterface.de/english/English1.html>). Pueden conectarse hasta 3 placas de este tipo en un mismo PC, que pueden ser fácilmente programadas en C++.

Otros circuitos comerciales

Describiremos a continuación una serie de circuitos neuronales disponibles comercialmente en la actualidad. **Nestor Inc.** (<http://www.nestor.com/>), empresa americana fundada por el Premio Nóbel Leon Cooper comercializa desde hace unos años el **Ni1000** (desarrollado en colaboración con Intel), que implementa los modelos RCE, PRCE y PNN; una neurona puede tener hasta 222 entradas, cada una de 5 bits

de resolución, y soporta 64 salidas, pudiendo alcanzar hasta 10.000 patrones por segundo. Este integrado neuronal ha sido uno de los de mayor éxito comercial [Lindsey 98], por ejemplo, se ha empleado en OCR [Clarkson 95]. Se comercializa un sistema de desarrollo y la placa aceleradora **Pci4000**, con uno, dos o cuatro procesadores Ni1000.

Por otro lado, la empresa italiana **Neuricam** (<http://www.neuricam.com/>) ha desarrollado la serie de chips neuronales **TOTEM** que implementan aprendizaje *Reactive Tabu Search*; de arquitectura SIMD, son capaces de ejecutar hasta 1500 millones de MAC de 32 bits en punto fijo por segundo (a 50MHz). Por ejemplo, el **NC3002** integra 32 neuronas por chip, con 256 pesos por neurona y opera a 40 MHz. También comercializa placas procesadoras basadas en estos chips, como la **TotemPCI**, que incluye dos procesadores. Un ejemplo de empleo en la actualidad es como detector de eventos en los experimentos de altas energías del FermiLab (USA).

Para finalizar, comentaremos que **Sensory Inc.** (<http://www.sensoryinc.com/>) desarrolla circuitos específicos para reconocimiento de habla (dependiente e independiente del hablante), orientados a aplicaciones de consumo y juguetes (por ejemplo, marcadores para teléfonos o el osito Kobi), debido a su bajo coste (en grandes cantidades estos chips cuestan menos de \$5 dólares). Integran en un mismo chip procesador, red neuronal para reconocimiento de habla y circuitería de conversión A/D y D/A (entre otros); el aspecto externo del chip es el mismo que el de un microcontrolador estándar. Algunos de sus productos son RSC164, RSC264, RSC364 o VoiceDialer, en 1998 se llevaban vendidos ya más de un millón de unidades de todos ellos. A modo de ejemplo, en [Medrano 00] describimos la construcción de un ratón de computador controlado con la voz mediante un **RSC164**.

5.11 EJEMPLO: NEUROEMULADOR BASADO EN FPGA

Como ejemplo de implementación de un neuroemulador describiremos la tarjeta diseñada y realizada en el Departamento de Ingeniería Electrónica y Comunicaciones de la Universidad de Zaragoza⁸ [Hernández 96, Martín del Brío 98, Medrano 98]. La tarjeta es un **coprocesador neuronal** basado en dispositivos programables tipo FPGA, que emula el modelo de **mapas autoorganizados**. Esta tarjeta es conectable al bus de un ordenador PC, y capaz de realizar todos los procesos asociados al modelo, tanto en aprendizaje como en recuerdo. La capacidad de reconfigurabilidad ha sido aplicada a la depuración y mejora del diseño del procesador elemental (PE) y de la unidad de control (CU), permitiendo reducir considerablemente el tiempo de desarrollo.

Como se desprende de la tabla 5.3, dos de las restricciones más importantes que tiene la implementación en tecnología digital son la ocupación de área de silicio y su

⁸ Los autores quieren expresar su agradecimiento a la Universidad de Zaragoza por el apoyo al desarrollo de esta tarjeta, en la forma de Proyecto de Investigación.

elevado consumo, en comparación con las realizaciones analógicas. Por ello, a la hora de abordar el diseño digital de una red neuronal es de gran importancia la obtención del modelo más simple posible de PE. Atendiendo a estas consideraciones, el PE diseñado se basa en el cálculo de la **distancia de Manhattan**, tal y como se describe en [Martín del Brío 94b], y trabaja con una **precisión de 8 bits**.

Bloques básicos del coprocesador neuronal reconfigurable

Como se ha indicado en la sección 5.6, los neuroprocesadores constan de cuatro bloques básicos, cuya estructura los define. Los bloques que conforman la tarjeta neuroprocesadora realizada (figura 5.23) poseen las siguientes características:

Procesador Elemental (PE). El procesador elemental, bloque básico de la red neuronal, efectúa las operaciones propias de las neuronas del mapa de Kohonen. En la fase de recuerdo sólo es preciso que actúen los módulos de cálculo de distancias (implementan la de Manhattan) y determinación de neurona ganadora (para lo que se emplea la técnica de aproximaciones sucesivas descrita en [Martín del Brío 94b]). El tiempo de cómputo depende del número de componentes de los patrones de entrada.

Cada PE está implementado sobre una FPGA de Xilinx tipo XC3042APC84-7, pudiendo implementar hasta 16 procesadores virtuales cada uno de ellos.

Unidad de Control (CU). Secuencia las distintas fases y operaciones que constituyen cada fase, genera la dirección de la RAM que utilizarán los PE, configura los PE para que simulen la neurona virtual adecuada, controla el flujo de datos entre el *host* y cada uno de los PE, determina cuándo deben decrementarse los valores del radio de vecindad y del valor de actualización, y determina la neurona ganadora.

El núcleo de este módulo es una máquina de estados que genera las señales de control. Consta de 4 bloques, que especifican el funcionamiento de cada una de las cuatro operaciones de la tarjeta (*Configuración, Recuerdo, Aprendizaje y Lectura de pesos y estados*). Su funcionamiento puede ser interrumpido en cualquier instante, devolviendo la tarjeta a una situación de reposo. Diseñada sobre una FPGA XC3090APC84-7, es capaz de controlar hasta 256 PE (limitado por la anchura del bus de direccionamiento de PE).

Unidad de Comunicaciones. Las comunicaciones entre los dispositivos que conforman la placa se realizan mediante una arquitectura orientada a bus (BBA, Figura 5.8 a) constituida por 5 buses internos (transmisión de datos, direccionamiento de PE, acceso y direccionamiento a memoria y control interno), y 3 externos, propios de cualquier microprocesador.

La tarjeta coprocesadora se halla diseñada para la conexión al bus de un ordenador PC que hace de *host*, ocupando 3 direcciones consecutivas de E/S, destinadas a la configuración de las FPGA, transmisión de datos entre tarjeta y *host*, y transmisión de órdenes a la tarjeta.

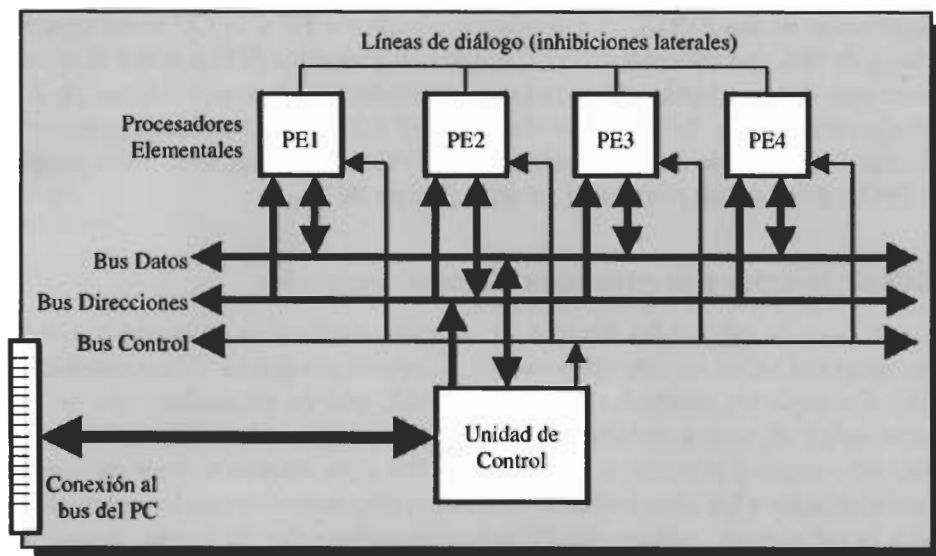


Figura 5.23. Diagrama de bloques de la tarjeta coprocesadora y buses

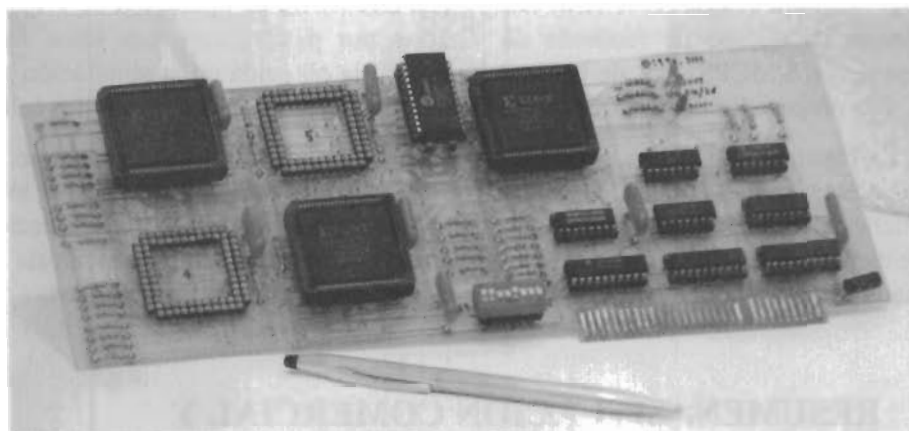


Figura 5.24. Neuroemulador construido en la Universidad de Zaragoza. La FPGA de la derecha hace de CU, y las otras dos de PE. Los dos zócalos vacíos permiten añadir dos FPGA más, para llegar a 4 PE (Fotografía de R. Aparicio y O. Medrano)

Unidad de Almacenamiento. Constituida por una memoria RAM de 2Kbytes, en ella se almacenan los valores de los pesos sinápticos de las neuronas. Cada chip de memoria RAM es compartido entre cuatro PE, permitiendo que cada uno pueda simular hasta 16 neuronas con un vector de 32 elementos.

Configuración de las FPGA. La configuración de los PE y la CU viene descrita en un fichero de bits, que informa a los LCA que componen las FPGA sobre la estructura hardware que deben adoptar. Este fichero, generado por el propio sistema de Xilinx, es enviado en serie a la dirección del mapa de E/S del bus del PC donde se encuentran conectadas las líneas de configuración de las FPGA [Lawman 95]. Para programar varias FPGA éstas deben conectarse en serie [Xilinx 96].

Análisis de tiempos y prestaciones del neuroemulador

A la hora de estimar los tiempos de cómputo implicados en la emulación de un modelo neuronal sobre un neuroprocesador, debemos considerar dos cuestiones: por un lado, los aspectos asociados a la red virtual, que es el modelo que se quiere proyectar sobre el neuroemulador, con sus características (dimensiones, número de patrones de entrada y elementos por patrón). Por otro, debemos tener en cuenta los aspectos asociados a las características físicas del dispositivo emulador sobre el que se proyecta la red virtual: número de PE implementados sobre la tarjeta, tiempo de un ciclo de máquina (habitualmente dos o cuatro ciclos de reloj), y de un ciclo de reloj.

Bajo estos supuestos, y tomando como ejemplo una red de 5x5 neuronas, con los 4 PE sobre la placa, y un problema con 150 patrones de 6 elementos, un tiempo de ciclo de máquina cuatro veces el de reloj y una frecuencia de operación de 7.15 MHz obtenemos un tiempo de recuerdo de 92.16µs por patrón, con una velocidad de proceso de 7.15 MCPS, más de 6 veces superior a la obtenida en la simulación sobre un PC con procesador 80486DX4 a 100MHz, ofreciendo resultados de similar calidad.

La Figura 5.24 muestra el prototipo de neuroemulador realizado, en el que, para reducir costes, se han empleado dispositivos de reducidas prestaciones, pese a lo cual se obtiene un rendimiento elevado. Realizar una tarjeta neuroprocesadora haciendo uso de ejemplares de FPGA más rápidos y de mayor capacidad, chip de memoria local para cada PE, o introduciendo más PE en la tarjeta, dispararía sus prestaciones.

5.12 RESUMEN. SITUACIÓN COMERCIAL Y TENDENCIAS

Hemos visto que gracias a los computadores disponibles a mitad de los años ochenta se pudieron realizar las primeras aplicaciones de ANS, lo que contribuyó al renacimiento de este campo. Sin embargo, en el estudio del DARPA publicado en 1988 se puso de manifiesto que la potencia de cálculo de los computadores entonces disponibles permitían tan solo la aplicación de ANS a problemas de relativa simplicidad (redes pequeñas), resultando inviable su empleo en tareas como reconocimiento de habla o visión en tiempo real. Además, algunos algoritmos de aprendizaje, como el BP habitual eran terriblemente lentos, y en ocasiones se requerían varios días de entrenamiento. Por ello **muchas empresas y grupos**

universitarios decidieron realizar mediante las tecnologías de integración VLSI chips neuronales y neurocomputadores. Multinacionales del sector de la electrónica e informática como Intel, IBM, Philips, Siemens y otras, pusieron en marcha programas de desarrollo de hardware neuronal, puesto que se preveía un rápido crecimiento de este nuevo sector. También entonces se fundaron empresas como HNC, BrainMaker o Adaptive Solutions, que desarrollaban hardware neuronal.

Sin embargo, años más tarde bastantes de estas empresas han abandonado el desarrollo de hardware neuronal, en gran parte **debido al incesante aumento en el rendimiento de los computadores convencionales**, como los PC. Por ejemplo, hacia 1990 se puso a disposición de nuestro laboratorio un PC/AT con procesador 80286 a 12 MHz, y representó para nosotros un gran avance poder adquirir tres años más tarde un 80386 a 25MHz con coprocesador matemático (y 8MB de RAM); sin embargo, a fecha de hoy Intel comercializa ya un **Pentium III a 1.13 GHz**. Para poder resolver con ANS problemas de cierta entidad a finales de los ochenta y principios de los noventa había que recurrir a supercomputadores o a las placas aceleradoras descritas en este capítulo (como las comercializadas por HNC, SAIC o BrainMaker). Sin embargo, el incesante progreso en los microprocesadores de propósito general [Martín de Brío 99], como la familia Pentium o la PowerPC, permite hoy en día disponer de máquinas sumamente potentes a precios reducidos; **una placa aceleradora o un neurocomputador** (como los de Adaptive Solutions, quizás el mayor éxito comercial de estas máquinas), **suponen un desembolso importante, pero pueden ser sobrepasados en unos cuantos meses por un nuevo Pentium**. En [Granado 99] se muestra que los microprocesadores de propósito general pueden abordar ya tareas relacionadas con redes neuronales de cierta complejidad incluso en tiempo real.

Por este motivo **muchas de las multinacionales han ido abandonando el campo del hardware neuronal**, centrándose en el negocio más lucrativo de los sistemas de propósito general (por ejemplo, Intel con los microprocesadores). Algunas de las empresas que nacieron para explotar comercialmente las posibilidades de las redes neuronales han desaparecido, pero muchas otras siguen trabajando con éxito, pero **orientadas más al desarrollo de aplicaciones que al hardware** (por ejemplo, la Hecht-Nielsen Corp., HNC, una de las pioneras, ha abandonado el hardware, pero tiene un enorme éxito como consultora, desarrollando numerosos proyectos con redes neuronales para muchas empresas). En definitiva, las ANS están establecidas ya como una herramienta más para la resolución de problemas complejos [Lindsey 98a, Werbos 98] (se emplean mucho pero normalmente no se cita, como tampoco se dice que en cierto producto se emplee la FFT) y **muchas de las aplicaciones se desarrollan como un software que se ejecuta sobre un computador convencional**, negocio que mueve muchos millones de Euros [Lindsey 98 a]. El tema de aplicaciones lo trataremos ampliamente en el siguiente capítulo.

No obstante, la realización hardware tiene en la actualidad claros nichos de aplicación, que exponemos a continuación, y empresas especializadas (Synaptics, Neuricam, Ligature, etc.) desarrollan y comercializan chips neuronales [Lindsey 98 a]:

- **Aplicaciones donde se requiere elevado rendimiento.** La detección de eventos en laboratorios de **física de altas energías**, como el CERN (Ginebra) y el FermiLab (USA), es un caso típico. Por ejemplo, en un experimento del acelerador LHC (*Large Hadron Collider*) se emplean 200.000 sensores; cada 25 ns cada uno proporciona un dato de 16 bits [Duranton 96]. Los eventos interesantes que se producen son raros y hay que extraerlos de la ingente masa de datos, para lo que se emplean rutinariamente chips neuronales como ETANN (Intel) y TOTEM (Neuricam), y neurocomputadores como CNAPS, clasificando patrones en menos de 10 μ s [Widrow 94, Lindsey 98]. Por otro lado, en [Freedman 94] se expone el empleo de hardware neuronal en el **avión de combate F-15** para ayuda al piloto en caso de alcance por fuego enemigo (la red neuronal electrónica aprenderá las nuevas condiciones de pilotaje más rápidamente que el ser humano). En ambos tipos de aplicaciones lo importante es la velocidad, quedando en un segundo plano la cuestión económica.
- **Circuitos de aplicación específica y bajo coste.** Los microcontroladores son chips estándar de muy bajo coste que integran procesador, memoria y periféricos [Martín del Brío 99], empleándose en el desarrollo de controles específicos en el automóvil (ABS, cierre centralizado...), electrodomésticos (lavadoras, hornos, TV...), comunicaciones (teléfonos, radio...), etc. Se han desarrollado chips neuronales para su empleo en controles de este tipo; por ejemplo, el escáner en forma de bolígrafo *Quicktionary*TM de Wizcom, lee y traduce texto gracias al chip neuronal **OCR-on-a-chip** de Ligature. Por otro lado, los chips de Sensory Inc. para **reconocimiento de voz** se comercializan por menos de \$5; se usan en telefonía, juguetes, aparatos de consumo, etc. Otro ejemplo son los *touchpad* para ordenadores portátiles de Synaptics, que se basan en una red analógica de estructura similar a la retina de silicio.
- **Emulación de sistemas biológicos.** Se desarrollan **circuitos neuromorfos** [Lande 98] que modelan con cierta fidelidad estructuras biológicas, como la clásica neurona de silicio [Mahowald 91a], o las retinas y cócleas ya descritas [Koch 96, Delbrück 00, Lande 00]. Circuitos de este tipo mejorados permitirían en un futuro sustituir tejidos sensoriales dañados [IEEE 96].

En resumen, el desarrollo de hardware neuronal ha tenido menos éxito que las aplicaciones software debido al gran desarrollo de microprocesadores y computadores convencionales. No obstante, existen nichos de aplicación claros para este tipo de circuitos (grandes rendimientos, sistemas específicos), donde se tiene un cierto éxito comercial. Sin embargo, todavía hoy las grandes aplicaciones (más de 1000 neuronas) son difícilmente abordables, por lo que a medio plazo, cuando se disponga de una tecnología adecuada, habrá que contar todavía con el desarrollo de hardware neuronal específico. Por ejemplo, el proyecto *Artificial Brain Project*, de Starlab (<http://www.starlab.org>) trata de construir un sistema con millones de neuronas mediante circuitos FPGA estándar, pero el tamaño y precio del sistema resultará enorme.

APLICACIONES DE LAS REDES NEURONALES ARTIFICIALES¹

Tras el estudio de algunos de los modelos neuronales más conocidos y la forma de realizarlos en la práctica (por simulación por computador o mediante hardware específico), en este último capítulo dedicado a las redes neuronales artificiales expondremos diferentes aspectos relacionados con su puesta en práctica para la **resolución de problemas reales**. Así, hablaremos del interés de utilizar ANS, la manera de desarrollar una aplicación basada en redes neuronales y las herramientas de simulación por software existentes (tanto comerciales como gratuitas). Asimismo, compararemos los ANS con otras técnicas y, finalmente, expondremos algunas aplicaciones reales desarrolladas con modelos neuronales. Muchos de los aspectos que se tratarán aquí se aplican **también al caso del desarrollo de sistemas borrosos**.

6.1 MOTIVACIÓN E INTERÉS DEL EMPLEO DE ANS

La lista de problemas en los que se han aplicado con éxito redes neuronales crece constantemente [Widrow 94], lo que se aprecia en cada nueva edición de congresos o revistas especializadas. Las preguntas que surgen inmediatamente son: *¿dónde radica el éxito de los ANS?, ¿podré beneficiarme de la (al parecer) gran potencia de estas nuevas herramientas?, ¿cuáles son las características que debe tener mi problema para que pueda plantearme el empleo de un ANS?* Para intentar dar respuesta a estas preguntas, vamos a describir un par de aplicaciones de ANS, destacando en cada una de ellas la motivación que ha llevado a emplear estas nuevas técnicas. Ambos casos de estudio aparecen en la referencia [Eberhart 90].

¹ En la realización de este capítulo ha colaborado Javier Blasco Alberto, del Laboratorio de Investigación en Tecnologías de la Combustión (LITEC), CSIC-Universidad de Zaragoza.

Caso 1: Detección de picos epilépticos en electroencefalogramas

Durante décadas, neurólogos, científicos e ingenieros han estado tratando de obtener un algoritmo *universal* de identificación de picos epilépticos en señales electroencefalográficas (EEG). No obstante, los expertos no se ponen de acuerdo a la hora de definir las características precisas de estos picos, lo cual imposibilita marcar unas pautas concretas para su análisis. Ahora bien, se dispone de un gran número de ejemplos o casos en los que se tienen identificados picos epilépticos. Recientes avances en sistemas expertos han permitido la construcción de herramientas que analizan estas señales, pero de forma *off-line* o fuera de línea (es decir, no son capaces de responder en tiempo real o en línea).

Una **motivación para la aplicación de ANS** sería ésta problemática asociada a las técnicas convencionales, ya que

- No existe un algoritmo adecuado.
- No hay acuerdo para definir de forma precisa un pico epiléptico.
- Los sistemas expertos convencionales funcionan *off-line*.

En cambio, los ANS pueden ser una buena solución porque:

- No hay que proporcionar ningún algoritmo que defina un pico epiléptico.
- Simplemente se proporciona al sistema ejemplos ya clasificados por los expertos.
- Además, una vez entrenada, la red es capaz de dar la respuesta en tiempo real, lo cual posibilita el análisis en línea (*on-line*) de las señales EEG.

Caso 2: Predicción de erupciones solares

En la Universidad de Colorado se había desarrollado un sistema experto para la predicción de erupciones solares, formado por unas setecientas reglas, y capaz de proporcionar una respuesta tan precisa como la de un experto en tan sólo cinco minutos; no obstante, para su desarrollo necesitaron un año de trabajo. Siguiendo el mismo esquema empleado en el desarrollo del sistema experto, en tan sólo una semana se construyó un ANS que logró mejor precisión que el sistema experto y que proporcionaba la respuesta en unos milisegundos.

La **motivación para la aplicación de ANS** en este caso sería el considerable ahorro en tiempo de desarrollo (y dinero), y también la mejora en las prestaciones del sistema. Muchos sistemas expertos son costosos de diseñar y lentos en proporcionar respuestas, por lo que en ocasiones están siendo reemplazados o complementados por técnicas como los ANS.

Conclusión: aplicabilidad de las redes neuronales artificiales

Una vez vistos estos dos primeros ejemplos, expondremos en términos generales las características que debe poseer un problema para que su resolución con redes neuronales proporcione buenos resultados, así como aquellas que sugieren que el empleo de estas técnicas puede no resultar aconsejable o viable.

a) Características que debe cumplir el problema:

- No se dispone de un conjunto de **reglas** sistemáticas que describan completamente el problema.
- En cambio, sí se dispone de **muchos ejemplos** o casos históricos (ésta es una condición indispensable para poder aplicar técnicas de ANS).
- Los datos procedentes del problema son imprecisos o incluyen **ruido** (por ejemplo, en el caso de un reconocedor de caracteres, las distintas imágenes correspondientes a una misma letra pueden aparecer giradas, ampliadas, reducidas, incompletas, distorsionadas, etc.).
- El problema es de **elevada dimensionalidad** (por ejemplo, la matriz de puntos que conforma una imagen puede ser un número demasiado grande como para que un sistema convencional que opera en serie haga un análisis de la imagen en tiempo real).
- Una circunstancia frecuente es que los métodos de ANS proporcionan una alternativa mucho más **rápida y sencilla** de desarrollar que otras técnicas convencionales.
- Si las condiciones de trabajo son cambiantes se puede hacer uso de la capacidad del ANS para adaptarse a esos cambios **re-entrenando** el sistema con nuevos ejemplos.

b) Características que hacen desaconsejable el empleo de ANS:

- Existe un **algoritmo** que resuelve con total eficacia el problema. Es el caso de los problemas puramente algorítmicos o numéricos (multiplicación de números, inversión de matrices, FFT, etc.).
- No se dispone de un **número adecuado de casos** (ejemplos) para entrenar la red neuronal.
- Tareas críticas o potencialmente peligrosas, cuya resolución deba ser siempre perfectamente predecible y explicable. A veces no resulta fácil interpretar la operación de la red neuronal o predecir con total fidelidad el resultado que pueda proporcionar en todos los casos posibles.

6.2 DESARROLLO DE UNA APLICACIÓN CON ANS

Resolver un problema mediante el uso de ANS supone aplicar una metodología que presenta aspectos comunes con las técnicas convencionales de tratamiento de datos y señal, pero también otros más particulares, que solamente se dan en el campo de los ANS. Seguidamente se van a exponer los pasos que suelen seguirse en el diseño de una aplicación neuronal (en la sección 6.6 se detallan estos mismos puntos para un ejemplo concreto); éstos han sido recopilados de los muy recomendables libros de [Masters 93, Refenes 95, Eberhart 90], entre otros, y completados con la experiencia de los autores.

(1) Planteamiento del problema. Divide y vencerás

Es necesario realizar una descripción detallada de nuestro problema para conseguir averiguar si algún aspecto de él podrá ser resuelto mediante un módulo neuronal. No debemos pensar que los ANS son la mejor solución en todos los casos, en muchas ocasiones será más interesante aplicar un algoritmo convencional. Lo aconsejable es **separar el problema en partes**, y resolver cada aspecto mediante el módulo que resulte más adecuado; **se aplicará una red neuronal solamente donde resulte conveniente**. Los problemas reales siempre requieren soluciones multi-modulares [Fogelman 98] y, como se ha comentado ya en la sección 6.1, algunos aspectos son abordables mediante técnicas neuronales, mientras que otros no son idóneos.

(2) Requerimientos del sistema

En este punto se debe responder de la forma más concreta posible a las especificaciones que debe cumplir el sistema. Por ejemplo, es necesario plantearse la cota de error que se desea alcanzar, el tipo de formulación que se va a aplicar (predicción, clasificación, series temporales, ajuste funcional, procesado de señales, etc.), la forma en la que se dispondrán los datos, el tiempo de respuesta requerido, los equipos informáticos necesarios o disponibles (a nivel de máquinas y de programas de simulación), etc. Todas estas aclaraciones ayudarán a despejar dudas sobre el modelo de ANS a escoger, la necesidad de trabajar con ANS a nivel de simulación, o bien de tarjeta aceleradora o circuito integrado específico (capítulo 5), etc.

(3) Revisión bibliográfica

Resulta una buena práctica realizar un sondeo en busca de alguna aplicación parecida a la que se nos plantea. Observando la estrategia seguida por otras personas en problemas similares al nuestro obtendremos puntos de arranque para ir ensayando sobre nuestros propios datos. En este sentido, es recomendable hacer uso de los recursos que Internet proporciona para obtener referencias recientes (o incluso programas de simulación).

(4) Elección del modelo de ANS

Una vez que hemos especificado con todo detalle las características de nuestro problema, debe elegirse un modelo de ANS con el que comenzar a hacer pruebas. En este sentido, las arquitecturas de ANS más empleadas son las siguientes:

- a) **Perceptrón multicapa (MLP)** con aprendizaje BP o similar, como caso de red de aprendizaje supervisado.
- b) **Mapas autoorganizados de Kohonen (SOFM)**, como red no supervisada.
- c) Otros modelos muy habituales en las aplicaciones prácticas son las **RBF** y el **LVQ**, y otros bastante empleados son SVM, GRNN, ART y CNN.

La red BP puede emplearse en tareas de ajuste funcional (predicción, series temporales, modelado) y también para clasificación. Ésta es la red neuronal por excelencia, empleada en más del setenta por ciento de los casos (!!!). Por su parte, los SOFM se aplican en análisis exploratorio de datos, es decir, para encontrar relaciones entre una gran masa de datos multidimensionales, clasificados previamente o no. Como ya sabemos, el SOFM realiza la proyección no lineal de estos datos sobre las dos dimensiones del mapa, pudiendo visualizar fácilmente sobre él relaciones de interés; un ejemplo muy ilustrativo es el análisis de la quiebra bancaria española de los años 1977-1985 ya descrito [Martín del Brío 93]. En [Fogelman 98] se afirma que en la actualidad los modelos MLP, SOFM, LVQ y RBF cubren el 90% de las aplicaciones prácticas de redes neuronales.

(5) Datos disponibles y selección de variables relevantes

Es preciso saber la forma en la que se va a disponer de los datos. Si están presentes todos a la vez se puede hacer un **aprendizaje por lotes (batch)**. En cambio, si los datos van llegando en tiempo real (por ejemplo, procedentes directamente del exterior, suministrados por sensores), será preciso emplear **aprendizaje en línea (on-line)**. También se necesita conocer la cantidad de ejemplos que se van a poder emplear en el entrenamiento, ya que en el caso de ser un número pequeño quizás haya que aplicar alguna técnica específica de entrenamiento para pocos patrones.

Respecto a la selección de las variables de entrada y de salida (en su caso) del ANS, hay que tener en cuenta los siguientes puntos [Refenes 95]:

- Teniendo en cuenta que los MLP son aproximadores universales [Funahashi 89], se podría optar por proporcionar a la red una gran cantidad de entradas con la esperanza de que fuera ella la que diera más peso a las más importantes y consiguiera realizar el ajuste funcional entradas-salidas deseado. Esta **aplicación a ciegas** puede incluso proporcionar resultados razonables, sobre todo en relación el poco tiempo necesario para conseguirlos. No obstante, este enfoque, en su intento de minimizar el error, induce en ocasiones al ANS a encontrar asociaciones entrada-salida no reales, lo que lleva a una pobre

generalización (es decir, el ANS daría una respuesta incorrecta para ejemplos nuevos, lo que indicaría su incapacidad para extrapolar las relaciones encontradas en los patrones de aprendizaje, véase el capítulo 2). Por lo tanto, es desaconsejable su empleo, en general.

- Si se quieren obtener resultados mejores, es preciso abandonar la postura anterior (todas las entradas que se quiera), y se deben **elegir cuidadosamente las variables** a emplear. Gran parte del éxito de la aproximación funcional reside en buena medida en una correcta selección de las entradas y salidas.
- Pocas variables o entradas restringen el espacio de búsqueda de parámetros, pero si la elección no se realiza con precaución puede viciarse la arquitectura de partida del ANS (tener un *bias* alto) y llevar a una pobre generalización.
- Muchas variables independientes implican una alta dimensión del espacio de búsqueda, lo cual puede conducir a un error de generalización elevado, pero en este caso por exceso de varianza en la arquitectura (capítulo 2).

Los dos últimos puntos suelen denominarse **dilema del sesgo frente a la varianza**, como ya expusimos en el capítulo 2. De entrada, parece importante la selección de las variables de entrada a emplear, pero, ¿cómo realizar esta selección? Un camino puede ser preguntar a un experto en el problema que nos ocupa, lo cual suele dar bastante buenos resultados. Otra posibilidad más formalizable es tomar prestadas de la estadística sus técnicas de construcción de modelos [Refenes 95]. En ellas se analizan multicolinearidades de las variables independientes (posibles entradas al ANS), y se mide la variabilidad de las dependientes (candidatos a salidas del ANS), quedándonos con aquellas variables independientes que más variabilidad producen sobre las dependientes.

(6) Elección de los conjuntos de aprendizaje y test

La condición que se debe imponer a los ejemplos (patrones) que se presentan al ANS para llevar a cabo el entrenamiento es que sean un número suficiente como para ser representativos del fenómeno que queremos modelar. Con esto nos referimos a **que queden cubiertas más o menos todas las situaciones que puedan darse**, es decir, que los valores de las entradas y salidas cubran uniformemente todo su rango; por ejemplo, en el caso de un problema de clasificación, que se disponga de un suficiente número de casos de cada categoría. El libro [Masters 93] ofrece una completa revisión de las características que debe cumplir el conjunto de aprendizaje.

En el caso particular de los MLP se suele emplear una técnica llamada **validación cruzada** [Masters 93] (capítulo 2), que consiste en dividir la totalidad de los patrones en dos grupos: uno de aprendizaje y otro para test. De esta forma, se procederá al entrenamiento del ANS empleando solamente los ejemplos del conjunto de aprendizaje, mientras que se comprobará de vez en cuando el error que comete el ANS al ser aplicado sobre los casos del de test.

En la fase inicial del entrenamiento, el ANS extrae los rasgos generales de los patrones del conjunto de aprendizaje. Como el conjunto de test debe poseer una información parecida a la del conjunto de aprendizaje, a medida que transcurren las iteraciones se observa que los errores de los dos conjuntos van disminuyendo. De hecho, el error en el conjunto de aprendizaje siempre irá descendiendo (ya que ésta es la misión del algoritmo de entrenamiento) hasta llegar a su cota mínima, y raramente empeorará. En cambio, se observa experimentalmente que el error del conjunto de test alcanza un valor mínimo, después del cual empeora. A partir de ese momento, el ANS está memorizando los casos del conjunto de aprendizaje (incluso el ruido presente en ellos) y se está alejando de la tónica general, es decir, pierde capacidad de generalización (de extrapolar al conjunto de test las características del de aprendizaje). Este fenómeno se denomina **sobre-ajuste o sobreentrenamiento** (capítulo 2), que debe evitarse puesto que supone obtener malos resultados en la futura aplicación del ANS sobre casos nuevos. Detener el entrenamiento cuando comienza a empeorar el error en test constituye la técnica denominada validación cruzada. Una discusión más completa sobre este asunto se realizó en la sección 2.6.

Por último, hay que recalcar que los conjuntos de aprendizaje y de test deben ser por separado representativos de lo que sucede en nuestro problema, es decir, ambos deben cubrir en lo posible los espacios de entrada y de salida, abarcando todas las posibilidades que pueden darse y que hay que aprender.

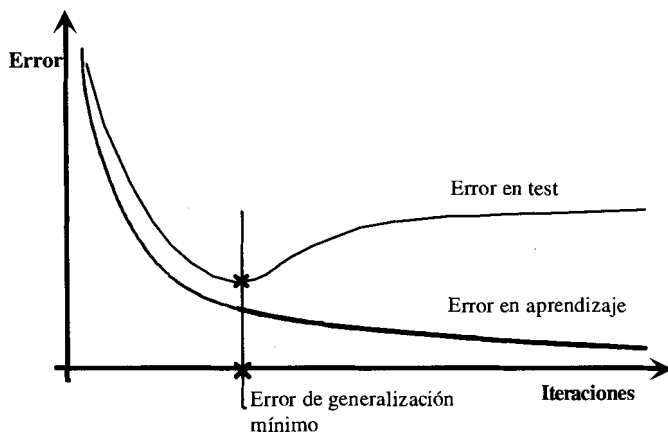


Figura 6.1. Evolución del error de aprendizaje y del error de test en función de las iteraciones, en un proceso de aprendizaje de una red unidireccional

(7) Preprocesamiento

Se denomina **preprocesamiento** al tratamiento previo de los datos de entrada y salida para adecuarlos a su tratamiento por la red neuronal. Para trabajar con ANS es muy aconsejable (aunque no imprescindible) conseguir que los datos que se proporcionan a la red posean las siguientes cualidades [Masters 93]:

- Buena distribución (es decir, sigan una distribución estándar o uniforme).
- Rangos de valores parecidos para todas las entradas.
- Rangos acotados dentro del intervalo de trabajo de la función de activación empleada en las capas ocultas y de salida del ANS.

Cuando indicamos que estos requisitos son aconsejables aunque no imprescindibles nos referimos a que si se trabaja con unos datos que no cumplen lo anterior el ANS podrá, en algunos casos, seguir dando buenos resultados, pero invertirá un tiempo uno o dos órdenes de magnitud mayor en conseguirlo. Si se desea cambiar el tipo de **distribución de los datos**, es preciso aplicar una **transformación**; el interés de ello radica en no enmascarar información: puede que moviéndonos en un pequeño intervalo de cierta entrada se produzcan variaciones importantes en la salida del ANS, mientras que en otro intervalo mayor las variaciones pueden no afectar tanto a la salida. Las transformaciones más frecuentes se basan en el empleo de logaritmos, raíces cuadradas y cúbicas.

Para modificar el **rango de valores** de las entradas se aplica un **escalado**. El objetivo es acotar sus valores entre $[0,+1]$ o $[-1,+1]$, límites de la función de activación, para evitar la saturación de ésta al realizar la suma ponderada de las entradas. Para ello se busca el mínimo y máximo del rango de las entradas y se transforma el intervalo de forma lineal al comentado anteriormente. Hay dos formas de obtener los factores de escalado con los que aplicar esta transformación (comentamos el caso $[0,+1]$):

- **Escalado por patrones (por filas)**, relativa a los valores del resto de las entradas de cada ejemplo (es decir, buscando el máximo y mínimo entre las entradas de cada ejemplo y asignándoles 1 y 0 respectivamente, y escalando en función de ello los restantes). Con este esquema cada ejemplo tendrá forzosamente al menos un uno, un cero y valores intermedios.
- **Escalado global por entradas (por columnas)**, relativa a los valores de cierta entrada para todos los ejemplos (análogo al anterior, pero fijándose en los valores de cierta entrada). En este caso nos encontraremos con ejemplos sin ceros ni unos.

El primer esquema tiene la ventaja de poner de manifiesto relaciones entre las entradas, en cambio, resulta catastrófico si las entradas tienen distinto rango de

variación (por ejemplo, la primera entrada entre 1000 y 1500, la segunda entre 0.01 y 0.02 y la tercera entre 0.0001 y 0.0005: la primera siempre obtendría el 1, la tercera el 0, y la segunda un valor cercano al 0, con lo cual anulamos toda la información). Por lo tanto, éste sólo se debe aplicar cuando las entradas a las que se refieren toman valores en intervalos parecidos y hay interés en mantener los valores relativos entre ellas. En el resto de los casos, se debe emplear el escalado global *por columnas*. No hay ningún inconveniente para aplicar en una misma ANS los dos tipos de escalados a la vez, uno a ciertas entradas y el otro a las demás.

Un escalado que proporciona con frecuencia buenos resultados es el de tipo global por columnas, aplicando dos transformaciones consecutivas:

- 1) En primer lugar se realiza un *estandarizado*, es decir, se transforman las variables de forma que presenten media cero y varianza unidad. Para ello, se calcula de la manera usual la media ($\bar{x}$) y la desviación estándar (σ) de cierta entrada, y se aplica la expresión

$$x' = \frac{x - \bar{x}}{\sigma} \quad (6.1)$$

- 2) A continuación, pueden escalarse los nuevos valores al intervalo $[0,+1]$ o $[-1,+1]$ mediante una transformación lineal.

Mediante este escalado conseguimos que todas las entradas tengan la misma importancia al igualar su variabilidad (gracias al estandarizado) y su rango de valores, sin importar el rango y distribución inicial.

Existen **otros tipos de preprocesamiento** más complejos, como la transformada de Fourier, análisis de componentes principales o técnicas de extracción de rasgos (*feature extraction*). Algo tan sencillo como cambiar las entradas por otras que sean cocientes de ciertas variables (ratios) puede dar muy buenos resultados.

Buena parte de lo comentado para las entradas se aplica también a las salidas de la red neuronal, pues también debería adecuarse su distribución, variabilidad y rangos de valores.

(8) Proceso de entrenamiento

Éste es el punto determinante de todo el desarrollo de la aplicación. En este proceso existe una interacción entre los dos conjuntos de patrones (aprendizaje y test), la estructura del ANS y el experimentador. Normalmente (excepto en las arquitecturas de tipo evolutivo, en las que automáticamente aparecen y desaparecen neuronas) se debe ensayar con diferentes topologías y con distintos parámetros de aprendizaje, midiendo el error de aprendizaje y el de generalización, hasta alcanzar los deseados. En el caso de no obtener unos resultados aceptables, habrá que volver a revisar alguna de las fases anteriores: puede que los conjuntos de aprendizaje y test no sean representativos, que no se hayan elegido bien las variables, o que éstas no hayan sido

preprocesadas o escaladas adecuadamente. Puede suceder también que no se haya elegido el modelo de red más apropiado o que su resolución con ANS no sea viable.

Los parámetros de la red que pueden modificarse en los experimentos son los siguientes: pesos iniciales, ritmo de aprendizaje, número de neuronas ocultas y parada del entrenamiento. A continuación comentaremos brevemente cada uno de ellos; se recomienda al lector la lectura de las secciones 2.5 y 2.6, donde se expusieron aspectos de este tipo para el caso concreto del MLP-BP.

- **Inicialización de los pesos.** El método más extendido consisten en realizar una inicialización aleatoria en un cierto intervalo, como por ejemplo, entre -0.3 y +0.3; recuérdese que en la sección 2.5.3 indicamos otras heurísticas para el caso concreto del MLP. No obstante, existen otros enfoques de inicialización más inteligente que la meramente aleatoria; se trata de aplicar algoritmos de minimización de funciones a esta elección de pesos iniciales, tal es el caso de técnicas como **regresión lineal**, **simulated annealing** y **algoritmos genéticos** [Masters 93].

- **Ritmo de aprendizaje.** Este parámetro desempeña un papel crucial en el proceso de entrenamiento del ANS, ya que controla el tamaño de los cambios en los pesos de las neuronas. Toma valores muy dispares, desde 0.5 a 0.00001 y aun menor (uno de los motivos de esta variabilidad reside en que en el aprendizaje *batch*, al acumular el cambio en los pesos propuesto por cada patrón, se obtiene un cambio mayor cuantos más patrones se tengan en el conjunto de aprendizaje). Elegir un ritmo de aprendizaje demasiado pequeño implica que el ANS realiza cambios pequeños en sus pesos, lo cual es perjudicial en dos sentidos: evita escapar de mínimos locales y disminuye la velocidad de convergencia. Por otra parte, decidirse por un valor alto puede ocasionar grandes variaciones en los pesos, lo cual puede conducir a inestabilidades en el ANS o a saturar sus neuronas. La conclusión es clara: interesa un ritmo de aprendizaje que varíe durante el entrenamiento, adaptándose a las necesidades. Hay muchos enfoques de este tipo, desde simples reglas heurísticas hasta un control mediante reglas borrosas de su valor. Al margen de estas técnicas más sofisticadas, se recomienda, en general, emplear el mayor ritmo de aprendizaje que no provoque excesivas oscilaciones en los errores mostrados por el ANS, modificándolo a mano si es necesario.

- **Neuronas ocultas.** Se trata de uno de los puntos más críticos del desarrollo, pues no existe una receta que indique el número de neuronas ocultas que para un problema dado deben emplearse. Gracias a algunos resultados, como el de [Funahashi 89], se tiene constancia teórica de que basta con una capa oculta para resolver cualquier problema de aproximación funcional con un perceptrón, aunque en ocasiones se recurra a dos por cuestiones prácticas. Debe recordarse (capítulo 2) que si se coloca un número excesivo de neuronas ocultas ocurrirá algo parecido a cuando en el ajuste de unos datos experimentales a un polinomio empleamos uno de grado demasiado elevado: sobran grados de libertad, y aunque ajusten perfectamente los

casos que nos proporcionan (conjunto de aprendizaje), nos apartamos de la tónica general y fallamos ante nuevos casos (conjunto de test). Por otro lado, si el número de neuronas no es suficiente no obtendremos un error aceptable ni siquiera para los datos que queremos ajustar. Para decidir el número de neuronas ocultas puede recurrirse a técnicas como las siguientes:

- **Recetas** [Masters 93]. Unas son de tipo geométrico, como por ejemplo que el ANS tenga aspecto de pirámide (el número de neuronas en cada capa va decreciendo desde las entradas hacia las salidas). Otras imponen condiciones relativas al número de patrones disponibles, intentando ajustar los grados de libertad a la cantidad de ejemplos, en aras de una buena generalización (se sugiere que el número de pesos de la red debe ser del orden de la décima parte del de patrones [Baum 89], capítulo 2). No obstante, ninguna de estas reglas es infalible.
- **Prueba y error**. Partiendo de un número sumamente pequeño de neuronas ocultas (por ejemplo, dos), se procede a realizar el entrenamiento con validación cruzada. El proceso se repite para distintas arquitecturas, cada vez con más neuronas ocultas, hasta llegar a la arquitectura que proporciona el resultado óptimo para los de aprendizaje y de test.
- **Métodos dinámicos** (arquitecturas evolutivas) [Refenes 95, Jutten 95, IEEE 99]. Consisten en que sea el propio algoritmo el que, en función de los ejemplos de aprendizaje, ajuste la arquitectura de la red mediante la creación, destrucción o compartición de neuronas o conexiones durante el entrenamiento (o al término de éste). La introducción de estas técnicas complica el algoritmo de aprendizaje, garantizando su convergencia pero disminuyendo su velocidad. En cambio, no asegura un determinado grado de generalización; además, alguna de estas técnicas requiere un ajuste cuidadoso de sus parámetros de control. La ventaja principal es que evita la clásica prueba/error para encontrar la topología ideal del ANS.
- **Parada del entrenamiento**. Tal y como se ha explicado en el método de validación cruzada, no interesa prolongar indefinidamente el entrenamiento, pues llega un momento en el que se pierde generalización y tan sólo se memorizan los detalles (ruido) del conjunto de aprendizaje. La decisión habitual es quedarse con los pesos del ANS en la iteración para la cual se obtuvo el mínimo error en el conjunto de test (véase la sección 2.6).

(9) Evaluación de los resultados

Finalizada la fase de entrenamiento y almacenados los pesos ideales, ya se está en disposición de aplicar el ANS sobre casos nuevos (no empleados en el entrenamiento) para medir su eficacia de forma completamente objetiva. Si se comprueba que se siguen obteniendo resultados dentro del margen de error deseado, se puede proceder a emplear el ANS dentro de su entorno de trabajo real.

6.3 PROGRAMAS DE SIMULACIÓN DE ANS

En el desarrollo de un ANS se hace uso de programas de simulación, que podemos clasificar en **tres tipos**: i) los **comerciales**, desarrollados por empresas y puestos a la venta; ii) los **de libre distribución**, realizados por grupos de investigadores de Universidades o por particulares, que se pueden obtener de forma gratuita a través de Internet (salvo si se trata de programas *shareware*, en cuyo caso hay que pagar una pequeña cuota); por último, iii) los **de producción propia**, que cada cual puede confeccionar.

A la hora de decidirse por determinado programa, sea comercial o gratuito, hay que buscar un producto que se adapte a nuestras necesidades o preferencias. Debemos averiguar las plataformas y sistemas operativos para los que se encuentra disponible (MS-Windows, UNIX, MacIntosh), los modelos de ANS y reglas de aprendizaje que contempla, el enfoque del programa (introductorio y genérico, o bien indicado para desarrollo de aplicaciones *serias*), la forma de especificar la red neuronal (interfaz gráfica orientada a ventanas y ratón, o lenguaje de descripción), si permite la generación de ejecutables o código fuente con los que reproducir más tarde la fase de recuerdo o de entrenamiento, y si admite la inclusión de rutinas externas de usuario.

Hay que destacar que **los programas de simulación resultan un excelente banco de pruebas**, ya que permiten explorar de forma rápida y gráfica las distintas arquitecturas de ANS hasta encontrar la que mejor se adapta al problema. No obstante, al estar diseñados para interactuar gráficamente con el usuario y ser de propósito general, suelen resultar más lentos que un programa desarrollado específicamente por uno mismo.

A continuación presentaremos algunos ejemplos de programas de simulación; además, en el CD adjunto (véase el apéndice A) se incluye la versión de demostración de uno de ellos (NeuroSolutions). Remitimos al lector interesado en una lista de programas más exhaustiva al fichero FAQ (*Frequently Asked Questions*) sobre redes neuronales disponible en <ftp://ftp.sas.com/pub/neural/FAQ.html>, en el que se realiza una revisión del software neuronal disponible (periódicamente actualizada). También aparecen revisiones de software en textos como [Arbib 98, Hammerstrom 93a].

6.3.1 Programas comerciales

Las ventajas de los programas comerciales son los entornos gráficos intuitivos y fáciles de usar que presentan, el elevado nivel de prestaciones que ofrecen, la disponibilidad de productos para aplicaciones especializadas (como reconocimiento de imágenes, predicción de series temporales, etc.), la posibilidad de cursos de formación a usuarios y apoyo por parte de la empresa y, en algunos casos, el gran número de modelos de redes que permiten simular y la existencia de tarjetas aceleradoras controladas a través de ellos. En el caso de los productos de cierta solera en el mercado, se podrían añadir además la tranquilidad de manejar versiones mejoradas y

la seguridad de futuras actualizaciones. El principal inconveniente es su elevado precio (aunque ofrecen descuentos para fines educativos) y, con frecuencia, el tratarse de productos cerrados en los que no se pueden manipular los modelos según el interés del usuario. Los siguientes son algunos de los paquetes software de uso más extendido disponibles hoy en día en el mercado:

- **NeuralWorks Professional II Plus** (*NeuralWare*, www.neuralware.com). Es uno de los programas clásicos y más completos, muy empleado desde hace años. Permite el trabajo con más de treinta modelos de ANS con diferentes reglas de aprendizaje (BP, ART, SOM, GRNN, PNN, LVQ, RBF...) y contempla muchos tipos de visualizaciones gráficas; permite además incorporar funciones externas en C. NeuralWorks comercializa otras herramientas más especializadas para redes neuronales, que se describen en su página web.
- **MATLAB Neural Network Toolbox** (*The MathWorks Inc.*, www.mathworks.com). MATLAB® es un entorno matemático que se ha convertido casi en un estándar en el campo de la ingeniería; en particular, son bien conocidos sus *toolboxes* (cajas de herramientas) que simplifican enormemente el trabajo en campos como el procesamiento de señal e imagen, identificación y control de sistemas, etc. Las *toolboxes* para redes neuronales han tenido una excelente acogida, estando bastante extendido su uso tanto en la enseñanza como en investigación. Contempla un variado número de ANS (BP, RBF, Elman, asociativas, SOM, LVQ, etc.), y al tratarse de rutinas en código fuente el usuario puede adaptarlas a sus necesidades (su gran ventaja); por contra, al trabajar en modo interpretado se pierde algo de velocidad. Hablaremos más sobre MATLAB en la parte dedicada a los sistemas borrosos.
- **NeuroSolutions** (*NeuroDimension Inc.*, www.nd.com). Se trata de un producto de gran calidad, modular, visual y fácil de usar. Posee una interfaz gráfica muy vistosa orientada a objetos (bloques funcionales), lo cual facilita la creación de nuevos tipos de ANS. Incluye MLP, Modular, Jordan, Elman, SOFM, PCA, RBF, *Time-Lag* y recurrentes. Además, permite la generación de código C++ para la fase de recuerdo y la de aprendizaje, y próximamente contará también con módulos neuroborrosos. Una **versión demostración** de este programa se incluye en el CD-ROM adjunto, y puede encontrarse una explicación a fondo en [Principio 00].
- **BrainMaker** (*California Scientific Software*, <http://www.brainmaker.com/>). Es uno de los programas más famosos, aunque se basa tan solo en el algoritmo BP, pero incluyendo variantes (como recurrencias) y contemplando técnicas diversas, como validación cruzada, análisis de sensibilidad, optimización por algoritmos genéticos, podado de pesos, etc. En tiempos, este programa permitía el trabajo con tarjetas aceleradoras o neuroemuladores (como CNAPS, capítulo 5) conectadas al bus del PC, obteniéndose así velocidades de procesamiento del orden de 1000 veces mayores que con simulación en PC.

Existen muchos otros simuladores comerciales; en el documento FAQ sobre redes neuronales (<ftp://ftp.sas.com/pub/neural/FAQ.html>) se describen cerca de cuarenta; allí remitimos al lector interesado en otros productos.

6.3.2 Programas de libre distribución

La principal ventaja de estos programas es que están disponibles de forma gratuita a través de Internet, proporcionando en la mayoría de los casos material de valor incalculable, como códigos fuente, manuales, ejemplos y, en ocasiones, incluso atención por correo electrónico para las dudas que puedan surgir sobre su instalación o manejo. A diferencia de los productos comerciales, los programas gratuitos se pueden permitir el lujo de tratar modelos más minoritarios (por ejemplo, el neocognitron, modelado biológico, redes con consciencia, etc.). A continuación pasamos a comentar dos ejemplos representativos de simuladores, uno general y otro específico:

- **SNNS** (*Stuttgart Neural Network Simulator*). Se trata de un simulador de carácter general desarrollado en la Universidad de Stuttgart (en la actualidad es mantenido en la Universidad de Tübingen), que ha tenido una elevada aceptación y repercusión (incluso existen grupos de usuarios). Contempla muchos modelos: BP (con infinidad de variantes), contra-propagación, *quickprop*, *backpercolation* 1, RBF, Rprop, ART1, ART2, ARTMAP, *Cascade Correlation*, *Recurrent Cascade Correlation*, *Dynamic LVQ*, BP *Through Time*, Hopfield, Jordan/Elman, autoasociativas, SOFM, TDNN, etc. Es ampliable por el usuario (funciones de activación, algoritmos de aprendizaje), genera código C e incluso permite repartir una simulación en un cluster de estaciones de trabajo. Inicialmente se desarrolló para estaciones UNIX, pero existe versión para PC (si bien, al parecer su instalación no resulta sencilla en el entorno PC). Viene acompañado por unos manuales de implementación y de usuario muy extensos y claros.

<http://www-ra.informatik.uni-tuebingen.de/SNNS/>

<http://inf.informatik.uni-stuttgart.de:80/ipvr/bv/projekte/snns/snns.html>

- **SOM_PACK y LVQ_PACK**. Se trata de dos simuladores de carácter específico desarrollados por el grupo del profesor Teuvo Kohonen en la Universidad Tecnológica de Helsinki. Son dos simuladores sobrios, pero muy rigurosos y completos, con una excelente documentación de consulta casi obligada para los que trabajan con estos modelos.

<http://www.cis.hut.fi/research/som-research/>

Más de cuarenta simuladores gratuitos adicionales se describen en el documento FAQ sobre redes neuronales ya citado (<ftp://ftp.sas.com/pub/neural/FAQ.html>).

6.4 COMPARACIÓN CON OTRAS TÉCNICAS

Una vez que el lector ya conoce los fundamentos de los ANS y los pasos que debe seguir para su aplicación, consideramos que es un buen momento para comentar brevemente las relaciones de los ANS (y del *soft computing* en general) con otras herramientas más clásicas relacionadas con el **procesamiento de información inteligente**: nos referimos especialmente a la inteligencia artificial (IA) y a la estadística. Concluiremos este punto realizando un análisis crítico de los mayores **inconvenientes que presentan los ANS**.

6.4.1 Redes neuronales e inteligencia artificial

Para la realización de esta sección nos hemos inspirado especialmente en las referencias [Eberhart 90, Barnden 98]. Podríamos definir la **inteligencia artificial (IA)** como un conjunto de algoritmos cuyo objetivo es imitar el razonamiento humano a través de una **lógica deductiva o manipulación de símbolos**. A diferencia de la inspiración biológica de los ANS (*cómo* funciona el cerebro), la IA profundiza en los modelos del cerebro provenientes de la psicología más tradicional (*qué* hace el cerebro sin importarle el *cómo*). Ambas técnicas tienen en común la aspiración de emular la naturaleza de la inteligencia humana, pero tiene que quedar claro que los ANS (o la *soft computing* en general) no pertenecen de lleno al campo de la IA tradicional, sino que presentan una visión más multidisciplinar y flexible sobre la inteligencia.

Uno de los productos más conocidos y potentes fruto de la investigación en IA son los llamados **sistemas expertos**. Como ya se ha comentado, éstos consisten en programas de ordenador que tratan de aprehender los conocimientos que un experto dispone sobre un campo muy concreto (concesión de créditos, diagnóstico de enfermedades de la piel, etc.). Un sistema experto incluye dos partes: un **motor de inferencia** (*inference engine*) y una **base de conocimiento** (*knowledge base*). En el primero están programadas las manipulaciones genéricas de tipo lógico que se van a aplicar, mientras que en el segundo está almacenada la información que se va a emplear en el problema (reglas y parámetros).

Los razonamientos se obtienen aplicando las reglas almacenadas en la base de conocimiento a la situación que se plantee, haciendo uso del motor de inferencia. Por tanto, existe una separación clara entre algoritmo (motor de inferencia) y memoria (datos almacenados en la base de conocimiento). En cambio, en los ANS el conocimiento emerge de las interconexiones de la estructura de la red de procesadores elementales (neuronas); por este motivo se ha venido en llamar **inteligencia computacional**. La intensidad de dichas conexiones se obtiene de un proceso de entrenamiento a partir de ejemplos, de modo que la *inteligencia* de los ANS no hay que programarla, sino que *emerge* del entrenamiento.

Para desarrollar un sistema con técnicas de IA se requieren dos esfuerzos enormes (y costosos en tiempo y dinero): uno de recopilación de reglas a partir del

conocimiento de los expertos en la materia, y otro esfuerzo posterior de codificación de estas reglas (muchas veces de tipo cualitativo) en la base de conocimiento. Cuanto más complejo sea el problema a modelar, mayor será el número de reglas que tendremos que suponer (o inventar) y de mayor grado de complejidad, de forma que la descripción del problema se pueda considerar aceptable. Por este motivo, la respuesta del sistema experto será lenta, puesto que deberá atravesar por unos frondosos árboles de decisión.

Sin embargo, **los tiempos de respuesta de un ANS son normalmente muy pequeños** (milisegundos), incluso si se simulan en un computador; lo costoso en este caso es el aprendizaje, pero suele realizarse tan sólo una vez. Además, se ha demostrado que los ANS son *solucionadores universales de problemas*, que en principio no requieren de un estudio tan profundo del proceso.

Así, en aquellos problemas complejos para los que no existen una serie de reglas que los describan y que implican el procesado masivo de información, los sistemas expertos se han mostrado incapaces de dar buenos resultados, mientras que los ANS han cosechado grandes éxitos y con un gasto mucho menor de tiempo y recursos (veremos muchos ejemplos en la sección 6.5).

Después de esta breve comparación entre IA y ANS, volveremos a destacar los diferentes enfoques de ambas técnicas: la IA opera de arriba hacia abajo (enfoque **descendente**) con reglas, conceptos y cálculos secuenciales, tal y como al parecer trabaja nuestra mitad izquierda del cerebro; mientras que los ANS, operan de abajo a arriba (enfoque **emergente**), interpretando de manera *intuitiva* y paralela las imágenes, sonidos y otros tipos de estímulos que llegan de forma masiva desde el exterior, como el lado derecho del cerebro.

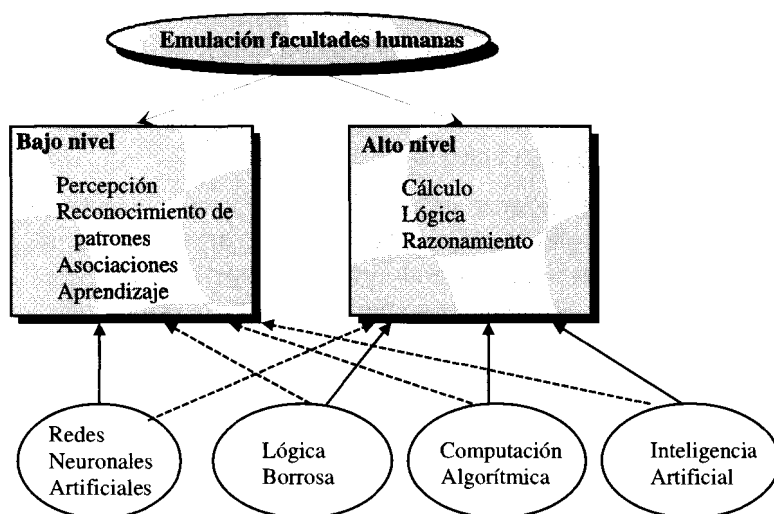


Figura 6.2. Relaciones entre las distintas técnicas

Por todo lo anterior, se puede concluir que la IA y los ANS representan enfoques diferentes para resolver problemas, a veces diferentes, otras similares. En realidad **no debe pensarse en ambas como tecnologías excluyentes, sino más bien complementarias**; los mejores resultados se obtendrán aplicando cada técnica (procedente de la IA, de los ANS, de la estadística o del procesamiento de señal) al aspecto del problema donde resulte más conveniente [Fogelman 98].

Por último, nos hemos centrado en esta sección en la comparación de redes neuronales y sistemas expertos. Sin embargo, existe otra técnica novedosa que también trata de resolver problemas similares, los sistemas basados en **lógica borrosa**, orientados a reglas como los sistemas expertos, pero de rapidez de cálculo semejante (o mayor) a las redes neuronales. Los sistemas borrosos resultan un excelente complemento a ambas técnicas, existiendo en la actualidad la clara tendencia de incorporar en un mismo sistema redes neuronales, lógica borrosa, sistemas expertos y técnicas estadísticas, en todas sus posibles combinaciones. Los sistemas borrosos serán tratados con mayor extensión en la segunda parte de este texto.

6.4.2 Redes neuronales y estadística

La estadística comprende un conjunto de métodos que sirven para recoger, organizar, resumir y analizar datos, así como para extraer conclusiones y tomar decisiones razonables basadas en tal análisis. Las redes neuronales quizá tengan muchos más aspectos en común con la estadística que con la IA, de hecho, los ANS han sido descritos en alguna ocasión como **técnicas de ajuste estadístico inspiradas en la biología**. Modelos procedentes de ambas disciplinas se emplean en ajuste funcional (perceptrón y regresión), en reducción de dimensionalidad (mapas de Kohonen y análisis de componentes principales) y otras tareas.

Se han realizado estudios comparando métodos estadísticos y neuronales, como el de [Croall 92], llegándose a la conclusión de que no se puede realizar la afirmación genérica de que los modelos neuronales sobrepasen siempre en eficiencia a las técnicas estadísticas. Los expertos en estadística se apoyan en resultados como el anterior y acusan a las técnicas de ANS de reinventar la rueda. Por ejemplo, en [Sarle 94] se señala el claro paralelismo entre multitud de modelos estadísticos y neuronales, de los cuales mostramos algunos en la tabla 6.1. No obstante, el autor admite que hay algunos modelos de ANS para los que no encuentra una técnica estadística equiparable [Sarle 94]. En el mismo artículo se comenta que los expertos en estadística echan en cara a los ANS que sus algoritmos de entrenamiento son poco eficientes, muy lentos, no están bien explicados todavía, necesitan mucho ajuste heurístico de parámetros (ritmo de aprendizaje, número de neuronas ocultas, etc.).

Pensamos que este punto de vista es un tanto extremista. Ni las redes neuronales son tan excelentes como en algún momento se ha tratado de mostrar, ni poseen tantos aspectos negativos como algunos estadísticos sugieren (véase [Flexer 95] para un análisis curioso de la amistad entre expertos neuronales y estadísticos).

Ambas técnicas tratan de resolver en ocasiones problemas similares, de modo que resulta lógico que se llegue a soluciones semejantes; por este motivo, parece más adecuada la cooperación entre ambas que el enfrentamiento. En este sentido, los ANS pueden beneficiarse de la estadística en numerosos aspectos, como por ejemplo, el empleo de técnicas estadísticas para el análisis de la relevancia de las variables de entrada, su utilización en una inicialización más inteligente de los pesos o en el análisis de la operación de la red. Muy interesantes estudios comparativos de ambas técnicas aparecen en [White 89a, 89b], cuya lectura recomendamos.

Pero en concreto, ¿qué ventajas ofrecen los ANS respecto de las técnicas estadísticas? En primer lugar, los métodos neuronales, en su forma más básica, son relativamente fáciles de emplear y la interpretación de sus resultados resulta asequible a usuarios no necesariamente expertos en el tema. Por otro lado, las redes neuronales normalmente no suelen imponer presupuestos de partida, como tipo de dependencia funcional o distribución gaussiana de los datos. Otra ventaja es la rápida respuesta de un ANS simulado [Croall 92]; cuando se requiere una respuesta más rápida aún, la red neuronal puede incluso realizarse electrónicamente en circuitos específicos paralelos. Además, las redes neuronales, gracias a su posibilidad de entrenamiento en línea, pueden utilizarse para aplicaciones de control industrial, en las que los patrones van llegando uno tras otro, tarea imposible para una herramienta estadística en la que se requiere la presencia de todos los datos simultáneamente desde el principio.

Modelo neuronal	Técnica estadística
Perceptrón simple (nodo tipo umbral)	Análisis discriminante
Perceptrón simple (nodo sigmoideo)	Regresión logística
Adalina	Regresión lineal
Perceptrón multicapa	Regresión no lineal simple Regresión no lineal multivariada
Aprendizaje hebbiano no supervisado	Análisis de componentes principales
Red simple de Kohonen (competitiva)	Análisis cluster mínimos cuadrados
Cuantificación de vectores LVQ	Análisis discriminante (vecindad)
Funciones de base radial (RBF)	Métodos de <i>kernel regression</i>
Mapas de Kohonen	Escalas multidimensionales

Redes neuronales sin paralelismo claro
Contrapropagación
Redes ART
Modelo de Hopfield

Tabla 6.1. Algunas relaciones entre técnicas neuronales y estadísticas [Sarle 94]

Existe una última motivación para el trabajo con redes neuronales manifestada de manera humorística por H. White [White 89b], estadístico que lleva muchos años trabajando en redes neuronales: *la aplicación de ANS resulta mucho más amena y creativa que la de las técnicas estadísticas*. Bromas aparte, podemos indicar a modo de resumen, que las técnicas estadísticas y neuronales están muy relacionadas, que ningún campo es a priori mejor que el otro (en algunos casos ambos son indistinguibles incluso), y que lo razonable es aplicar en cada caso la solución más idónea.

6.4.3 Inconvenientes de las redes neuronales

Obviamente, los ANS no son la panacea que resuelve todos los problemas, sino que están más enfocados a un determinado tipo de tareas, como las comentadas al final de la sección 6.1. Además, el campo de los ANS es relativamente joven, y algunos de los siguientes **problemas** están aún por resolver:

- Constituyen un método de resolución de problemas **demasiado creativo**, es decir, dadas las especificaciones de un problema, se desconoce la topología (arquitectura) del ANS que la va a solucionar de forma más eficiente. Hay que utilizar el método de prueba y error.
- Es difícil averiguar por qué un ANS no es capaz de ajustar los datos que se le proporcionan. Por tanto, dado un problema al que le aplicamos ANS, puede resultar dificultoso averiguar **por qué no funciona** correctamente.
- Los modelos neuronales precisan elevados **requisitos de cómputo**. Si bien esta afirmación es cierta en modo aprendizaje, usualmente en fase de ejecución son los métodos más rápidos, incluso si se simulan en un computador.
- Una vez entrenada una red neuronal, en general resulta difícil interpretar su funcionamiento. Es el denominado comportamiento como **caja negra** de la red neuronal.

Este último punto es particularmente importante, ya que en las aplicaciones reales al usuario final le gustaría saber qué es lo que está haciendo exactamente la red neuronal, para de esta manera sentirse seguro de su operación, especialmente en el caso de aplicaciones críticas (por ejemplo, en el control de una central de energía eléctrica o pilotaje de un avión). Es ésta un área de intenso trabajo, en la que se trata de **extraer las reglas que ha aprendido la red neuronal en el proceso de aprendizaje**. A menudo se emplea para ello la equivalencia existente entre algunos modelos de redes neuronales unidireccionales y los sistemas basados en lógica borrosa [Jang 92-97, Benítez 97, Reyneri 99, Figueiredo 99, Li 00], aunque se han propuesto también otras técnicas no basadas en lógica borrosa [Tsukimoto 00, Setiono 00]. Este tema se tratará en el capítulo 9.

6.5 APLICACIONES REALES DE LOS ANS

Las primeras aplicaciones de redes neuronales a problemas reales se inician en los años sesenta; desde entonces se aplica el modelo de red lineal adalina como filtro adaptativo de manera rutinaria (por ejemplo, en los módems convencionales). Los modelos no lineales de redes neuronales permiten abordar un abanico de aplicaciones mucho más amplio, pero debido a los recursos de cómputo requeridos sus aplicaciones reales no surgen hasta finales de los años ochenta. Una de las primeras aplicaciones data de junio de 1989 [Shea 89], cuando la empresa norteamericana SAIC (*Science Applications International Corporation*) presenta un detector de explosivos en los equipajes facturados en los aeropuertos. Desde aquella fecha, las aplicaciones de ANS a problemas industriales y comerciales se han convertido en algo habitual.

En la actualidad, la aplicación de redes neuronales puede considerarse que ha alcanzado ya su **madurez** [Werbos 98], y en muchas ocasiones ya ni se cita que en determinado producto se emplean estas técnicas, por ser algo rutinario. Por ejemplo, la mitad de los sistemas para OCR (*Optical Character Recognition*, reconocimiento de letras y números impresos a partir de sus imágenes digitalizadas) emplean redes neuronales (pero no se hace propaganda de ello) [Werbos 98]. Otros ejemplos de aplicaciones actuales de redes neuronales son el aterrizaje automático de un avión Jumbo y el desarrollo por Ford del primer vehículo que cumple con el estándar de emisiones ultrabajas [Werbos 98, James 98]. En [IEEE 97] se describen detalladamente muchas otras aplicaciones reales, como la detección de fraudes en el empleo de tarjetas de crédito (VISA) o la predicción de consumo eléctrico. En la sección 6.5.2 proporcionaremos muchos más ejemplos concretos.

6.5.1 Informes sobre el estado de la aplicación de ANS

Sin duda **los Estados Unidos lideran el campo de las redes neuronales**, tanto en desarrollos teóricos como en aplicaciones industriales y comerciales. Como se destaca en el informe [NeuroNet 95], aunque Europa se encuentra bien posicionada en cuanto a investigación académica (prácticamente a la par que Estados Unidos y el sureste asiático), falla en el ámbito de la transferencia de los conocimientos teóricos hacia la empresa, por lo que se está perdiendo un área de negocio de grandes posibilidades. Por este motivo, se han promovido en los últimos años diversas **iniciativas a nivel europeo que tratan de salvar el espacio existente entre los grupos académicos y las empresas**.

Una de las primeras iniciativas fue la creación de la red de excelencia **NeuroNet**, *The European Network of Excellence in Neural Networks*, financiada por el cuarto programa marco de la Unión Europea, la cual agrupa a numerosos centros académicos y empresas europeas. La consulta de su página web es muy recomendable (<http://www.kcl.ac.uk/neuronet/>).

Otra iniciativa emprendida ha sido **SIENA** (*Stimulation Initiative for European Neural Networks Applications*) [SIENA 96], proyecto ESPRIT de la Unión Europea, cuyo objetivo era el análisis del estado de explotación comercial de las redes neuronales por las empresas europeas, y estimular el interés en estos temas entre las mismas. Este estudio se realizó durante año y medio por parte de seis grupos de cinco países europeos, estando España representada por el Instituto de Ingeniería del Conocimiento (<http://www.iic.uam.es/>). Los informes finales de SIENA están disponibles en Internet (<http://www.mbfys.kun.nl/snn/Research/siena/index.html> y <http://www.mbfys.kun.nl/snn/Research/siena/whitebook/siena/index.html>). En este estudio se identificaron unos 175 suministradores de tecnología neuronal en Europa, estimándose un mercado de unos 110 a 140 millones de Euros en 1996, con un crecimiento anual de un 20 a un 30%.

En el informe final de SIENA se incluyeron numerosos ejemplos de aplicaciones de redes neuronales a problemas reales por empresas europeas. En España se identificaron entonces una veintena de empresas que utilizaban estas tecnologías, y unas 14 suministradoras, siendo el área principal de aplicación el OCR. Las siguientes son algunas de las aplicaciones comerciales desarrolladas en España que aparecen en el informe final: detección de fraudes en el empleo de tarjetas de crédito de la empresa VISA, gestión del suministro de agua potable del Canal de Isabel II de Madrid y sistemas OCR para el procesamiento de los impresos de demandas de empleo para El Corte Inglés y para las órdenes de venta de Reebook.

En el año 1998 la fundación española de origen empresarial **COTEC** (<http://www.cotec.es/>), constituida para el fomento de la innovación tecnológica en la empresa española, dedicó a las redes neuronales el número 13 de la serie *Documentos COTEC Sobre Oportunidades Tecnológicas* [COTEC 98]. Este interesante estudio (en el que participamos nosotros) está disponible en Internet como documento PDF, e incluye una introducción a las redes neuronales y sus aplicaciones, una relación de grupos suministradores de esta tecnología en España, e incluso un resumen del informe SIENA.

6.5.2 Listado de aplicaciones

La mayoría de las aplicaciones comerciales actuales se basan en los siguientes modelos de redes neuronales: MLP, RBF, mapas autoorganizados y LVQ. A estas arquitecturas neuronales habría que añadir otras dos herramientas pertenecientes a la inteligencia computacional ampliamente utilizadas: los algoritmos genéticos y los sistemas borrosos (de ellos hablaremos más adelante). Los ejemplos de aplicación de redes neuronales que comentamos a continuación corresponden siempre a productos comerciales o a aplicaciones en la industria, que organizaremos en seis categorías. Muchos de estos ejemplos han sido extraídos de [Widrow 94], como representativos de las diversas aplicaciones de los ANS. No obstante, pueden encontrarse muchos ejemplos más recientes en las siguientes direcciones Internet:

<http://www.emsl.pnl.gov:2080/proj/neuron/neural/products/index.html>

<http://www.mbfys.kun.nl/snn/Research/siena/index.html>

<http://www.mbfys.kun.nl/snn/Research/siena/whitebook/siena/index.html>

<ftp://ftp.sas.com/pub/neural/FAQ.html>

(i) Redes neuronales lineales

Telecomunicaciones. Es en este campo donde los ANS han conseguido la aplicación industrial más significativa hasta nuestros días: se han utilizado para construir ecualizadores lineales y canceladores de ecos, que son empleados en los **módems** que trabajan con transmisión de alta velocidad. En ambos casos se emplea una red neuronal de una sola neurona. Así, millones de módems en todo el mundo contienen una adalina.

Anulación de ruido y vibraciones. Mediante una adalina, que desempeña el papel de actuador adaptativo, se generan ruidos y vibraciones opuestas a las recibidas, y de esta forma se consigue amortiguar los ruidos y vibraciones provocados por sistemas de aire acondicionado y grandes instalaciones industriales.

(ii) Clasificación de patrones

Fraudes con tarjetas de crédito. El fraude cometido con tarjetas de crédito es una cuestión preocupante y creciente que ha lanzado a varios bancos y compañías de tarjetas de crédito (como American Express, Mellon Bank y First American Bank) al uso de ANS en la detección de estos fraudes mediante el estudio de pautas en su uso fraudulento. En [IEEE 97] se puede encontrar descrito el caso de VISA.

Reconocimiento de caracteres impresos (OCR). Existen un gran número de productos comerciales desarrollados con ANS dedicados a OCR [Hammerstrom 93]. Entre estas empresas cabe destacar Sharp Corp., Mitsubishi Electric Corp., VeriFone Inc., Hecht-Nielsen Corp. (HNC), Nestor Inc., Calera Recognition Systems Inc., Caere Corp. y Audre Recognition Systems. Es éste uno de los campos en los que los ANS han cosechado un mayor éxito. Parte de estos productos están basados en software (por ejemplo, el OCR de Sharp, empleado para reconocer caracteres japoneses mediante LVQ, o el de Synaptics, para caracteres chinos), y otros en circuitos integrados específicos (como el lector de números de talones bancarios Onyx Check Reader desarrollado por VeriFone, donde se emplea un chip neuronal analógico diseñado por Synaptics, o el ya citado Quicktionary, de Wizcom, escáner con forma de bolígrafo, traductor y lector, basado en OCR-on-a-chip de Ligature).

Reconocimiento de caracteres manuscritos. La empresa HNC comercializa un producto de ANS llamado Quickstrokes Automated Data Entry System mediante el

cual se pueden leer formularios rellenos a mano. Este producto se está utilizando actualmente en el servicio de pedidos de Avon y en las oficinas recaudadoras de impuestos del estado de Wyoming. La motivación para esta última aplicación era que el retraso en el ingreso de talones bancarios ocasionaba unas pérdidas anuales de 300.000 dólares que se habrían percibido de otro modo en forma de intereses. La empresa Nestor tiene en el mercado un producto de software de ANS llamado NestorWriter de idéntica utilidad.

Reconocimiento del habla. Se están realizando numerosos trabajos en este sentido. Por ejemplo, Sensory Circuits (véase el capítulo 5) comercializa desde 1995 la serie de circuitos integrados para reconocimiento de habla RSC, que implementan una combinación de algoritmos neuronales y estadísticos en un único chip. Su reducidísimo precio (menos de 5 dólares si es adquirido en grandes cantidades) hace que sea empleado en, por ejemplo, electrodomésticos y juguetes. Es éste uno de los aspectos más interesantes tanto de los ANS como de la lógica borrosa: poder incorporar inteligencia en sistemas de reducido tamaño y coste.

Control de calidad. Se aplican en muchos métodos de control y aseguramiento de la calidad en las industrias. Algunos ejemplos son la detección de niveles de contaminantes a partir de datos espectroscópicos en plantas químicas [Kestelyn 90], la clasificación de anomalías en altavoces [AI expert 90], y la evaluación del grado de pureza del zumo de naranja. La multinacional europea Philips usa redes neuronales en la evaluación de ruido en radiocassettes de automóvil [Nelson 99].

Detección de sucesos en aceleradores de partículas. En los grandes laboratorios de física de altas energías (CERN de Ginebra [Flam 92], y FermiLab de Batavia, Illinois) se están empleando circuitos integrados analógicos de ANS para decidir en tiempo real sobre la relevancia de la enorme cantidad de sucesos que son detectados (<http://www1.cern.ch/NeuralNets/nnwInHepHard.html>). Esta técnica ha demostrado ser especialmente útil y barata en aquellos experimentos para los que se emplean criterios de selección de sucesos complejos.

Prospecciones petrolíferas. Algunas compañías petrolíferas (Arco, Texaco) están empleando ANS para determinar la localización de bolsas subterráneas de gas y petróleo [Schwartz 92].

Lucha contra el tráfico de drogas. La Agencia de Información del estado de Carolina del Norte emplea un software de ANS en un PC que ayuda a un grupo de forenses a identificar muestras de cocaína procedentes de una cierta red de tráfico [Casale 93].

Aplicaciones médicas. La empresa Neuromedical Systems Inc. comercializa productos para realizar exploraciones de cáncer y otros chequeos. Por ejemplo, el denominado Papnet que ayuda a los citólogos a marcar las células cancerosas,

reduciendo de forma dramática los errores en esta clasificación. Actualmente, este sistema está siendo usado por el Ministerio de Alimentación y Drogas de los Estados Unidos [Fuochi 92].

(iii) Predicción y análisis financiero

Una referencia imprescindible en este campo es [Refenes 95]. Exponemos a continuación algunos ejemplos concretos tomados de otras referencias.

Concesión de préstamos. El entorno Creditview, desarrollado en el Chase Manhattan Bank [Marose 90], evalúa los riesgos que corre la entidad financiera al conceder un préstamo a un determinado cliente.

Análisis de mercados. La empresa Churchill System ha desarrollado un sistema denominado Target Marketing System, que es empleado actualmente por Veratex Corp. para optimizar estrategias de mercado y reducir gastos de publicidad a través del descarte de clientes poco rentables [Hall 92]. Algo parecido hace la editorial Spiegel Inc., la cual ha diseñado un sistema (empleando el software de simulación de NeuralWare Inc.) para determinar qué clientes deben seguir recibiendo o empezar a recibir sus catálogos de pedidos. El director de marketing de esta empresa afirma que con esta técnica esperan ahorrarse un millón de dólares anuales [Schwartz 92].

Reservas de vuelos. El sistema The Airline Marketing Assistant/Tactician desarrollado por BehavHeuristics Inc., utiliza ANS para predecir la demanda de billetes. Está siendo empleado por Nationair Canada y USAir [Hall 92].

(iv) Control y optimización

Siemens viene aplicando redes neuronales y lógica borrosa al control industrial desde hace tiempo (líneas de galvanizado, laminadoras e incluso OCR), y sus autómatas programables cuentan con módulos neuro-borrosos. Presentaremos a continuación algunos otros ejemplos de aplicaciones a control de procesos:

Fabricación de celulosa y papel. Siemens ha aplicado redes y *fuzzy* a la optimización del proceso de fabricación de celulosa en la planta de Celulose do Caima, en Portugal [Höhfeld 93, Poppe 95]. En [Edwards 99] se muestra también un ejemplo de aplicación de ANS a la industria del papel por Tullis Russell & Co (UK).

Hornos de fundiciones. El producto Intelligent Arc Furnace, desarrollado por Neural Applications Corp. [Fuochi 92], posiciona de forma precisa el electrodo de un horno de arco de voltaje empleado en la fundición de acero. Según esta empresa, los ahorros conseguidos al aplicar esta nueva técnica de control se cifran en un millón de dólares anuales por horno, al aumentar su rendimiento y reducir el consumo eléctrico

y el desgaste del electrodo. Este sistema de control se está instalando actualmente en hornos de todo el mundo.

Industria de semiconductores. La empresa Kopin Corp. ha empleado ANS para controlar el espesor de la capa de dopantes y su concentración, con lo que ha conseguido reducir los defectos en la fabricación de células solares en más de un factor dos. Intel también aplica redes neuronales en la manufactura de circuitos integrados [May 94]. En este caso se aprovecha la capacidad del BP de ajuste no lineal a partir de muestras experimentales entrada-salida, para modelar, monitorizar o controlar los procesos altamente no lineales relacionados con la manufactura de semiconductores (deposición, ataque por plasma, etc.).

Control de procesos químicos. Los ANS están siendo empleados en la industria química [Ferguson 92] para reducir residuos, aumentar la calidad del producto, incrementar el rendimiento de la planta, determinar el punto de operación de cierto proceso, predecir rendimientos y detectar fallos. Tal es el caso del producto de Pavilion Technologies denominado Process Insights, que es utilizado por empresas como Eastman Kodak.

Refinería de petróleo. La refinería Texaco's Puget Sound Refinery, que produce 120.000 barriles de crudo al día, ha integrado ANS en el proceso de control de su planta. El papel del ANS es proporcionar estabilidad a procesos que duran casi un día, para los que se requieren condiciones muy precisas de temperatura, presión y flujos, tal y como se cita en el número de junio de 1990 de AI Expert.

(v) Aplicaciones militares

Guiado automático de misiles. En el U.S Naval Air Warfare Center se emplean ANS analógicas para el guiado automático de misiles y otras aplicaciones militares [Shandle 93], puesto que han llegado a la conclusión de que los ANS presentan enormes ventajas cuando se requiere una rápida toma de decisiones.

Combate aéreo. Se han desarrollado sistemas compuestos por ANS que analizan datos de situaciones de vuelo normal y de combate para poder proporcionar ayuda al piloto en el manejo del avión ante inminentes acciones enemigas [Schwartz 92b]. Esta aplicación se realizaba sobre el avión de combate táctico avanzado Lockheed's YF-22.

(vi) Otras aplicaciones

Predicción. Es un campo en el que las redes neuronales proporcionan excelentes resultados, por ejemplo, en la predicción de consumo eléctrico [García 95, Martín del Brío 95a, IEEE 97], gas [Khotanzad 00], flujo de ríos [Atiya 99], etc.

Máquinas fotocopadoras. La compañía Ricoh Corp. emplea con éxito técnicas de ANS para mantener constante la calidad de la copia de sus máquinas fotocopadoras. Dicha calidad depende de forma compleja y altamente no lineal de una serie de parámetros que le son proporcionados al ANS (temperatura, humedad, tiempo de inactividad entre copias, tiempo transcurrido desde la reposición del cartucho de tóner, etc.). El algoritmo empleado es, cómo no, el popular BP, con técnicas de podado de pesos.

Fallos en motores eléctricos. La empresa Siemens ha desarrollado un sistema neuronal capaz de predecir, con mucha fiabilidad y bajo coste, los fallos de grandes motores de inducción [Shandle 93]. El porcentaje de aciertos en dicha predicción resulta ser de un 80% a un 90%, frente al 30% obtenido anteriormente con otras técnicas.

Conducir camiones. El procedimiento para conducir marcha atrás un camión con remolque es un problema altamente no lineal. En este contexto se empleó un ANS para que aprendiera por sí mismo a conducir hacia atrás el vehículo con su remolque, empleando datos de la simulación por ordenador de un camión [Nguyen 92]. El ANS fue capaz de manejar el vehículo independiente de las condiciones de partida, incluso si inicialmente el remolque se encontraba atravesado. Gran parte de la experiencia obtenida en este desarrollo es directamente trasladable a un gran número de problemas de control no lineal.

Automoción. El uso de ANS en este campo es muy variado. Como ya se ha comentado, Ford Motor Co. ha desarrollado el primer vehículo de emisiones ultrabajas empleando redes neuronales [James 99, Werbos 98]. General Motors ha empleado ANS para modelar las preferencias de sus clientes, lo cual sirve de información a sus ingenieros a la hora de diseñar nuevos modelos. En Renault se emplean mapas autoorganizados para detectar averías en el encendido de los automóviles [Hèrault 94], y Citroën emplea redes neuronales en la determinación de la calidad del material utilizado en la confección de los asientos de los vehículos. La empresa Nestor Traffic System (<http://www.nestor.com>) comercializa sistemas para gestión de tráfico.

Aplicaciones biomédicas. Se aplica ANS al estudio de secuencias de aminoácidos en proteínas, nucleótidos en ADN y ARN, clasificación de señales electrocardiográficas y electroencefalográficas, predicción de la reacción de pacientes ante determinados tratamientos, predicción de accidentes relacionados con anestesia, detección de arritmias, predicción de mortandad de pacientes, identificación de cáncer de pecho a partir de mamografías, modelización de la esquizofrenia, clasificación de imágenes médicas, y un largo etcétera.

Síntesis de nuevos medicamentos. Se está empleando ANS para predecir las propiedades curativas de nuevas sustancias sin tener que recurrir para ello a la experimentación en animales, un proceso éticamente discutible, además de lento y

costoso. Por ejemplo, en la lucha contra el cáncer se ha empleado el clásico BP en un programa comercial de simulación (NeuralWare's Professional II/PLUS) para clasificar una serie de medicamentos en función de su mecanismo de acción sobre cultivos con células de tumores humanos [Weinstein 92], que alcanza una precisión sorprendente.

6.6 EJEMPLO DE APLICACIÓN DE ANS: PREVISIÓN DE LA DEMANDA DE CONSUMO ELÉCTRICO

A continuación, vamos a describir como ejemplo de desarrollo una aplicación de redes neuronales realizada por los autores en el contexto de un proyecto de investigación llevado a cabo en colaboración con un conjunto de empresas eléctricas españolas, liderado por Eléctricas Reunidas de Zaragoza S.A.² [Martín del Brío 95a, García 95]. Iremos explicando los pasos emprendidos en el desarrollo de la aplicación basada en redes neuronales, siguiendo el esquema planteado en la sección 6.2.

(1) Planteamiento del problema

Nuestro objetivo consistía en realizar un sistema capaz de **predecir la demanda de consumo eléctrico horario** en el mercado de la compañía eléctrica E.R.Z. (Comunidad de Aragón, esencialmente). El interés de esta previsión radica en que las empresas eléctricas deben conocer con antelación las necesidades de su mercado para poder planear la generación y distribución futura de la energía eléctrica (por una parte, la electricidad producida en exceso se pierde o puede ocasionar inestabilidades en la red; por otra, podemos imaginarnos lo que significa no producir la electricidad suficiente para cubrir las necesidades).

El método utilizado por la empresa estaba basado en técnicas estadísticas convencionales, con las que para obtener unos resultados aceptables se requería la actuación de un experto cuyo papel era ajustar continuamente los múltiples parámetros del programa, en especial, los causados por cambios en la estructura del mercado.

(2) Requerimientos del sistema

El producto a desarrollar debe proporcionar una previsión de consumo eléctrico, de forma automática y no supervisada, para cada una de las veinticuatro horas del día. La antelación con la que se requería esta predicción era de dos tipos: a una semana vista y a dos días. Además, el sistema debe ser capaz de adaptarse automáticamente (sin intervención humana) a los cambios que observe en el mercado (incorporación de nuevas industrias, aumento de la población, etc.).

² Proyecto de Investigación Electrotécnico (PIE) n° 150028, financiado por OCIDE.

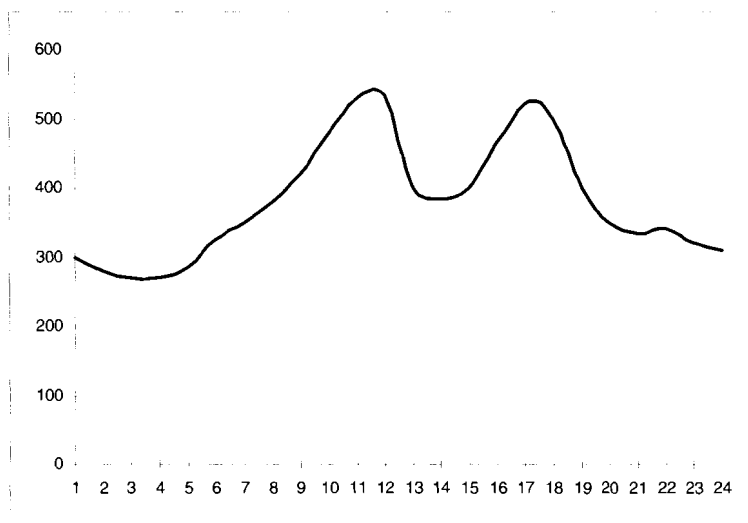


Figura 6.3. Perfil típico de consumo de energía eléctrica en Aragón (megawatios) a lo largo de las 24 horas del día. El consumo más elevado tiene lugar alrededor del mediodía y a media tarde (máxima actividad económica). Los perfiles cambian de forma en función de si se trata de un día laborable, festivo, anterior o posterior

Nos encontramos, por tanto, ante un problema de predicción fuera de línea, es decir, no en tiempo real a medida que van llegando datos, sino a partir de datos históricos, y sin importar excesivamente el tiempo de respuesta. Se deben predecir veinticuatro cantidades que representan el valor de la potencia eléctrica consumida en cada una de las horas del día.

Dado que no existe un algoritmo conocido que resuelva el problema con las condiciones de contorno expuestas, y teniendo en cuenta que los ANS han demostrado conseguir buenos resultados en el campo de la predicción, experimentamos con la aplicación de estas técnicas.

(3) Revisión bibliográfica

Realizamos una búsqueda de trabajos similares al que nosotros íbamos a acometer. Encontramos ejemplos en la literatura que abordaban el problema desde distintos enfoques, que podían ser resumidos en dos: series temporales y modelado del consumo en función de variables climáticas (la influencia del clima en el consumo eléctrico es bastante clara).

(4) Elección del modelo de ANS

Por diversas razones, nos inclinamos por el modelado del consumo a través de las variables climáticas; de este modo, podrán hacerse previsiones de demanda

eléctrica sólo con disponer de las previsiones climatológicas de los Institutos Meteorológicos.

Antes de realizar este modelado del consumo eléctrico fue necesario separar los días en ciertos grupos, por ejemplo, los días laborables de los festivos, con el objeto de que el cambio de actividad social e industrial no enmascarara la dependencia del consumo respecto del clima (es decir, que el descenso del consumo de electricidad que se da en un domingo respecto a un martes no es debido a que haga más frío, sino a que hay menos actividad). Por tanto, teníamos que resolver dos problemas: por una parte uno de clasificación de patrones (tipos de día según el patrón de consumo eléctrico), por otra, un problema de ajuste funcional o modelado (consumo respecto del clima, para cada uno de los tipos de días separados). Para cada uno de estos dos problemas utilizaríamos una arquitectura neuronal diferente.

Decidimos emplear como base de cada arquitectura los dos modelos de ANS más comunes: los **mapas autoorganizados de Kohonen** (SOFM) y el **perceptrón multicapa** (MLP) entrenado con el algoritmo BP. Mostraríamos a la primera arquitectura patrones de consumo diario para comprobar cómo distribuía cada ejemplo en el mapa de salida. Por otra parte, la arquitectura basada en el BP asociaría situaciones climáticas (entradas del ANS) con valores de consumo eléctrico (salidas del ANS).

Decidimos utilizar ANS simuladas en ordenadores convencionales (estaciones de trabajo y PC compatibles) puesto que, tal y como fue planteado en principio, el problema no requería de hardware específico (no así el caso de la predicción en línea, fuera de nuestro propósito).

(5) Datos disponibles y selección de variables relevantes

Por un lado, la empresa eléctrica nos proporcionaba ficheros con datos históricos de consumo eléctrico horario de unos cuantos años atrás (1988-93). Por otro lado, el Instituto Meteorológico hacía lo propio con un buen número de variables climáticas. En ningún momento estábamos recogiendo datos ni proporcionando respuestas en tiempo real; tampoco necesitábamos la ayuda de un experto para que clasificara los patrones: simplemente nos apoyábamos en los datos históricos de consumo y clima.

Para cuantificar la importancia de las variables a emplear en cada una de las dos redes (tipos de días de consumo mediante SOFM, y ajuste de variables climáticas al consumo mediante BP), decidimos seguir dos estrategias: por un lado, preguntamos a ingenieros de la empresa conocedores del tema, y por otro, llevamos a cabo diferentes análisis estadísticos sobre los datos (estudio de correlaciones, regresiones multilíneas, análisis de componentes principales, etc.). Por ambos caminos llegamos a similares conclusiones.

(6) Elección de los conjuntos de aprendizaje y test

En el caso de la clasificación de tipos de días (mapas de Kohonen), empleamos datos de un año entero en dos clasificaciones sucesivas: primero por estaciones (invierno, verano...), y dentro de cada estación, por tipo de día (laborable, festivo...) [Martín del Brío 95a]. En el ajuste consumo-clima (BP) decidimos emplear todo un año como conjunto de aprendizaje y el siguiente como conjunto de test, para cada uno de los tipos de día obtenidos [García 95].

(7) Preprocesamiento

Tratamos cada entrada y cada salida por separado, y aplicamos de forma sucesiva los dos preprocesamientos recomendados: primero restando la media y dividiendo por la desviación estándar (estandarización, ecuación 6.1), y dividiendo a continuación por el valor máximo resultante en cada variable. Con el primer escalado conseguimos uniformizar el rango de variación de todas las entradas y, por lo tanto, equiparamos su importancia (por ejemplo, los valores de la entrada *número_de_horas_de_sol*, con un rango de 0-12 horas, eran mucho menores que los de la *visibilidad*, que tomaba valores de 0-800 decímetros); con el segundo mantenemos las entradas y salidas aproximadamente dentro del rango dinámico de variación de la función de activación elegida $[-1,+1]$. El preprocesamiento también incluía potenciar ciertas variables que sabíamos deberían tener un papel relevante en la clasificación, por ejemplo, elevándolas al cuadrado (ésta es una forma de incorporar información apriorística en el modelo).

(8) Proceso de entrenamiento

Seguimos la estrategia de comenzar realizando simulaciones de arquitecturas con un número reducido de neuronas ocultas (ninguna, dos, cuatro...), e ir aumentando su número progresivamente. De esta manera observamos que diez neuronas ocultas era un número que proporcionaba buenos resultados. Empleamos validación cruzada como criterio de parada del proceso del entrenamiento, haciendo uso de un conjunto de test (se detenía el entrenamiento cuando su error aumentaba), lo cual nos servía además para medir la capacidad de generalización de la red. Como medida del error utilizábamos el error cuadrático medio o RMS (*Root Mean Square Error*) promediado para todos los patrones y las salidas.

Para tener una idea de los tiempos de cálculo necesarios, indicaremos que haciendo uso de un PC compatible con microprocesador i80486DX2 a 66 MHz bajo MS-DOS, el sistema (todas las potencias horarias para todos los tipos de días) podía ser entrenado en aproximadamente un día de CPU (que podría reducirse a unas pocas horas si se llevase a cabo hoy en día sobre un computador con un microprocesador actual). Por supuesto, una vez entrenado, dadas unas determinadas variables de entrada, el sistema proporciona la predicción de demanda de energía en pocos segundos.

(9) Evaluación de los resultados

Los resultados que se obtenían en conjunto eran buenos, consiguiendo una clasificación de los tipos de días con una precisión superior al noventa por ciento, y alcanzando un error en la previsión de la demanda de consumo eléctrico horario de entorno al 3%. Además, el sistema neuronal permite representar gráficamente la relación que ha encontrado entre el consumo eléctrico y cada una de las variables climáticas, como por ejemplo la temperatura, información de gran interés para los ingenieros que gestionan la distribución. Podemos señalar que el sistema final es completamente automático y no supervisado, alcanzando errores de predicción comparables a los del sistema clásico, que requiere la supervisión y corrección manual por técnicos de la empresa.

6.7 CONCLUSIONES

Tras presentar en este capítulo abundante información sobre la aplicación de redes neuronales a la resolución de problemas reales, se hace necesario extraer unas conclusiones finales que fijen una serie de conceptos. Algunas se han extraído de las referencias [Croall 92, Fogelman 98], otras son conclusiones a las que hemos llegado nosotros tras años de experiencia en el trabajo con ANS.

- a) Para resolver con eficiencia un problema, éste debe **comprenderse bien**. Rara vez existen soluciones simples a problemas complejos.
- b) Si se comprende bien el problema, **introduciendo explícitamente el conocimiento apriorístico** en el modelo se obtiene un sistema más eficiente.
- c) **Todo problema debe descomponerse en partes**.
- d) En cada parte se debe **ensayar con distintos métodos**, neuronales o no, para quedarse con el más eficaz. En ese sentido, **deben realizarse comparaciones objetivas**. Lo habitual es que la red neuronal trate con la parte *dura de roer* del problema (la que sea altamente no lineal, no del todo comprendida, etc.).
- e) En la línea del punto anterior, una clara tendencia actual es **integrar en una aplicación diversas soluciones**, neuronales, borrosas, estadísticas, procesamiento de señal, etc.
- f) Debe huirse de las **aplicaciones a ciegas** de los ANS, pues de ese modo no se alcanzará una solución tan eficaz como la que podría obtenerse con una metodología más elaborada. No obstante, incluso este tipo de aplicación puede llevar en ocasiones a un resultado aceptable, pues las redes neuronales ofrecen una buena relación eficiencia/coste, debido especialmente a los relativamente reducidos tiempos de desarrollo.

- g) Los programas comerciales de simulación proporcionan una interesante visión del problema, y son útiles en la etapa de desarrollo, aunque casi siempre es necesario **crear finalmente un software específico a la aplicación en desarrollo.**
- h) Si la aplicación precisa además tratar muchos datos en tiempo real, o bien se requiere un circuito dedicado, pequeño y de bajo consumo, puede resultar necesario el empleo de **hardware neuronal específico.**

Finalmente, hemos comentado que los ANS deberán emplearse allí donde ofrezcan ventajas; una red neuronal no tiene por qué ser siempre la solución idónea. En este sentido **debe tenerse en cuenta una perspectiva global en la que se incluyan aspectos económicos, velocidad de desarrollo, robustez, fiabilidad, etc., y no sólo el rendimiento del sistema.** Cuando se tienen en cuenta todos estos factores, las redes neuronales suelen salir favorecidas, especialmente debido a la ya citada muy buena relación eficiencia/coste que ofrecen.

Con estas conclusiones finales hemos tratado de desmitificar la aplicación de nuevas tecnologías, como las redes neuronales (o los sistemas borrosos, que estudiaremos a continuación), que en muchas ocasiones se presentan de manera gratuita y poco rigurosa como las herramientas que resolverán todos nuestros problemas. La realidad es que no existen soluciones fáciles a problemas difíciles. Las redes neuronales constituyen así un útil y muy potente conjunto de herramientas, que podemos añadir al gran número de métodos de procesamiento y control disponibles, y que deberán utilizarse donde sea apropiado.

Sistemas Borrosos

P A R T E II

CAPÍTULO 7

LÓGICA BORROSA

Estudiados los sistemas basados en redes neuronales (ANS), que emulan simplificadaamente la estructura del cerebro para reproducir sus capacidades, en esta segunda parte del texto trataremos los **sistemas basados en lógica borrosa**, que más bien emulan la manera en que el cerebro razona o piensa.

La denominada lógica borrosa (*fuzzy logic*) permite tratar información imprecisa, como *estatura media*, *temperatura baja* o *mucha fuerza*, en términos de conjuntos borrosos o difusos (imprecisos, en definitiva). Veremos que estos conjuntos borrosos se combinan en reglas para definir acciones, como por ejemplo, *Si la temperatura es alta entonces enfría mucho*. De esta manera, los sistemas de control basados en lógica borrosa combinan unas variables de entrada (definidas en términos de conjuntos borrosos), por medio de grupos de reglas que producen uno o varios valores de salida.

En los próximos capítulos veremos también que los sistemas basados en lógica borrosa pueden ser aplicados a similares problemas que las redes neuronales, de modo que resultarán especialmente interesantes para problemas no lineales o no bien definidos. De la misma manera, los sistemas borrosos permiten modelar cualquier proceso no lineal, y aprender de los datos haciendo uso de determinados algoritmos de aprendizaje (a veces tomados de otros campos, como las propias redes neuronales o los algoritmos genéticos). No obstante, a diferencia de los sistemas neuronales, los basados en lógica borrosa permiten utilizar fácilmente el conocimiento de los expertos en un tema, bien directamente, bien como punto de partida para una optimización automática, al formalizar el conocimiento a veces ambiguo de un experto (o el sentido común) de una forma realizable. Además, gracias a la simplicidad de los cálculos necesarios (sumas y comparaciones, fundamentalmente), normalmente pueden realizarse en sistemas baratos y rápidos.

7.1 INTRODUCCIÓN

Hablando ya en términos más rigurosos, la **teoría de conjuntos borrosos** parte de la teoría clásica de conjuntos, añadiendo una función de pertenencia al conjunto, definida ésta como un número real entre 0 y 1. Así, se introduce el concepto de conjunto o subconjunto borroso asociado a un determinado valor lingüístico, definido por una palabra, adjetivo o etiqueta lingüística A . Para cada conjunto o subconjunto borroso se define una **función de pertenencia o inclusión** $\mu_A(t)$, que indica el grado en que la variable t está incluida en el concepto representado por la etiqueta A . Como puede verse en la Figura 7.1 para el valor lingüístico *estatura_de_una_persona* podrían definirse tres subconjuntos borrosos, cada uno identificado por una etiqueta, {Bajo, Medio, Alto}, y con una función de inclusión o pertenencia $\{\mu_{Bajo}(t) \ \mu_{Medio}(t) \ \mu_{Alto}(t)\}$ (situadas en la figura a la izquierda, centro y derecha, respectivamente).

Los conjuntos borrosos permiten agrupar objetos o sucesos por el valor de una cierta magnitud; por ejemplo, las personas pueden ser agrupadas por su altura. Así, si definimos el conjunto clásico de las personas de estatura baja como las que miden menos de 1.65 metros, resulta que alguien de 1.64 es bajo, mientras que alguien de 1.66 no lo es; esta descripción que proporciona la teoría clásica de conjuntos no resulta perfectamente satisfactoria, ya que su estatura sólo se diferencia en 2 cm. Veremos que una descripción en términos de conjuntos borrosos resulta más adecuada en casos de este tipo, por ejemplo, podrían introducirse los términos *bajo*, *medio* y *alto*, y definirse mediante funciones de pertenencia o inclusión, que al variar de forma continua en el rango de 0 a 1 (Figura 7.1) nos indicarían si una persona es baja (valor entorno a 1.0 para etiqueta *bajo*), baja tirando a media (por ejemplo, valor 0.6 para *bajo* y 0.4 para *medio*), claramente alta (por ejemplo, valor 0.8 para *alto*), etc.

Sistemas de control borroso

Hay que señalar que dentro de los sistemas borrosos se incluyen diversas teorías, como la teoría de conjuntos borrosos [Combettes 93], extensión de la teoría de conjuntos clásica, o la lógica borrosa, que puede ser considerada una ampliación de las lógicas n -valuadas propuestas por Lukasiewicz en 1930, y que son a su vez extensión de la lógica trivaluada (verdadero, falso e indeterminado) [Kosko 93]. No obstante, quizás la principal aplicación actual de la lógica borrosa sean los sistemas de control basados en lógica borrosa o **sistemas de control borroso**, que utilizan las expresiones de la lógica borrosa para formular reglas orientadas al control de sistemas [Brubaker 92, Nass 92, Passino 98]. Dichos sistemas de control borroso pueden considerarse una extensión de los sistemas expertos, pero superando los problemas prácticos que éstos presentan en el razonamiento en tiempo real, causados por la explosión exponencial de las necesidades de cálculo requeridas para el análisis lógico completo de las amplias bases de reglas que manejan.

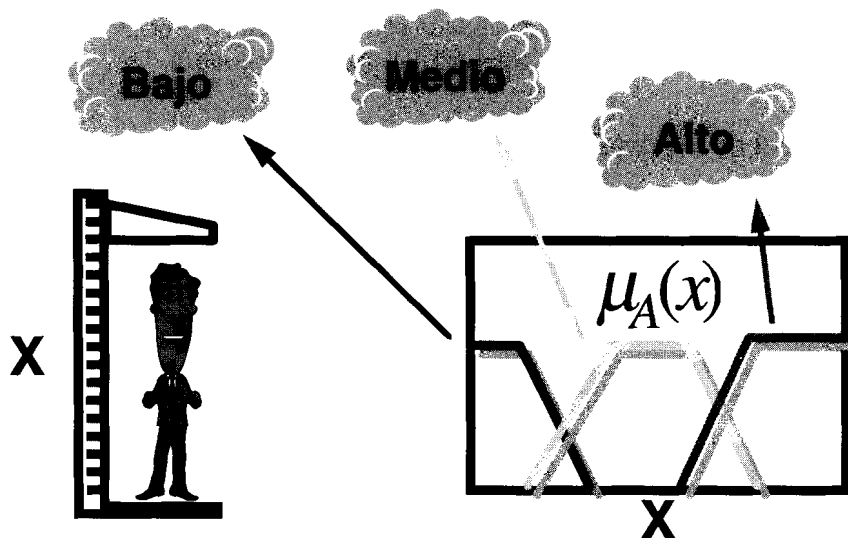


Figura 7.1. Ejemplo de conjuntos borrosos para la variable Estatura

Adelantaremos que este control de sistemas puede ser realizado a diferentes niveles. En el nivel inferior un controlador borroso puede realizar el control en bucle cerrado de una determinada magnitud física del sistema, con el fin de mantenerla en torno a un valor de referencia. A modo de ejemplo, un controlador de este tipo puede decidir la potencia que se ha de suministrar al sistema de calefacción de una habitación para mantener la temperatura en un valor de referencia (por ejemplo, 21°C), utilizando como información la temperatura actual en la habitación y en el exterior de la vivienda. Por otro lado, aplicado a los niveles superiores de planificación, un controlador puede aconsejar los grados de almacenamiento necesarios para mantener la producción prevista, con los mínimos costes y teniendo en cuenta los datos históricos.

Estos métodos de control pueden aplicarse también en brazos articulados y vehículos autónomos, en los cuales los modelos matemáticos significativos son muy complejos. En muchos de estos casos interesa combinar las propiedades de un control basado en el modelo del sistema con el de reglas heurísticas, las cuales pueden emplearse para seleccionar o ajustar automáticamente sus parámetros. Asimismo, las técnicas de razonamiento aproximado resultan interesantes para los niveles superiores de control y planificación de robots cuando el entorno no es conocido en forma precisa. Una introducción simple a estas técnicas puede encontrarse en [Barron 93, Brubaker 93], y una visión más completa en [Terano 92].

Para desarrollar estos sistemas de control se precisa de herramientas de diseño de controladores que faciliten la adquisición de conocimiento y el análisis del controlador resultante, incluyendo las propiedades dinámicas de estabilidad y

robustez. En caso contrario, el diseño puede convertirse en un proceso muy tedioso, y no garantizarse el comportamiento correcto del sistema de control.

Breve historia de los sistemas borrosos

Las bases teóricas de la lógica borrosa (*fuzzy logic*) fueron enunciadas en 1965 por Lotfi A. Zadeh [Zadeh 65], profesor de Ingeniería Eléctrica en la Universidad de California en Berkeley, pero Zadeh no presenta la teoría básica de los controladores borrosos hasta 1973 [Zadeh 73, 88] (recientemente Zadeh ha interpretado los sistemas *fuzzy* como computación con palabras, y ha introducido la teoría de computación con percepciones [Zadeh 99]; véase el Prólogo de la página xix redactado por él). Aunque al principio su trabajo fue recibido fríamente (especialmente en los Estados Unidos), a partir de él otros investigadores comenzaron a aplicar la lógica borrosa a diversos procesos. Así, desde Mamdani [Mamdani 74, 83], quien aplica la lógica borrosa a un sistema de control de vapor, se han venido sucediendo numerosas aplicaciones. Otra de las más clásicas quizás sea la de Smidth y otros, que en 1980 aplican esta técnica al control de hornos rotativos en una cementera [García Cerezo 91].

Sin duda alguna, donde más éxito han tenido los sistemas borrosos ha sido en Japón [Tsukamoto 79]. En este sentido, una historia breve sería:

1972: Grupo de trabajo en sistemas *fuzzy* en el Instituto de Tecnología de Tokio.

1972-1981: Informe anual *General Problems on Fuzzy Systems*

1972: *Workshop* sobre sistemas borrosos por la sociedad para la instrumentación e ingenieros de control.

1983: Aplicación del control borroso al proceso de purificación de agua, Fuji Elec. & TIT.

1984: Capítulo IFSA (*International Fuzzy Systems Association*) de Japón. Primer *simposium* anual sobre Sistemas Borrosos.

1987: Aplicación del control borroso al tren metropolitano de la ciudad de Sendai (Japón), desarrollado por Hitachi.

1987: Primera *moda Fuzzy*. Segundo congreso IFSA, Tokio.

1988: *Workshop* internacional sobre aplicación de sistemas borrosos en Iizuka.

1989-1994: Proyecto LIFE (*Laboratory for International Fuzzy Engineering*), Ministerio de Comercio Internacional e Industria de Japón (MITI).

1989-1993: Proyecto de investigación en sistemas borrosos (STA).

1989: Sociedad para la teoría y sistemas *fuzzy* (SOFT).

1989: Aplicación de control borroso a unidad de suministro de agua caliente para uso doméstico, por Matsushita.

- 1989: Premio Honda al profesor Zadeh.
- 1990: Conferencia sobre lógica borrosa & redes neuronales, Iizuka.
- 1990: Reunión chino-japonesa sobre conjuntos y sistemas borrosos.
- 1993: *Second IEEE International Conference on Fuzzy Systems*, San Francisco
- 1994: *Third IEEE International Conference on Fuzzy Systems*, Orlando
- 1995: *Fourth IEEE International Conference on Fuzzy Systems*, Yokohama
- 1996: *Fifth IEEE International Conference on Fuzzy Systems*, New Orleans
- 1997: *Sixth IEEE Int. Fuzzy Systems Conference FUZZ-IEEE'97*, Barcelona
- 1998: *IEEE Int. Fuzzy Systems Conference FUZZ-IEEE'98*, Anchorage
- 1999: *IEEE Int. Fuzzy Systems Conference FUZZ-IEEE'99*, Seúl
- 2000: *IEEE Int. Fuzzy Systems Conf. FUZZ-IEEE'2000*, San Antonio (Texas)

Especial mención merece la creación de LIFE (*Laboratory for International Fuzzy Engineering research*) en marzo de 1989, auspiciado por el Ministerio de Comercio Internacional e Industria de Japón (MITI). Su capital es de compañías privadas japonesas y del propio Ministerio al 50%, y su Presidente está en el Instituto de Tecnología de Tokio (TIT). En su sede trabajan en la actualidad alrededor de 30 investigadores a tiempo completo. En los Estados Unidos (y Europa) solamente se empezó a dar importancia a la lógica borrosa cuando desde Japón empezó a llegar información sobre numerosas aplicaciones prácticas. A partir de entonces empresas norteamericanas como NASA, Boeing, Ford, Rochwell o Bell comenzaron a aplicar la lógica borrosa; por ejemplo, Ford Motor Co. experimenta con un sistema de aparcamiento automático para camiones con remolque.

Plan del capítulo

Presentadas unas primeras ideas sobre los sistemas borrosos y su historia, en los siguientes apartados de este capítulo expondremos los principales conceptos que aparecen tanto en lógica borrosa como en los conjuntos borrosos, explicando con detenimiento las funciones de inclusión, las reglas y su utilización. Asimismo, se introducirán los conceptos de partición borrosa y diversas medidas borrosas. Finalmente, explicaremos la diferencia entre los conceptos de borrosidad y probabilidad. Una vez hayamos introducido todos los conceptos fundamentales, en el capítulo siguiente desarrollaremos un ejemplo sencillo de controlador borroso que los ilustre; será entonces cuando el lector encuentre el justo sentido de muchos de los términos que se introducirán más formalmente a lo largo del presente capítulo.

Muchos de los conceptos que se expondrán proceden de [Wang 94]; otros textos completos y recomendables son [Passino 98, Jang 97, Lin 96].

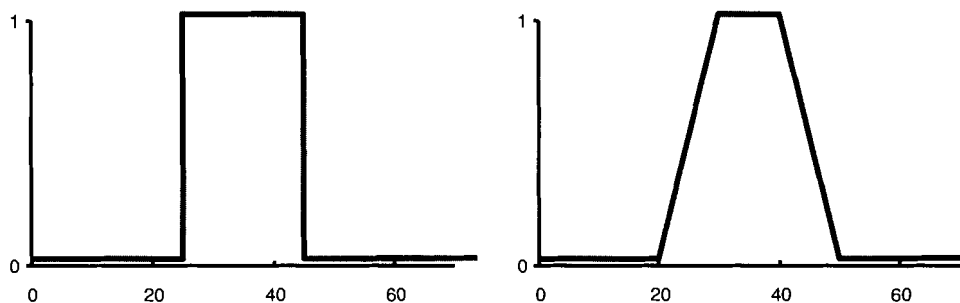


Figura 7.2. Funciones de inclusión de conjuntos clásico (izquierda) y borroso (derecha) para edad adulta. Una persona de 25 años en términos clásicos habría que definirla como adulta o no adulta, en términos borrosos podría decirse que se incluye en aproximadamente un 0.5 (50 %) al conjunto edad adulta

7.2 CONJUNTOS BORROSOS

En los conjuntos clásicos algo está incluido completamente en él o no lo está en absoluto (recuérdese el ejemplo de *persona alta* y *persona baja*). Esta situación puede describirse asignando un 1 a todos los elementos incluidos en el conjunto y un 0 a los no incluidos. A la función que asigna estos valores la denominaremos **función de inclusión o pertenencia** (*membership function*). Veremos cómo los conjuntos borrosos permiten describir el grado de pertenencia o inclusión de un objeto (o el valor de una variable) al concepto dado por la etiqueta que le da nombre, asignando un número real entre 0 y 1 (así, por ejemplo, una persona podrá ser *medio baja* o *muy alta*).

Sea U un conjunto de objetos, por ejemplo, $U = \mathcal{R}^n$, que se denominará **universo de discurso**. En término matemáticos [Wang 94], un **conjunto borroso** F en U queda caracterizado por una función de inclusión μ_F que toma valores en el rango $[0,1]$, es decir, $\mu_F: U \rightarrow [0,1]$; donde $\mu_F(u)$ representa el grado en el que $u \in U$ pertenece al conjunto borroso F . Ello representa la generalización del concepto clásico de conjunto (abrupto), en el que la función de pertenencia toma solamente los valores 0 o 1; por el contrario, para uno borroso, la función puede tomar también valores intermedios.

A modo de ejemplo, para el conjunto de las personas se pueden definir subconjuntos borrosos en función de la edad. El subconjunto de los *Adultos* puede definirse, como se puede observar en la Figura 7.2, asignando una función de inclusión abrupta para el conjunto clásico *Adulto* = *edad entre 25 y 45*. Definido en términos borrosos, la función de inclusión de este conjunto toma valor 1 entre 30 y 40, 0 para los menores de 20 o para los mayores de 50, y valores intermedios entre 20 y 30 y entre 40 y 50 (Figura 7.2).

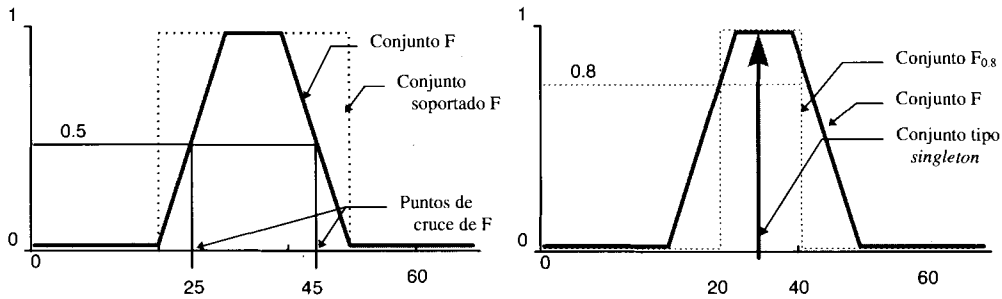


Figura 7.3. Términos relativos a los conjuntos borrosos

Dado un cierto conjunto borroso F , se definen los siguientes términos. El **conjunto soportado** es el conjunto (clásico) de todos los valores de U para los que $\mu_F(u) > 0$. Los **puntos de cruce** son aquellos valores para los que $\mu_F(u) = 0.5$. Se dice que un conjunto borroso es de tipo **singleton** si su conjunto soportado es de un sólo valor (véase la Figura 7.3).

Asimismo, se denomina **conjunto α -corte** F_α de un conjunto borroso F , al conjunto clásico de todos los puntos u de U para los que se cumple $\mu_F(u) > \alpha$. Por otro lado, se dice que un conjunto borroso está **normalizado** si el máximo de su función de inclusión es 1; obviamente, un conjunto borroso puede **normalizarse** multiplicando su función de inclusión por un coeficiente fijo para que sea de tipo normalizado.

7.3 FUNCIONES DE INCLUSIÓN DE CONJUNTOS BORROSOS

La **función de inclusión o pertenencia** (*membership function*) de un conjunto borroso consiste en un conjunto de pares ordenados $F = \{ (u, \mu_F(u)) / u \in U \}$ si la variable es discreta, o una función continua si no lo es. Como ya se ha comentado, el valor de $\mu_F(u)$ indica el grado en que el valor u de la variable U está incluida en el concepto representado por la etiqueta F . Para la definición de estas funciones de pertenencia se utilizan convencionalmente ciertas familias de formas estándar, por coincidir con el significado lingüístico de las etiquetas más utilizadas. Las más frecuentes son la función de tipo trapezoidal, singleton, triangular, S , exponencial y tipo π , que pasamos a describir.

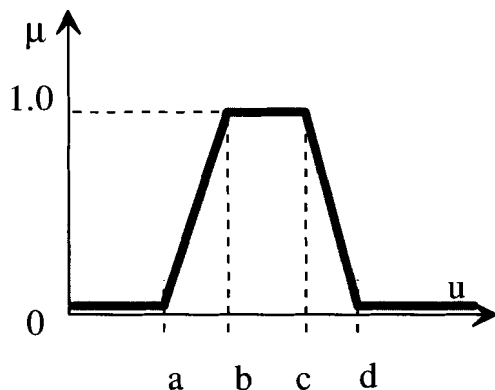


Figura 7.4. Función de pertenencia de tipo trapezoidal

La **función de tipo trapezoidal** se define por cuatro puntos a , b , c , d . Esta función es cero para valores menores de a y mayores de d , vale uno entre b y c , y toma valores en $[0,1]$ entre a y b , y entre c y d . Se utiliza habitualmente en sistemas borrosos sencillos, pues permite definir un conjunto **borroso** con pocos datos, y calcular su valor de pertenencia con pocos cálculos. Se emplea especialmente en sistemas basados en microprocesador, pues con similar formato pueden codificarse también funciones de tipo S, función de tipo π , triangular y singleton, según se distribuyan los puntos a , b , c y d de la figura (por ejemplo, juntando b y c tenemos una triangular). Se define con

$$S(u; a, b, c, d) = \begin{cases} 0 & u < a \\ \left(\frac{u-a}{b-a} \right) & a \leq u \leq b \\ 1 & b \leq u \leq c \\ \left(\frac{d-u}{d-c} \right) & c \leq u \leq d \\ 0 & u > d \end{cases} \quad (7.1)$$

Esta función resulta adecuada para modelar propiedades que comprenden un rango de valores (*adulto*, *normal*, *adecuada*...). Para modelar una función triangular se hace $b=c$, para una función de tipo S (pero no suave) se hace $c=d=\max(U)$, y para una función de tipo singleton $a=b=c=d$ (Figura 7.4).

La función de tipo **singleton** tiene valor 1 sólo para un punto a y 0 para el resto. Se utiliza habitualmente en sistemas borrosos simples para definir los conjuntos borrosos de las particiones de las variables de salida, pues permite simplificar los cálculos y requiere menos memoria para almacenar la base de reglas. Se define con:

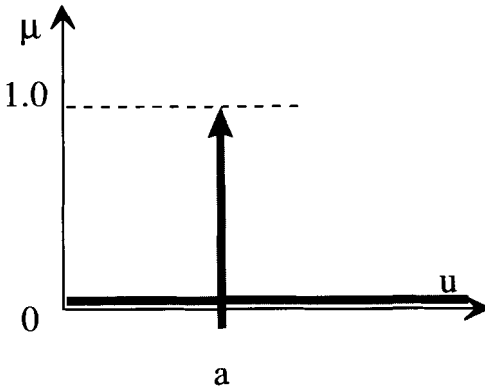


Figura 7.5. Función de tipo singleton

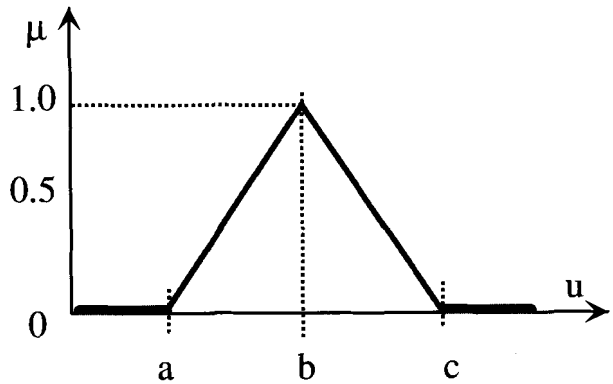


Figura 7.6. Función de tipo T (triangular)

$$S(u; a) = \begin{cases} 1 & u = a \\ 0 & u \neq a \end{cases} \quad (7.2)$$

La **función de tipo T (triangular)** puede definirse como:

$$T(u; a, b, c) = \begin{cases} 0 & u < a \\ \frac{u-a}{b-a} & a \leq u \leq b \\ \frac{c-u}{c-b} & b \leq u \leq c \\ 0 & u > c \end{cases} \quad (7.3)$$

Esta función es adecuada para modelar propiedades con un valor de inclusión distinto de cero para un rango de valores estrecho en torno a un punto b .

La **función de tipo S** puede definirse como:

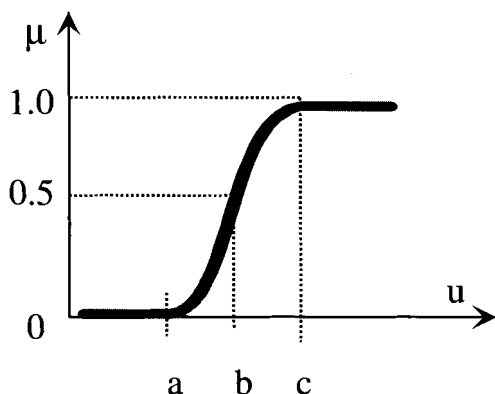


Figura 7.7. Función de tipo S

$$S(u; a, b, c) = \begin{cases} 0 & u < a \\ 2\left(\frac{u-a}{c-a}\right)^2 & a \leq u \leq b \\ 1 - 2\left(\frac{u-a}{c-a}\right)^2 & b \leq u \leq c \\ 1 & u > c \end{cases} \quad (7.4)$$

Esta función resulta adecuada para modelar propiedades como *grande*, *mucho*, *positivo*... Se caracteriza por tener un valor de inclusión distinto de 0 para un rango de valores por encima de cierto punto a , siendo 0 por debajo de a y 1 para valores mayores de c . Su punto de cruce (valor 0.5) es $b=(a+c)/2$; y entre los puntos a y c es de tipo cuadrático (suave). También se han utilizado funciones exponenciales para definir funciones de tipo S, como

$$S(u; k, c) = \frac{1}{1 + \exp(-k(u - b))} \quad (7.5)$$

Por último, la **función de tipo π** puede definirse de la forma siguiente:

$$\pi(u; b, c) = \begin{cases} S(u; c-b, c-b/2, c) & u \leq c \\ 1 - S(u; c-b, c-b/2, c) & u \geq c \end{cases} \quad (7.6)$$

Esta función tiene forma de campana, y resulta adecuada para los conjuntos definidos en tono a un valor c , como *medio*, *normal*, *cero*... Pueden definirse también utilizando expresiones analíticas exponenciales o cuadráticas, como la bien conocida campana de Gauss.

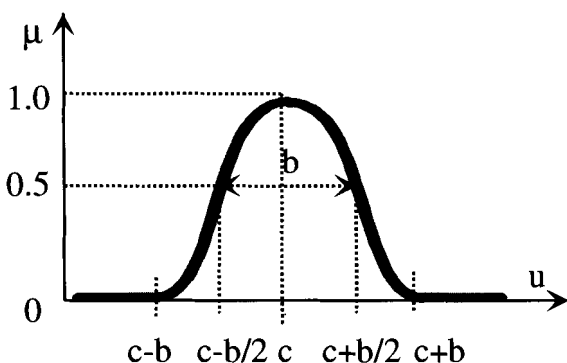


Figura 7.8. Función de pertenencia de tipo π (con forma de campana)

7.4 VARIABLE LINGÜÍSTICA

Se denomina **variable lingüística** a aquella que puede tomar por valor términos del lenguaje natural, como *mucho*, *poco*, *positivo*, *negativo*, etc., que son las palabras que desempeñan el papel de etiquetas en un conjunto borroso. Aunque el objetivo principal de este concepto es expresar de manera formal el hecho de que pueden asignarse como valor de una variable palabras tomadas del lenguaje natural, no obstante a una variable lingüística podrán asignarse también valores numéricos. Así, en una expresión como *la temperatura es fría*, la variable temperatura debe ser entendida como una variable lingüística, pues se le asigna como valor el conjunto borroso *fría*, pero además esta variable puede también tomar valores numéricos como *la temperatura es 4°C*.

En términos más formales, una variable lingüística se define por una tupla $(A, T(A), U, G, M)$, donde A es el nombre de la variable, $T(A)$ es el conjunto de términos que nombran los valores x que puede tomar A , valores que son conjuntos borrosos en U ; el conjunto de valores numéricos que puede tomar para una variable discreta, o el rango de valores posibles para una continua, es lo que se conoce como el universo de discurso de la variable x , y se nombra como U ; por último, G es una regla sintáctica para la generación de los nombres de los valores de x , y M es una regla semántica para asociar un significado a cada valor.

El siguiente ejemplo permitirá comprender el sentido de estos términos formales. *Temperatura* puede considerarse como una variable lingüística, de modo que $A = \text{temperatura}$. $T(\text{temperatura})$ es el conjunto de todos los términos que pueden hacer referencia a la temperatura, como *muy fría*, *fría*, *normal*, *alta*, *muy alta*, pero también *agradable*, *suave*, *cortante*, etc. El universo de discurso U de esta variable va, en general, desde el cero absoluto al infinito, pero en aplicaciones normales se

suele restringir al rango de temperaturas que pueden presentarse en ella (por ejemplo, temperaturas entre 0 y 40°C).

7.5 PARTICIONES BORROSAS

Dada una variable A , definida en un rango entre u_1 y u_2 , es posible establecer en ella diversas particiones. Se conoce por **partición** a un conjunto de los conjuntos borrosos que se han definido para la variable A . Una partición de A es uno de los subconjuntos que pueden formarse con los elementos (términos) de $T(A)$. Así, para la variable estatura una posible partición sería la correspondiente a la Figura 7.1, con tres subconjuntos borrosos, cada uno identificado por una etiqueta, {Bajo, Medio, Alto}, y una función de inclusión o pertenencia, $\{\mu_{\text{Bajo}}(t), \mu_{\text{Medio}}(t), \mu_{\text{Alto}}(t)\}$. Se dice que una **partición es completa** si para todos los valores posibles de U existe en la partición un conjunto con pertenencia no nula (es decir, los conjuntos definidos cubren todo U); así, completitud es el porcentaje de los elementos de U para los que existe en la partición un conjunto con pertenencia no nula frente al total de elementos de U . Se dice que dos conjuntos borrosos están **solapados** si su intersección es no nula; de este modo, el solapamiento de un conjunto borroso es la relación del número de elementos que comparte con otros conjuntos de la misma partición, respecto del número total de elementos que lo forman.

Para la realización de controladores basados en lógica borrosa se han de definir particiones de las variables del controlador. Normalmente se recomienda que estas particiones sean completas, con un solapamiento del 20% al 50%, y en número impar. Normalmente se emplean particiones de 3 o 7 conjuntos, pues la complejidad no es excesiva y permiten una precisión suficiente en la descripción de los valores de la variable. Además, se recomienda definir conjuntos de tipo T (triangulares) en torno a puntos singulares, como el cero. Los nombres de los conjuntos borrosos que forman una partición se suelen expresar en forma abreviada por sus iniciales; así, una partición típica como {Negativo Grande, Negativo Pequeño, Cero, Positivo Pequeño, Positivo Grande} se representa como {NG, NP, CE, PP, PG} o, en inglés, {NL, NS, ZE, PS, PL} (*Negative Large, Negative Small, Zero, Positive Small, Positive Large*).

7.6 MEDIDAS BORROSAS

Dado un conjunto borroso A , se definen ciertas magnitudes medibles del conjunto, que se conocen como **medidas borrosas**. Una de las principales es la **borrosidad**. Si llamamos C al conjunto discreto de los valores x en los que $\mu_A(x) > 0$, la borrosidad indica la distancia de A al conjunto discreto C . En otras palabras, la magnitud borrosidad mide cuál es el grado de borrosidad de un conjunto.

Por otro lado, la **distancia entre dos conjuntos borrosos** A y C se puede definir utilizando diversas medidas. Las más frecuentes son las siguientes:

- **Hamming**
$$f(A) = \sum |\mu_A(x) - \mu_C(x)| \quad (7.7)$$

- **Euclídea**
$$f(A) = (\sum (\mu_A(x) - \mu_C(x))^2)^{1/2} \quad (7.8)$$

- **Minkowski**
$$f(A) = (\sum (\mu_A(x) - \mu_C(x))^w)^{1/w} \text{ con } w \in [1, \infty] \quad (7.9)$$

Otra medida que puede definirse es la **similitud**, la cual mide el parecido entre dos conjuntos, y en su forma básica es una extensión de la distancia entre conjuntos. Por su parte, la **entropía borrosa** nos informa sobre cuánta información aporta este conjunto a la descripción de la variable x [Wang 94]. Se define para un conjunto borroso A como

$$f(A) = -\sum \{\mu_A(x) \log \mu_A(x) + [1 - \mu_A(x)] \log [1 - \mu_A(x)]\} \quad (7.10)$$

Por último, el **agrupamiento borroso o clustering** es una técnica que se introduce para alcanzar una determinada representación de un espacio vectorial de vectores de entrada. Se basa en la medición de las distancias euclídeas entre vectores, y se utiliza para determinar las reglas borrosas que describen un sistema desconocido o *caja negra*. Uno de los métodos más conocidos para realizar el agrupamiento borroso es el método denominado de las k -medias (*k-means*), aunque existen muchas otras técnicas, como las basadas en la entropía, o bien en los métodos de minimización energética [Wang 94].

7.7 OPERACIONES BORROSAS

A los subconjuntos borrosos se les puede aplicar determinados operadores, o bien pueden realizarse operaciones entre ellos. Al aplicar un operador sobre un sólo conjunto borroso se obtiene otro conjunto borroso; de la misma manera al combinar dos o más subconjuntos mediante alguna operación, se obtendrá otro conjunto.

Sean los subconjuntos borrosos identificados por las etiquetas A y B , asociados a una variable lingüística x , para ellos pueden definirse **tres operaciones básicas**: complemento, unión e intersección. Estas operaciones básicas pueden expresarse de la siguiente manera en términos de las funciones de pertenencia de los conjuntos borrosos A y B (que coinciden con las operaciones del mismo nombre que se definen habitualmente para los conjuntos clásicos):

- **Complemento** $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$ (7.11)

- **Unión** $\mu_{A \cup B}(x) = \max[\mu_A(x), \mu_B(x)]$ (7.12)

- **Intersección** $\mu_{A \cap B}(x) = \min[\mu_A(x), \mu_B(x)]$ (7.13)

Es importante resaltar que el funcionamiento de estas operaciones básicas coincide con el de las correspondientes a las de la teoría clásica de conjuntos; de hecho, la teoría de conjuntos borrosos se reduce a la teoría clásica si reducimos la incertidumbre a 0, y admitimos sólo los valores 0 y 1 para las funciones de pertenencia a un conjunto (0, no pertenece; 1, pertenece). Es decir, **la teoría clásica de conjuntos puede considerarse un caso particular de la borrosa.**

Además, pueden definirse también otras operaciones, como las que se muestran en la tabla 7.1.

Las funciones correspondientes a la definición de la unión y la intersección pueden generalizarse [Alegre 91], a condición de cumplir ciertas restricciones. Las funciones que cumplen estas condiciones se conocen respectivamente como **Conorma Triangular (T-Conorma)** y **Norma Triangular (T-Norma)**. Algunas de las más usadas son las siguientes:

Conormas	Normas	(7.14)
$MAX(a, b)$	$MIN(a, b)$	
$(a + b - ab)$	(ab)	
$a \dot{+} b = MIN(1, a + b)$	$a * b = MAX(0, a + b - 1)$	

Como en la lógica clásica, las conormas y normas cumplen las leyes de Morgan que las relacionan. Se puede demostrar además que las funciones $MAX(.)$ y $MIN(.)$ son las más restrictivas de todas las posibles.

Por otro lado, las dos últimas operaciones de la tabla 7.1 (concentración y dilatación) se conocen como **modificadores**, ya que permiten formalizar el tipo de modificadores aplicados sobre un mismo término en el lenguaje común, como por ejemplo *muy* o *más o menos*. Por ejemplo, para el conjunto borroso F en U , $F = \text{frío}$, el conjunto *muy frío* puede definirse de la forma siguiente:

$$\mu_{\text{muy } F}(u) = (\mu_F(u))^2 \quad (7.15)$$

	Operación	Rango
Igualdad	$\mu_A(x) = \mu_B(x)$	$x \in U$
Unión	$\mu_{A \cup B}(x) = \max[\mu_A(x), \mu_B(x)]$	$\forall x \in U$
Intersección	$\mu_{A \cap B}(x) = \min[\mu_A(x), \mu_B(x)]$	$\forall x \in U$
Complemento	$\overline{\mu_A}(x) = 1 - \mu_A(x)$	$x \in U$
Norma	$\mu_{Norma(A)}(x) = \frac{\mu_A(x)}{\max[\mu_A(x)]}$	$x \in U$
Concentración	$\mu_{Conc(A)}(x) = (\mu_A(x))^2$	$x \in U$
Dilatación	$\mu_{Dilat(A)}(x) = (\mu_A(x))^{0.5}$	$x \in U$

Tabla 7.1. Operaciones entre conjuntos borrosos

ya que al calcular el cuadrado de un número entre 0 y 1 se obtiene un valor más pequeño, con lo que esta operación causa una *concentración* de la función de pertenencia original (la hace más estrecha), lo que implica disponer de una función más exigente para decidir que un valor es *frío*, representando así el término *muy frío*. Por otra parte, el conjunto *más o menos frío* puede definirse como

$$\mu_{mas\ o\ menos\ F}(u) = (\mu_F(u))^{0.5} \quad (7.16)$$

pues al calcular la raíz cuadrada de un número entre 0 y 1 se obtiene un valor más grande, es decir, esta operación causa una *dilatación* sobre la función de pertenencia de partida, siendo de esta manera menos exigente para decidir si un valor corresponde a *frío*, por lo que se tendría el término *más o menos frío*.

7.8 INFERENCIA BORROSA

También como en el caso de la lógica clásica, la borrosa se ocupa del razonamiento formal con proposiciones, pero a diferencia de ésta, los valores de las proposiciones pueden tomar valores intermedios entre verdadero y falso.

De la misma forma que se define un isomorfismo entre la lógica y la teoría de conjuntos clásica, es posible también definir un isomorfismo entre la lógica y la teoría de conjuntos borrosos, y de éstas a su vez con un Álgebra de Boole. De esta forma, los conjuntos borrosos también representan predicados en la lógica proposicional. El

objeto de la lógica borrosa es proporcionar un soporte formal al razonamiento basado en el lenguaje natural, que se caracteriza por tratarse de un razonamiento de tipo aproximado, que hace uso de unas proposiciones que a su vez expresan información de carácter impreciso.

7.8.1 Principio de extensión

El principio de extensión permite convertir conceptos no borrosos en borrosos, siendo además la base de la inferencia en los sistemas borrosos. Sean U y V dos universos de discurso, y f una función de U a V . En general, para una conjunto borroso A en U el **principio de extensión** define un conjunto borroso B en V dado por

$$\mu_B(v) = \sup_{u \in f^{-1}(v)} [\mu_A(u)] \quad (7.17)$$

es decir $\mu_B(v)$ es el máximo de $\mu_A(u)$ para todos los $u \in U$ que cumplen que $f(u)=v$, donde $v \in V$ y suponemos que $f^{-1}(v)$ no es vacío. Si $f^{-1}(v)$ es vacío para algún $v \in V$, definiremos $\mu_B(v)=0$.

7.8.2 Relación borrosa

Para dos universos de discurso U y V , una **relación borrosa** se define como un conjunto borroso R en el espacio $U \times V$, cuya función de inclusión se denota como $\mu_R(u, v)$, con $u \in U$ y $v \in V$.

Puede definirse también la **composición Sup-Star** $R \circ S$ [Wang 94] para dos relaciones borrosas R y S en $U \times V$ y $V \times W$, respectivamente, como otra relación borrosa con la siguiente función de inclusión

$$\mu_{R \circ S}(u, w) = \sup_{v \in V} [\mu_R(u, v) * \mu_S(v, w)] \quad (7.18)$$

donde $u \in U$, $v \in V$, $w \in W$, y el operador $*$ puede ser cualquier t-norma definida en (7.14).

En resumen, la composición sup-star de dos relaciones borrosas $R \circ S$ es un conjunto borroso en $U \times W$. También es posible que S sea en vez de una relación simplemente un conjunto borroso en V , en cuyo caso la expresión $\mu_S(v, w)$, en (7.18) será simplemente $\mu_S(v)$, y $\mu_{R \circ S}(u, w)$, será $\mu_{R \circ S}(u)$. Veremos algo más adelante que la relación sup-star es uno de los principios básicos para el desarrollo de un sistema de inferencia borrosa.

Dependiendo de la elección particular para el operador $*$ (ecuaciones 7.14), se pueden tener distintos casos particulares de relaciones borrosas, siendo otras de las más habituales la **sup-min** (operador MIN) y la **sup-producto** (operador producto) [Wang 94].

7.8.3 *Modus Ponens Generalizado y Modus Tolens Generalizado*

Las **reglas borrosas** son básicamente de tipo IF-THEN (Si-Entonces) y expresan una relación o proposición borrosa. En lógica borrosa el razonamiento no es preciso, sino aproximado, lo cual quiere decir que se puede inferir de una regla una conclusión aunque el antecedente (premisa) no se cumpla plenamente. Existen dos métodos básicos de inferencia entre reglas o **leyes de inferencia**, el ***modus ponens generalizado* (GMP)** y el ***modus tolens generalizado* (GMT)**, que representan extensiones o generalizaciones del razonamiento clásico. **El GMP se conoce como razonamiento directo** y puede resumirse de la forma siguiente:

(Conocimiento):	<i>Si x es A Entonces y es B</i>
(Hecho):	<i>x es A'</i>

(Consecuencia):	<i>y es B'</i>

Donde A, A', B y B' son conjuntos borrosos. Esta relación se expresa también como $B' = A' \circ R$.

	x es A'	y es B'
criterio 1	x es A	y es B
criterio 2-1	x es muy A	y es muy B
criterio 2-2	x es muy A	y es B
criterio 3-1	x es más o menos A	y es más o menos B
criterio 3-2	x es más o menos A	y es B
criterio 4-1	x no es A	y es desconocido
criterio 4-2	x no es A	y no es B

Tabla 7.2. Criterios intuitivos para GMP

La tabla 7.2 muestra diversos criterios que pueden aplicarse para la selección del conjunto B' de la consecuencia en función del conjunto borroso A' empleado en el hecho. El ***modus ponens* generalizado** es equivalente al ***modus ponens* clásico** para el criterio 1 (es decir, ambos coinciden si $A=A'$ y $B=B'$). Por otra parte, los criterios 1 a 3-2 están de acuerdo con el sentido común. Por último, debe destacarse que aunque el criterio 4-2 no es válido en lógica formal, el hecho es que se utiliza en el razonamiento común (mucha gente lo usa en la vida corriente).

El GMT se conoce como **razonamiento inverso** y puede resumirse de la forma siguiente:

(Conocimiento):	Si x es A Entonces y es B
(Hecho):	y es B'

(Consecuencia):	x es A'

Lo que se expresa como $A' = B'$ o R

La tabla 7.3 muestra diversos criterios que pueden aplicarse para la selección del A' de la consecuencia en función del conjunto borroso B' utilizado en el hecho. El GMT generalizado es equivalente al *modus tolens* clásico para el criterio 5 (coincide si $A' = \text{no } A$ y $B' = \text{no } B$); por su parte, los criterios 5 a 8-2 están de acuerdo con el sentido común. Finalmente, hay que destacar como en el caso anterior que el criterio 8-2 no es válido en la lógica formal, pero se utiliza en el razonamiento común.

	y es B'	x es A'
criterio 5	y no es B	x no es A
criterio 6	y no es muy B	x no es muy A
criterio 7	y no es más o menos B	x no es más o menos A
criterio 8-1	y es B	x es desconocido
criterio 8-2	y es B	x es A

Tabla 7.3. Criterios intuitivos para GMT

7.8.4 Implicación borrosa

Si se definen dos conjuntos borrosos A y B en U y V , respectivamente, una **implicación borrosa** de A en B , que se indica con $A \rightarrow B$, es una relación borrosa en $U \times V$, que puede venir definida por alguna de las siguientes funciones de inclusión empleadas en la literatura de lógica borrosa:

• **Conjunción borrosa** $\mu_{A \rightarrow B}(u, v) = \mu_A(u) * \mu_B(v)$ (7.19)

• **Disyunción borrosa** $\mu_{A \rightarrow B}(u, v) = \mu_A(u) \dot{+} \mu_B(v)$ (7.20)

• **Implicación material** $\mu_{A \rightarrow B}(u, v) = \mu_{\bar{A}}(u) \dot{+} \mu_B(v)$ (7.21)

• **Cálculo proposicional** $\mu_{A \rightarrow B}(u, v) = \mu_{\bar{A}}(u) \dot{+} \mu_{A * B}(v)$ (7.22)

• **Modus ponens generalizado** $\mu_{A \rightarrow B}(u, v) = \sup \left\{ c \in [0, 1] \mid \mu_A(u) * c \leq \mu_B(v) \right\}$ (7.23)

• **Modus tolens generalizado** $\mu_{A \rightarrow B}(u, v) = \inf \left\{ c \in [0, 1] \mid \mu_B(v) \dot{+} c \leq \mu_A(u) \right\}$ (7.24)

7.9 REGLAS BORROSAS

Las **reglas borrosas** combinan uno o más conjuntos borrosos de entrada, llamados **antecedentes o premisas**, y les asocian un conjunto borroso de salida, llamado **consecuente o consecuencia**. Los conjuntos borrosos de la premisa se asocian mediante conjuntivas lógicas como *y*, *o*, etc. Una regla típica, de tipo IF-THEN, para un sistema de control sería “*Si error es positivo_pequeño y derivada_de_error es negativo_pequeño Entonces acción es positiva_pequeña*”, que se suele expresar abreviadamente mediante expresiones del tipo *Si E es PP y dE es NP Entonces U es PP*.

Las reglas borrosas permiten expresar el conocimiento que se dispone sobre la relación entre antecedentes y consecuentes. Para expresar este conocimiento de forma completa normalmente se precisa de varias reglas, que se agrupan formando lo que se conoce como una **base de reglas**, es decir, el conjunto de reglas que expresan las relaciones conocidas entre antecedentes y consecuentes.

La base de reglas se puede representar bien como una tabla de las reglas que la forman, o bien como una **memoria asociativa borrosa o FAM** (*Fuzzy Associative Memory*). Las FAM son matrices que representan la consecuencia de cada regla definida para cada combinación de dos entradas; de ello se verá un ejemplo en el capítulo siguiente (véase la Figura 8.5). Las FAM permiten realizar una representación gráfica clara de las relaciones entre dos variables lingüísticas de entrada y la variable lingüística de salida, pero requiere que se indique explícitamente todas las reglas que se pueden formar con estas dos variables de entrada. Cuando el número de conjuntos

de cada una de las particiones de entrada crece las FAM se hacen difícilmente manejables. Es posible también definir FAM de más de dos dimensiones, pero su tamaño se hace rápidamente excesivo y son más difíciles aún de manejar. En su lugar se suele trabajar con varias FAM de dimensión dos, para así definir subconjuntos de reglas que asocien las entradas de dos en dos en la base de reglas general.

Formalmente, una **base de reglas borrosa** es una colección de reglas $R^{(l)}$ con el formato

$$R^{(l)}: \text{ IF } x_1 \text{ is } F_1^l \text{ and ...and } x_n \text{ is } F_n^l \text{ THEN } y \text{ is } G^l \quad (7.25)$$

donde F_i^l y G^l son conjuntos borrosos en $U_i \subset \mathfrak{X}$ y $V \subset \mathfrak{Y}$, respectivamente, y $\mathbf{x} = (x_1, \dots, x_n)^T \in U_1 \times \dots \times U_n$ e $y \in V$ son variables lingüísticas. Este formato de reglas se conoce como **borroso puro o de tipo Mamdani**, por ser quien primero las propuso en 1974 para realizar un controlador borroso que estabiliza un sistema en torno a su punto de trabajo. Otro formato frecuente para las reglas es el llamado **de tipo Sugeno** [Sugeno 85a, 85b]. En este caso, la función de salida es una combinación lineal de las variables de entrada, o en un caso más general, una función genérica de las variables de entrada [Takagi 83].

$$R^{(l)}: \text{ IF } x_1 \text{ is } F_1^l \text{ and...and } x_n \text{ is } F_n^l \text{ THEN } y^l = f^l(\mathbf{x}) \quad (7.26)$$

Si llamamos M al número de reglas IF-THEN de la base de reglas entonces $l = 1, 2, \dots, M$ en las ecuaciones (7.25) y (7.26). El vector $\mathbf{x}$ representa el conjunto de las entradas, mientras que y es la salida del sistema borroso. Los sistemas borrosos descritos con n entradas x_i y una sola salida y , se conocen como **MISO** (*Multiple Input Single Output*), mientras que los que tienen varias salidas (de 1 hasta k) se conocen como **MIMO** (*Multiple Input Multiple Output*). Para estos últimos sistemas, se puede generalizar el formato anterior de las reglas, o bien descomponerlo en k sistemas de tipo MISO.

7.10 DISPOSITIVOS DE INFERENCIA BORROSA

Se llaman **dispositivos de inferencia borrosa** a los sistemas que interpretan las reglas de tipo IF-THEN de una base de reglas, con el fin de obtener los valores de salida a partir de los actuales valores de las variables lingüísticas de entrada al sistema. En un sistema borroso las reglas del tipo de la ecuación (7.25) se interpretan como una implicación borrosa (véase la sección 7.8.4) de $F_1^l \times \dots \times F_n^l \rightarrow G^l$ en $U \times V$, con $U \equiv U_1 \times \dots \times U_n \subset \mathfrak{X}^n$, $V \subset \mathfrak{Y}$. Si llamamos A' a la entrada en U del dispositivo de inferencia borrosa, cada regla l define un conjunto borroso B^l en V utilizando la composición Sup-Star (ecuación 7.18)

$$\mu_{B'}(y) = \sup_{\mathbf{x} \in U} \left[\mu_{F_1' \times \dots \times F_n' \rightarrow G'}(\mathbf{x}, y) * \mu_{A'}(\mathbf{x}) \right] \quad (7.27)$$

En la Sección 7.8.4 vimos seis formas diferentes de implicación borrosa (7.19-24), por lo que podemos proponer seis interpretaciones de la ecuación (7.27), para la ejecución de la implicación borrosa definida por una regla del tipo de la ecuación (7.25), dependiendo de las normas y conormas concretas que se empleen (7.14). Para simplificar las ecuaciones siguientes llamaremos $F_1' \times \dots \times F_n' \equiv A$ y $G' \equiv B$, con lo que la ecuación (7.25) puede expresarse simplemente como $A \rightarrow B$.

• **Implicación borrosa por la regla del mínimo:**

$$\mu_{A \rightarrow B}(\mathbf{x}, y) = \min[\mu_A(\mathbf{x}), \mu_B(y)] \quad (7.28)$$

• **Implicación borrosa por la regla del producto:**

$$\mu_{A \rightarrow B}(\mathbf{x}, y) = \mu_A(\mathbf{x}) \mu_B(y) \quad (7.29)$$

• **Implicación borrosa por la regla aritmética:**

$$\mu_{A \rightarrow B}(\mathbf{x}, y) = \min[1, 1 - \mu_A(\mathbf{x}) + \mu_B(y)] \quad (7.30)$$

• **Implicación borrosa por la regla Max-min:**

$$\mu_{A \rightarrow B}(\mathbf{x}, y) = \max\{\min[\mu_A(\mathbf{x}), \mu_B(y)], 1 - \mu_A(\mathbf{x})\} \quad (7.31)$$

• **Implicación borrosa por la regla Booleana:**

$$\mu_{A \rightarrow B}(\mathbf{x}, y) = \max\{1 - \mu_A(\mathbf{x}), \mu_B(y)\} \quad (7.32)$$

• **Implicación borrosa por la regla de Goguen:**

$$\mu_{A \rightarrow B}(\mathbf{x}, y) = \begin{cases} 1 & \mu_A(\mathbf{x}) \leq \mu_B(y) \\ \mu_B(y) / \mu_A(\mathbf{x}) & \mu_A(\mathbf{x}) > \mu_B(y) \end{cases} \quad (7.33)$$

En las ecuaciones 7.28 a 7.33 aparece el término $\mu_A(\mathbf{x}) = \mu_{F_1' \times \dots \times F_n'}(\mathbf{x})$, que a su vez puede ser definido por la regla del mínimo

$$\mu_{F_1' \times \dots \times F_n'}(\mathbf{x}) = \min\{\mu_{F_1'}(\mathbf{x}), \dots, \mu_{F_n'}(\mathbf{x})\} \quad (7.34)$$

o por la regla del producto

$$\mu_{F_1' \times \dots \times F_n'}(\mathbf{x}) = \mu_{F_1'}(\mathbf{x}) \dots \mu_{F_n'}(\mathbf{x}) \quad (7.35)$$

Una completa descripción de estas posibilidades y de las diferencias entre ellas puede verse en [Li-Xin 93].

Para concluir, la **salida final** de un dispositivo de inferencia borrosa puede consistir en:

- a) M conjuntos borrosos B^l , con $l=1, 2, \dots, M$, según la ecuación (7.27), cada uno de los cuales es el resultado de aplicar la entrada A' a cada una de las M reglas de la base de reglas.
- b) Un único conjunto borroso B' , que es la unión de los M conjuntos borrosos B^l calculado según

$$\mu_{B'}(y) = \mu_{B^1}(y) + \dots + \mu_{B^M}(y) \quad (7.36)$$

- c) M escalares y^l , con $l=1, 2, \dots, M$, si las reglas son del tipo Sugeno, según la ecuación (7.26), cada uno de los cuales es el resultado de aplicar la entrada A' a cada una de las M reglas de la base de reglas.

7.11 BORROSIFICADOR (*FUZZIFIER*)

El **borrosificador** establece una relación entre puntos de entrada no borrosos al sistema $\mathbf{x} = (x_1, \dots, x_n)^T$, y sus correspondientes conjuntos borrosos A en U (las variables procedentes del exterior serán, en general, valores no borrosos, y habrá que borrosificarlas previamente). Se pueden utilizar diversas estrategias de borrosificación:

a) **Borrosificador singleton.** Es el método de borrosificación más utilizado, principalmente en sistemas de control, y consiste en considerar los propios valores discretos como conjuntos borrosos. De otra forma, para cada valor de entrada $\mathbf{x}$ se define un conjunto A' que lo soporta, con función de pertenencia $\mu_{A'}(\mathbf{x}')$, de modo que $\mu_{A'}(\mathbf{x})=1$, ($\mathbf{x}'=\mathbf{x}$), y $\mu_{A'}(\mathbf{x}')=0$, para todos los otros $\mathbf{x}' \in U$ en los que $\mathbf{x}' \neq \mathbf{x}$.

b) **Borrosificador no singleton.** En este método de borrosificación se utiliza una función exponencial del tipo siguiente

$$\mu_{A'}(x') = a \cdot \exp \left[- \left(\frac{x' - x}{\sigma} \right)^2 \right] \quad (7.37)$$

función con forma de campana, centrada en el valor x de entrada, de anchura σ y amplitud a . Volveremos a ella en la sección 7.13.

7.12 DESBORROSIFICADOR (DEFUZZIFIER)

El **desborrosificador** es la función que transforma un conjunto borroso en V , normalmente salida de un dispositivo de inferencia borrosa, en un valor no borroso $y \in V$. Para esta tarea se utilizan diversos métodos:

a) **Desborrosificador por máximo**, definido como

$$y = \arg \sup_{y \in V} (\mu_{B^l}(y)) \quad (7.38)$$

es decir, y es el punto de V en que $\mu_{G^l}(y)$ alcanza su valor máximo, donde $\mu_{B^l}(y)$ está definido según la ecuación 7.36 (la unión de los B^l de salida).

b) **Desborrosificador por media de centros**, definido como

$$y = \frac{\sum_{l=1}^M \bar{y}^l (\mu_{B^l}(\bar{y}^l))}{\sum_{l=1}^M (\mu_{B^l}(\bar{y}^l))} \quad (7.39)$$

donde $\bar{y}^l$ representa el centro del conjunto borroso G^l (definido como el punto de V en el que $\mu_{G^l}(y)$ alcanza su valor máximo), y $\mu_{B^l}(y)$ está definido según 7.27.

c) **Desborrosificador por centro de área**, definido como

$$y = \frac{\sum_{l=1}^M M^l(\mu_{B^l}(\bar{y}^l))}{\sum_{l=1}^M A^l(\mu_{B^l}(\bar{y}^l))} = \frac{\sum_{l=1}^M \int_V \mu_{B^l}(\bar{y}^l)^2 dy M^l(\mu_{B^l}(\bar{y}^l))}{\sum_{l=1}^M \int_V \mu_{B^l}(\bar{y}^l) dy} \quad (7.40)$$

donde M^l es el momento (en torno al eje y del universo de discurso de la salida V) de la función de inclusión del conjunto borroso G^l , A^l es el área, y $\mu_{B^l}(y)$ está definida según la ecuación 7.27.

Estos métodos de desborrosificación son los empleados para obtener el valor de salida no borrosa de un dispositivo de inferencia borrosa que utiliza reglas de tipo Mamdani (ecuación 7.25). Si las reglas utilizadas son del tipo Sugeno (ecuación 7.26), el valor de salida no borrosa se obtiene como media ponderada de las salidas de cada regla de la base de reglas según

$$y = \frac{\sum_{l=1}^M y^l (\mu_{A^l}(\mathbf{x}))}{\sum_{l=1}^M (\mu_{A^l}(\mathbf{x}))} \quad (7.41)$$

donde y^l es la salida de la regla l , y el término $\mu_{A^l}(\mathbf{x})$ se calcula utilizando 7.34 o 7.35 (reglas del mínimo y del producto, respectivamente). Este valor y^l de la salida

de una regla del tipo Sugeno (ecuación 7.26) se calcula frecuentemente como una combinación lineal de las entradas

$$y^l = f_l(\underline{x}) = a_{l,0} + \sum_{i=1}^n a_{l,i} x_i \quad (7.42)$$

7.13 DESARROLLO DE SISTEMAS BORROSOS

Tras describir el amplísimo abanico de posibilidades para los distintos aspectos de un sistema borroso, expondremos algunas de las elecciones más comunes. La selección de los detalles concretos de implementación del sistema borroso dependerá de diversos condicionantes. Hablando en términos generales, los principales son:

- **Eficiencia computacional.** Para problemas complejos, con muchas variables lingüísticas o muchas reglas, o en realizaciones en microcontroladores de poca capacidad de cálculo y poca memoria, resulta fundamental seleccionar métodos que no requieran muchos cálculos o mucha memoria. Así, son preferibles en este caso funciones de inclusión triangulares o trapezoidales frente a las exponenciales, y el cálculo de máximos frente a multiplicaciones. Además, las funciones de inclusión de tipo singleton para la salida producen sistemas más simples, aunque son más sensibles al ruido de las entradas.

- **Facilidad de adaptación.** En aplicaciones en las que se requiere que el sistema pueda realizar aprendizaje puede ser necesario que la función de salida $y = f(\mathbf{x})$ sea derivable respecto de los parámetros que se han de ajustar. En este caso, por el contrario, son preferibles funciones de inclusión exponenciales frente a las triangulares o trapezoidales, y las multiplicaciones frente al cálculo de máximos.

Opciones más habituales en el desarrollo de sistemas borrosos

Expondremos a continuación alguna de las opciones más utilizadas para el diseño de sistemas borrosos. La deducción de las expresiones que proporcionaremos se puede consultar en [Wang 94] (aunque invitamos al lector a tratar de deducirlas él mismo haciendo uso de lo expuesto hasta el momento sobre lógica borrosa).

a) Un sistema de lógica borrosa con desborrosificador por media de centros (ecuación 7.39), implicación borrosa por la regla del producto (7.29 y 7.35), y borrosificador singleton, produce la siguiente función de salida

$$f(\mathbf{x}) = \frac{\sum_{l=1}^M y^l \left(\prod_{i=1}^n \mu_{F_i^l}(x_i) \right)}{\sum_{l=1}^M \left(\prod_{i=1}^n \mu_{F_i^l}(x_i) \right)} \quad (7.43)$$

donde $\bar{y}^l$ es el centro del conjunto borroso G^l (el punto de V en que $\mu_{G^l}(y)$ alcanza su máximo valor, que se asume como $\mu_{G^l}(y) = 1$).

b) Un sistema de lógica borrosa con desborrosificador por media de centros (ecuación 7.39), implicación borrosa por la regla del mínimo (7.28 y 7.34), y borrosificador singleton, produce la siguiente función de salida

$$f(\mathbf{x}) = \frac{\sum_{l=1}^M \bar{y}^l \left[\min(\mu_{F_1^l}(x_1), \dots, \mu_{F_n^l}(x_n)) \right]}{\sum_{l=1}^M \left[\min(\mu_{F_1^l}(x_1), \dots, \mu_{F_n^l}(x_n)) \right]} \quad (7.44)$$

donde $\bar{y}^l$ es, de nuevo, el centro del conjunto borroso G^l .

Como se ha indicado antes, en aplicaciones en las que se requiere que el sistema posea capacidad de aprendizaje puede resultar necesario que la función de salida $y = f(\mathbf{x})$ sea derivable respecto de los parámetros que se han de ajustar. Una elección frecuente en este caso es el empleo de funciones de inclusión gaussianas

$$\mu_{F_i^l}(x_i) = a_i^l \exp \left[- \left(\frac{x_i - \bar{x}_i^l}{\sigma_i^l} \right)^2 \right] \quad (7.45)$$

donde a_i^l y σ_i^l son los parámetros que dan la forma concreta de la gaussiana (amplitud y anchura, respectivamente). A partir de este tipo de función de pertenencia puede obtenerse el sistema borroso.

c) Un sistema de lógica borrosa con desborrosificador por media de centros (ecuación 7.39), implicación borrosa por la regla del producto (7.29 y 7.35), y borrosificador singleton con funciones de tipo gaussiano, produce la siguiente función de salida

$$f(\mathbf{x}) = \frac{\sum_{l=1}^M \bar{y}^l \left(\prod_{i=1}^n a_i^l \exp \left[- \left(\frac{x_i - \bar{x}_i^l}{\sigma_i^l} \right)^2 \right] \right)}{\sum_{l=1}^M \left(\prod_{i=1}^n a_i^l \exp \left[- \left(\frac{x_i - \bar{x}_i^l}{\sigma_i^l} \right)^2 \right] \right)} \quad (7.46)$$

Este sistema es uno de los más utilizados en sistemas con entrenamiento. Los parámetros de este sistema son $\bar{y}^l$, a_i^l , $\bar{x}_i^l$ y σ_i^l , que suelen estar sometidos a ciertas restricciones: $\bar{y}^l \in V$, $a_i^l \in (0,1)$, $\bar{x}_i^l \in U_i$ y $\sigma_i^l > 0$.

En el siguiente capítulo expondremos un ejemplo concreto y detallado de desarrollo de un sistema borroso, lo que ilustrará muchos de los conceptos expuestos de una manera formal en el presente capítulo.

7.14 BORROSIDAD Y PROBABILIDAD

Se ha de evitar desde el principio confundir la función de pertenencia de un conjunto borroso con una función de densidad de probabilidad. Debe tenerse siempre presente que la función de pertenencia de un conjunto borroso indica hasta qué punto cierto valor de una magnitud puede ser incluido en un conjunto borroso, mientras que la probabilidad, por su parte, indica la frecuencia con que los diversos valores de una magnitud se presentan.

Explicándolo con el clásico ejemplo de la botella, la función de pertenencia indica el grado en que podemos incluir una cierta botella dentro del conjunto de las *botellas vacías* y en el de las *botellas llenas*, mientras que la probabilidad nos informa sobre cuántas botellas de las encontradas podremos incluir en cada uno de dichos conjuntos. Una probabilidad 0.33 de *botellas vacías* nos indica que de cada 100 botellas que tomemos 33 estarán vacías, mientras que una pertenencia de 0.33 al conjunto *botellas vacías* indicará que nuestra botella incluye un tercio de litro del líquido de que se trate (supuesta una botella de un litro de capacidad).

Aunque muchas de las expresiones matemáticas de la lógica borrosa son similares a otras del campo de la probabilidad, su sentido es bien distinto. Las funciones de pertenencia a un conjunto son fijadas arbitrariamente por el observador, indicando el significado que éste asigna a cada uno de las variables lingüísticas que definen los conjuntos. Por el contrario, la probabilidad se determina por la observación de la ocurrencia de los valores de una magnitud, en algunos casos se realiza la medida de esta probabilidad, y en otros se supone un modelo y se comprueba su validez.

CAPÍTULO 8

SISTEMAS DE CONTROL BORROSO

Expuestos ya de un modo más o menos formal los conceptos básicos relacionados con la lógica y sistemas borrosos, en este capítulo abordaremos el estudio específico de los sistemas de control basados en lógica borrosa.

En primer lugar desarrollaremos con cierto detalle un sencillo ejemplo de control borroso; este caso de estudio servirá para aclarar muchas de las ideas expuestas con anterioridad de forma más abstracta. A continuación, pasaremos a realizar una clasificación de los diferentes tipos de sistemas de control borroso disponibles, que serán explicados con mayor o menor detenimiento. Es de destacar que el control borroso, principal aplicación de los sistemas borrosos, aparte de un tema de estudio académico, resulta muy importante desde un punto de vista industrial, campo en el que existen desde hace tiempo multitud de aplicaciones de estos sistemas en funcionamiento.

8.1 INTRODUCCIÓN AL CONTROL BORROSO

Los sistemas expertos de control borroso basados en reglas, conocidos como **controladores borrosos o FLC** (*Fuzzy Logic Controllers*), o también, **sistemas de inferencia borrosa o FIS** (*Fuzzy Inference Systems*, FIS), son sin duda la aplicación más extendida de la lógica borrosa. Una introducción a este tema puede verse en [Conner 93, Brubaker 92, Passino 98], donde se muestran ejemplos de controladores borrosos.

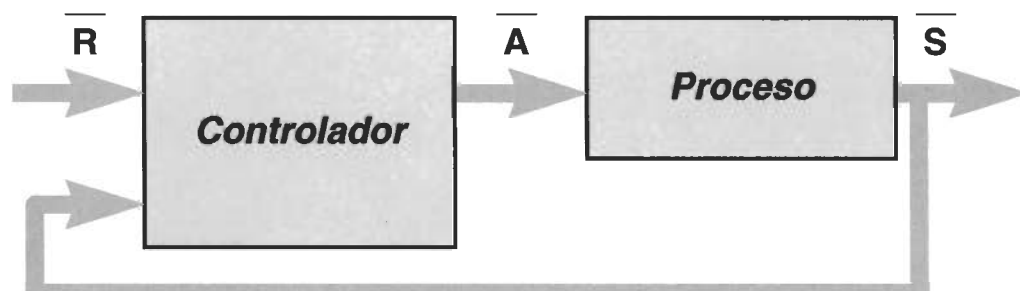


Figura 8.1. Control directo de un proceso o sistema

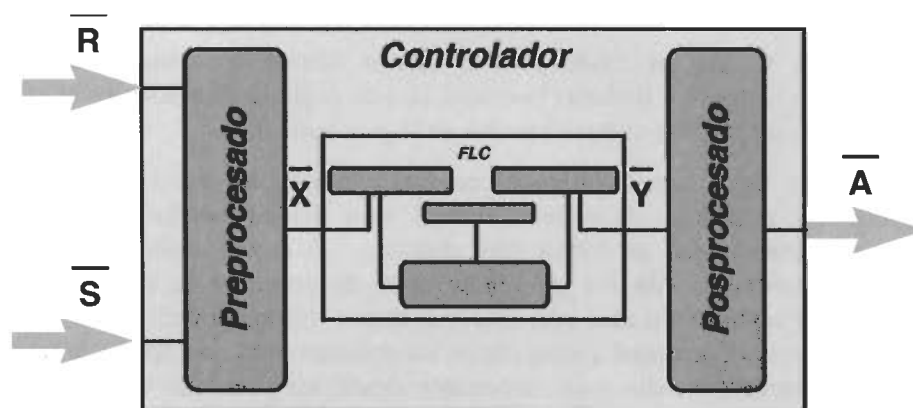


Figura 8.2. Estructura de un controlador (el núcleo FLC es el controlador borroso)

Como se puede observar en la Figura 8.1, para controlar un proceso o sistema se hace uso de un módulo controlador, que recibe como entradas una o varias variables de control llamadas generalmente referencias, $\bar{R}$, y una o varias variables de salida del propio proceso, $\bar{S}$, produciendo como salida una o varias variables, que se conocen como actuadores $\bar{A}$. Normalmente el objetivo del control es mantener $\bar{R} = \bar{S}$. Por ejemplo, en el caso de una calefacción doméstica, el controlador recibe una consigna de temperatura que fija el usuario, y mide la temperatura de la habitación por medio de un sensor. En función de los valores de estas dos entradas, el controlador conecta o desconecta el sistema de calentamiento, si la calefacción es eléctrica actuando sobre los radiadores de cada habitación, y si es de gas o fuel encendiendo o apagando el quemador.

La estructura típica de un controlador basado en un sistema borroso puede verse en la Figura 8.2. Un primer bloque realiza un preprocesado de las variables de entrada, que proporciona el vector de entradas al controlador borroso o FLC. El

controlador borroso aplica la entrada que recibe a la base de reglas, para obtener la salida. Finalmente, esta salida puede requerir un procesado final (postprocesado), con el fin de adecuarla al proceso que se ha de controlar. Como se verá con más detalle en el capítulo siguiente, el tipo de preprocesado y postprocesado determina la clase de controlador, e influye de forma considerable en sus propiedades.

La estructura interna de un controlador borroso o FLC se muestra en la Figura 8.3. Un primer elemento llamado borrosificador realiza la conversión de valores discretos a términos borrosos (la descripción de este elemento puede verse en la sección 7.11). Su salida es utilizada por el dispositivo de inferencia borrosa para aplicarla a cada una de las reglas de la base de reglas, siguiendo el método de inferencia seleccionado (véase la sección 7.10). La salida de este bloque pueden ser M conjuntos borrosos B^l , con $l=1,2,\dots,M$, según la ecuación 7.26, o bien un único conjunto borroso B' , que es la unión de los M conjuntos borrosos según 7.36, o bien M escalares y^l , con $l=1,2,\dots,M$, si las reglas son del tipo Sugeno (ecuación 7.26). Finalmente, el desborrosificador transformará estos conjuntos borrosos en un valor no borroso $y \in V$ (véase la sección 7.12).

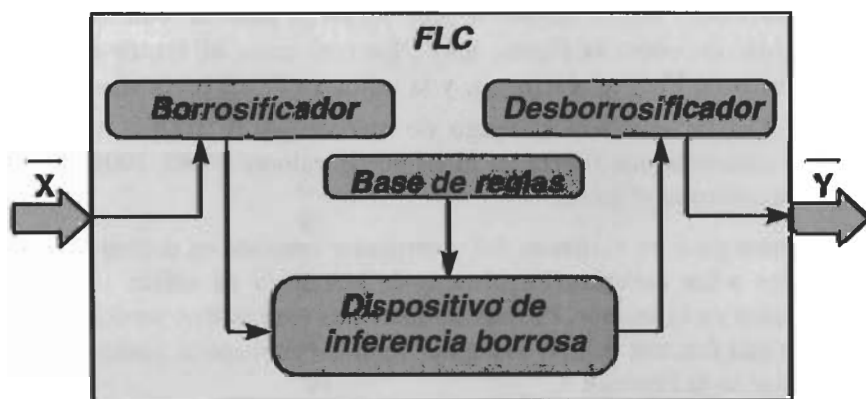


Figura 8.3. Estructura de un controlador borroso o FLC

8.2 UN PRIMER EJEMPLO

Vamos a desarrollar a continuación un ejemplo completo de desarrollo de un controlador basado en un sistema de lógica borrosa, haciendo uso de muchos de los conceptos introducidos en el capítulo anterior. El ejemplo elegido es el control de un sistema no lineal simple conocido como **péndulo inverso**.

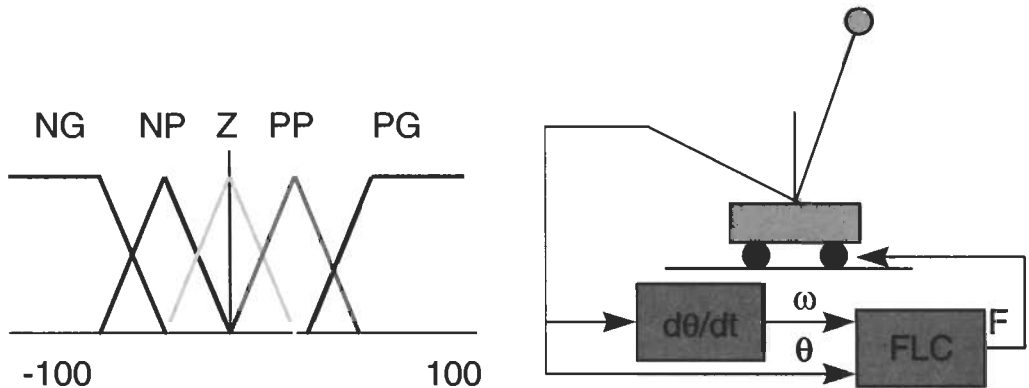


Figura 8.4. Control del péndulo inverso

Se trata de mantener en posición vertical una varilla que se apoya en un carrito movido por un motor, sistema que se asemeja a un péndulo invertido (Figura 8.4). El sistema controlador fijará la fuerza F que debe realizar el motor del carrito; las entradas al controlador son el ángulo θ que forma el péndulo con la vertical y su velocidad angular ω (véase la Figura 8.4). En este caso, el vector de entradas al controlador borroso o FLC es $x = (\theta, \omega)$, y la salida $y = F$. El preprocesado calcula la derivada y realiza un escalado al rango de valores $[-100, 100]$. A la salida, el postprocesado convierte una fuerza en el rango de valores $[-100, 100]$ a los valores adecuados para controlar el motor.

El siguiente paso en el diseño del controlador consiste en definir las particiones correspondientes a las variables lingüísticas de entrada y de salida. Siguiendo los consejos indicados en la sección 7.5, seleccionaremos para ambas particiones de cinco elementos, con una función de tipo triangular centrada en torno al punto central, como se puede apreciar en la Figura 8.4.

El siguiente paso consiste en seleccionar el tipo de reglas a utilizar en el controlador, en principio podemos elegir entre las de tipo Mamdani o las de tipo Sugeno. Las reglas de tipo Mamdani (ecuación 7.25) permiten expresar el conocimiento previo disponible sobre el sistema, expresando así el adquirido durante el proceso de optimización. Por su parte, las reglas de tipo Sugeno (ecuación 7.26) simplifican los cálculos de la salida, pero en general no resultan tan adecuadas para expresar el conocimiento de los expertos. Como en este problema disponemos de un conocimiento intuitivo de las acciones de control a realizar sobre el sistema, nos decantaremos por el empleo de reglas tipo Mamdani.

A continuación se ha de definir la base de reglas, las cuales asociarán a cada una de las posibles combinaciones de las entradas un valor de salida. Como se ha descrito en la sección 7.9 (Reglas borrosas), es posible describir esta base de reglas

con el uso de una memoria asociativa borrosa o FAM, como la de la Figura 8.5. Utilizamos las abreviaturas ya conocidas {NG, NP, Z, PP, PG}, para denotar {Negativo Grande, Negativo Pequeño, Cero, Positivo Pequeño, Positivo Grande}.

Las reglas pueden presentarse también en el formato de la ecuación (7.25). A modo de ejemplo, las dos siguientes reglas pueden extraerse de la FAM (Figura 8.5):

R1: Si θ es Z y ω es Z Entonces F es Z

R2: Si θ es PP y ω es NP Entonces F es NP

La regla *R1* expresa que si el ángulo es cero y la velocidad angular también, entonces no hay que ejercer fuerza alguna (puesto que la varilla estará en equilibrio). La regla *R2*, por su parte, indica que si el ángulo es pequeño positivo y la velocidad pequeña negativa, entonces la fuerza que se deberá ejercer debe ser pequeña negativa para tender al equilibrio.

Desarrollada la base de reglas borrosas, se han de seleccionar después los métodos de borrosificación, de inferencia y de desborrosificación. Para esta elección se han de considerar fundamentalmente los aspectos relativos a la eficiencia computacional y a la facilidad de adaptación, expuestos en la sección 7.12. Así, en el caso frecuente de realizar un sistema simple basado en un microcontrolador de 8 bits y sin funciones de aprendizaje, la opción más adecuada será la del sistema descrito en la ecuación 7.44, es decir, un sistema de lógica borrosa con desborrosificador por media de centros (7.39), implicación borrosa por la regla del mínimo (7.28) y (7.34), y borrosificador tipo singleton.

		θ				
		NG	NP	Z	PP	PG
ω	NG	NG	NP	NP	NP	NP
	NP	NP	Z	PP	NP	Z
	Z	NP	PP	Z	PG	PP
	PP	Z	PP	PP	Z	PP
	PG	PP	PP	PP	PP	PG

Figura 8.5. FAM para el control del péndulo inverso. A cada combinación de las variables de entrada, θ y ω , se asocia una consecuencia

Para entender cómo operaría el sistema borroso diseñado, es decir, dadas unas determinadas entradas cómo se obtiene la salida, realizaremos los cálculos detallados para un caso simplificado, considerando únicamente las dos reglas *R1* y *R2* anteriores, y dadas las entradas $\mathbf{x} = (\theta, \omega) = (10, -5)$. La premisa de cada regla se calcula, como muestra la Figura 8.6, siguiendo la ecuación 7.34

$$\mu_{F_1' \times \dots \times F_n'}(\mathbf{x}) = \min\{\mu_{F_1'}(\mathbf{x}), \dots, \mu_{F_n'}(\mathbf{x})\}$$

es decir, como el mínimo de las funciones de inclusión para cada término. Según esto, $\mu_{A^i}(\mathbf{x}) \equiv \mu_{F_1' \times \dots \times F_n'}(\mathbf{x})$, con $\mathbf{x} = (10, -5)$, valdrá para cada regla

$$\begin{aligned} R1: \quad \alpha_1 &= \mu_{A^1}(\mathbf{x}) = \min\{0.6, 0.8\} = 0.6 \\ R2: \quad \alpha_2 &= \mu_{A^2}(\mathbf{x}) = \min\{0.6, 0.2\} = 0.2 \end{aligned} \quad (8.1)$$

Por otra parte, la implicación borrosa por la regla del mínimo viene según (7.28)

$$\mu_{A \rightarrow B}(\mathbf{x}, y) \equiv \mu_{F_1' \times \dots \times F_n' \rightarrow G'}(\mathbf{x}, y) = \min[\mu_{F_1' \times \dots \times F_n'}(\mathbf{x}), \mu_{G'}(y)]$$

y el conjunto de salida correspondiente a cada regla *l*, $\mu_{B^l}(y)$, viene dado a su vez por la ecuación 7.27

$$\mu_{B^l}(y) = \sup_{\mathbf{x} \in U} [\mu_{F_1' \times \dots \times F_n' \rightarrow G'}(\mathbf{x}, y) * \mu_{A^l}(\mathbf{x})]$$

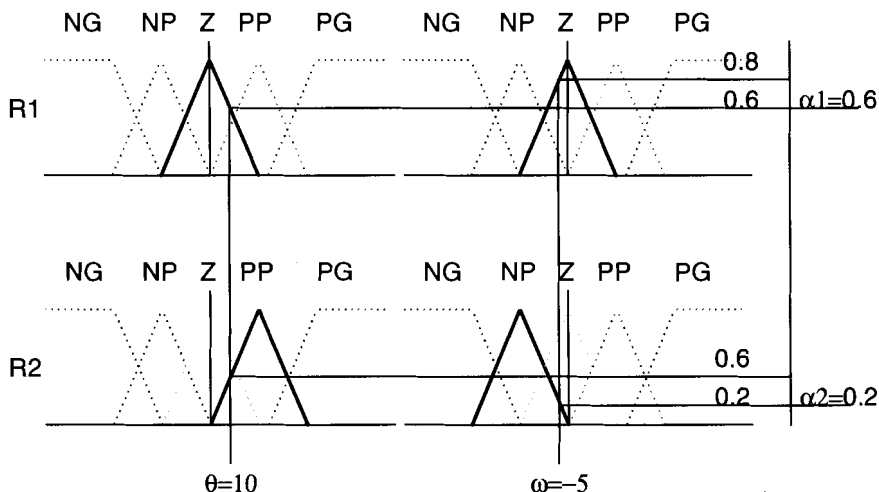


Figura 8.6. Cálculo de las premisas de las reglas *R1* y *R2*

siendo A' la entrada en U del dispositivo de inferencia borrosa. Combinando todo ello, se tiene

$$\mu_{B^i}(y) = \sup_{x \in U} [\min[\mu_{F_1^i \times \dots \times F_n^i}(x), \mu_{G^i}(y), \mu_{A^i}(x)]] \quad (8.2)$$

Si utilizamos un borrosificador singleton, se cumple que $\mu_{A^i}(x') = 1$ sólo para $x' = x$, siendo $\mu_{A^i}(x') = 0$ para todos los demás valores en U . Por ello, la ecuación anterior queda reducida a

$$\mu_{B^i}(y) = \min[\mu_{F_1^i \times \dots \times F_n^i}(x'), \mu_{G^i}(y)] \quad (8.3)$$

lo que supone el *truncado* de los conjuntos borrosos de salida a los valores obtenidos de la premisa (Figura 8.7).

En la Figura 8.7 pueden verse los conjuntos de salida correspondientes a las reglas $R1$ y $R2$, $\mu_{B^1}(y)$ y $\mu_{B^2}(y)$ (Z y NP , respectivamente). Se puede ver también la salida global del dispositivo de inferencia borrosa, el conjunto borroso $\mu_{B^i}(y)$ obtenido al aplicar (7.36), que es la unión de los conjuntos borrosos de cada regla.

$$\mu_{B^i}(y) = \mu_{B^1}(y) \dot{+} \dots \dot{+} \mu_{B^M}(y)$$

Por último, queda por calcular el valor final de la salida, siguiendo el procedimiento de la media de centros (7.44) (recordemos que se denota por $\bar{y}^i$ al centro del conjunto borroso de salida G^i).

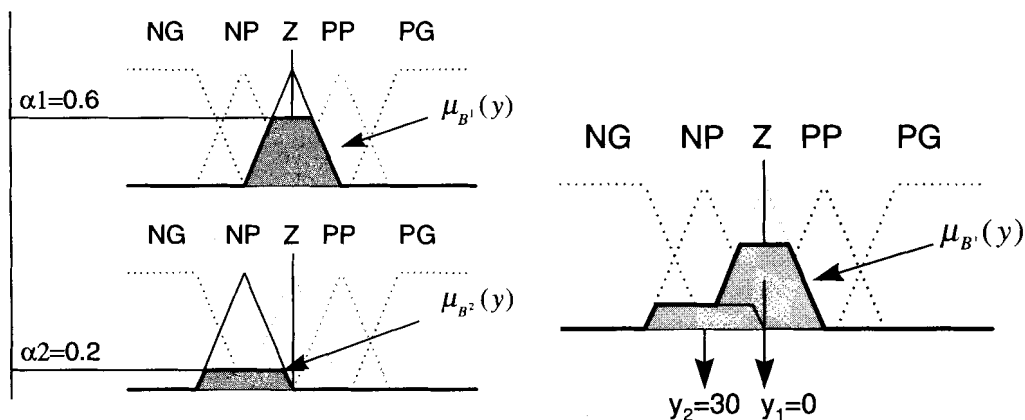


Figura 8.7. Conjuntos borrosos de salida, truncados a los valores de las premisas de las reglas $R1$ y $R2$, y cálculo de la consecuencia (fuerza F) por media de centros

$$f(\mathbf{x}) = \frac{\sum_{l=1}^M y^l \left[\min(\mu_{F_1^l}(x_1), \dots, \mu_{F_n^l}(x_n)) \right]}{\sum_{l=1}^M \left[\min(\mu_{F_1^l}(x_1), \dots, \mu_{F_n^l}(x_n)) \right]} = \frac{0.6 * 0 + 0.2 * 30}{0.6 + 0.2} = 7.5$$

Ello viene a indicar que la regla *R1*, cuya salida es *Z* (de centro 0), se cumple en una cantidad de 0.6, mientras que la regla *R2*, cuya salida es *NP* (de centro 30), se cumple en 0.2. La salida *F* es una media ponderada de los centros de los conjuntos de salida correspondientes a cada regla, es decir, la *F* que actúa sobre el sistema no es ni 0 ni 30, sino el promedio ponderado 7.5.

8.3 TIPOS DE CONTROLADORES BORROSOS

La arquitectura del controlador a utilizar depende de la aplicación concreta a llevar a cabo. No resulta fácil realizar una clasificación genérica de todas las arquitecturas posibles de controladores basados en lógica borrosa; no obstante, consideraremos los siguientes tres grandes grupos de controladores, que a continuación comentaremos con mayor detalle:

- 1) Controladores borrosos directos sin optimización (sección 8.3.1).
- 2) Controladores borrosos directos con optimización (sección 8.3.2).
- 3) Controladores borrosos híbridos (sección 8.3.3).

8.3.1 Controladores borrosos directos sin optimización

La estructura típica de uno de estos controladores borrosos puede verse en la Figura 8.2. En ella, un primer bloque realiza el preprocesado de las variables de entrada para proporcionar las entradas al controlador borroso. En el caso más simple este preprocesado puede consistir en un simple escalado de las magnitudes que se miden.

Si el controlador se realiza digitalmente, normalmente con un microprocesador, y alguna de las entradas es analógica, se necesitará amplificarla y convertirla a valores digitales con un circuito de acondicionamiento adecuado. En este caso, la amplificación analógica supone un primer escalado de la magnitud física que se mide, y la conversión a digital representa un segundo escalado que convierte la tensión analógica a un número dentro de un cierto rango (por ejemplo, para el caso de un conversor A/D de 8 bits el rango irá de 0 a 255). Para el caso de un escalado a valores digitales discretos es importante considerar la resolución empleada en la representación de la variable, con el fin de asegurar que no se pierde demasiada información en esta transformación.

El tipo de preprocesado de las entradas define la clase de controlador; en este sentido, los tipos más usuales de controladores son

- 1) Controlador proporcional: $X = f(e)$
- 2) Controlador integral: $X = f(s)$
- 3) Proporcional-derivativo: $X = f(e, \Delta e)$
- 4) Proporcional-integral: $X = f(e, s)$
- 5) Con realimentación no lineal: $X = f(R, S)$

En las expresiones anteriores se denota con e el error, con Δe su derivada, y con s su integral. R representa las variables de control y S las salidas del proceso.

Los controladores de los tipos 1 al 4 han sido muy utilizados por su similitud con los controladores clásicos; permiten formular reglas independientes del punto de referencia R utilizando sólo el error e , su derivada Δe , o la integral s . Por su parte, los controladores del tipo 5 permiten definir comportamientos diferentes según el punto de referencia. Esto resulta de especial importancia en el control de sistemas no lineales, en los que su comportamiento cambia según el punto de operación. Este tipo de controladores permite también definir reglas para controlar el sistema en caso de averías o de funcionamiento anormal, que permitan llevarlo a un estado seguro sin grandes daños.

Algunos ejemplos de controladores borrosos directos se pueden ver en [Maeng 93], donde se controla un sistema de grúa sobre ménsulas en una nave industrial; en [Reyes de los Mozos 93], donde se emplea para compensar un oscilador a cristal; y en [Jihong 93a], donde se utiliza para el control de un sistema de segundo orden.

Una buena visión sobre los sistemas de control borroso puede encontrarse en la referencia [García Cerezo 93], y un estudio en profundidad sobre el control de procesos con lógica borroso se realiza en [Driankov 93]. Un tema importante al considerar los controladores borrosos son los criterios de evaluación, que permitan una adecuada comparación; estos temas se estudian en [Williams Tom 93], donde se analizan los tiempos de respuesta de diversas implementaciones; y también se tratan en [Boverie 93], donde se propone un problema patrón para la evaluación de los controladores.

Los controladores directos permiten realizar control de sistemas utilizando una descripción lingüística de las reglas de control. Estas reglas han de obtenerse del conocimiento que disponen los expertos sobre el control del sistema, o bien por procedimientos heurísticos, siendo esta solución en muchos casos suficiente para obtener un buen control del sistema. Cuando se precisa una solución más eficiente, o no se dispone de este conocimiento previo, se han de utilizar los controladores borrosos directos con optimización, que pasamos a exponer.

8.3.2 Controladores borrosos directos con optimización

Éstos parten de la estructura de los controladores borrosos directos sin optimización, añadiendo elementos que permiten ajustar sus parámetros internos con el fin de mejorar su eficiencia. La estructura de este tipo de controladores borrosos puede verse en la Figura 8.8.

Un primer elemento realiza la evaluación del funcionamiento del controlador para permitir al bloque siguiente decidir sobre las modificaciones a realizar. El bloque de evaluación puede calcular la sobreoscilación del sistema en torno al punto de trabajo, o el tiempo que tarda en estabilizarse el sistema tras una variación del mismo. Estos parámetros son utilizados por el bloque de ajuste para realizar las modificaciones necesarias en el FLC. Según el tipo de ajuste se distinguen diversos tipos de optimizaciones:

- Controladores borrosos auto-organizados.** En estos controladores los ajustes se realizan en la base de reglas del FLC.
- Controladores borrosos con auto-aprendizaje.** En este grupo se incluyen aquellos en los que se modifican parámetros escalares, del controlador o de los bloques de preprocesado o postprocesado.
- Controladores basados en modelado borroso.** Controladores que actúan como modeladores de la dinámica inversa del proceso.

Pasaremos a continuación a hablar algo más detenidamente de cada uno de ellos.

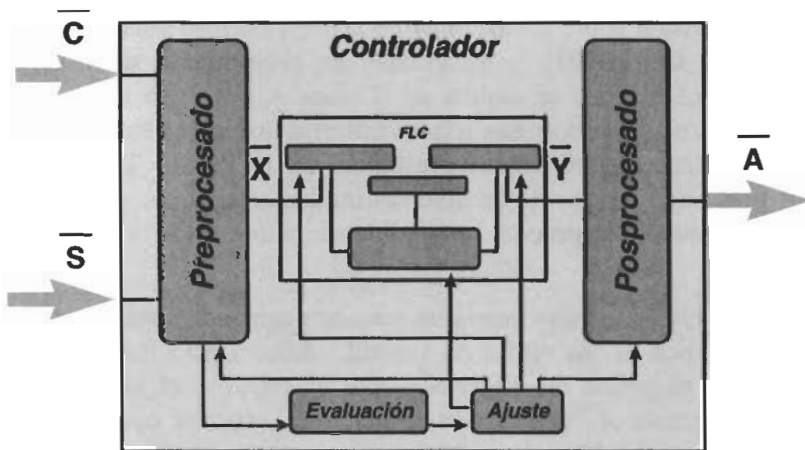


Figura 8.8. Estructura de un controlador borroso directo con optimización

Un **controlador borroso auto-organizado o SOC** (*Self-Organizing Controller*) es un sistema capaz de modificar automáticamente, y sin intervención humana, su base de conocimientos. A partir de las discrepancias con respecto a unos determinados criterios prefijados (que calcula el bloque de evaluación), el bloque de ajuste del control autoorganizado establece las oportunas modificaciones en la base de reglas.

Introducidos por Mamdani en 1979, los SOC se están mostrando de especial interés en multitud de aplicaciones; por ejemplo, en [Scharf 85] y en [Mandic 85] se introducen los SOC para control de robots. Los enfoques seguidos para el desarrollo de los SOC se han basado en criterios de evaluación local, es decir, orientados a acciones de control individuales.

El bloque de ajuste suele ser a sus vez un FLC que implementa una tabla de decisión que, en función de los criterios de evaluación, devuelve una indicación sobre la corrección requerida respecto a la respuesta ideal. Por lo tanto, este bloque de ajuste representa de algún modo la idea del diseñador respecto de la tolerancia mínima sobre la respuesta. Frecuentemente, este mecanismo de ajuste no modifica en realidad las reglas ya adquiridas, sino que incluye otras nuevas en la base de reglas. Por ello, los SOC presentan una serie de problemas relacionados con la convergencia del algoritmo, la generación de un excesivo número de nuevas reglas de control, la falta de optimización del protocolo lingüístico, etc.

Se han propuesto métodos de optimización de la base de reglas basados en algoritmos genéticos, como puede verse en [Fukuda 93, Sae-Hie 93, Herrera 93a 93b, Katai 93, Daihee 94]; más adelante se describirán estos métodos con cierto detalle. Otras técnicas de reducción del tamaño de la base de reglas se basan en el cálculo de la distancia a la trayectoria en el espacio de estado [Lembessis 93], o la minimización del tamaño de la base por criterios de entropía [Sánchez 93]. También se utiliza la equivalencia con modelos de redes neuronales para la depuración de la base de reglas [Taechon 93].

Parte de los problemas originados por la continua generación de reglas para cada operación (entre 60 y 100), pueden ser eliminados diseñando sistemas de evaluación alternativos, limitando el número de términos en los antecedentes, e incluyendo modificadores con capacidad de optimización del protocolo de control.

En [García Cerezo 87] se introduce un controlador borroso auto-organizado basado en un evaluador global de características y en un modelo no lineal del proceso bajo control. En este caso, la evaluación de características se realiza de forma cíclica sobre las trayectorias del sistema realimentado mediante un algoritmo borroso. En éste se utilizan cuantificadores borrosos del tipo *Mayor parte, Todos, Algunos, El 90%...*, con objeto de realizar la evaluación. Las reglas utilizadas son del tipo:

*Si mayoría puntos_de_trayectoria_sobre_regla_R son inaceptables
Entonces evaluar_la_actuación_precisa*

En [Shao 88] se introducen algunas mejoras en cuanto al comportamiento en aprendizaje del esquema propuesto por Mamdani, aplicándolo al control de un servomecanismo. En [Tanscheit 88] se presentan nuevos experimentos en control y seguimiento de trayectorias en robots. En [Zeungnan 91] se utiliza un SOC para el seguimiento visual en robots. Por último, en [Hayashi 91] se propone un regulador PI auto-sintonizado que incluye un paso adicional de escalado y que mejora el número de pasos requeridos en la modificación del protocolo. Como se comprueba en [Braae 79] y en [Aracil 89], la dinámica del sistema en bucle cerrado se ve notablemente afectada por los factores de escalado (fundamentalmente en la actuación).

Pasaremos a continuación al segundo tipo, los **controladores borrosos con autoaprendizaje**; en éstos no se modifican los términos lingüísticos, sino que se ajustan los parámetros escalares del controlador, o de los bloques de preprocesado o postprocesado. Una visión general de estos sistemas puede verse en [Cox 93a]. En su mayor parte responden a procedimientos y técnicas de aprendizaje del tipo del conocido descenso por el gradiente [Jang 92-93b, Shann 93, Furuhashi 93], similar al estudiado para los modelos de redes neuronales artificiales. En la mayoría de los casos el ajuste se centra en modificar las funciones de inclusión de las particiones de salida y, con menos frecuencia, las de entrada; al modificar estas funciones se ajusta el significado dado a las reglas, sin cambiar la estructura definida por el experto en el control del sistema.

El último tipo de controlador borroso directo con optimización al que haremos referencia es a aquellos que establecen **modelos borrosos del proceso a controlar**. Estas técnicas tienen sus orígenes en los primeros trabajos de Takagi y Sugeno [Takagi 83] y Sugeno y Nishida [Sugeno 85a], sobre identificación de sistemas mediante lógica borrosa y su aplicación al modelado de sistemas. Como en casos anteriores, la aplicación de las técnicas de modelado precisan una serie de simplificaciones sobre los parámetros relativos a antecedentes y consecuentes. Por ejemplo, en [Sugeno 85a] y en [Ichihashi 91], se emplean reglas llamadas de tipo Sugeno, mientras que en Graham y Newell [Graham 88] se hace uso de una técnica de modelado basada en la modificación directa de la matriz de relación.

Básicamente, los sistemas de control basados en modelos borrosos actúan como modeladores de la dinámica inversa del proceso. Este modelo puede ser realizado fuera de línea, como en el caso de Graham, o bien realizarse en tiempo real mediante un proceso de autoaprendizaje. En [Nakamori 91] se propone un método de diseño de un controlador predictivo mediante el uso de modelos borrosos de sistemas. En este caso, el diseño consta de dos fases: la primera fase de modelado permite la construcción de un modelo de la dinámica del proceso, mientras que la segunda consiste en la implantación de un esquema de control predictivo.

Se han propuesto también métodos para la elaboración de modelos borrosos a partir de ejemplos [Sugeno 93c]; también se han utilizado en combinación con redes neuronales en el proceso de modelado [Minho 93]. El control por modelado borroso

se ha utilizado en diversas aplicaciones, como el control de un grupo de elevadores [Chang-Bum 93], y la predicción de demanda eléctrica [Iokibe 93].

8.3.3 Controladores borrosos híbridos

Se denominan así aquellos sistemas de control formados por dos controladores interconectados, de los cuales uno es convencional (como los PID) y el otro es borroso. El primero se encarga básicamente del control, garantizando un comportamiento estable, mientras que el controlador borroso actúa en paralelo, introduciendo el componente heurístico en el proceso [Sung-Kwun 93].

Este segundo controlador borroso también puede emplearse para el ajuste de los parámetros del controlador convencional, como se indica en [Zhao 93]; usualmente, sus acciones se orientan a la mejora de ciertas características, como reducción de oscilaciones, mejoras del tiempo de establecimiento, etc. [Tanaka 93a]. Por ejemplo, en el control de sistemas de alto orden se puede conseguir la eliminación de las sobreoscilaciones con el uso de controladores híbridos [Jihong 93b].

Las primeras referencias a este tipo de estructuras aparecen en [Willaeys 80], pero su mayor presencia es sin duda a nivel industrial. Una buena parte de los reguladores borrosos industriales han sido realizados mediante esta técnica de hibridación; un ejemplo es el regulador de temperatura E5AF de OMRON, regulador PID combinado con un regulador borroso que funciona en paralelo [Bruijn 93].

Estas estrategias han sido muy utilizadas en la industria; por su similitud con los controles clásicos, el técnico de proceso se siente apoyado por su propia experiencia con controladores convencionales, y a su vez, se adapta a los nuevos planteamientos introducidos por el controlador borroso. En sectores como la industria química y nuclear, en donde la seguridad del funcionamiento es fundamental, se han utilizado estos controladores por estar bien establecido su funcionamiento.

También se incluyen en este tipo de controladores los formados por la combinación de lógica borrosa con redes neuronales [Buckley 93] y con sistemas expertos [Branko 91]; en este sentido, se han definido diversos tipos de equivalencias [Jang 93a]. Por último, hay que indicar que también se han realizado estudios comparativos de la eficiencia de las diversas soluciones [Sztandera 93].

CAPÍTULO 9

APRENDIZAJE EN SISTEMAS BORROSOS

Hasta el momento hemos establecido las bases de la lógica y sistemas borrosos, y hemos estudiado los sistemas borrosos de control o FLC, quizás su campo aplicación más importante, especialmente a nivel industrial. Como se ha apuntado ya en el capítulo anterior, los FLC, lejos de ser los entes estáticos considerados hasta hace unos años, pueden ser entrenados para optimizar su funcionamiento. Debe tenerse siempre presente que, como las redes neuronales artificiales, los FLC son aproximadores funcionales genéricos, es decir, dado un cierto nivel de error, se puede encontrar un FLC que aproxime cualquier función con un error menor que el fijado, y que para ello pueden hacer uso de diversas técnicas, algunas procedentes del campo de las redes neuronales (sistemas neuro-borrosos), y otras de otros campos, como los métodos estadísticos o los algoritmos genéticos.

Algunos de estos algoritmos de aprendizaje aplicables a los sistemas borrosos serán estudiados en el presente capítulo, incidiendo especialmente en los basados en el ya conocido BP y en algoritmos genéticos. Se tiene así lo que se viene a denominar un **sistema borroso adaptativo**.

9.1 INTRODUCCIÓN

Existen numerosos algoritmos de aprendizaje o entrenamiento aplicables al caso de los sistemas borrosos; algunos de los más importantes son los siguientes:

- Retropropagación (*backpropagation* o BP, véase el capítulo 2).
- Algoritmos genéticos (se explicarán en este capítulo).
- Mínimos cuadrados ortogonales (*orthogonal least squares*).
- Tablas de búsqueda (*look-up table*).

- Agrupamiento (*clustering*) por vecino más cercano (*nearest neighborhood*).

En general, todos los métodos de entrenamiento utilizados en redes neuronales artificiales pueden ser trasladados al campo de los FLC. Ésta ha sido la línea más habitualmente empleada en estos años, obteniéndose así sistemas en ocasiones denominados **neuroborrosos**, que tratan de aunar la capacidad de aprendizaje de las redes neuronales, con la facilidad de trabajo con información incierta y su traducción a reglas, de los sistemas borrosos. Muchos de estos sistemas neuroborrosos parten de bases similares al aprendizaje en el campo de las redes neuronales, como las de tipo BP (capítulo 2), aspecto que será tratado en la sección 9.2. Remitiremos al lector interesado en el resto de los métodos a los libros de [Wang 94, Lin 96, Jang 97]. En particular, en [Jang 97] se realiza un amplio estudio del **ANFIS** (*Adaptive Neuro-Fuzzy Inference System*), uno de los modelos neuroborrosos de mayor repercusión.

En el resto del capítulo nos centraremos más en los esquemas de aprendizaje basados en algoritmos genéticos (secciones 9.3 y 9.4), que están cobrando creciente importancia. Adelantaremos que las técnicas basadas en algoritmos genéticos o GA (*Genetic Algorithms*) resultan adecuadas para realizar una búsqueda global de la mejor solución dentro del espacio de búsqueda. A diferencia de los métodos de tipo BP, que son locales, los GA buscan la mejor solución posible simultáneamente en diversas zonas del espacio, lo que permite evitar el estancamiento en mínimos locales típico de muchos modelos neuronales. A modo de ilustración de lo que este hecho significa, los métodos de tipo BP se han comparado con encontrar la altura mínima en una superficie (valle) dejando rodar una bola ladera abajo desde un cierto punto de partida (que puede ser mejor o peor, según su elección), y esperar a ver dónde se detiene. Frente a ello, los GA son comparables a dejar llover sobre la superficie y observar globalmente dónde se acumula el agua.

Por otro lado, los métodos de tipo BP suelen proporcionar mejores resultados cuando se dispone de una solución cercana a la optima. Además, no necesitan tantos recursos de memoria como los GA, aunque exigen disponer de la derivada de la salida frente a los parámetros que se han de ajustar. A modo de resumen, dependiendo del tipo de problema, una solución resultará más adecuada que otra.

9.2 RETROPROPAGACIÓN (BP)

Las técnicas de optimización de FLC con BP [Wang 94, Jang 92-93b] son básicamente similares a las utilizadas en redes neuronales artificiales. En esta sección seguiremos el esquema de [Wang 94].

Observando la ecuación 7.46, correspondiente a un sistema borroso con borrosificador singleton, funciones gaussianas, implicación por producto y desborrosificador por media de centros

$$f(\mathbf{x}) = \frac{\sum_{l=1}^M \bar{y}^l \left(\prod_{i=1}^n a_i^l \exp \left[- \left(\frac{x_i - \bar{x}_i^l}{\sigma_i^l} \right)^2 \right] \right)}{\sum_{l=1}^M \left(\prod_{i=1}^n a_i^l \exp \left[- \left(\frac{x_i - \bar{x}_i^l}{\sigma_i^l} \right)^2 \right] \right)}$$

se puede apreciar que un sistema de lógica borrosa podría representarse como una red unidireccional (*feed-forward*) de tres capas. Desde este punto de vista, resulta directo aplicar la idea genérica del algoritmo BP (capítulo 2) al caso concreto del ajuste de los parámetros del sistema borroso $\bar{y}^l$, $\bar{x}_i^l$ y σ_i^l .

Así, supongamos se tiene un conjunto de pares-ejemplos $(\mathbf{x}^p, t^p)$, en donde a cada vector de entrada $\mathbf{x}^p \in U \subset \mathfrak{R}^n$ se asocia una salida objetivo $t^p \in V \subset \mathfrak{R}$; $p = 1, 2, \dots$ marca el número de patrón de aprendizaje. El objetivo de la optimización es encontrar los parámetros $\bar{y}^l$, $\bar{x}_i^l$ y σ_i^l de la función $f(\mathbf{x})$, del tipo de la ecuación 7.46, que minimicen la siguiente expresión que da el error cometido por el sistema

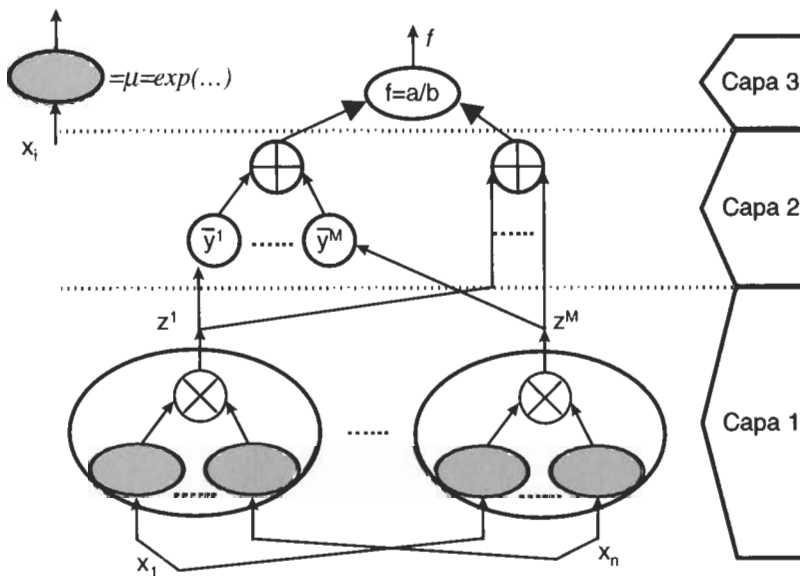


Figura 9.1. Representación de un sistema borroso como una red neuronal de tres capas [Wang 94]

$$e^p = \frac{1}{2} [f(\mathbf{x}^p) - t^p]^2 \quad (9.1)$$

En adelante usaremos e , f y t , para indicar e^p , $f(\mathbf{x}^p)$, y t^p , por simplicidad de notación. Para el entrenamiento (ajuste) del parámetros $\bar{y}^l$, centro del conjunto G^l de salida, se haría uso de la expresión recursiva

$$\bar{y}^l(k+1) = \bar{y}^l(k) - \alpha \left(\frac{\delta e}{\delta \bar{y}^l} \right)_k \quad (9.2)$$

donde $l=1, 2, \dots, M$ representan los distintos conjuntos borrosos de salida, y $k=1, 2, \dots$ son las iteraciones del proceso. El término α es el parámetro **ritmo de aprendizaje**. Como se puede observar en la Figura 9.1, f depende de $\bar{y}^l$ sólo por el término a definido según

$$\begin{aligned} a &\equiv \sum_{l=1}^M (\bar{y}^l z^l) \\ b &\equiv \sum_{l=1}^M z^l \\ z^l &= \prod_{i=1}^n a_i^l \exp \left(- \left(\frac{x_i - \bar{x}_i^j}{\sigma_i^j} \right)^2 \right) \end{aligned} \quad (9.3)$$

Utilizando la regla de la cadena, se obtiene para el entrenamiento de los centros de los conjuntos borrosos de salida la siguiente expresión

$$\bar{y}^l(k+1) = \bar{y}^l(k) - \alpha \frac{f - t}{b} \quad (9.4)$$

De manera similar para el entrenamiento de $\bar{x}_i^l$, centroides de las gaussianas que representan las funciones de pertenencia, se utiliza la expresión recursiva

$$\bar{x}_i^l(k+1) = \bar{x}_i^l(k) - \alpha \left. \frac{\delta e}{\delta \bar{x}_i^l} \right|_k \quad (9.5)$$

donde de nuevo se tiene $l=1, 2, \dots, M$, $k=1, 2, \dots$. Como se puede observar en la ecuación 7.46 (Figura 9.1), f depende de $\bar{x}_i^l$ sólo por el término z^l . Haciendo uso de la regla de la cadena se obtiene

$$\bar{x}_i^l(k+1) = \bar{x}_i^l(k) - \alpha \frac{f - t}{b} \left(\bar{y}^l - f \right) z^l \frac{2 \left(x_i^p - \bar{x}_i^l(k) \right)^2}{\sigma_i^{l2}(k)} \quad (9.6)$$

De manera similar para el parámetro σ_i^l , que representa la anchura de la gaussiana

$$\sigma_i^l(k+1) = \sigma_i^l(k) - \alpha \frac{f - t}{b} \left(\bar{y}^l - f \right) z^l \frac{2 \left(x_i^p - \bar{x}_i^l(k) \right)^2}{\sigma_i^{l3}(k)} \quad (9.7)$$

Debe tenerse en cuenta que, en realidad, se ha empleado en la notación un único ritmo de aprendizaje α , pero muy bien pudiera tenerse un ritmo diferente para la actualización de cada parámetro.

Como en el caso de las redes neuronales, el entrenamiento se realiza en dos pasos, una fase hacia adelante y otra hacia atrás (retropropagación). Para cada entrada $\mathbf{x}^p$, se propaga su valor hacia adelante (en este caso en el FLC en vez de en la red), obteniéndose las salidas z^l ($l=1,2,\dots,M$) y los valores a y b definidos. Después se calcula hacia atrás la actualización de los valores de $\bar{y}^l$, $\bar{x}_i^l$ y σ_i^l , con $i=1,2,\dots,n$, y $l=1,2,\dots,M$. Esto se repite para cada uno de los pares $(\mathbf{x}^p, t^p)$, correspondientes a cada patrón de aprendizaje $p=1, 2, \dots$, hasta que se obtiene el error deseado.

La equivalencia establecida entre ciertos modelos neuronales y borrosos [Wang 94, Jang 92-95], como la que acabamos de estudiar, se puede traducir de diversas maneras. Por una parte, de esta forma los sistemas borrosos aprovechan la capacidad de aprendizaje de la red neuronal para optimizar su funcionamiento. Por otro lado, este esquema podría ser empleado para **extraer las reglas** que una red neuronal (por ejemplo, tipo BP) ha encontrado en el entrenamiento, eliminando uno de los grandes problemas de los ANS: su operación en forma de caja negra. La combinación de redes neuronales y sistemas borrosos es actualmente un campo de intenso trabajo [Cox 93a, Furuhashi 93, Horikawa 90, Jang 92-95, Khan 93b, Shann 93, Sztandera 93, Taecheon 93, Wang 94]. En concreto, en las referencias recientes [Jang 92-97, Benítez 97, Reyneri 99, Figueiredo 99, Li 00] se hace uso de la equivalencia existente entre algunos modelos de redes neuronales y FLC para extraer las reglas encontradas por la red en su entrenamiento.

9.3 ALGORITMOS GENÉTICOS

Los **algoritmos genéticos o GA** (*Genetic Algorithms*) están inspirados en los métodos de solución de problemas que ha utilizado la naturaleza para hacer evolucionar los seres vivos, adaptándolos a los diferentes entornos o habitats. La técnica empleada por la naturaleza consiste principalmente en la codificación de las características de los seres vivos en el genoma, y su evolución a través de la reproducción sexual y las mutaciones. Esta idea básica, que permite que los seres

vivos se adapten al entorno en el que viven (y que no es otra cosa que su optimización para unas condiciones de contorno dadas), anima el uso de algoritmos basados en estas técnicas como metodología general de optimización de sistemas. Los primeros trabajos en esta área son de [Hollstien 71], aunque hay que esperar un tiempo hasta que se proponga en la literatura una codificación basada en una cadena de bits [Holland 75].

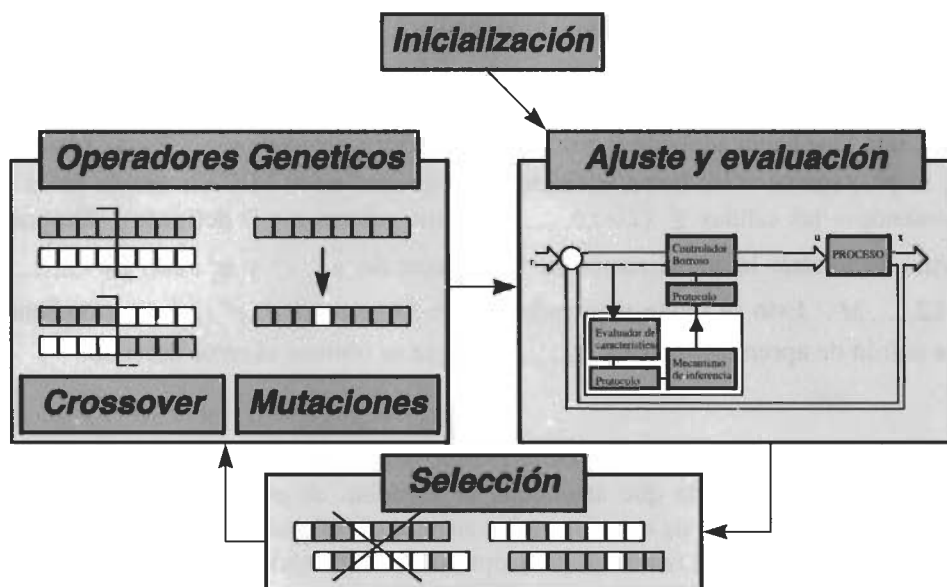


Figura 9.2. Estructura genérica y funcionamiento de un algoritmo genético

La estructura del algoritmo puede verse en la Figura 9.2. El proceso de optimización comienza con la **iniciación**, consistente en la generación de una población de individuos. Hay dos interpretaciones para el concepto de **individuo y población** en la optimización de controladores. La primera, habitualmente llamada **Pitts** [Goldberg 89], que es la más utilizada, y que nosotros seguiremos en adelante, entiende por **población** un conjunto de individuos; en nuestro caso, cada individuo será un controlador completo. En este marco, se realizan las siguientes definiciones:

- **Genoma:** Todos los parámetros que definen a todos y cada uno de los individuos de la población.
- **Genotipo:** La parte del genoma que describe a un individuo concreto. En esta interpretación son los genes que describen un controlador concreto.
- **Fenotipo:** La expresión de un genotipo. En esta interpretación es un controlador concreto.

- **Gen:** Cada uno de los parámetros que describen a un individuo. En esta interpretación los genes codifican parámetros de un controlador concreto, que en el caso de un FLC pueden ser, por ejemplo, la ganancia del escalado, la forma o el tipo de función de pertenencia de una entrada, una salida, regla, etc. Una descripción más detallada de este asunto se verá más adelante.

La segunda interpretación, habitualmente llamada **Michigan** [Goldberg 89], consiste en considerar como individuo las habilidades de un controlador, de forma que la población representa el conjunto de todas sus habilidades. Según esta interpretación, el genoma es la codificación de estas habilidades; en el caso de un FLC, una habilidad puede estar formada por un conjunto de reglas que definen su comportamiento ante cierta situación. Este tipo de interpretación se ha utilizado en el control de sistemas en tiempo real y en robótica. Por ejemplo, en el caso de la robótica, un individuo está formado por el conjunto de reglas que permiten al robot agarrar un objeto; la población está formada por individuos especializados en ciertas tareas, y durante la evolución se seleccionan los mejores para cada tarea.

No obstante, nosotros seguiremos la primera de las interpretaciones (cada individuo es un controlador concreto). Durante la fase de inicialización se puede generar una población, bien de forma aleatoria, bien partiendo de un controlador ya existente. Esto último permite partir de soluciones desarrolladas por otros procedimientos, tanto automáticos como manuales. El caso más frecuente, no obstante, consiste en partir de una población aleatoria pues, como se ha explicado antes, los GA son especialmente indicados para realizar la búsqueda global de soluciones óptimas en un gran espacio.

La fase siguiente del proceso es la **evaluación**, en la cual se deja que cada uno de los controladores que forman la población actúe controlando el sistema, normalmente mediante una simulación, siendo evaluados mediante una función de eficiencia o idoneidad (*fitness*). Ésta es normalmente la etapa que más tiempo requiere, pues se han de simular cada uno de los controladores de la población durante el tiempo necesario para evaluar su eficiencia, y en un número de situaciones de control suficiente. La definición de esta función de eficiencia es fundamental en el éxito del uso de los GA para la resolución de un problema dado. A partir de ella, el diseñador decide qué comportamientos no se han de potenciar, cuáles sí, y en qué medida. Una descripción más detallada del diseño de esta función se verá más adelante.

La fase siguiente es la de **selección**, en la cual se simula el proceso de selección natural de los individuos en cada generación. En este sentido, se deben seleccionar qué individuos han de transmitir su genotipo a la generación siguiente. Dada una población de M individuos, son posibles diversas alternativas:

- a) **Sólo el mejor se selecciona.** De toda la población se elige al individuo que tiene una mejor evaluación, y solamente es empleado su genoma para crear la siguiente generación.

- b) **Sólo los mejores se seleccionan.** De toda la población se eligen los n individuos ($n \ll M$) que tienen una mejor evaluación, y para crear la siguiente generación solamente se utiliza su genoma.
- c) **Todos pueden seleccionarse.** Cada individuo de la población se selecciona con un probabilidad mayor cuanto mejor sea su evaluación.

La primera de las alternativas permite una rápida convergencia a la solución, pero puede estancarse en un mínimo local. Ello resulta especialmente probable en aquellos problemas en los que se debe alcanzar una solución de compromiso entre varios objetivos contrapuestos, ya que si se selecciona sólo el mejor el genoma se empobrece (en cuanto a variedad), y los individuos pueden tender a especializarse en conseguir parte de los objetivos, dando malos resultados en el resto. Este problema puede resolverse con el uso de la segunda alternativa, y una adecuada elección de la relación n/M . La última alternativa es adecuada para buscar soluciones a problemas con un gran espacio de búsqueda, y para los que no se conoce una buena aproximación a la solución; cuenta con el inconveniente de requerir más tiempo de cálculo y poblaciones mayores, por lo que en algunos casos resulta inaplicable.

A la fase de selección le sigue la aplicación de una secuencia de **operadores genéticos**, que simulan el proceso de reproducción sexual de los seres vivos. Aplicando estos operadores sobre el genoma que se ha seleccionado, se obtienen un genoma final para la siguiente generación. Esta fase queda, pues, definida por la secuencia de operadores seleccionada, la cual sirve además para identificar el algoritmo genético utilizado. Una descripción completa de estos operadores se verá en próximas secciones.

Con el genoma final, correspondiente a las siguientes generaciones, se expresa un fenotipo, reconstruyendo cada uno de los controladores que forman la población y **procediendo nuevamente a su evaluación**. Este proceso se repite un número fijo de veces, o bien hasta que la evaluación se estabiliza. Una descripción de estos métodos puede verse en [Herrera 96, Holland 92].

9.3.1 ¿Qué optimizar?

Un aspecto clave en el empleo de GA en la optimización de un controlador es decidir qué se ha de optimizar y cómo codificarlo en forma de genoma. Respecto a qué se ha de optimizar, caben cuatro opciones básicas:

- a) Optimización de coeficientes.
 - Ganancias de entrada y salida.
 - Partición de las variables de entrada.
 - Partición de las variables de salida.
- b) Optimización de reglas.

c) Optimización de la estructura de la base de reglas.

d) Optimización completa.

- Codificación de desordenada.

El primer caso (a), **optimización de coeficientes**, consiste en seleccionar de un FLC inicial un conjunto limitado de **parámetros escalares**, y limitar la optimización a encontrar la mejor combinación de estos valores, pero sin modificar la base de reglas. Para realizar una buena selección de los parámetros hay que comprender la influencia de cada uno en el comportamiento del FLC. Un caso particular es la **modificación de las ganancias** de entrada y salida de un controlador, las cuales afectan a su funcionamiento en todo el espacio de entradas, modificando de forma simultánea el significado de todas las particiones afectadas; esta modificación del funcionamiento es similar a la experimentada por los controladores clásicos al modificar las ganancias (un estudio de ese efecto puede verse en [Santos 96]).

Por otra parte, el efecto de **modificar las particiones** de las variables de entrada o de salida es más complejo, y se debe considerar su funcionamiento. Para un FLC con n entradas, $\mathbf{x}=\{x_i/1\leq i\leq n\}$, podemos considerar que para el espacio de las entrada de dimensión $\mathbb{R}^n$, cada valor lingüístico o conjunto borroso representa un subespacio del espacio n -dimensional. De esta forma, el espacio de las entradas queda dividido en tantos subespacios borrosos como valores lingüísticos se definan en la base de reglas. Las proposiciones de la base de reglas definen finalmente una aplicación entre los subespacios borrosos y las salidas; de esta manera se puede obtener la aproximación funcional requerida en base a una adecuada definición de los valores lingüísticos que parcelan el espacio de las entradas.

Cada conjunto borroso de una partición de entrada afecta al comportamiento en una región amplia. Para el caso típico de un FLC con dos entradas, x_1 y x_2 , y una salida y , el modificar una función de pertenencia de las definidas para la variable x_1 afecta al comportamiento definido por las reglas de toda una columna (Figura 9.3). De la misma manera, el modificar una función de pertenencia de las definidas para la variable x_2 afecta al comportamiento definido por las reglas de una fila.

Por otro lado, se ha de convenir también qué es lo que se **codifica de los conjuntos borrosos**. Se puede asumir, por ejemplo, que para cada variable hay un número fijo de conjuntos borrosos definidos, y que todos son de tipo triangular salvo el primero, que es de tipo Z, y el último, de tipo S. En este supuesto puede ser suficiente con codificar para su ajuste la posición de los máximos y utilizar una anchura fija. Es posible también codificar la anchura de cada uno, o incluso el tipo.

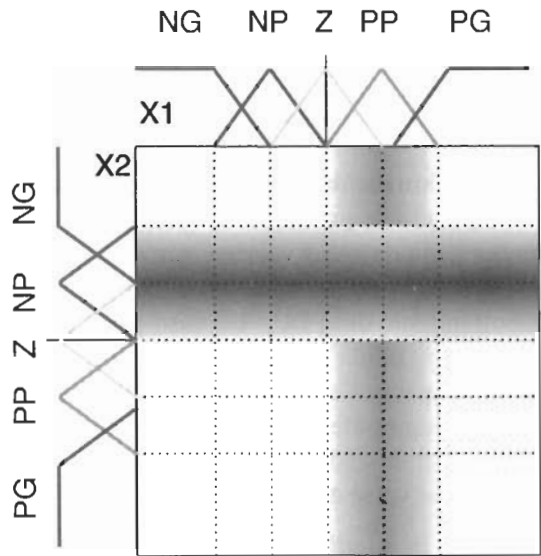


Figura 9.3. Influencia de un cambio en las particiones de entrada en la FAM. El cambio en X1 afecta a toda una columna, y en X2 a una fila

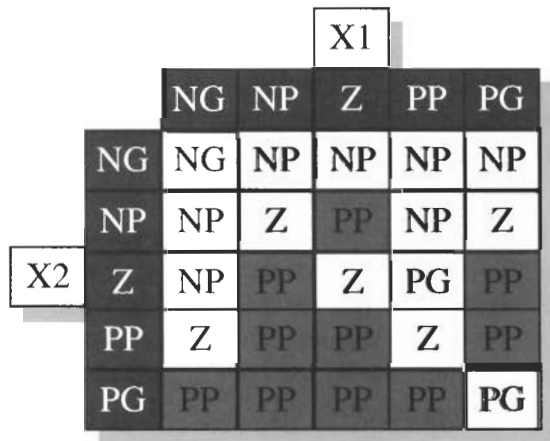


Figura 9.4. Influencia de un cambio en las particiones de salida de la FAM. Ejemplo de cambio en el conjunto borroso de salida PP

Como puede verse en la Figura 9.4, el cambiar uno de los conjuntos borrosos de salida afecta al comportamiento del controlador en las zonas afectadas por reglas que lo incluyen en la consecuencia. Con frecuencia estas regiones coinciden con las que controlan un cierto comportamiento global, por lo que al actuar sobre ese conjunto se actúa globalmente sobre el comportamiento.

X1

	NG	NP	Z	PF	PG
NG	NG	NP	NP	NP	NP
NP	NP	Z	PP		Z
Z	NP	PP		PG	PP
PP	Z	PP	PP	Z	PP
PG	PP	PP	PP	PP	PG

X2

•R1: if X1 is Z and X2 is Z then Y is Z

•R2: if X1 is PP and X2 is NP then Y is NP

Figura 9.5. Influencia de las reglas en la FAM

La segunda alternativa importante (b) para la optimización de un FLC es la **optimización de reglas**. En este caso, la zona de influencia de estos cambios es menor que en el de las modificaciones anteriores (Figura 9.5), ya que supone modificar las premisas de las reglas que forman la FAM sin cambiar la definición de las particiones, lo que permite ajustar el comportamiento del FLC en cada una de las regiones de la FAM controladas.

La tercera alternativa (c) es optimizar el comportamiento del controlador **modificando la estructura de la base de reglas**. Ello supone permitir la realización de simplificaciones en la premisa, eliminando entradas no significativas, lo cual es de especial interés en el caso de FLC con muchas entradas, en los que no es posible definir todas las reglas posibles. En estos casos se comienza definiendo reglas genéricas, con sólo dos o tres variables de entrada, a las cuales se añaden después más reglas en zonas más reducidas, en las que se necesite un comportamiento mejor. Dentro de estas técnicas de optimización se incluyen también los mínimos cuadrados, tablas de búsqueda y agrupamiento por vecino más próximo, ya mencionadas.

Finalmente, la cuarta alternativa (d) nos lleva a plantear **una optimización simultánea** de todos estos aspectos. Para que este planteamiento sea viable resulta de especial significación el tipo de codificación empleada, tema que abordaremos en la siguiente sección.

9.3.2 Codificación

El método más utilizado para la **codificación de los genes** es el uso de una cadena de longitud y secuencia fijas. Los parámetros del controlador que se ha decidido ajustar, según los criterios que se han expuesto antes, se almacenan en esta

cadena como secuencias fijas de números. De esta manera, el número de genes (en nuestro caso parámetros) resulta fijo, y su valor se guarda siempre en la misma posición de la cadena. Por ejemplo, como se puede observar en la Figura 9.6, una partición mediante cinco conjuntos borrosos de anchura fija puede codificarse como una secuencia de cinco números que indican la posición de cada conjunto.

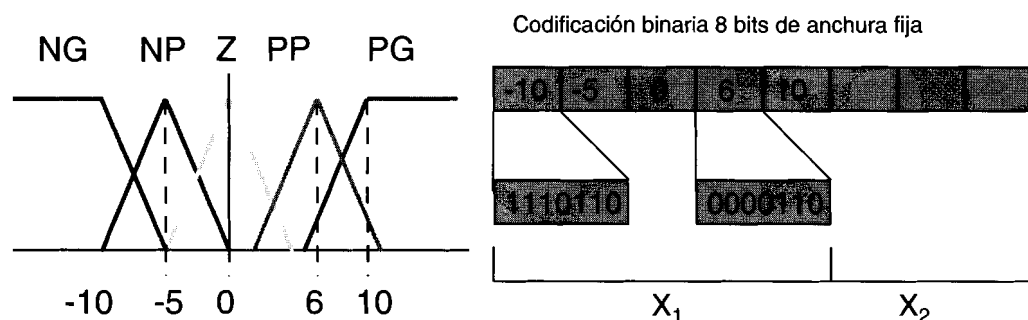


Figura 9.6. Codificación fija de los conjuntos borrosos. En este caso, el valor en la cadena indica la posición de cada conjunto borroso, codificada en 8 bits

Por otro lado, también se ha de decidir cómo codificar estos números; son habituales codificaciones binarias de 8 bits (como en el ejemplo) y de 16, aunque también puede hacerse uso de una codificación en coma fija o flotante. Asimismo, puede emplearse una codificación basada en reservar cierto número de bits para cada gen, y codificar su valor mediante el número de unos presentes, con independencia de su posición. El tipo de codificación seleccionado para los genes influirá en cómo implementar los operadores (como el mutación), así como en la cantidad de memoria necesaria para almacenar el genoma.

Para la codificación de la estructura de la base de reglas se hace uso de una secuencia de índices a los conjuntos borrosos de la partición de salida. Para el ejemplo antes expuesto de un FLC con dos entradas x_1 y x_2 , y una salida y , la codificación de la FAM de uno de los controladores que forman la población puede observarse en la Figura 9.7. Cada conjunto borroso de salida tiene asignado un código (1, 2, 3...), y los códigos de salida correspondientes a cada fila de la FAM se colocan consecutivos en la cadena (1,2,2,2,2; 2,3,4,3,4; ...).

De esta manera, para codificar en el ejemplo la base de reglas de cada uno de los controladores de la población se necesita una cadena de 25 enteros, uno para cada regla de las posibles. Este número indica cuál de los conjuntos de la partición de salida se utiliza en la consecuencia de la regla, mientras que las premisas quedan definidas por la posición en la cadena. Esta codificación resulta simple para el caso de un FLC sencillo, con pocas entradas y una complejidad moderada en las particiones,

pero debe tenerse en cuenta que en el caso de un controlador con n entradas y p particiones hay un total de r reglas posibles, cantidad dada por la expresión

$$r = p^n \quad (9.8)$$

Así, se tiene que para 5 entradas y 7 particiones, el total de reglas será $r=16807$, mientras que para 10 entradas y 7 particiones el total de reglas ascenderá ya a 282.475.249, en cuyo caso el esquema no resulta viable. Para un controlador de este nivel de complejidad es preferible utilizar reglas que incluyan en su premisa sólo una parte de las entradas, lo que permitirá crear una base de reglas más manejable. En este caso resultará necesario utilizar una codificación para las reglas de tipo variable, en vez de la fija empleada hasta el momento; los GA adecuados para estos sistemas se conocen como **algoritmos genéticos desordenados**, que estudiaremos más adelante. Este tipo de algoritmos permiten codificar una base de regla de estructura no uniforme, y optimizarla de forma eficiente.

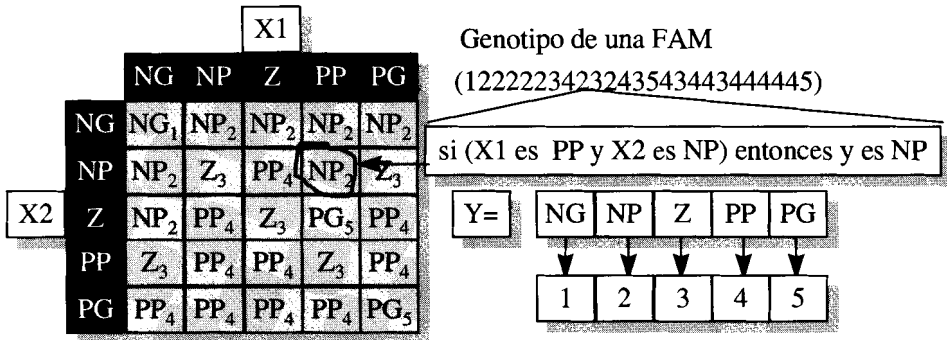


Figura 9.7. Codificación fija de una FAM como secuencia de valores de salida (por filas)

9.3.3 Operadores cruzamiento y mutación

Los **operadores** utilizados en los GA realizan modificaciones sobre el genoma, simulando la evolución del genoma de los seres vivos causada por la reproducción sexual y otros factores, como las mutaciones. Los principales operadores utilizados en los GA son el cruzamiento y la mutación.

El **cruzamiento** permite simular la combinación del genoma entre los individuos. Este puede realizarse sobre parejas de individuos o sobre toda la población. Cuando se realiza sobre parejas de individuos (padres), se copia el genoma

en dos memorias, de forma aleatoria se selecciona un punto en esta memoria, y se cruzan las dos mitades de cada copia de sus genotipos (véase la Figura 9.8).

Cuando se realiza sobre toda la población, el cruzamiento puede realizarse de forma simultánea sobre todo el genoma por el procedimiento conocido como ruleta rusa (véase la Figura 9.9): primero se copia el genoma en dos memorias circulares, a continuación uno de ellos gira un ángulo aleatorio, luego se selecciona en esta memoria un punto de forma también aleatoria y, finalmente, se cruzan las dos mitades de cada copia del genoma.

Ejecutado de una manera u otra, el cruzamiento permite que las mejores cualidades de los padres se combinen en sus descendientes. En problemas en los que se pretende que el controlador haya de alcanzar simultáneamente varios objetivos, se espera que los descendientes en algún momento de la evolución puedan heredar de cada uno de sus padres el conjunto de reglas que permiten lograr cada uno de los objetivos.

No obstante, se sabe que durante la reproducción de los seres vivos se producen errores en la copia del genoma, a los que asignaremos una cierta probabilidad que llamaremos p . Este efecto introduce diversidad en el genoma, y permite explorar nuevas soluciones a los problemas (véase la Figura 9.10). Ello se modela mediante el **operador mutación**, el cual supone seleccionar un punto de mutación y modificar el parámetro almacenado con una probabilidad p . Si la codificación es de tipo binaria, se modifica uno de los bits, mientras que si la codificación es de tipo como fija o flotante se suele realizar un cambio aleatorio dentro de un rango limitado.

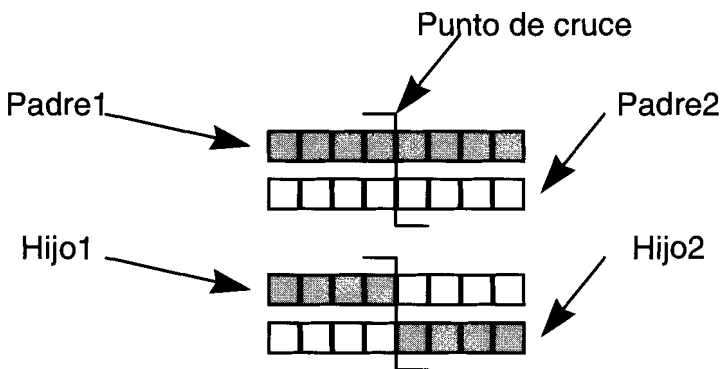


Figura 9.8. Cruzamiento del genoma entre individuos

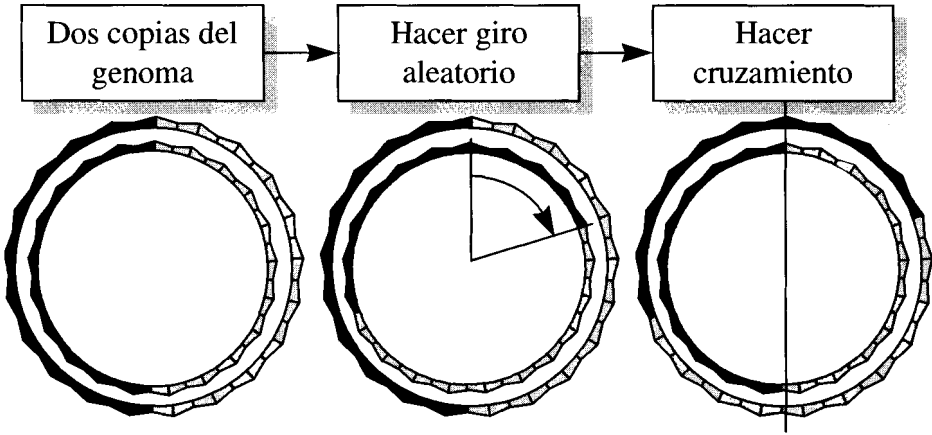


Figura 9.9. Las tres fases del cruzamiento del genoma de toda la población

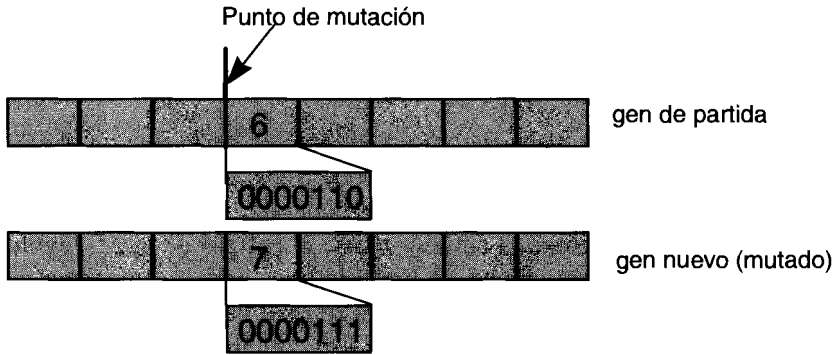


Figura 9.10. Operador mutación actuando sobre un punto del genoma

9.3.4 Diseño de la función de idoneidad

Para especificar un algoritmo genético es necesario definir una función de idoneidad y un método de codificación; con la **función de idoneidad (*fitness*)** se establece el objetivo que se desea alcanzar con el algoritmo (traducimos *fitness* por idoneidad; el sentido de función de idoneidad sería similar a función coste o de mérito, que hemos empleado ya anteriormente). Existen dos casos típicos, según se disponga o no de un conjunto de pares entrada-salida de referencia (o patrones, empleando la ya familiar jerga neuronal) que deba aproximar el controlador. Si se dispone de este fichero con pares (x_i, y_i) de referencia, la función de idoneidad se puede calcular

restando a un máximo de la idoneidad F_{max} , la suma de las distancias euclidianas entre las salidas deseadas y_i (u objetivos, equivalentes a los t_i ya utilizados antes) y las que proporciona el controlador y . El objetivo es, pues, maximizar la cantidad

$$F(y, y_i) = F_{max} - \sum_i (y_i - y(x_i))^2 \quad (9.9)$$

Este tipo de optimización, como muy bien sabemos ya, se conoce como aprendizaje supervisado, puesto que se conocen las acciones-objetivos que el controlador ha de tomar. Si se puede obtener la derivada de la función de idoneidad frente a los parámetros a optimizar es preferible un método basado en descenso por el gradiente, como los de tipo BP, que pueden resultar más rápidos en este caso.

En general, **los algoritmos genéticos son más útiles en problemas con gran número de variables a optimizar, y en los que no se conocen las salidas concretas deseables para cada entrada**, por lo que la evaluación del controlador se ha de realizar por los resultados de sus acciones sobre el sistema bajo control. Un caso típico sería el del control para el mantenimiento de un punto de funcionamiento, cuyo objetivo es alcanzar un determinado punto de funcionamiento en un tiempo T_s , con una sobreoscilación S_o , y una amplitud de oscilación residual A_m , mínima

$$F = F_{max} - C_T * T_s - C_S * S_o - C_A * A_m \quad (9.10)$$

El anterior representa un claro ejemplo de definición de la función de idoneidad con el fin de optimizar simultáneamente varios objetivos, aunque encontrar un buen compromiso entre todos ellos resultará difícil, en general. Así, si se hace uso de una función de idoneidad estática como la anterior existe el peligro de que se evolucione a un sistema muy optimizado para alguno de los objetivos, pero malo para otros. Para evitarlo, puede utilizarse una técnica basada en el reparto de refuerzos; además, suele resultar conveniente evaluar la idoneidad de cada controlador C_K en diversos entornos E_i . Así, la idoneidad total de un controlador C_K sería en este caso la suma de los refuerzos obtenidos en cada uno de los entornos

$$F(C_K) = \sum_j \frac{\sum_i R_j(C_K, E_i)}{R_j} \quad (9.11)$$

siendo R_j el refuerzo asignado al objetivo j para toda la población, por lo que debe cumplirse

$$R_j = \sum_K \sum_i R_j(C_K, E_i) \quad (9.12)$$

constante que será fijada al comienzo del algoritmo con el fin de ponderar globalmente la importancia de cada uno de los objetivos de optimización. Con este procedimiento el refuerzo que corresponde a cada objetivo se reparte inicialmente sólo

entre los pocos individuos que se especializan en él, obteniendo buenos resultados en casi todos los entornos, aunque fallen en otros concretos. A medida que la media de los individuos mejora en ese objetivo disminuye el refuerzo de los especialistas.

Es importante destacar que la elección de los entornos de test E_i resulta fundamental para asegurar un funcionamiento adecuado del controlador en todos los ambientes. Si es necesario, y a medida que la población de controladores mejora su idoneidad sobre los entornos utilizados, puede hacerse uso de un segundo algoritmo genético que se ocupe de evolucionar los entornos de test hacia entornos más difíciles para los controladores. La idoneidad de un entorno E_i se puede calcular como la inversa del refuerzo de los controladores para cada uno de los objetivos j

$$F(E_i) = \frac{1}{\sum_j \sum_K R_j(C_K, E_i)} \quad (9.13)$$

Una descripción detallada de estas técnicas aplicadas al control de robots puede verse en la referencia [Herrera 96].

9.4 ALGORITMOS GENÉTICOS DESORDENADOS

Mientras que los algoritmos genéticos clásicos (los estudiados hasta el momento) codifican el genoma en una cadena de longitud fija, los **algoritmos genéticos desordenados** [Goldberg 89] utilizan un genoma de longitud variable en el que los genes pueden colocarse en cualquier posición. Los GA desordenados permiten evolucionar sistemas complejos de forma más eficiente al independizar el significado de los genes de su posición, e introducir operadores capaces de realizar simplificaciones en la base de reglas, o bien añadirlas si es preciso.

En general, las características que permiten resolver un problema de forma óptima suelen ser debidas a más de un gen. Un determinado grupo de genes conforma lo que se denomina un **esquema**, que no es otra cosa que una sección de la cadena de bits del genoma, caracterizado por una secuencia de tres símbolos 0, 1, *. El símbolo * indica que el valor concreto de los bits rotulados con él no influyen en este esquema; por ejemplo, el esquema 0**1 corresponde a cualquier secuencia de cuatro bits en los que el primero es el carácter 0 y el último 1, sin que importe el valor concreto de los bits segundo y tercero.

La **idoneidad de un esquema** se calculará como promedio de las cadenas que especifica. Se puede demostrar que el número m de esquemas h en la siguiente generación $t+1$ viene dado por la expresión [Goldberg 89]

$$m(h, t+1) \geq m(h, t) \frac{f(h)}{\bar{f}} \left[1 - p_c \frac{\delta(h)}{l-1} \right] \quad (9.14)$$

donde $f(h)$ es la idoneidad media de un esquema h , y $\bar{f}$ es la idoneidad media de la población; la longitud total de la población es l , y $\delta(h)$ es la longitud del esquema h , calculada como la distancia de los bits extremos distintos de *. Por último, p_c representa la probabilidad de supervivencia tras el cruzamiento. Según esta expresión, el número de esquemas con mejor idoneidad que la media y con una longitud reducida aumentará en la siguiente generación (para un estudio más detallado recomendamos al lector la consulta de la referencia [Goldberg 89]).

Ésta es la clave de por qué los algoritmos genéticos desordenados permiten una evolución más eficiente en el caso de sistemas complejos. En un sistema complejo las características que se han de optimizar dependen de varios parámetros, que en ocasiones pueden situarse bastante separados en la cadena, lo que hace que los esquemas genéticos que codifican esas características posean una gran longitud en el caso de los algoritmos genéticos clásicos. Esa gran longitud hace que no aumenten en cada generación, con lo que la optimización no tiene éxito. Por otro lado, en el caso de los algoritmos genéticos desordenados puede reducirse notablemente la longitud de estos esquemas al permitir que los parámetros de una misma característica se codifiquen juntos. Si, por ejemplo, los parámetros 1 y 5 con valor 1 tienen una alta idoneidad, este esquema en la codificación clásica se indicaría con $1***1$, y su $\delta(h)$ sería de 5. En la codificación de los algoritmos genéticos desordenados los parámetros en un momento dado pueden aparecer en un orden distinto, por ejemplo $(5,1)(1,1)(3,0)(2,1)(4,0)$, lo que estaría dentro de un esquema $11***$, cuyo $\delta(h)$ vale 2, inferior al anterior, por lo que la probabilidad de aumentar en la siguiente generación crece, y la idoneidad de la población mejora.

9.4.1 Codificación

En los algoritmos genéticos desordenados cada gen se codifica con un par de números enteros: el primero codifica su significado y el segundo el valor. Este par de valores se suelen denominar **cláusula**. (véase la Figura 9.11). A modo de ejemplo, en el caso antes indicado de un FLC con dos entradas x_1 y x_2 , y una salida y , en la codificación de los conjuntos borrosos empleados el primer término de la cláusula indica a qué variable de entrada o de salida se refiere, mientras que el segundo término establece el conjunto borroso de la partición correspondiente.

Las reglas de un FLC se codifican como un conjunto desordenado de pares de enteros, cada uno de los cuales codifica una cláusula. En la Figura 9.12 aparecen diversos ejemplos de reglas; en ella puede verse que es posible codificar las reglas con independencia de la posición de las cláusulas, así, la cadena $(1,2)(2,1)(3,3)$ significa lo mismo que $(2,1)(3,3)(1,2)$. Además, este método de codificación permite expresar reglas que no incluyan todos los términos, como es el caso de la $(1,2)(3,3)$. Esta última característica, conocida como **subespecificación**, resulta especialmente útil en el caso de los FLC complejos, pues veremos que permite reducir el tamaño de la base de reglas de una forma importante. Como situación extrema de subespecificación,

puede aparecer una regla sin cláusulas de salida, en cuyo caso se añade una de forma aleatoria. Por otra parte, si aparece una regla sin cláusulas de entrada, la premisa sería siempre cierta; para evitarlo, se añade también una cláusula de forma aleatoria.

Como resultado de la aplicación de los operadores genéticos al genoma puede suceder que aparezca más de una cláusula en una misma regla refiriéndose a la misma variable lingüística. Esta circunstancia, conocida como **sobreespecificación**, hace necesario definir un método para interpretar estas reglas que resuelva la situación. Como se puede observar en la Figura 9.13, es posible emplear dos reglas de interpretación. La más frecuente es la conocida como *la primera es la que vale*, según la cual si varias cláusulas de una regla se refieren a la misma variable, sólo la primera se tiene en consideración. Otra regla de interpretación es el *or borroso*, es decir, si varias cláusulas de una regla se refieren a la misma variable se unen en la premisa con una conjuntiva *or*.

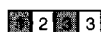
• $X1=(\text{negativo, zero, positivo})$	$X1= \text{zero}$
	
• $X2=(\text{pequeño, medio, grande})$	$X2= \text{grande}$
	
• $Y=(\text{izquierda, centro, derecha})$	$Y= \text{izquierda}$
	

Figura 9.11. Cláusulas (tipo de variable, conjunto borroso) para la codificación desordenada de los conjuntos borrosos

■ $\text{regla}_i = \{\text{clausula}_{x1}, \text{clausula}_{x2}, \text{clausula}_y\}$

 if $x1=\text{zero}$ and $x2=\text{pequeño}$ then $y=\text{derecha}$

 if $x1=\text{zero}$ and $x2=\text{pequeño}$ then $y=\text{derecha}$

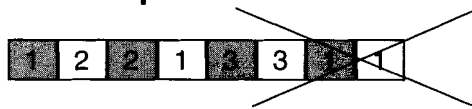
 if $x1=\text{zero}$ then $y=\text{derecha}$

■ Base de reglas = $\{\text{regla}_1, \text{regla}_2, \dots, \text{regla}_n\}$



Figura 9.12. Codificación desordenada de las reglas

■ La primera es la que vale



if $x_1 = \text{zero}$ and $x_2 = \text{pequeño}$ then $y = \text{derecha}$

■ OR borroso



if $x_1 = \text{zero}$ or $x_1 = \text{negativo}$ and $x_2 = \text{pequeño}$ then $y = \text{derecha}$

Figura 9.13. Ilustración de las dos reglas de resolución de una sobreespecificación

A modo de resumen de lo dicho hasta el momento, la codificación de la base de reglas en el caso de los GA desordenados conforma un conjunto desordenado de reglas, cada una de las cuales es codificada por una serie de cláusulas. La subespecificación que puede surgir de este esquema es muy importante para los FLC complejos, ya que es la característica que permite sustituir varias reglas completas por una más simple. Por ejemplo, la regla (1,1)(3,2), puede sustituir a las siguientes tres reglas (1,1)(2,1)(3,2), (1,1)(2,2)(3,2) y (1,1)(2,3)(3,2). En algunos casos puede resultar necesario definir acciones particulares en una zona afectada por una regla simple; para ello puede utilizarse una estructura de prioridad jerárquica, según la cual las reglas generales se inhiben cuando hay reglas más específicas definidas. También es posible asignar a cada regla un nivel de prioridad independiente de su nivel jerárquico.

Como en el caso de las reglas individuales, también en la base de reglas puede darse subespecificación y sobreespecificación. Se dice que ocurre **sobreespecificación en la base de reglas** cuando aparece más de una regla con la misma premisa, pero distinta consecuencia. En este caso, o bien se toma como salida final el promedio, o bien se puede aplicar la técnica *la primera es la que vale*, y no considerar el resto de las reglas. La **subespecificación** se produce en este caso cuando no hay reglas definidas para alguna región del espacio de entradas. Como resultado de la aplicación de los operadores genéticos al genoma no puede asegurarse una cobertura completa del espacio de entradas, por lo que se hace necesario definir una acción por defecto, o bien repetir la última acción calculada.

9.4.2 Operadores *corta y empalma*

En los algoritmos genéticos desordenados el operador cruzamiento se sustituye por el **operador corta y empalma**, mientras que el **operador mutación** se mantiene. El operador corta y empalma (Figura 9.14) se realiza tomando aleatoriamente dos puntos de corte en el genotipo de los padres, cuyas posiciones son independientes entre sí. El empalme se realiza concatenando las dos mitades de padres distintos resultantes (1-4, 1-3 y 3-2 de la Figura 9.14), aunque en este caso es posible concatenar también mitades del mismo padre (lo que representa un cambio de orden), como sucede en la cadena 4-3 del ejemplo.

El operador mutación en el caso de los algoritmos genéticos desordenados no cambia respecto del comentado en los clásicos (recordemos que éste consiste en intercambiar 0 y 1 con una pequeña probabilidad). La salvedad es que la mutación de las cláusulas se aplica solamente al segundo miembro, el que codifica el valor, y no al primero, que indica el significado. También es posible aplicar la mutación a las reglas, con una probabilidad muy baja, eliminando reglas o añadiendo otras completamente aleatorias a la base de reglas.

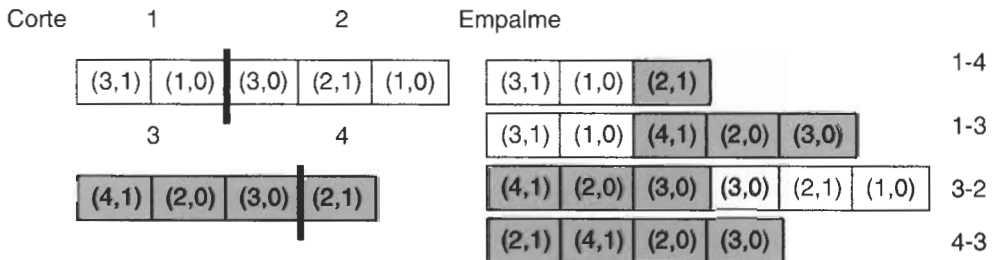


Figura 9.14. Ejemplos de aplicación del operador corta y empalma

Con esta exposición de las bases de los algoritmos genéticos damos por concluido el capítulo dedicado al aprendizaje, entrenamiento u optimización de controladores borrosos. Destacaremos que los algoritmos genéticos se están empleando cada vez más en la optimización de FLC (unos cuantos ejemplos pueden verse en [Fukuda 93, Herrera 93a, 96, Daihee 94, Lin 96, Jang 97]), y que desde hace algún tiempo se vienen empleando técnicas basadas en redes neuronales. En la última línea señalada, en las referencias [Cox 93a, Furuhashi 93, Horikawa 90, Jang 92-95, Khan 93b, Shann 93, Sztandera 93, Taecheon 93, Wang 94] se exponen diversos **sistemas mixtos neuronales-borrosos**, que tratan de combinar la capacidad de aprendizaje de las redes neuronales, con la habilidad en el manejo de incertezas y la expresión del conocimiento en forma de reglas de los sistemas borrosos. Además, en el recomendable libro de Wang [Wang 94] se muestra el empleo de técnicas estadísticas más convencionales (*clustering*, mínimos cuadrados, etc.) en el entrenamiento de controladores borrosos. Finalmente, en los libros [Jang 97, Lin 96] se expone una visión unitaria y completa de los sistemas neuro-borrosos.

Concluiremos subrayando una vez más el siguiente hecho: en general, no se puede afirmar que una técnica sea ni mejor ni peor que otra, simplemente cada problema o aspecto de un problema requiere una solución diferente, y cada técnica está más indicada para ciertos tipos de problemas que para otros. Por ejemplo, cuando se dispone de un buen punto de partida el empleo de BP puede ser interesante, pero si no es así, existe una elevada probabilidad de que el BP quede estancado en un mínimo local, en cuyo caso un algoritmo genético podría ser una técnica más apropiada.

CAPÍTULO 10

IMPLEMENTACIÓN DE SISTEMAS BORROSOS

Tras exponer en los capítulos anteriores las bases de los sistemas borrosos, controladores borrosos y sistemas borrosos adaptativos, en el presente expondremos las diversas maneras disponibles para realizar en la práctica un sistema basado en lógica borrosa. Como en el caso de las redes neuronales, un sistema borroso podrá implementarse como programa ejecutable por un microprocesador convencional (o microcontrolador), o podrá realizarse en hardware específico. La gran diferencia con las redes neuronales es que un sistema borroso precisa en general de recursos de cálculo relativamente reducidos, por lo que muchas veces podrá implementarse como programa ejecutado en un sencillo (y barato) microcontrolador de 8 bits.

En este capítulo comenzaremos por considerar los entornos de desarrollo de sistemas borrosos, mostrando el por qué son necesarios, así como algunos de los actualmente disponibles, que serán descritos con cierto detenimiento. Veremos después diversos métodos de realización hardware de sistemas borrosos, incluyendo los denominados aceleradores de procesamiento.

10.1 INTRODUCCIÓN

El diseño de un controlador basado en lógica borrosa supone establecer un compromiso entre diversos criterios de diseño: velocidad, precisión y flexibilidad, principalmente (Figura 10.1). Para conseguir los resultados deseados debe plantearse la velocidad de respuesta del sistema de control, la cual vendrá limitada por otros factores, como el grado de precisión requerido o la flexibilidad del diseño. Así, si deseamos una alta precisión en el control necesitaremos una gran cantidad de conjuntos para cada variable y un alto número de reglas, lo que exigirá una elevada cantidad de cálculos, causando aumento del tiempo de respuesta.

Si además deseamos que el sistema de control tenga flexibilidad de adaptación a los caminos del sistema y aprender de los errores cometidos, serán necesarios muchos más cálculos adicionales, que también aumentarán el tiempo de respuesta. Evidentemente, estas opciones influyen en el coste del sistema final, y en muchos casos será ésta la mayor restricción del diseño.

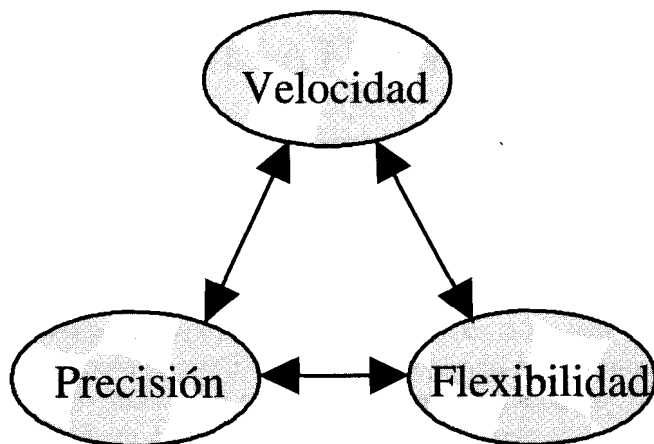


Figura 10.1. Compromiso de diseño. Falta considerar otro fundamental, que suponemos implícito: el coste

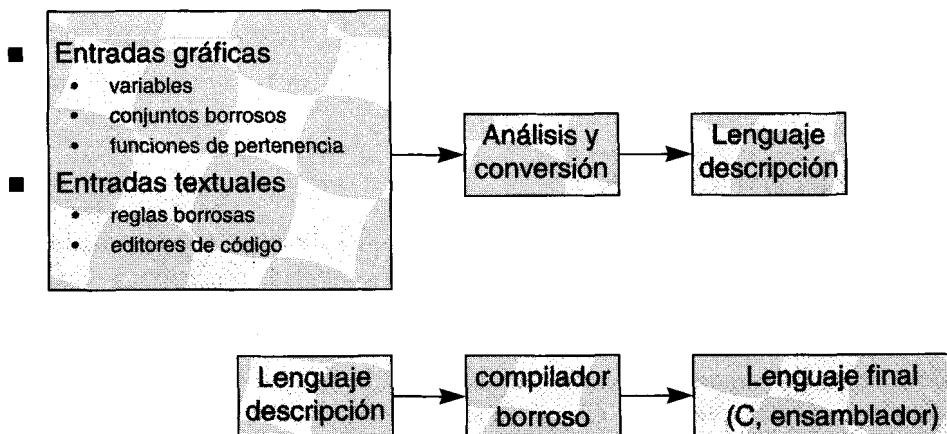


Figura 10.2. Proceso de diseño de un controlador borroso

Una vez decididas las prestaciones requeridas, se ha de utilizar una herramienta de desarrollo para el diseño del sistema y seleccionar la plataforma de implementación adecuada. Las herramientas de desarrollo suelen utilizar un lenguaje de descripción para independizar el diseño de la plataforma, como puede observarse en el esquema del proceso de diseño de un FLC mostrado en la Figura 10.2. En este sentido, los sistemas de desarrollo disponibles comercialmente más utilizados son FuzzyTECH, MATLAB, TILShell y FIDE. De éstas y de otras herramientas software hablaremos en las secciones siguientes, mientras que en la última sección de este capítulo expondremos algunas formas de realizar en hardware sistemas borrosos.

10.2 ENTORNOS DE DESARROLLO

En esta sección comenzaremos con el relato de la evolución de los sistemas electrónicos e informáticos hacia la complejidad que actualmente presentan, para mostrar cómo se ha hecho cada vez más necesario disponer de **sistemas integrados para el desarrollo de aplicaciones**. Posteriormente describiremos los distintos entornos de desarrollo disponibles para sistemas borrosos.

Evolución de los sistemas electrónicos e informáticos

El rápido aumento de la potencia de cálculo y procesamiento de las máquinas actuales, así como el de su capacidad de almacenamiento de información, ha permitido afrontar problemas paulatinamente más complejos. Tradicionalmente el desarrollo de aplicaciones ha tenido dos límites, la capacidad de la máquina y la capacidad del hombre que la usa. Inicialmente el límite estuvo impuesto fundamentalmente por la capacidad de las máquinas disponibles, pues las restricciones de memoria y almacenamiento condicionaban el tamaño de las aplicaciones. Debido a ello, con las computadoras de los cincuenta y parte de los sesenta resultaba imposible plantear sistemas de interfaz gráfica. En esta época la memoria se medía en kilobytes, los medios gráficos eran muy caros y no se disponía de dispositivos de entrada cómodos. Estos condicionantes llevaron a diseñar durante muchos años **sistemas operativos de tipo textual**, y herramientas de desarrollo basadas en el procesamiento de textos. En condiciones de este tipo nacieron los lenguajes de programación más clásicos, FORTRAN, PASCAL, C, y BASIC, y algunos de los sistemas operativos actualmente más difundidos, UNIX, VMS y MS-DOS.

Cada uno de estos sistemas y lenguajes se diseñaron para aplicaciones y usuarios distintos; cada máquina tenía que programarse en su lenguaje nativo, por medio de pulsadores primero, y de tarjetas y cintas perforadas después. Estos métodos resultaban muy complejos para el programador, y hacían su tarea lenta y poco productiva. Además, la dependencia de los programas de la máquina para la que se desarrollaban limitaba su utilidad y obligaba a continuos esfuerzos de actualización y aprendizaje de los nuevos sistemas.

Los sistemas operativos textuales se desarrollaron para facilitar la relación con el usuario, tanto en la entrada de los datos e instrucciones como en la presentación de los resultados. Dado el elevado coste de las grandes máquinas, se pretendía mejorar su aprovechamiento facilitando el acceso simultáneo de muchos usuarios a un mismo computador, para asegurar un flujo continuo de tareas a procesar por la unidad central. Estos sistemas operativos estaban asociados a una máquina o a un grupo de máquinas de un mismo fabricante, lo que facilitaba la actualización de los programas y aplicaciones desarrolladas, pero limitaba su validez general; un ejemplo de este tipo de sistemas es el clásico VMS de DEC. Para facilitar la relación de los usuarios con máquinas de diversos fabricantes y la compartición de recursos y datos entre ellas se comenzó el desarrollo de sistemas operativos independientes de la máquina, permitiendo al tiempo la simulación de multitarea y el acceso multiusuario. En esta línea se encuentra el sistema operativo UNIX.

Con la aparición a comienzos de los setenta de los microprocesadores monopastilla (*single-chip*) se hace ya posible la realización de **ordenadores personales** a precios cada vez más reducidos. Esto difunde una nueva línea de sistemas operativos pequeños monotarea y monousuario para estas máquinas. De estos sistemas destaca el MS-DOS de Microsoft.

Paralelamente a la evolución de los sistemas operativos se desarrollan **lenguajes de programación orientados a diversas aplicaciones**. Todos ellos tienen en común la descripción de las tareas a realizar por la máquina de una forma secuencial y a través de ficheros de texto. Estos ficheros codifican un lenguaje de expresiones formales que otro programa, llamado compilador, se ha de encargar de traducir al lenguaje nativo que la máquina puede ejecutar. Cada lenguaje se orienta a aplicaciones distintas: FORTRAN a cálculo científico (con fuerte carga matemática), COBOL o RPG a aplicaciones de gestión, Forth, MODULA y parcialmente C, a control o acceso a la máquina a bajo nivel.

Estos lenguajes se basan en métodos bien estructurados de análisis de lenguajes de gramáticas formales basados en expresiones regulares. Para la elaboración de una aplicación, que se concreta en uno o más ficheros que pueden ser ejecutados directamente por la máquina, se siguen normalmente varios pasos intermedios: se parte de uno o varios ficheros textuales, llamados normalmente fuentes, cada uno de estos ficheros es convertido a un formato intermedio para facilitar la tarea de la conexión de los diversos módulos, la cual realiza un programa llamado enlazador (*linker*). El uso de este formato intermedio permite la realización de un programa complejo por un equipo de personas, trabajando cada una en un subconjunto de los ficheros fuente. También permite el uso de los programas de alto nivel más adecuados para la realización de las tareas específicas de cada módulo de la aplicación. Esta estructura de trabajo ha permitido la elaboración de un enorme número de aplicaciones de los ordenadores en muy diversos campos, incluido el del control y supervisión de procesos y sistemas.

Con el continuo aumento de las capacidades de las máquinas se ha producido también un aumento en el tamaño y la complejidad de las tareas realizadas. Se ha observado entonces que la limitación en el tamaño y complejidad de los sistemas a desarrollar no estaba ya en la capacidad de las máquinas para almacenar y procesar los datos y las instrucciones, sino en la capacidad de los equipos y personas que las elaboran para controlar y comprender la complejidad de los sistemas que se crean. Inicialmente las soluciones propuestas para este problema fueron las denominadas **técnicas de programación estructurada**, con diversas aproximaciones ascendentes y descendentes. La idea básica consiste en dividir la tarea a realizar en diversas subtareas más simples, hasta conseguir llegar a definir la tarea total como un árbol de subtareas cuya comprensión, desarrollo y verificación resulte viable [Wirtyh 80]. Esta es la aproximación hoy en día generalmente más aceptada.

Frente a esta técnica de construcción de aplicaciones descendente se propone el método ascendente. En este caso, se parte de bloques pequeños bien definidos y de funcionamiento probado. A estos bloques se van añadiendo sucesivamente bloques o capas más complejas o con funciones de más alto nivel.

Ambas metodologías adolecen de un mismo defecto: en sistemas complejos el comportamiento global no se define sólo por la yuxtaposición de elementos simples, sino que surge de la interacción de los elementos que lo componen. Las técnicas de programación estructurada aseguran el buen funcionamiento de los elementos, pero no simplifican la tarea de análisis y depuración de las interacciones entre los bloques.

Para resolver estos problemas y favorecer el diseño de sistemas de control complejo y las interfaces gráficas, se han desarrollado las técnicas de **programación orientada al objeto u OOP** (*Objetc Oriented Programming*). Estos sistemas se modelan de forma natural en base a entidades que interaccionan entre sí, o responden a las acciones del usuario. Para el caso del trabajo con sistemas complejos, resulta muy útil dicha división en entidades más simples que interaccionan. Estos lenguajes mantienen las ideas de la programación estructurada, a la cual añaden la interacción entre bloques, permitiendo definir y limitar las interacciones entre los objetos, lo cual permite también organizar mejor el trabajo de diseño en equipo; asimismo, resultan de especial interés en los sistemas que han de trabajar en tiempo real (un repaso a este tema, así como los sistemas disponibles, puede encontrarse en [Willians 92]).

Paralelamente al problema de la gran dificultad del desarrollo y depuración de sistemas complejos, se ha planteado la continua necesidad del aumento de la velocidad de procesamiento en aplicaciones que tratan grandes cantidades de información, como pueda ser el caso del procesamiento de señal e imagen en tiempo real. Aunque se han conseguido grandes avances en la velocidad de las máquinas genéricas ejecutando software, en ocasiones debe recurrirse al desarrollo de hardware específico (como el ya estudiado en el capítulo 5 para el caso de las redes neuronales) y al empleo de sistemas de procesamiento masivamente paralelos.

Dada la complejidad de del hardware que puede implementarse en la actualidad (millones de puertas lógicas), se hace uso de lenguajes y compiladores para la descripción y diseño de estos sistemas, como VHDL o VERILOG. En estos lenguajes el sistema (hardware) se describe como un conjunto de procesos que se ejecutan de forma concurrente. Su interés radica en que se dispone de compiladores que son capaces de traducir una descripción del sistema en circuitos digitales que lo implementan, de manera automática y fiable. Una descripción del sistema utilizando código sintetizable (subconjunto del general) puede ser directamente implementada en una gran variedad de circuitos digitales, cada uno orientado a optimizar aspectos como velocidad, consumo, tamaño o coste. En la actualidad se dispone de herramientas software que generan una descripción utilizando código sintetizable de un FLC cumpliendo ciertas restricciones iniciales.

La introducción de entornos de desarrollo

El continuo aumento de complejidad de los sistemas de desarrollo ha forzado a introducir el concepto de **entorno de desarrollo**. Los sistemas iniciales de base puramente textual se limitaban a conjuntos de programas aislados que procesaban ficheros, produciendo otros ficheros. Estos sistemas resultaban muy complejos de manejar, y requerían de un largo período de aprendizaje y de alta especialización. Todo ello limitaba la productividad de los equipos de desarrollo y, en ultimo término, la dimensión máxima de los sistemas que pueden realizarse a un coste razonable. Los primeros entornos de desarrollo se orientaban a lenguajes textuales, e incluían editores básicos y ayudas para automatización de las tareas de depuración. Paulatinamente se han ido mejorando tanto las herramientas de edición, como las de depuración, y se han desarrollado sistemas automáticos de optimización y depuración tanto de los ficheros fuente como del código final generado. Este continuo aumento del número de herramientas y utilidades implicadas en el desarrollo de sistemas ha hecho cada vez más necesario el uso de entornos que integren y faciliten la tarea del programador individual, así como la integración del trabajo de los equipos de desarrollo.

Las nuevas tecnologías de desarrollo de aplicaciones se han beneficiado de la experiencia de los sistemas clásicos y, por orientarse a aplicaciones complejas y requerir la integración con técnicas clásicas, se han diseñado mayoritariamente con el apoyo de entornos de desarrollo más o menos elaborados. Sin embargo, por ser tecnologías recientes y de origen diverso, apenas se dispone aún de sistemas que faciliten la integración en un mismo sistema de las diversas metodologías de trabajo disponibles (redes neuronales, sistemas borrosos, algoritmos genéticos, etc.). En este sentido, quizás sea MATLAB (www.mathworks.com) el entorno más completo actualmente, pues permite el trabajo desde un mismo entorno con técnicas clásicas y novedosas (redes neuronales, sistemas borrosos, wavelets...). Su principal limitación es el limitado soporte para desarrollo de sistemas de tiempo real.

Para el campo de los sistemas basados en lógica borrosa, que en estos momentos nos ocupa, se han introducido diversos sistemas de desarrollo de propósito

específico. Resulta difícil realizar una lista completa de todos los disponibles, pues continuamente aparecen nuevos. En los puntos siguientes se describirán algunos de los más difundidos, agrupándolos en aquellos que se apoyan en conocidos programas matemáticos (MATLAB, Mathematica...), y los realizados expresamente para lógica borrosa.

10.2.1 Entornos de tipo matemático

Como herramientas para la investigación en el desarrollo de sistemas de control y procesamiento de señal, se han utilizado frecuentemente entornos de base matemática que facilitan la evaluación de diversas técnicas, sin obligar inicialmente al desarrollo de programas específicos, haciendo uso de paquetes matemáticos diversos (conjuntos de rutinas, utilidades...) incluidos en estos sistemas matemáticos de propósito general. Así, dado un sistema de desarrollo matemático de propósito general, se suelen suministrar por separado estos paquetes o conjuntos de utilidades específicos para tratamiento de señal, control de sistemas, trabajo con redes neuronales, lógica borrosa, etc.

La mayoría de estos sistemas matemáticos tienen su origen en entornos textuales, aunque en sus versiones actuales se han adaptado a los entornos gráficos disponibles hoy en día, tanto en su interacción con el usuario, como en la presentación de los resultados de las aplicaciones desarrolladas. No obstante, la mayoría conservan su base textual, más o menos oculta, tras la interfaz de usuario.

MATLAB (www.mathworks.com)

MATLAB (*Matrix Laboratory*) es probablemente el entorno de desarrollo matemático más extendido para aplicaciones de control y procesamiento de señal, especialmente en ambientes universitarios, donde se utiliza para la simulación del control de sistemas. Originariamente se escribió en FORTRAN para grandes ordenadores (*mainframes*), con objeto de proporcionar un acceso simple al software de cálculo matricial desarrollado en los proyectos LINPACK y EISPACK.

Este entorno puede considerarse una especie de sistema interactivo cuyos elementos básicos son matrices, cuyo dimensionamiento se realiza dinámicamente. La solución de los problemas se expresa en MATLAB con expresiones matemáticas similares a las habituales, de modo que en principio no requiere programación (aunque sí pueden realizarse programas a base de ficheros de macros).

MATLAB se utiliza habitualmente en entornos de ingeniería para analizar y desarrollar prototipos de algoritmos o computaciones numéricas, utilizando una formulación matricial que se adapta bien tanto a la teoría de control automático clásico, como al procesamiento digital de señales. Posee una amplia librería de funciones matemáticas, operaciones aritméticas, operadores lógicos y relacionales, funciones para el análisis de datos, funciones especiales para matrices, cálculo

polinomial y análisis de sistemas no lineales. Dispone también de funciones de control de flujo típicas de los lenguajes de programación, así como de una librería básica de funciones gráficas de tipo 2D y 3D para el análisis de los datos.

Se dispone actualmente de versiones para un gran número de máquinas, desde ordenadores potentes, como estaciones de trabajo (por ejemplo, Sun/SPARC), hasta ordenadores personales, como PC con sistema Windows y Macintosh.

La utilización de MATLAB de forma interactiva se basa en la definición de variables globales, que siempre son matrices, y el enunciado de operaciones entre ellas, que siempre se acumulan en otras matrices. Se dispone de muchas funciones, como las clásicas que calculan el determinante de una matriz o su inversa, no obstante, la característica más importante de MATLAB es que pueden definirse nuevas funciones como secuencias de comandos ya definidos, almacenándolos en un fichero de tipo *M-file*. Los *M-files* pueden ser usados para crear guiones (*scripts*) para automatizar largas secuencias de comandos. Existen amplias colecciones de estos *M-files* o *toolboxes* (caja de herramientas), que han sido escritas para aplicaciones especiales, y que se venden por separado. Por ejemplo, la *Signal Processing Toolbox* contiene un gran número de funciones para procesamiento de señal; la *Control System Toolbox*, orientada al análisis y síntesis de sistemas automáticos de control basados en la teorías clásicas; o la librería *System Identification Toolbox*, que se especializa en la identificación de modelos de datos de entrada y salida observados en el sistema. En los últimos años se han incorporado el *Neural Networks Toolbox* (capítulo 6) y el *Fuzzy Logic Toolbox*, que permiten el trabajo con redes neuronales y lógica borrosa, respectivamente, con relativa sencillez y buenas salidas gráficas.

Como complemento a MATLAB se dispone de SIMULINK, un entorno gráfico orientado a la simulación de sistemas dinámicos no lineales. Para el modelado de sistemas dispone de diferentes bloques lineales y no lineales, tanto en el campo continuo como en el discreto. Los sistemas se describen como bloques que se interconectan, organizados jerárquicamente. Para el análisis de sistemas se pueden utilizar diversos algoritmos de resolución de ecuaciones diferenciales, como Runge-Kutta de tercer orden o Runge-Kutta-Fehlberg de quinto orden. También permite la extracción de modelos lineales en torno a un punto de operación del sistema no lineal, y posee herramientas para la determinación de puntos de equilibrio.

Este conjunto de herramientas resulta muy útil para las simulaciones requeridas en el proceso de diseño de un sistema complejo, pues permiten explorar con relativa sencillez distintas soluciones. Sin embargo, para aplicaciones en tiempo real en ingeniería puede plantear problemas por la gran carga computacional requerida. Finalmente, indicaremos que se dispone de *toolboxes* para la conversión de sistemas descritos en forma de *M-files* o ficheros de SIMULINK a sistemas autónomos basados en procesadores de tipo 486 o superiores. Se puede también incluir el sistema desarrollado en aplicaciones realizadas en C o C++ mediante diversos compiladores, e incluso se dispone de una *toolbox* para adquisición de datos del mundo real.

Las herramientas de MATLAB *fuzzy logic toolbox* permiten diseñar sistemas basados en lógica borrosa con potentes técnicas de entrenamiento, así como su integración en sistemas de control complejo simulables en SIMULINK. Para comenzar a utilizar esta herramienta se pueden ejecutar las siguientes órdenes:

help fuzzy, informa de las funciones disponibles en esta toolbox.

demo toolbox fuzzy, da acceso a las principales demostraciones de esta toolbox.

fuzzy, editor gráfico de los FIS (*Fuzzy Inference Systems*).

anfisedit, interfaz gráfica para crear, entrenar y probar FIS de tipo Sugeno.

anfis, rutina que permite entrenar FIS tipo Sugeno con técnicas de aprendizaje del tipo del conocido descenso por el gradiente [Jang 92-93b, 97], similar al estudiado para los modelos de redes neuronales artificiales (modelo ANFIS).

genfis1, rutina que crea FIS tipo Sugeno a partir de datos de entrenamiento.

genfis2, ídem utilizando técnicas de *fuzzy subtractive clustering*.

Mathematica (Wolfram Research, www.wolfram.com)

El entorno *Mathematica* es probablemente uno de los paquetes software de tipo matemático más extendido, siendo empleado por investigadores, ingenieros y estudiantes de escuelas de ingeniería y ciencias. Cubre no solo las áreas científicas y matemáticas, sino que también se emplea en áreas tecnológicas y empresariales. La primera versión de Mathematica se anunció en 1988, y se realizó para el MacIntosh de Apple; posteriormente se realizaron versiones para estaciones de trabajo, y en 1989 se realizó la primera versión para PC compatible (basado en máquinas 386). A finales de 1990 se disponía ya en una amplia variedad de ordenadores, como CONVEX, DEC VAX, RISC, HP/Apollo, PC compatibles (DOS y Windows), IBM RISC, MIPS, NeXT, Silicon Graphics, y Sun/SPARC. Una completa descripción de este entorno puede encontrarse en [Wolfram 91]. Stephen Wolfram, autor de este libro, es el desarrollador principal de Mathematica; trabajó originariamente en física de altas energías, física cuántica teórica y cosmología, actualmente se dedica al análisis de sistemas complejos, y es pionero de los modelos computacionales de autómatas celulares. En 1986 fundó *Complex Systems* la principal revista de este campo.

El entorno Mathematica está dividido en dos partes: el núcleo (*kernel*), que realiza los cálculos, y un panel frontal (*front end*), que se ocupa de la interacción con el usuario. El núcleo de Mathematica es idéntico en todas las computadoras en las que se ejecuta, en cambio, el panel frontal está adaptado a las características de cada máquina. En la mayor parte de las implementaciones de Mathematica el panel frontal soporta un tipo de documento interactivo llamado *notebooks*. Estos documentos están formados por textos conectados jerárquicamente, junto con gráficos que pueden ser animados, y expresiones matemáticas que pueden ser utilizadas para realizar cálculos.

Mathematica puede conectar con otros programas estándar para permitir el intercambio de datos. Así, los gráficos en Mathematica se representan en lenguaje postscript para que puedan ser intercambiados con otros programas. Además, puede leer datos en diversos formatos y generar salidas en formatos C, FORTRAN y T_EX. Puede también comunicar con otros programas realizados en lenguaje de alto nivel usando el estándar de comunicación MathLink (el panel frontal de Mathematica usa MathLink para comunicar con el núcleo de la aplicación); esta comunicación puede realizarse en un ordenador aislado o a través de una red entre diferentes ordenadores. Se puede usar MathLink también para controlar programas externos, así como preparar entradas y comandos usando el lenguaje de Mathematica, y enviar estos a un programa externo vía MathLink. Finalmente puede utilizarse Mathematica para mostrar o analizar los resultados obtenidos. Por último, también se puede hacer uso de MathLink para crear programas que llaman a Mathematica, como si de una subrutina se tratase. De esta manera pueden crearse nuevos paneles frontales.

El entorno Mathematica puede ser usado para realizar cálculos numéricos y simbólicos, visualizar funciones, modelar y analizar datos, representar conocimientos, y generar documentos interactivos, y a través de MathLink puede interactuar con otros programas.

Asimismo, dispone de un lenguaje de programación de alto nivel con el que pueden escribirse programas. La principal cualidad de Mathematica es la enorme cantidad de funciones ya definidas de que dispone. Se pueden realizar programas de diversos estilos, pues soporta la programación procedural con estructuras de bloques, condicionales, iteraciones y recursión; también puede realizarse una programación funcional, haciendo uso de las funciones y de los operadores funcionales. Finalmente se puede llevar a cabo una programación basada en reglas, con reconocimiento de patrones y orientación al objeto. Una más completa información sobre las capacidades de programación de Mathematica pueden encontrarse en [Maeder 89].

La principal limitación de Mathematica viene de su potencia, pues requiere enormes recursos de máquina, tanto en capacidad de disco como en memoria. Por la generalidad de sus algoritmos, los programas escritos con Mathematica resultan más lentos que los realizados con lenguajes de programación convencionales, aunque su desarrollo es más rápido. Por estas circunstancias, no resulta un entorno adecuado para realizar control en línea. La gran variedad de funciones disponibles hace también que sea largo el tiempo de aprendizaje de la herramienta. En resumen, se trata de una buena herramienta para análisis teóricos e investigación básica.

MathCAD (MathSoft, www.mathsoft.com)

El entorno matemático MathCAD es más reciente que los anteriores, por lo cual recoge desde el principio de su diseño la conocida filosofía WYSIWYG (*What you see is what you get*, es decir *Lo que se ve es lo que se obtiene*). Este estilo de interfaz gráfica de usuario o GUI (*Graphical User Interface*) es propio de los procesadores de textos actuales, y supone que no hay diferencias entre la presentación del documento

en pantalla y lo que se obtiene en una copia impresa. Esta idea se extiende en MathCAD no sólo a los textos, sino a ecuaciones, gráficos y demás objetos que forman un documento. Además de una buena apariencia para las fórmulas, MathCAD permite realizar cálculos reales con ellas. A diferencia de otros editores matemáticos, las ecuaciones se introducen tal y como se escribirían normalmente. Se dispone de todas las funciones matemáticas habituales y de un alfabeto griego completo.

Para facilitar la tarea del usuario, se dispone de consultas en línea, con instrucciones paso a paso y ejemplos. Se pueden añadir también una amplia gama de manuales electrónicos para áreas específicas (ingeniería eléctrica, electrónica, química, civil y mecánica), estos manuales se conocen como MathTOOLS. Se dispone también de manuales para métodos numéricos y estadística.

MathCAD permite la confección de documentos de tipo técnico o matemático tal y como se escribirían sobre un papel o pizarra. Además las fórmulas y ecuaciones del documento pueden ser calculadas por MathCAD tanto de forma numérica como simbólica. Los resultados de los cálculos numéricos y las funciones definidas pueden visualizarse en gráficas de diversos tipos.

Dentro del cálculo simbólico, MathCAD realiza integración y diferenciación de expresiones, puede calcular la inversa, traspuesta y determinante de una matriz, factorizar y simplificar expresiones, así como encontrar soluciones de ecuaciones (todo ello simbólicamente). Para facilitar los cálculos dispone de una amplia gama de funciones ya definidas: trigonométricas, exponenciales, hiperbólicas, Bessel, estadísticas y de procesamiento digital de señal (en una y en dos dimensiones). Permite también la operación con matrices y vectores numéricos, tanto reales como complejas. Por último, el usuario puede definir libremente nuevas funciones.

Desde el punto de vista de su conexión con otras aplicaciones, permite la importación y exportación de datos de forma sencilla; por ejemplo, soporta las interfaces OLE y DDE propios de MS-Windows, y permite también la importación de archivos en formato HPGL.

Todas estas cualidades hacen de MathCAD una herramienta adecuada para el análisis de datos y sistemas, pudiéndose emplear en la formulación e investigación de algoritmos de control de sistemas y para su modelado. Por su facilidad de su uso, resulta cómodo para la formulación y verificación rápida de nuevas hipótesis y nuevos modelos de control. Al no disponer de conexión con aplicaciones de control y no considerar el tiempo como una variable de los cálculos, resulta más complejo su empleo en control en línea.

10.2.2. Entornos de lógica borrosa

Expuesas en la sección anterior algunas herramientas de tipo matemático, susceptibles de ser empleadas en el desarrollo de sistemas borrosos y/o neuronales, describiremos a continuación algunos de los entornos más difundidos orientados específicamente a lógica borrosa. Nuestro objetivo no es realizar un repaso exhaustivo de todos los entornos disponibles, sino mostrar las características de algunos de los más utilizados. Se indicarán también diversos entornos experimentales para el desarrollo de sistemas borrosos propuestos por centros académicos.

FuzzyTECH (INFORM GmbH, www.fuzzytech.com)

En el CD-ROM que acompaña este texto se incluye una versión de demostración del entorno fuzzyTECH, así como abundante información de su uso, características y aplicaciones. El entorno fuzzyTECH es uno de los más difundidos y completos para el desarrollo de sistemas basados en lógica borrosa. En el apéndice A se encuentra una descripción del procedimiento de instalación.

El entorno fuzzyTECH fue desarrollado por la compañía *INFORM Software GmbH*, el cual surgió del trabajo de un grupo de investigadores dirigido por el profesor Hans Zimmermann, de la Universidad de Aachen (Alemania). Zimmermann, uno de los pioneros de la lógica borrosa en Europa, es presidente y fundador de la *International Fuzzy Systems Association* (IFSA), la principal organización internacional para la investigación y aplicación de los sistemas basados en lógica borrosa. Es también editor principal de la revista *Fuzzy Sets and Systems Journal*, una de las más importantes publicaciones científicas dentro de esta área.

El entorno fuzzyTECH está formado por un GUI común basado en MS-Windows que, entre otras cosas, permite la edición gráfica de las variables lingüísticas para cada una de las variables del sistema, con una precisión seleccionable según el tipo de implementación final seleccionado (8 o 16 bits). Se pueden utilizar gran variedad de funciones de inclusión, además de las de tipo S, Z, Lambda y Pi. Posee también un editor gráfico de reglas, con un formato similar al de una hoja de cálculo. Se pueden asociar a las reglas diversos métodos de inferencia estándar, y también pueden asignarse pesos individuales a cada una de las reglas. Estos pesos pueden ajustarse de forma automática utilizando un entrenamiento basado en FAM. Admite varias variables de entrada y de salida por módulo, y permite el empleo de diversos tipos de desborrosificación (máximo de centros, centro de áreas, singletons y área central).

FuzzyTECH dispone de varios asistentes para facilitar las tareas más frecuentes del diseño, como asistente para la estructura, para las variables y para las reglas. Incluye además una ventana de gestión del proyecto con estructura de árbol, un sistema personalizable de generación automática de documentación del proyecto según el estándar IEC1131-7, y un gestor de versiones integrado.

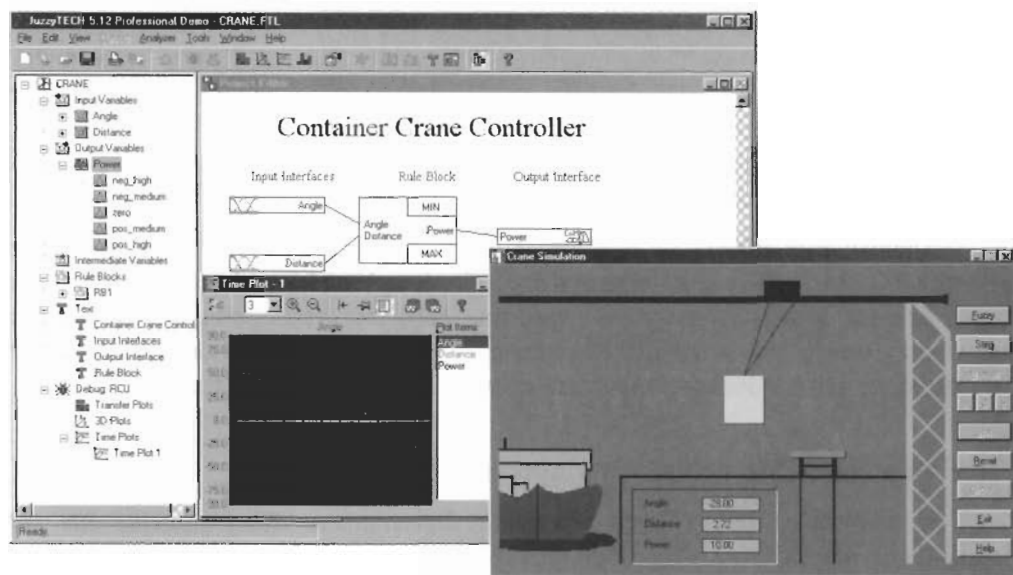


Figura 10.3. Entorno fuzzyTECH

El entorno admite la simulación fuera de línea y en tiempo real. Dentro del primer tipo, permite la depuración interactiva con visualización de los flujos de inferencia, y optimización interactiva de los parámetros del sistema. Admite también la simulación basada en datos pregrabados del sistema, y visualiza la respuesta del sistema en tiempo real.

El modelado del sistema puede realizarse a través de la conexión por DDE con un programa específico para cada aplicación, que ha de ser creado por el usuario, o con su conexión a los principales programas de simulación como InTouch™, InControl™, CiTECT™, LabVIEW™, Matlab/Simulink™, VisSim™, WinFACT™, WinCCT™, FIX™ o BridgeVIEW™. Puede conectarse utilizando DDE, DLL o ActiveX con otros programas, como Simulink™, InTouch™, Excel™, Access™ o VisualBasic™. Además, permite el análisis gráfico avanzado, visualizando la superficie de control y la activación de las reglas.

Para la simulación en tiempo, real permite la conexión en línea con sistemas basados en diversos microprocesadores. El sistema así dispuesto recibe los datos del microprocesador, que los transmite a la aplicación, ésta los procesa y actúa sobre las salidas del microprocesador para controlar el sistema.

Como cualidad importante de este entorno destacaremos que dispone de un gran número de versiones con una interfaz de usuario común, adaptadas a un gran número de implementaciones. La versión MCU-C produce código C según el estándar ANSI, para su utilización en aplicaciones basadas en estaciones de trabajo o PC. Se dispone también de versiones que generan código para los microcontroladores ST6 de SGS

Thompson, 8051, MCS-96 de Intel, 81C99 y 80C166 de Siemens, PIC de Microchip, e incluso para los DSP de Texas Instruments (familias TMS320C2X y TMS320C5X de coma fija, y TMS320C3X y TMS320C4X de coma flotante). Esto supone una ampliación considerable de las posibilidades de este entorno, dado el bajo costo de los microcontroladores y la gran capacidad de cálculo los DSP (microprocesadores orientados específicamente al procesamiento de señal).

Además, este entorno ha sido seleccionado por diversos fabricantes de sistemas industriales como herramienta para el desarrollo de aplicaciones de control basadas en lógica borrosa sobre sus sistemas. Para una completa descripción de los sistemas soportados, consúltese www.fuzzytech.com/E/links.html; una visión general de las características del producto se encuentra en www.fuzzytech.com/E/ftpo.html.

FIDE (Aptronix-Motorola, www.aptronix.com).

El entorno FIDE (*Fuzzy Inference Development Environment*) ha sido desarrollado por la compañía americana Aptronix, en colaboración con Motorola. Este entorno se basa en un lenguaje de descripción de controladores llamado FIL (*Fuzzy Inference Language*), que a su vez se compone de tres sublenguajes, para cada uno de los bloques posibles en el entorno. El lenguaje FIU (*Fuzzy Inference Unit*) permite describir las unidades de inferencia borrosas formadas por conjuntos de reglas que se aplican sobre variables, para las que se pueden definir diversos adjetivos. Por ejemplo, para el problema de controlar un péndulo inverso podríamos definir los diversos adjetivos de la variable que indican el ángulo del péndulo como:

```
invar pend_angle " " : -127 ( 1 ) 127 [
$ pend_angle is an input variable
  N_Large (@-127, 1, @-112, 1, @-64, 0),
  N_Medium (@-112, 0, @-64, 1, @-16, 0),
  N_Small (@-64, 0, @-16, 1, @16, 0),
  Cero "pend_zr",
  P_Small (@-16, 0, @16, 1, @64, 0),
  P_Medium (@16, 0, @64, 1, @112, 0),
  P_Large (@64, 0, @112, 1, @127, 1)
];                                $ end of labels
```

Que correspondería a un conjunto de ángulos tal y como se visualiza en la Figura 10.4.

Las reglas en el lenguaje FIU se expresan en un estilo similar al inglés natural. También pueden introducirse utilizando una interfaz de usuario de tipo hoja de cálculo. Para el caso del péndulo inverso un conjunto de reglas válidas sería:

```
if pend_angle is N_Large and pend_velocity is N_Large then force is N_Large;
if pend_angle is N_Large and pend_velocity is N_Medium then force is N_Large;
if pend_angle is N_Large and pend_velocity is N_Small then force is N_Large;
```

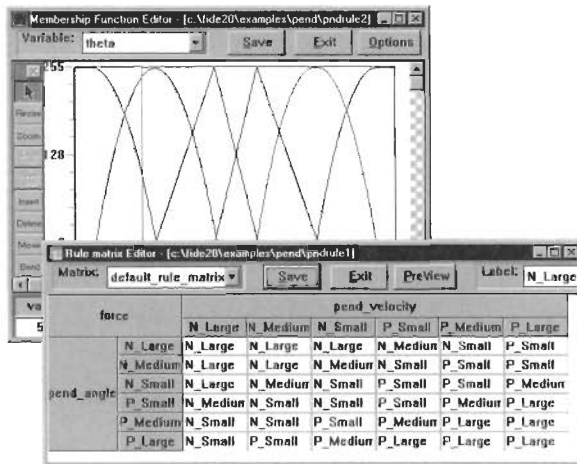


Figura 10.4. Entorno FIDE: funciones de inclusión y editor de reglas

Para el preprocesado de las variables del sistema se define un lenguaje llamado FOU. Mediante este sencillo lenguaje es posible definir diversos escalados y operaciones aritméticas con las variables. Se dispone también de diversas funciones matemáticas elementales para aplicar a las variables, y pueden definirse también tablas de una o más direcciones para modelar la respuesta de un bloque. Finalmente, el lenguaje FEU proporciona una interfaz para acceder a módulos escritos en lenguaje C.

Así, el entorno FIDE se estructura en torno a un editor de textos multiventana elemental, que incluye un compilador del lenguaje FIL. Desde esta aplicación MS-Windows es posible llamar a los cuatro bloques básicos que componen el entorno: COMPILER, DEBUGGER, COMPOSER y RTC. El bloque COMPILER compila los ficheros fuente del lenguaje FIL y genera un código objeto para el resto de las aplicaciones. La aplicación DEBUGGER posee tres herramientas: *Tracer*, *Analyzer* y *Simulator*. *Tracer* permite obtener el valor de las salidas para unos valores concretos de las entradas, y seguir paso a paso el proceso de inferencia que produce los valores de salida. *Analyzer* realiza el análisis de la función de transferencia del sistema, así como la visualización de la función de salida desde diversas perspectivas. Es posible realizar una representación tridimensional de la superficie de salida (Figura 10.5), o bien visualizaciones bidimensionales y topológicas. Finalmente, *Simulator* simula el comportamiento dinámico de las unidades para unos datos de test (Figura 10.6).

Por otra parte, la aplicación COMPOSER permite al usuario crear sistemas complejos que combinan módulos borrosos con otros no borrosos, utilizando el denominado FCL (*Fide Composer Language*). Esta herramienta proporciona un editor gráfico para el diseño de los sistemas de inferencia. A modo de ejemplo, en la Figura 10.7 puede verse la descripción mediante COMPOSER de un sistema de inferencia borrosa aplicado al control de un péndulo inverso.

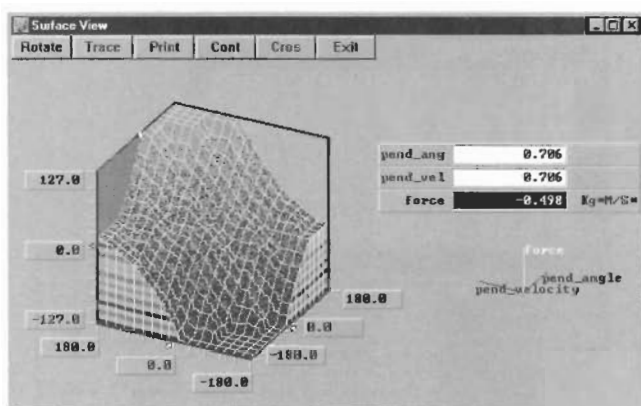


Figura 10.5. FIDE: Superficie de salida del controlador fuzzy

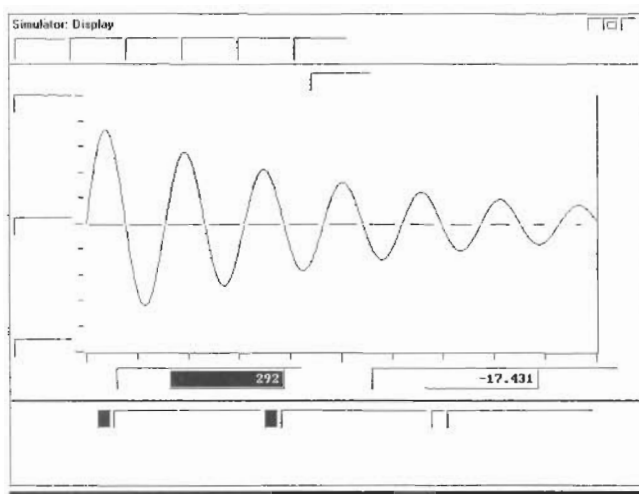


Figura 10.6. FIDE: Comportamiento dinámico del controlador fuzzy

Finalmente, la aplicación RTC permite convertir la descripción en código objeto creada por el compilador de fuente en el código máquina adecuado para diversos microcontroladores de Motorola. En la versión actual es posible generar código para los bien conocidos M6805, M68HC05, M68HC08, M68HC11, M68HC16, M6833X. También se puede generar código para MATLAB y ficheros fuente C ANSI.

La simulación del entorno externo se realiza por medio de un programa aparte; los fuentes C de este programa son creados automáticamente por el entorno. Mediante la aplicación COMPOSER resulta también posible realizar la compilación y ejecución

de este programa (el compilador utilizado en este caso es el extendido Borland C). El programa así generado se ejecuta en MS-DOS.

Como puede apreciarse, existen algunas dificultades en el uso de este sistema de desarrollo: por un lado, la diversidad de formatos a utilizar en los diferentes ficheros complica el aprendizaje y utilización del entorno; por último, al permitir solamente el empleo de una familia de microcontroladores (Motorola) su aplicación resulta limitada.

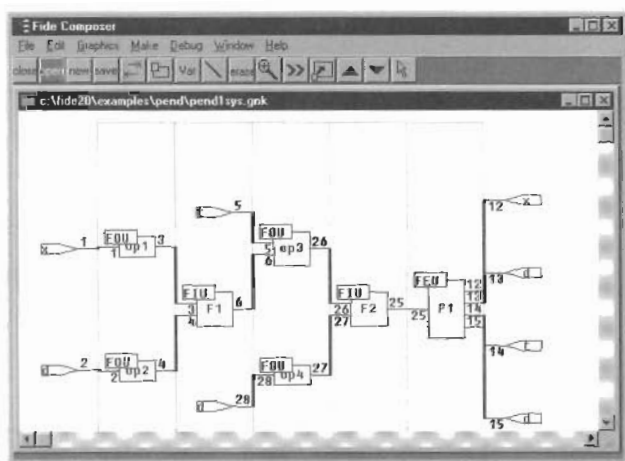


Figura 10.7. Presentación de Composer, el editor gráfico de FIDE

TILShell (Togai Infralogic, www.ortech-engr.com/fuzzy/togai.html).

El entorno TILShell, de la compañía **Togai InfraLogic**, es uno de los entornos para desarrollo de sistemas basados en lógica borrosa más completos. Dispone de una interfaz de usuario que permite la definición gráfica del sistema en base a bloques funcionales conectados. El sistema se completa además con una amplia gama de herramientas para la implementación física de los controladores, y mediante *Fuzzy-C Development System (FCDS)* puede generarse código fuente C para la implementación de los sistemas definidos con TILShell. Por último, puede implementarse el sistema borroso diseñado en un acelerador de cálculo basado en el circuito acelerador FC110 o, mediante MicroFPL, en microcontroladores estándar de 8 y 16 bits.

El entorno dispone también de una herramienta para la generación automática de las reglas, utilizando la definición de las entradas y salidas del sistema. Es posible también implementar los controladores borrosos en circuitos integrados ASICs propiedad de la compañía haciendo uso del *Fuzzy Computational Acceleration (FCA)*. La estructura general del entorno puede verse en la Figura 10.8.

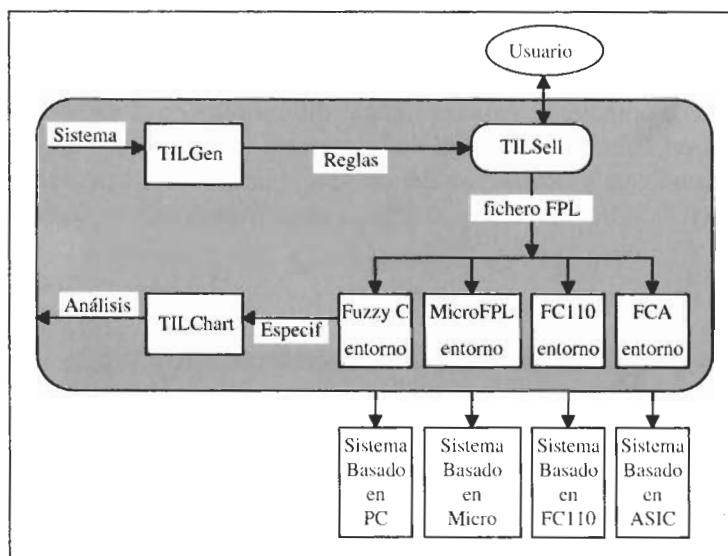


Figura 10.8. Conjunto de aplicaciones del sistema de Togai InfraLogic

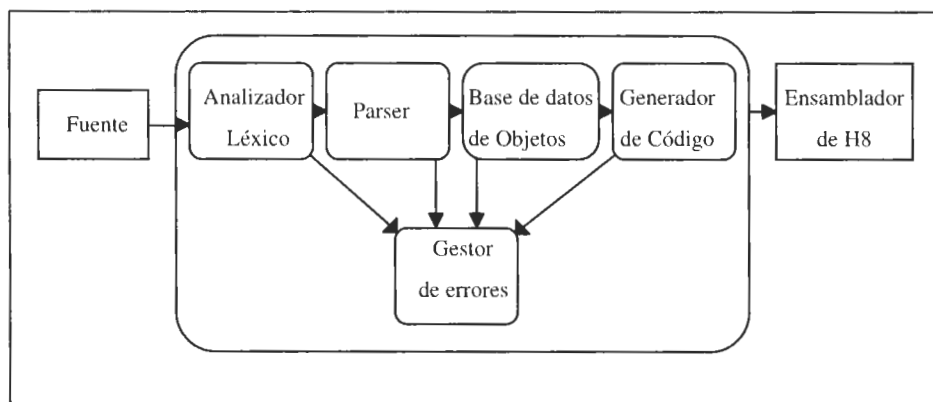


Figura 10.9. Esquema del interpretador de FPL para el microcontrolador H8

El entorno está basado en un lenguaje de especificación de controladores borrosos llamado FPL, que es independiente del soporte físico del controlador. El entorno dispone de compiladores que traducen los programas escritos en formato FPL a los diversos lenguajes finales, como C, código máquina del FC110, o de otros microcontroladores, como el 8051 de Intel o el 68HC11 de Motorola. Es posible también la utilización de un sistema de inferencia basado en el microcontrolador de 8 bits H8 de Hitachi; un compilador adecuado convierte los fuentes de FPL a un código binario para interpretar por el MicroFPL (Figura 10.9).

El entorno TILShell permite la simulación gráfica de sistemas. El sistema externo se simula como una aplicación MS-Windows que comunica los datos con el entorno por DDE. Se tienen tres formas de definir las reglas: mediante un editor de textos interno, con formato similar a una hoja de cálculo, y con un editor matricial de dos dimensiones.

Las funciones de inclusión pueden ser de tipo S, a tramos o con expresiones matemáticas. Se admiten diversos métodos de desborrosificación, incluyendo el método del centro de masas y el Takagi-Sugeno. Por último, dispone de un potente gestor gráfico de proyectos, que permite gestionar todos los aspectos que definen una aplicación, la base de reglas y su agrupamiento, los adjetivos para cada variable y sus funciones de inclusión y la conexión de los bloques.

Cubicalc (HyperLogic, www.hyperlogic.com)

El entorno de desarrollo para sistemas basados en lógica borrosa Cubicalc, con GUI para MS-Windows, está basado en un lenguaje de especificación simple orientado a lógica borrosa. Posee un editor gráfico de las funciones de inclusión, o adjetivos asociados a cada variable del sistema, y la base de reglas se define utilizando un editor estándar y en un formato textual intuitivo. Un ejemplo de sistema para trazar una trayectoria podría ser el siguiente

```
If RadiusError is Positive then make SteeringAngle Negative;
If RadiusError is NearZero then make SteeringAngle NearZero;
If RadiusError is Negative then make SteeringAngle Positive;
```

También es posible definir un preprocesado de las variables. Siguiendo con el mismo ejemplo, se tendría

```
# Calculate current range error.
# When close enough, restart
RadiusError = car_radius - desired_radius;
when (abs (RadiusError)<.01)
  Restart = AsKYN("Close enough?");
end
```

La simulación de la respuesta del sistema externo puede realizarse también mediante un fichero de texto:

```
car_radius = sqrt( car_radius * car_radius +V * V
  + 2* car_radius *V * sind(SteeringAngle));
when (car_radius<> 0.0)
  car_azimuth = angle360( car_azimuth
  + asind((V * cosd(SteeringAngle)) / car_radius));
end
```

CubiCalc dispone también de un amplio conjunto de gráficas polares, cartesianas y de barras para la visualización de los resultados de la simulación. Una vez depurado el sistema puede transferirse a un fuente C o C++ que pueda ser ejecutado en microcontroladores de 8 bits o en miembros de la familia x86, utilizando CubiCalc RTC. Se puede fijar la precisión de los cálculos utilizando coma flotante o enteros de 8 bits, lo que permite combinar los sistemas desarrollados mediante este entorno con otras librerías. Dentro de Windows puede ser utilizado por otras aplicaciones C, C++, VisualBasic o Excel a través de DLL. Para sistemas basados en microcontroladores de 8 bits puede generar código para incluir en ROM.

Con Cubicalc es posible utilizar la tarjeta CubiCard para realizar la interacción directa del entorno ejecutado en Windows con el sistema externo. Esta tarjeta dispone de 16 entradas analógicas u 8 diferenciales, con una precisión de 8 bits y ganancia programable. Posee también 8 entradas y 8 salidas digitales, y dos salidas analógicas de 8 bits. Asimismo, cuenta también con tres contadores de 16 bits programables.

Entornos académicos de lógica borrosa

Lo primero a tener en cuenta es que la mayoría de los sistemas comerciales disponibles han surgido de grupos o profesores universitarios, como TILSell, del profesor Togai, o fuzzyTECH, del profesor Hans Zimmermann, quienes han fundado empresas para la comercialización de estas tecnologías.

Quizás debido a ello no existan demasiados entornos académicos (de dominio público) en el área de los sistemas basados en lógica borrosa. No obstante, algunos libros incluyen librerías o pequeños entornos, como el de Bart Kosko [Kosko 92], y algunos grupos universitarios han elaborado entornos de dominio publico, como es el caso del profesor Herrera, de la Universidad de Granada [Herrera 93b]. No obstante, remitimos al lector interesado a Internet, donde sin duda encontrará información actualizada (puede comenzarse la búsqueda a partir de las direcciones suministradas en el apéndice B).

10.3 CODIFICACIÓN EN C

La simulación de un FLC en un computador o en un microcontrolador es uno de los procedimientos más utilizados para su implementación; cuando las restricciones de velocidad lo permiten, resulta la forma más barata y eficiente. Normalmente estos programas se realizan en lenguaje C; en los buscadores de Internet (como www.altavista.com, www.yahoo.com o alguna de las direcciones del apéndice B) y en la bibliografía es fácil encontrar ficheros fuentes en C de realizaciones simples de FLC.

A modo de ejemplo, para implementar un FLC de tipo MIMO (sección 7.9) podría utilizarse una definición simple como la siguiente:

```
typedef struct
{
    byte ni;      /* Número de variables de entrada */
    byte no;      /* Número de variables de salida */
    byte nr;      /* Número de reglas */
    word *in;     /* Lista de las variables de entrada */
    word *out;    /* Lista de las variables de salida */
    word *r;      /* Lista de reglas ("ni" adjIn, "no" adjOut, ...)*
} sistema;
```

La función que calcula la salida podría ser

```
void run_sis(sistema *si) // Ejecuta un sistema fuzzy
{
    byte i,j;
    word *p;           /* Puntero al contenido de cada regla */
    byte val;          /* Valoración de la regla */

    for (p=(word *)sv,i=0;i<2*si->no;i++,p++) *p=0;
    for (p=(word *)vt,i=0;i< si->no;i++,p++) *p=0;

    p=si->r;

    for (i=0;i<si->nr;i++)
    {
        /* Para cada regla */
        val=0xff;
        for (j=0;j<si->ni;j++)
        {
            /* Para cada entrada */
            if (*p)
                val=min(val,si->in[j],ai[*p]);
            p++;
        }
        for (j=0;j<si->no;j++)
        {
            /* Para cada salida */
            if (*p)
            {
                vt[j]+=val;
                sv[j]+=((word)val*ao[*p]);
            }
            p++;
        } /* for (regla.salida) */
    } /* for (regla) */

    for (i=0;i<si->no;i++)
        if (vt[i])
            v[si->out[i]]=sv[i]/vt[i];
} /* run_sis */
```

Y, por su parte, la función que calcula el grado de pertenencia a una función podría ser

```
byte is(byte w,byte *a) /* Devuelve 0..0xff */
{
  if (w>=a[1] && w<=a[2]) return(0xff);
  if (w<=a[0] || w>=a[3]) return(0);
  if (w<a[1])
    return(((w-a[0])<<8)/(a[1]-a[0]));
  return(((a[3]-w)<<8)/(a[3]-a[2]));
}
```

Este tipo de codificación es eficiente para sistemas simples, pero puede resultar de difícil mantenimiento en sistemas complejos, en cuyo caso es preferible utilizar una codificación basada en un lenguaje de más alto nivel, como C++, y en una librería de clases específica.

10.4 CODIFICACIÓN EN C++

Como ejemplo de la codificación de un controlador complejo basado en un lenguaje de más alto nivel, como C++, y en una librería de clases específica, describiremos el modelo de sistemas de respuesta autónoma ARS, incluido en el entorno de desarrollo IDEA, realizado en la Universidad de Zaragoza por uno de los autores [Sanz 94, 96b].

10.4.1 El modelo ARS (Sistemas de Respuesta Autónoma)

El concepto de **Sistemas de Respuesta Autónoma o ARS** (*Autonomous Response Systems*), se introduce para permitir el modelado y desarrollo de sistemas artificiales capaces de actuar y responder de forma autónoma frente a estímulos externos en función de su adiestramiento e instintos. Este tipo de sistemas pretende ser una aproximación de los sistemas nerviosos de los animales, capaces de responder con secuencias de actuaciones ante las modificaciones observadas en su medio.

Este concepto se introduce como base del lenguaje de alto nivel **EDA** (Especificación y Desarrollo de ARS), permitiendo la descripción de sistemas complejos como conjuntos de objetos conectados. En este esquema, los objetos heredan propiedades de padres a hijos, por ello, los objetos hijos o derivados de un **conjunto de neuronas artificiales o ANS** (*Artificial Neural Set*¹) comparten ciertas

¹ En esta Sección utilizaremos el término ANS, o *Artificial Neural Set*, para denotar un conjunto de neuronas perteneciente a un sistema nervioso; en este sentido un ANS equivaldría a un ganglio de los que conforman los sistemas nerviosos de los invertebrados, como veremos un poco más adelante. El ANS tal y como lo entenderemos en esta sección no debe confundirse con el término ANS general, procedente de *Artificial Neural Systems*, con el que se hace referencia en este libro (y en muchos otros) al campo de las redes neuronales, en general, y a los diversos modelos concretos de redes neuronales artificiales (BP, Kohonen, etc.), en particular.

propiedades o métodos que permiten utilizarlos con independencia de su estructura interna. La principal característica de los ANS es que se conectan entre sí con el uso de objetos que denominaremos de **tipo Vinculo**². La conexión de dos ANS mediante un vínculo supone la compartición de los datos contenidos en él. Cada ANS dispone de una lista de vínculos de entrada, y otra de salida: los vínculos de entrada, que determinan las salidas, se pueden leer pero no modificar; los vínculo de salida se modifican almacenando las salidas del ANS [Sanz 94].

Como se puede observar en la Figura 10.11, los ARS son hijos de la clase *Comp* por lo que pueden contener otros ANS interconectados. Como son también descendientes de *ANS* pueden conectarse con otros ANS. El papel de la clase ARS es permitir la conexión con el exterior, modelado con un objeto de la clase *World*.

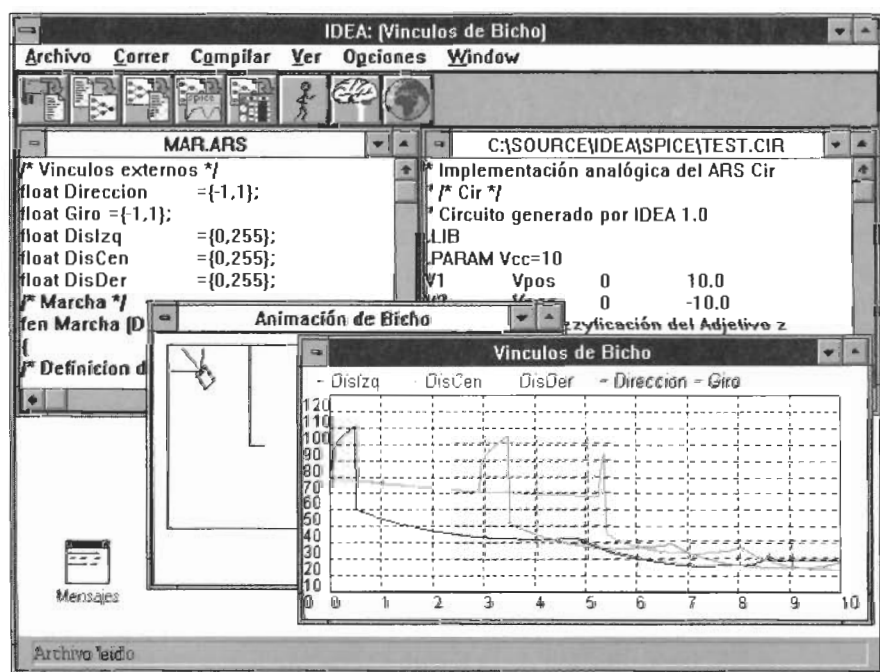


Figura 10.10. Entorno IDEA (Universidad de Zaragoza [Sanz 94, 96b])

² Por tratarse de un término (objeto, clase) que se maneja dentro del entorno de programación, el cual no contempla acentos, utilizaremos *Vinculo*, con mayúscula y cursiva, pero sin acento, para referirnos a él. En general, en esta sección denotaremos con cursiva los términos que nombran clases, funciones, variables, etc., del entorno informático que se describe.

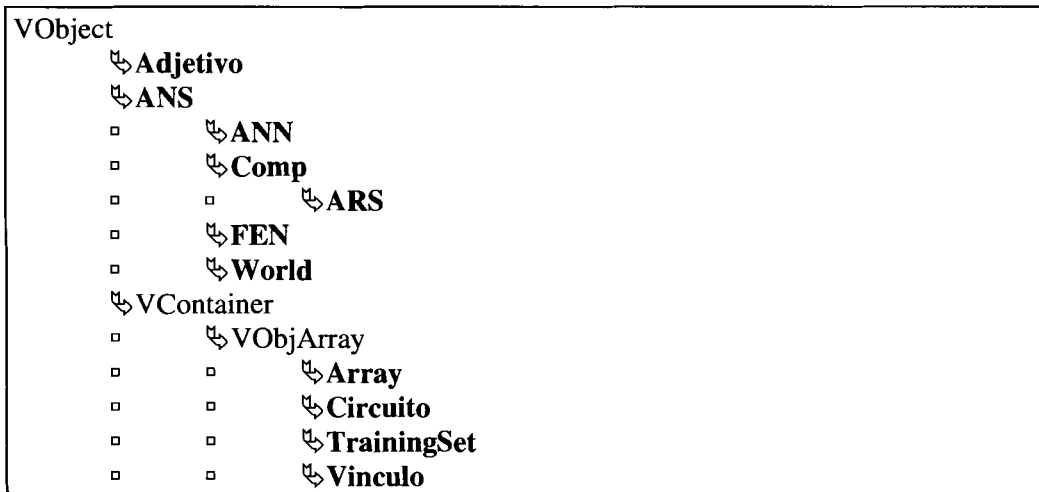


Figura 10.11. Clases del modelo ARS, sistemas de respuesta autónoma

Por lo tanto, los ARS se forman por la conexión de conjuntos de neuronas artificiales o ANS. Esta estructura surge del estudio de los sistemas nerviosos de los invertebrados más simples, que poseen solamente de 10.000 a 100.000 neuronas (frente a las 10^{11} de los seres humanos). Se ha observado además en estos sistemas que las neuronas se agrupan formando **ganglios** (conjuntos de neuronas) de entre 500 a 1.500 unidades, en los cuales pueden identificarse las neuronas por su conexión con otras, pues éstas no varían a lo largo de los individuos de una especie. Esta propiedad de la estructura de conexión, conocida como **unicidad neuronal**, fue propuesta ya en 1912 por el biólogo alemán R. Goldsmidt, a partir de sus trabajos sobre el sistema nervioso de un gusano primitivo, el parásito intestinal *Ascaris*. Goldsmidt observó que su cerebro estaba formado por varios ganglios, con un total de 162 células, cantidad que no variaba de un individuo a otro, con cada célula ocupando siempre la misma posición.

La obra de Goldsmidt pasó muchos años relegada [Kandel 87], pero en los años setenta dos grupos de investigación de la Facultad de Medicina de Harvard volvieron sobre el tema de la unicidad neuronal (cada uno por su cuenta): M. Otsuka, E. A. Kravitz y D. D. Potter, trabajando con la langosta, y W. T Frazier, I. Kupfermann, R. M. Waziri, R. E Coggeshall y R. Kandel, con el gran caracol marino *Aplysia* (en estos invertebrados superiores existía similar uniformidad, aunque algo menor [Kandel 76]). Inmediatamente se halló constancia de este hecho en diversos otros invertebrados, como sanguijuelas, cangrejos de río, saltamontes, grillos y muchas formas de caracoles [Kandel 79, Bentley 74]. En esta línea, los ANS se relacionarían con los ganglios de los sistemas biológicos, pues también en ellos aparecen circuitos neuronales bien definidos que realizan funciones específicas.

A partir de la constatación de que los invertebrados consiguen sobrevivir en un entorno hostil y cambiante con unos sistemas nerviosos tan simples, surge la idea de emularlos con el uso de ARS, y también la necesidad de unas herramientas adecuadas para su desarrollo, pues los actuales medios de simulación digital o analógica de redes neuronales artificiales permiten hoy llegar a simular sistemas de parecido tamaño al que podemos encontrar en el *Ascaris* o en la *Aplysia*, como se vio en el capítulo 5.

Así, entenderemos que un Sistema de Respuesta Autónoma o ARS es un sistema artificial capaz de responder de forma autónoma ante estímulos exteriores (Figura 10.12). **El objetivo de los ARS y de su entorno de desarrollo IDEA** (*Integrated Development Enviroment for ARS*) es el disponer de un modelo que permita la implementación de ARS adiestrados en términos de reglas y controlados por instintos, y no a través de algoritmos imperativos o sistemas de razonamiento formal. Sistemas de este tipo deberían ser capaces de ejecutar tareas simples, con flexibilidad suficiente para adaptarse a entornos cambiantes o indefinidos. Un claro ejemplo sería un robot móvil para la gestión de un almacén, con instintos y habilidades similares a los de la hormiga, capaz de moverse por el almacén sin tropezar, recoger las cargas y organizarlas, regresando periódicamente a la despensa para alimentarse (recargar sus baterías) [Sanz 93b].

Como muestra la Figura 10.12, donde aparece la estructura general de un ARS, las entradas procedentes del entorno son captadas por los sentidos, que son ANS (conjuntos de neuronas artificiales). La información de los sentidos externos, y también de los internos, junto con los instintos, son procesados por el módulo ANS *voluntad*, que decide las acciones a tomar. Dichas acciones son coordinadas por un ANS final [Beer 90].

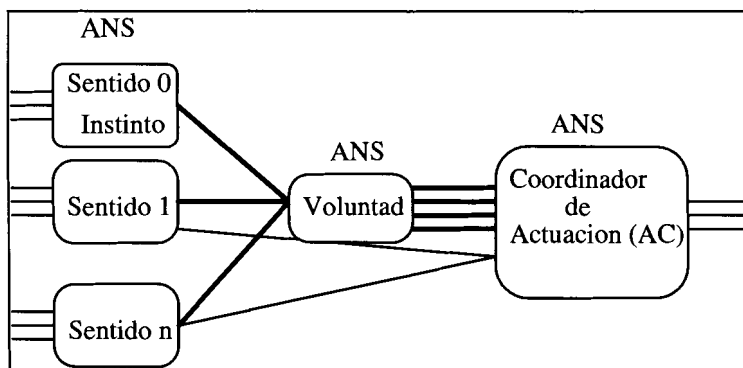


Figura 10.12. Estructura general de los bloques de un ARS

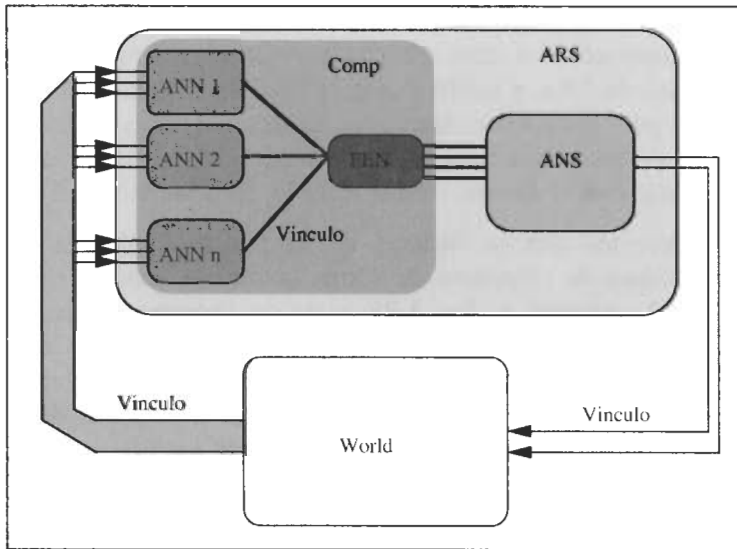


Figura 10.13. Conexiones internas y externas de un ARS desarrollado con el entorno IDEA

Por otro lado, la estructura interna de los sistemas de control basados en ARS que se desarrollan con el entorno IDEA es la siguiente: un objeto *World* modela el exterior del sistema de control; a este objeto se conecta un ARS que modela el sistema de control, el cual contiene todos los ANS de diversos tipos que realizarán el control, e incluye también las conexiones con el exterior, y conexiones internas entre ANS; el objeto ARS puede contener objetos de tipo *Comp* (complejos), que a su vez pueden englobar a otros ANS de forma jerárquica. Recordemos que la conexión entre los diversos tipos de ANS se realiza con el uso de vínculos, objetos de tipo *Vínculo*. Un ejemplo de esta estructura puede verse en la Figura 10.13.

10.4.2 Objeto *Vínculo*

El objeto *Vínculo* se crea para el encapsulado de las interacciones entre los ANS contenidos en el ARS, y de este ARS con el exterior. Dentro de la arquitectura general de los controladores basados en ARS, el sistema de control interactúa con el exterior recibiendo un cierto número de entradas del sistema externo, produciendo salidas que actúan sobre el exterior. Además, este ARS está constituido por diversos ANS, que pueden contener a su vez otros más. Estos ANS a su vez se comunican o interactúan entre sí conectando las salidas de unos a las entradas de los siguientes, conexiones que suponen la compartición de ciertas estructuras de datos o variables. En la implementación de un controlador ARS simulada por software, las conexiones son estructuras de datos que almacenan los valores que los ANS intercambian,

mientras que en una implementación cableada en circuito analógico los vínculos son conexiones físicas, elementales o como buses de conexión entre los diversos subcircuitos del controlador.

El objeto *Vinculo* permite, pues, tratar estas conexiones entre los objetos con independencia de la implementación física del controlador. Será el compilador el que decida la implementación del vínculo, bien como una estructura de datos, cuando se trate de simular el compilador, bien como uno o más nodos del circuito, si se realiza en un circuito electrónico. Este objeto *Vinculo* se ha diseñado para permitir el almacenamiento de datos complejos, de una o más dimensiones; de esta manera, la conexión entre dos *ANS* puede ser un simple valor analógico o todo un conjunto de bits, como un mapa de bits resultado de un sensor de imagen. En la actual versión 1.0 se ha implementado el objeto *Vinculo* como unidimensional y de tipo *float*.

La clase *Vinculo* deriva de *VObjArray*, lo que le permite almacenar no sólo el valor actual del vínculo, sino la evolución de éste a lo largo de una simulación del entorno. Cuando se añade un elemento al vínculo se incrementa un contador que indica número de muestras (*nSample*). Si se ha alcanzado el tamaño de memoria reservado para el objeto *Vinculo*, éste se expande (la función *size()* da el número de elementos que se han reservado, y *nSamples()* devuelve el número de muestras que se han almacenado ya en el vínculo.

Una propiedad importante de la clase *Vinculo* es que un objeto de tipo *Vinculo* asociado a la lista de entradas o salidas de un *ANS* puede conectarse a otro *Vinculo* externo. Esto significa que este objeto *Vinculo* tomará todos los valores de las variables del *Vinculo* al que se conecta, conservando sólo el nombre con el que se definió (que se puede obtener con *GetDefName()*). Esta propiedad de conexión permite que un *ANS* pueda acceder a los datos de un *Vinculo* externo, por el simple hecho de conectar sus *Vinculos* de entrada y de salida a los externos. Esta función la realiza el compilador, que tras hacer una copia del *ANS*, tal como fue definido, lo conecta al *ANS* de tipo *Comp* o *ARS*, conectando cada uno de sus vínculos según el ejemplo:

```
pVincInp = pANS ->getVinculoEntrada(nInp);  
pVincInpOri = pOriANS->getVinculoEntrada(nInp);  
pVincInp->setName(szVincName);  
pVincInp->Connect(pVincInpOri);  
nInp++;
```

El puntero *pANS* apunta al objeto *ANS* que se está conectando, y *pOriANS* al producido por la definición que de este objeto se hizo. En este caso, para la conexión de una entrada de la lista de vínculos de entrada se obtienen punteros a los vínculos a conectar de cada objeto con los punteros *pVincInp* y *pVincInpOri*. La conexión se

realiza con el método *Connect*, que conecta el vínculo apuntado por *pVincInp* al apuntado por *pVincInpOri*.

10.4.3 Objeto *World*

El objetivo de la clase *World* del entorno IDEA es permitir la simulación de las respuestas que el sistema va a producir en función de los estímulos que el controlador ARS produce; también permite la visualización de los resultados intermedios de la simulación, mostrando al usuario cómo evoluciona el sistema. Así pues, el objeto *World* tiene dos tipos de funciones fundamentales, las encargadas de simular la respuesta del sistema externo a controlar, y las que se ocupan de la visualización de los resultados de la simulación.

Dentro del primer tipo de funciones están *Init()* y *Run(integer T,float dT)*. La primera es llamada por el entorno cuando comienza la simulación, y es la que se encarga de iniciar las variables internas del objeto, fijando el estado de partida de la simulación. La segunda es la encargada realmente de la simulación de la respuesta del entorno a los estímulos de control; recibe como parámetros un contador *T*, que le indica el número de iteraciones realizadas, y una variable *dT*, que da la duración temporal del paso actual de simulación. En esta rutina se tiene acceso a los vínculos de entrada y de salida a través de los *VObjArray* que los almacenan. Para un acceso más rápido a los datos se dispone también de vectores con los punteros de los datos de las entradas y las salidas, *ppfEntradas* y *ppfSalidas*.

Las funciones *PintaEscena()* y *NewFrame()* se encargan de la visualización de los resultados de la simulación. La primera es llamada por el entorno al comienzo de una animación, con el objeto de dibujar aquellas partes de la visualización que no se van a modificar a lo largo de los cálculos (por ejemplo, si se visualiza una gráfica cartesiana dibujará los ejes y las leyendas de la gráfica). Al final, esta función llama a *NewFrame()*, la cual dibuja aquellas partes de la animación que varían de una interacción a otra. Para realizar las gráficas el objeto *World* dispone de un puntero a un objeto de la clase *VPort*, que permite dibujar en el área cliente del objeto de la clase *MdiEdit*, que visualiza los resultados de la animación.

10.4.4 Objeto *FEN* (red borrosa equivalente)

El objeto de la clase *FEN* (*Fuzzy Equivalent Network*) es el encapsulado de un controlador borroso equivalente. Recordemos que los controladores ARS están formados por neuronas agrupadas en ANS, conjuntos homogéneos con una función determinada. Los *FEN* suponen una extensión del concepto de ANS, como conjunto de neuronas conectadas entre sí, que poseen un conjunto de entradas y salidas. La utilización de este objeto es independiente de su implementación interna, lo que permite definir los *FEN* en base a reglas de lógica borrosa, con independencia de la implementación real con que se realice. En la versión 1.0 del entorno IDEA la

implementación realizada de los *FEN* tiene la forma de un controlador borroso clásico, que aplica su base de reglas al conjunto de las entradas para obtener, por inferencia y desborrosificación, las salidas. Se han definido en la bibliografía diversos métodos para convertir este tipo de controladores en redes de neuronas con un funcionamiento global equivalente (consúltese el capítulo 9).

La clase *FEN* almacena la base de reglas en dos partes. Por un lado, las premisas de las reglas se almacenan en un objeto *Array* apuntado por *oaPremisas*, el número total de reglas se guarda en *nNumRules*, y la premisa de cada regla se almacena en *oaPremisas* como un vector de enteros, en el que cada elemento apunta al *Array* de los adjetivos de los vínculos de entrada, y tiene tantos índices como vínculos de entrada tenga el *FEN* (lo que permite codificar secuencias de afirmaciones sobre las entradas). Si se aplica a un vínculo un adjetivo dentro de la premisa, éste se codifica con el índice de ese adjetivo, mientras que si no se utiliza en la premisa ningún adjetivo, el índice en *oaPremisas* se hace -1. Por otro lado, los adjetivos definidos en el *FEN* se almacenan en un *VObjArray* llamado *oaListasAdjetivos*, objeto que contiene un *Array* por cada vínculo. A su vez esos *Array* contienen punteros a los adjetivos definidos para ese vínculo.

Las consecuencias de las reglas se almacenan en dos tipos de datos, según el tipo de regla definida para el controlador. En la versión 1.0 del entorno se han implementado los dos tipos de controladores borrosos más habituales. En el caso del basado en reglas tipo Mamdani [Mamdani 74], las consecuencias de las reglas son afirmaciones lingüísticas sobre las salidas:

```
if temperatura is alta then {refrigerar=mucho};
```

Este tipo de controlador, internamente llamado *TIPO_FEN_B*, almacena la estructura de las consecuencias de las reglas en un *Array* llamado *oaConsecuencias*. La codificación en este *Array* es idéntica a las utilizadas para las premisas.

El otro tipo de controlador implementado en esta versión es el tipo Sugeno [Sugeno 85], en el que las consecuencias son funciones lineales de los vínculos de entrada. Internamente este tipo de controlador se denomina *TIPO_FEN_C*. Una regla de este tipo sería:

```
if temperatura is alta then {refrigerar=temperatura*10};
```

En la definición de un *FEN* sólo se admiten reglas de un tipo, y en ambos casos las premisas se componen con el operador *and* borroso. La norma utilizada en esta versión es la MIN, y para la desborrosificación de las reglas de tipo Mamdani se utiliza el método de la media ponderada de los centros.

Para la introducción en el *FEN* de un nuevo valor lingüístico se utiliza la función:

```
boolean AddRegla(int *in IndAdjRegla, float *fCoefCons)
```

que añade el adjetivo *Adj* al vínculo indicado por el índice *nInd*. Una vez añadidos todos los adjetivos al *FEN* se puede comenzar la definición de reglas. Para los *FEN* de *TIPO_FEN_B* se utiliza la función

```
boolean AddRegla(int *nIndAdjRegla, int *nIndAdjSal)
```

y para los *TIPO_FEN_C* se utiliza la función

```
boolean AddRegla(int *nIndAdjRegla, float *fCoefCons)
```

El cálculo de las salidas en función de las entradas se realiza como en cualquier otro ANS con la función *Run*, que llama a *RunFuzzyB* o a *RunFuzzyC* según el tipo de reglas utilizadas. Si se desea aumentar el número de tipos de reglas utilizables, o los métodos de inferencia o desborrosificación a emplear, se habrán de modificar estas funciones.

10.4.5 Objeto ANN

El objeto de la clase *ANN* es el encapsulado de los ANS que implementan una red de neuronas artificiales (*Artifitial Neural Network*) de tipo BP (*BackPropagation*). Se ha seleccionado este tipo de red neuronal por ser el más utilizado en las aplicaciones de control (recordemos que aproximadamente en el 70% de los problemas en los que se utilizan redes neuronales se hace uso de BP). En futuras versiones del entorno se ampliarán los modelos de red neuronal disponibles. Por último, para el entrenamiento de los ANN se hace uso del objeto *TrainingSet* heredado de la clase *ANS*.

10.5 REALIZACIÓN HARDWARE DE SISTEMAS BORROSOS. ACELERADORES

Hasta el momento hemos centrado nuestra atención en las herramientas de desarrollo de sistemas borrosos, consistentes normalmente en un paquete software integrado en el que se incluye un simulador, que permiten el diseño y experimentación con sistemas borrosos desde un computador (a menudo un PC). En esta última sección mostraremos cómo puede llevarse a cabo la implementación final de un sistema de control basado en lógica borrosa, que se incorporará a un sistema real en funcionamiento.

Implementación de sistemas borrosos en microprocesadores convencionales

Como se indica en [Legg 92], una de las principales ventajas de los sistemas de inferencia borrosa para control es la **simplicidad de su implementación en microcontroladores convencionales de 8 bits**. Es de destacar que en el caso de las de redes neuronales ello no suele ser posible, debido a los elevadísimos recursos de cálculo (procesamiento y memoria) que requieren (véase el capítulo 5).

En general, puede afirmarse que aquellos sistemas en los cuales los tiempos de respuesta del controlador sean del orden o mayores de medio segundo, pueden ser realizados programando el sistema de control borroso en microcontroladores convencionales de 8 bits. Una descripción de las técnicas utilizables en este enfoque puede encontrarse en [Adcok 92, Viot 93, Sibigtroth 93a, 93b]. Obviamente, el tiempo de respuesta del controlador borroso dependerá del tiempo necesario para la evaluación de las reglas y la inferencia de las salidas.

Una de las operaciones más críticas a realizar en cuanto a recursos requeridos es la desborrosificación, pues el método del centroide (uno de los más empleados) requiere cierta capacidad de cálculo. Para la aceleración de su ejecución resulta de gran ayuda disponer de instrucciones eficientes de multiplicación y acumulación (MAC), que algunos microcontroladores, o versiones especiales, pueden realizar (como el 68HC11 o el 68HC16 de Motorola). También resultan muy interesantes para estas aplicaciones los DSP (como los TMS320 de Texas Instruments) pues, como ya vimos en el capítulo 5, incluyen un módulo específico para la realización de la operación MAC, de gran importancia en el campo del procesamiento de señal (para el cual se introducen originalmente), pero también para la emulación de redes neuronales y sistemas borrosos. Como ya se ha comentado al principio de este capítulo, muchas herramientas de desarrollo de sistemas borrosos generan código para éstos y otros microprocesadores.

En esta línea, es de destacar que el microcontrolador de Motorola M68HC12 (www.mcu.motsp.com), introducido en 1996 como actualización del clásico 68HC11 (ambos se asientan sobre similares bases, y el software desarrollado para el HC11 puede ejecutarse en un HC12), incluye ya dentro de su juego de instrucciones algunas específicas para facilitar la realización de las operaciones más típicas que deben llevarse a cabo cuando se trata de implementar un sistema borroso mediante un microprocesador.

Como en el caso de los sistemas neuronales artificiales, algunas herramientas de desarrollo de lógica borrosa realizan la aceleración de los cálculos de las simulaciones mediante el empleo de **tarjetas coprocesadoras o emuladores**, adaptables al bus de un computador convencional, que normalmente se basan en microprocesadores RISC de muy elevadas prestaciones. Un ejemplo de ello es la tarjeta DASH!860 (Myriad Solutions), basada en el microprocesador i860 de Intel (potente CPU de 64 bits con arquitectura RISC, que utiliza un reloj de 40MHz); con ella se pueden alcanzar hasta

80 MFLOPS. No obstante, dada la creciente potencia de los computadores estándar (como los PC), este tipo de emuladores está cayendo en desuso.

Implementación en microprocesadores borrosos

Como en el caso de las redes neuronales, para aquellas aplicaciones en las que se requieren tiempos de respuesta más cortos que los que puedan proporcionar un microprocesador de propósito general, se hace uso de **circuitos integrados o microprocesadores específicamente desarrollados para aplicaciones con lógica borrosa**. Entre estos caben destacar NLX230 de Neuralogix, SAE 81C99 y 80C166-S de Siemens, MSM91U112 de OKI, T/FC150 de Toshiba, el FC110 de Togai Infralogic, y el FP-3000 de OMROM. Algunos de ellos están pensados para la realización de tarjetas aceleradoras, pero otros son verdaderos microprocesadores o coprocesadores borrosos, que pueden emplearse en controles específicos.

El **NLX230** de Neuralogix es uno de los microprocesadores específicos para aplicaciones de lógica borrosa más simple. Permite realizar inferencia con reglas basadas en funciones de inclusión triangulares, alcanzando velocidades de 30 MFLIPS (*Mega Fuzzy Logic Inferences Per Second*), a un coste bajo, por trabajar con buses de 8 bits. Neuralogix dispone también de otros integrados para la aceleración de sistemas basados en lógica borrosa, como el NLX110, un comparador borroso de patrones, o el NLX113 y el NLX112, que computan correlaciones de datos a alta velocidad (50 MHz), calculando la distancia de Hamming entre los datos y los patrones de referencia. Estos sistemas pueden utilizarse con entornos específicos basados en tarjetas conectables a PC, como la ADS230, basada en el NLX230, o la ADS110, basada en el NLX110.

El **MSM91U112** de OKI posee una interfaz digital para facilitar su conexión con microprocesadores estándar. Dispone de 8 variables de salida, es posible definir 8 funciones de inclusión para cada una, y puede configurarse en cuatro conjuntos de 32 reglas o en uno único de 128 reglas; por último, cada conjunto de reglas tiene asociada una salida. Alcanza las 25 KFLIPS para un reloj de 10 MHz. [Zakai 93].

El **T/FC150** de Toshiba proporciona una resolución de 10 bits para las 8 entradas y única salida disponibles. Las funciones de inclusión se definen con una resolución de 8 bits, utilizando 17 tipos básicos definidos con 7 parámetros; realiza inferencia min/max y pueden seleccionarse dos métodos de desborrosificación, alcanzando hasta 4.4 MFLIPS. Se dispone de dos tarjetas que utilizan este microcontrolador, la LFZY1 para PC, y la MFZY1, para el MULTIBUS II.

Entre los microprocesadores específicos para aplicaciones borrosas más extendidos se encuentra el **FC110** de Togai Infralogic, que puede ser utilizado como microprocesador autónomo o en conexión con microcontroladores estándar, como los de la familia 68XX de Motorola u 8051 de Intel, y puede ser programado mediante las herramientas de alto nivel de Togai. Su principal característica es que permite el uso de todo tipo de funciones de inclusión, sin limitación de forma ni tamaño. La base de

reglas está contenida en una RAM de 256 bytes incluida en el chip, y los datos y variables se encuentran en otra memoria RAM de 256 bytes, compartida con el microprocesador que hace el papel de *host*. Puede controlar más de 800 reglas, con más de 256 argumentos en cada una, y permite alcanzar los 0.2 MFLIPS. Se dispone también de una gama de tarjetas basadas en este microcontrolador borroso, que están soportadas por las herramientas de desarrollo de Togai, desde la *FC110 Development Module*, hasta tarjetas aceleradoras para los buses AT y VME.

El microprocesador FC110 se encuentra también disponible como macrocelda para la realización de circuitos integrados ASIC, lo que permite su empleo con lógica definida por el usuario, o con otras celdas de microprocesadores estándar integrados en un sólo circuito. Con éstas realizaciones se puede evaluar un sistema de 40 reglas de dos entradas y 1 salida a un ritmo de 800.000 reglas por segundo.

Las grandes compañías del sector electrónico están realizando un gran trabajo en el campo de los sistemas borrosos; así, hemos visto ya el microprocesador borroso de Toshiba, o el nuevo microcontrolador 68HC12 de Motorola, que incluye instrucciones de lógica borrosa. La multinacional Siemens está realizando un enorme esfuerzo, tanto en la aplicación de la lógica borrosa en la industria como en el desarrollo de circuitos borrosos; su microprocesador de 16 bits 80C166-S incluye instrucciones convencionales y borrosas, pudiendo combinar algoritmos borrosos y no borrosos [Reyero 95]. Por su parte, SGS-Thomson ha desarrollado el microprocesador WARP (*Weight Associative Rule Processor*) *fuzzy logic controller*, para aplicaciones de lógica borrosa, que viene acompañado por el sistema de desarrollo FuzzyStudio WARP, que incluye software y tarjeta de desarrollo (la cual contiene un chip borroso WARP, EPROM y circuito programador).

Por último, la empresa japonesa OMROM ha desarrollado el FP-3000, que opera como coprocesador borroso de un *host*, y a partir del cual ha diseñado la tarjeta FB-30AT, conectables al bus de un PC convencional; el sistema de desarrollo FS-10AT, toma como base esta tarjeta, y permite programar el FP-3000. OMROM comercializa también módulos de control borroso para sus **autómatas programables**, como el módulo C200H-FZ001 *Fuzzy Logic Unit*, para los conocidos autómatas de la serie SYSMAC C200H [Reyero 95]. De esta manera se facilita la aplicación de control borroso en tareas de automatización industrial basadas en autómatas o PLC.

Implementación de controladores borrosos y neuro-borrosos en circuitos

En la bibliografía pueden encontrarse una enorme cantidad de realizaciones de controladores borrosos, utilizando todas las técnicas de implementación disponibles, ya discutidas ampliamente en la parte dedicada a redes neuronales.

Así, en [Shenoi 93] aparece el caso de un controlador borroso basado en un microprocesador estándar como el 8031, mientras que en [Sasaki 93] puede verse un ejemplo de realización en circuito integrado específico. En la referencia [Eichfeld 93] se presenta un coprocesador borroso de 8 bits, que permite 256 entradas, 64 salidas y

16384 reglas, alcanzando una velocidad de pico de 7.9 MFLIPS. En [Katashiro 93] se presenta un microprocesador borroso que posee 128 entradas con una resolución de 8 bits, permitiendo evaluar hasta 128 reglas a un ritmo de 114 KFLIPS.

El VY86C570 [Wang 93] de Togai (www.ortech-engr.com/fuzzy/togai.html) incluye en un único chip un núcleo FCA (*Fuzzy Computational Accelerator*) de 12-bit, una memoria de 4K x 12 OCTD (*Observation, Conclusion, and Temporary Data*), y su base de reglas puede almacenar 200 reglas en una memoria compartida (que puede ser accedida desde un microprocesador externo). Este hardware realiza los cálculos con 12 bit de resolución y su consumo es inferior a 250 mW a 20MHz; está disponible en encapsulado PLCC de 68 patillas o en placa para PC. Su tiempo de inferencia es de 70 ms, con método de desborrosificación seleccionable. El número de ejemplos que pueden presentarse es enorme; pero, hablando en términos generales, puede decirse que para lograr un aumento de velocidad debe sacrificarse flexibilidad y precisión. Un análisis de estas relaciones puede encontrarse en [Tanaka 93].

Para la realización de controladores borrosos en circuito integrado se vienen utilizando tanto técnicas digitales como analógicas. El empleo de un enfoque u otro presenta las consabidas ventajas e inconvenientes, ampliamente comentados en el capítulo 5 a la hora de hablar de la implementación de redes neuronales. Entre las realizaciones mediante técnicas digitales, además de los coprocesadores borrosos mostrados antes, podemos citar la expuesta en [Ungering 93], donde se describe el uso de una arquitectura basada en la técnica de modulación de anchura de pulsos, que reduce el área de silicio necesaria en la implementación microelectrónica. En [Gabrielli 99] se describe un rápido procesador borroso de 7 bits de precisión.

Dentro de las técnicas analógicas una de las más empleadas es la basada en espejos de corriente [Patyra 93, Huertas 93]. Se han propuesto también técnicas híbridas; así, en [Ruiz 93] se propone una interfaz digital para la definición de funciones de inclusión programables, y un procesamiento interno basado en un diseño en modo corriente. En [Dualibe 00] se presenta otro procesador borroso analógico.

Las realizaciones citadas hacen uso de tecnología CMOS, pero en [Ishizuka 93] se propone el empleo de tecnología bipolar para lograr una mayor velocidad. También es posible la utilización de otras tecnologías para la implementación de sistemas de lógica borrosa, como los dispositivos lógicos programables o PLD (*Programmable Logic Devices*) [Leong 93].

Existen también realizaciones que tratan de unir los campos de redes neuronales y lógica borrosa. Por ejemplo, en [Song Han 93] se muestra el empleo de la resistencia variable de los CMOS, para la realización tanto de borrosificadores como de sinapsis en redes neuronales, para construir sistemas híbridos neuro-borrosos. Dentro de los sistemas neuro-borrosos cabe destacar el entorno **Neufuz, de National Semiconductors**, cuyo software de desarrollo hace uso de técnicas neuronales para realizar aprendizaje a partir de patrones, que luego es plasmado en un sistema borroso convencional; el entorno de trabajo acaba generando el equivalente del sistema borroso en código ensamblador para el microcontrolador de propósito general COP8

de National Semiconductors, o bien código C estándar. Una descripción de las bases de este sistema puede encontrarse en [Khan 93a, 93b]

Todos los sistemas descritos hasta el momento se basan en el modelo convencional de sistema de inferencia borrosa o FIS, descrito en capítulos anteriores. No obstante, existen otros esquemas posibles, como las máquinas borrosas de estados finitos; en [Grantner 93] puede verse un análisis de estas técnicas para su implementación microelectrónica.

Finalmente, indicaremos que en las actas de congresos especializados es donde puede encontrarse la información más reciente sobre realizaciones de hardware *fuzzy*. Un ejemplo es [ISCAS 2000], donde se presenta, entre otros, un controlador borroso de 5.26 MFLIPS [Dualibe 00]. Las actas de los congresos anuales *IEEE International Fuzzy Systems Conference FUZZ-IEEE* (1997 en Barcelona, 1998 en Anchorage, 1999 en Seúl y en San Antonio en el 2000) son fundamentales en este sentido. Todas estas actas las comercializa, como es habitual, la IEEE Press (www.ieee.org).

A modo de resumen final podemos comentar que en gran medida los sistemas borrosos se implementan como un software que se ejecuta sobre un PC, como un código que se graba en memoria ROM para ser ejecutado por microcontroladores estándar (como los de Siemens, Motorola, PIC, National, etc.) o casi estándar (como el 68HC12), o bien como código para módulos especiales de autómatas programables convencionales (Siemens, OMROM). Debido a los limitados recursos de cálculo que los sistemas borrosos requieren solamente se es necesario hardware específico en casos muy concretos [Gabrielli 99].

CAPÍTULO 11

APLICACIONES DE LOS SISTEMAS BORROSOS

En este último capítulo mostraremos algunas aplicaciones de los sistemas borrosos, tanto académicas como reales, relacionadas con procesamiento de datos y control. Como podemos deducir de lo estudiado hasta el momento, el campo que claramente cuenta con mayor número de aplicaciones basadas en lógica borrosa es el de control, existiendo numerosas aplicaciones en funcionamiento en la industria.

Para finalizar, expondremos el caso del desarrollo del robot autónomo controlado con reglas borrosas bautizado como PILAR, diseñado y realizado en la Universidad de Zaragoza.

11.1 INTRODUCCIÓN. *SOFT COMPUTING*, O IMITANDO A LA NATURALEZA

Los nuevos modelos de procesamiento y control estudiados en este libro, redes neuronales (*neural networks*), lógica borrosa (*fuzzy logic*) y algoritmos genéticos (*genetic algorithms*), junto con algunos otros de relativa novedad, se engloban en la denominada *soft computing* [Zadeh 94, Jang 97] o **inteligencia computacional** [Marks 93, IEEE 99] (véase el Prólogo de L. A. Zadeh), que tienen en común el constituir paradigmas de procesamiento muy diferentes a la convencional *hard computing*, basada en computadores von Neumann (serie) y la separación de hardware y software. Estas nuevas técnicas se inspiran en las soluciones que la naturaleza ha encontrado durante millones de años de evolución a numerosos problemas tecnológicos que involucran el tratamiento de cantidades masivas de información, redundante, imprecisa y ruidosa, problemas a los que en la actualidad se enfrenta el ser humano (visión, habla, control en ambiente natural). No obstante, debe quedar perfectamente claro que estas nuevas técnicas no vienen a suplantarse a las más

tradicionales, sino más bien a completarlas en aquellos problemas donde menos eficacia proporcionan.

Es fácil deducir de lo expuesto a lo largo del libro que tanto las redes neuronales artificiales como los sistemas borrosos, constituyen en la actualidad áreas de investigación y desarrollo (I+D) muy activas, no sólo desde un punto de vista académico, sino comercial e industrial. Recordemos que grandes compañías del sector electrónico e informático, como Siemens, Motorola, SGS-Thomson, Toshiba, National Semiconductors, OMROM, etc., trabajan en redes neuronales y sistemas borrosos. No obstante, debemos adelantar que, por los motivos que expondremos, los sistemas borrosos cuentan con un mayor número de aplicaciones reales actualmente en explotación.

En la línea anterior, mención especial merece el interés del mundo empresarial japonés en la aplicación de la lógica borrosa en la industria y en aparatos de consumo. Recordemos que, como se comentó en el capítulo 7, en los Estados Unidos y Europa solamente se empezó a dar importancia a la lógica borrosa cuando desde Japón empezó a llegar abundante información sobre numerosas aplicaciones prácticas de esta técnica, desarrolladas y comercializadas por compañías japonesas.

11.2 INTERÉS DEL EMPLEO DE LA LÓGICA BORROSA. FUSIÓN DE TECNOLOGÍAS

En los capítulos dedicados a las redes neuronales (ANS) estudiamos sus áreas de aplicación. En este sentido podemos decir que los sistemas basados en lógica borrosa pueden ser aplicados prácticamente a los mismos problemas que las redes neuronales. De esta manera, resultarán especialmente interesantes para los problemas no lineales o no bien definidos. Recordemos que según vimos en el capítulo 9, los sistemas basados en lógica borrosa, también como los neuronales, pueden modelar cualquier proceso no lineal, y aprender de los datos haciendo uso de determinados algoritmos de aprendizaje, a veces tomados de otros campos, como las redes neuronales o los algoritmos genéticos.

El otro gran denominador común de ambas técnicas es su orientación hacia el tratamiento de tareas que involucran el procesamiento de cantidades masivas de información, de tipo redundante, imprecisa y con ruido, que aparecen en problemas tecnológicos cruciales a los que en la actualidad se enfrenta el ser humano.

No obstante, frente a las características comunes citadas, existen también importantes diferencias. Por ejemplo, los sistemas basados en lógica borrosa permiten utilizar el conocimiento que los expertos disponen sobre un tema, bien directamente, bien como punto de partida para una optimización automática. En el caso de las redes neuronales, esto resulta más difícil o menos directo. Puede decirse que lógica borrosa permite formalizar tanto el conocimiento ambiguo de un experto como el sentido común, de una forma tecnológicamente realizable.

Una importante ventaja de los sistemas borrosos es que, gracias a la simplicidad de los cálculos requeridos (sumas y comparaciones, fundamentalmente), normalmente pueden realizarse en sistemas baratos y rápidos, con lo que pueden implementarse en sistemas específicos (por ejemplo, para el control inteligente de un horno microondas, o de un sistema de frenado ABS). Éste es uno de los motivos fundamentales del hecho constatado de la existencia en la actualidad de muchas más aplicaciones prácticas en funcionamiento basadas en lógica borrosa que en redes neuronales.

No obstante las ventajas e inconvenientes que cada enfoque puede presentar, el futuro apunta en la dirección de combinar distintas técnicas para resolver problemas complejos. Los problemas tecnológicos de mundo real resultan en general de gran complejidad, por lo que para su resolución conviene que sean divididos en partes más simples, de manera que cada una pueda ser resuelta mediante la técnica más indicada, procedente del campo de la estadística, procesamiento de señal, reconocimiento de patrones, redes neuronales, sistemas borrosos, algoritmos genéticos, o cualquier otra.

Debemos recordar en esta línea de razonamiento una de las conclusiones presentadas en el capítulo 6 dedicado a las aplicaciones de las redes neuronales: no hay panacea, no existen soluciones simples a problemas complicados. Estas nuevas técnicas emergentes aportan características sumamente interesantes, pero por sí solas no resolverán todos nuestros problemas tecnológicos, sino que contribuirán (y de forma importante) en determinados aspectos, pero otros seguirán siendo mejor abordados mediante técnicas *tradicionales*. En este sentido, no conviene forzar la aplicación de cierta nueva técnica a determinado problema simplemente por su novedad, sino que ello debe realizarse solamente en aras de conseguir un mayor rendimiento o sencillez de implementación.

Este último es otro de los aspectos destacables de los sistemas borrosos, su relativa sencillez de aplicación. *A veces mediante un sistema borroso no se logra un rendimiento superior (ni inferior) al que se alcanza con un enfoque clásico, pero el tiempo de desarrollo es con frecuencia inferior, y el sistema final resulta más barato.*

Para concluir, y retomando la línea de la fusión de tecnologías, merece la pena recordar el intenso trabajo que se desarrolla en sistemas neuro-borrosos. Como estudiamos en el capítulo 9, los sistemas borrosos pueden aprovechar la capacidad de aprendizaje de la red neuronal para optimizar su funcionamiento. Por otro lado, la equivalencia que se establece entre ciertos modelos neuronales y borrosos [Jang 92-95, 97, Rayneri 99] puede ser empleada para extraer las reglas que una red neuronal (por ejemplo, tipo BP) ha encontrado en el entrenamiento, eliminando uno de los grandes problemas clásicamente achacado a los sistemas neuronales artificiales, su operación en forma de caja negra. Por todo ello, la combinación de redes neuronales y sistemas borrosos es un campo de intenso trabajo en la actualidad [Cox 93a, Khan 93b, Wang 94, Jang 92-97, Benítez 97, Reyneri 99, Figueiredo 99, Li 00] del que ambas técnicas se benefician.

11.3 ALGUNAS APLICACIONES DE LOS SISTEMAS BORROSOS

En esta sección enumeraremos algunos ejemplos de aplicaciones prácticas de los sistemas borrosos. La relación no pretende ser exhaustiva, sino simplemente ilustrativa de los diferentes tipos de problemas que en la actualidad se abordan mediante el uso de las técnicas procedentes de la lógica borrosa. Remitimos al lector interesado en profundizar en este tema al libro de C. von Altrock [Altrock 95], en el que se exponen numerosas aplicaciones de muy diferentes campos, algunas de ellas ampliamente comentadas. Por otro lado, el libro de R. Reyero y C. F. Nicolás [Reyero 95] ilustra muy bien el tipo de aplicaciones de control industrial que actualmente se desarrollan con lógica borrosa. En los libros [Jang 97, Passino 98] se exponen en detalle ejemplos de resolución de problemas con sistemas borrosos. Finalmente, en el CD-ROM que acompaña este libro y en las direcciones que se incluyen en el apéndice B puede encontrarse abundante información y ejemplos sobre estas técnicas.

Haciendo un poco de historia, la primera aplicación práctica operativa de la lógica borrosa la desarrolló E. Mamdani en Europa [Mamdani 74, 83], realizando el control borroso de un sistema de vapor de una planta industrial. Otra de las más clásicas quizás sea la de Smidth y otros, que en 1980 aplican técnicas de lógica borrosa al control de hornos rotativos en una cementera [García Cerezo 91]. El **control en planta industrial** sigue siendo hoy en día uno de los campos de aplicación más destacables.

Los sistemas basados en lógica borrosa se vienen utilizando en aplicaciones procedentes de diversos campos. Así, en el **área médica** se emplea en diagnóstico, acupuntura [Zhang 93], análisis de ritmos cardíacos [Borshevich 93], o de la arterioestenosis coronaria [Cios 91]. Dentro del **apoyo a la toma de decisiones**, otra de las grandes áreas de aplicación de estos sistemas, se han utilizado, por ejemplo, en la búsqueda de caminos críticos en la ejecución de proyectos [Nasution 94], y asesoramiento a la inversión [Cox 93b].

Sin duda, el principal campo de aplicación de la lógica borrosa es el de **control**, a partir del empleo de las expresiones de la lógica borrosa para formular reglas orientadas al control de sistemas [Brubaker 92, Nass 92, Passino 98]. Dichos sistemas de control borroso pueden considerarse una extensión de los sistemas expertos, pero superando los problemas que éstos presentan para el razonamiento en tiempo real, ocasionados por la explosión exponencial de las necesidades de cálculo requerido para un análisis lógico completo de las amplias bases de reglas que éstos manejan.

En el campo del **control de sistemas en tiempo real** destaca el control de un helicóptero por órdenes pronunciadas de viva voz [Sugeno 93a, 93b], y el control con derrapaje controlado de un modelo de coche de carreras de [Altrock 93]. Dentro del **sector del automóvil** existen gran número de patentes sobre sistemas de frenado, y cambios de marcha automáticos [King 89].

El área de los **aparatos de consumo** es otra de las más destacables, hasta el punto de que no es raro encontrarse con propaganda del tipo *Incluye inteligencia artificial fuzzy*, en escaparates o en catálogos (por ejemplo, en los de las cámaras de video de Hitachi). En el sector de los electrodomésticos se han diseñado un buen número de aplicaciones neuro-borrosas como lavadoras (Matsushita, Hitachi, Siemens, AEG), tostadoras de pan, controles de calefacción y aire acondicionado [Hellenthal 93].

Para el **control de maquinaria** destaca el ya clásico control de frenado del metro de Sendai (Japón), realizado por Hitachi, y que opera desde julio de 1987. Se han aplicado sistemas borrosos en el control de una máquina de perforación de túneles [Nikkei 89], y en el control de ascensores (Mitsubishi-Elec., Hitachi, Fuji Tech) y grúas para contenedores (Hitachi). Se han aplicado también al **procesado de imágenes** [Krishanapuram 92, Furukawa 93], y **reconocimiento de caracteres**; por ejemplo, en [Gowan 93] se muestra un sistema que reconoce los números de los cheques bancarios, para lo cual hace uso de un sensor CCD de 64×1 píxeles, y un microcontrolador M68HC11, en el que el sistema borroso se materializa por programa.

11.4 ROBOTS MÓVILES Y NAVEGACIÓN AUTÓNOMA

En esta última sección expondremos las bases de la aplicación de lógica borrosa a robots móviles y navegación autónoma, y describiremos una aplicación desarrollada en la Universidad de Zaragoza por uno de los autores [Sanz 96a].

Los robots móviles pueden caracterizarse en términos de su movilidad, autonomía, inteligencia y conocimiento previo del entorno. El sistema de navegación del robot incluye tareas de planificación, percepción y control de movimientos. La primera comprende la planificación de la misión, planificación de la ruta, planificación de la trayectoria [Zhao 94], y las funciones específicas para evitar obstáculos imprevistos, no considerados en las planificaciones anteriores [Seok 93, Lee Jang 93].

La percepción se ocupa de tareas relacionadas con la detección de obstáculos, construcción de mapas, modelado del entorno y reconocimiento de objetos. Por último, el control de movimientos convierte las órdenes de control en acciones, utilizando para ello la diferencia entre la posición deseada y la posición actual del robot (ésta última, a menudo estimada). Se incluyen también tareas de monitorización de la ejecución.

Existen diversas posibilidades de aplicación de técnicas de control borroso. La mayoría de los trabajos existentes parten de una identificación del escenario, con extracción de características del entorno y obtención de valores de variables de situación del robot (distancia a la pared, distancia a la entrada de estacionamiento, etc.) [Ayache 89, González 92]. Las reglas de control borroso relacionan valores

cualitativos de estas variables con las acciones de control, típicamente, el ángulo de dirección del vehículo.

En algunos casos, en vez de utilizarse directamente la variable de control en los consecuentes de las reglas, se emplea una combinación lineal de los valores de las variables de situación, empleando unos coeficientes, que se calculan mediante la identificación de las actuaciones de un operador humano trabajando en simulación. En este punto, conviene mencionar el interés de la aplicación de técnicas de aprendizaje, tal como se ha estudiado en capítulos previos. En cualquier caso, una vez obtenidos los valores inferidos por cada regla, la acción final de control resulta de una suma ponderada normalizada de los resultados de cada regla.

Por otra parte, cabe mencionar que el conjunto de reglas puede descomponerse en subconjuntos relativos a operaciones diferentes (conducir hacia delante, girar a la izquierda en intersección...), o bien a maniobras diferentes (aparcar en paralelo, aparcar en batería) [Gachet 93]. En [Ruspini 93] se utiliza esta técnica para permitir que un robot evolucione en el entorno de una oficina.

Las aplicaciones mencionadas en los párrafos anteriores cabe considerarlas como de control borroso directo. Otra posibilidad consiste en aplicar las técnicas de razonamiento aproximado para seguir una trayectoria previamente definida. Por ejemplo, considérese un sistema de navegación que realiza sucesivamente los siguientes pasos: planificación de la ruta, generación de un camino con buenas propiedades cinemáticas y, finalmente, control de seguimiento de la trayectoria. Para este último paso existen a su vez diferentes métodos. En efecto, la generación del ángulo de dirección (curvatura) puede realizarse mediante métodos geométricos, o considerando también la dinámica.

De acuerdo con el esquema del párrafo anterior, la aplicación de métodos de control borroso puede realizarse directamente (sustituyendo los métodos geométricos y dinámicos), o en combinación con ellos. De esta forma, es posible compensar errores de seguimiento mediante reglas que tienen como antecedentes la desviación de ángulo y la variación de la desviación del ángulo, y como consecuentes la modificación de la dirección para compensar el error [Kawaji 93].

Asimismo, otra estrategia muy interesante consiste en seleccionar mediante técnicas de razonamiento aproximado los parámetros del control, que típicamente se eligen mediante consideraciones heurísticas. Por ejemplo, los métodos geométricos o dinámicos de seguimiento de trayectorias tienen como parámetro importante el número de puntos, o la distancia sobre el camino que pretende seguir (*look ahead*). Este parámetro puede elegirse mediante técnicas de razonamiento aproximado, teniendo en cuenta las características del camino, la velocidad, y la posición del vehículo con respecto al camino.

Los comentarios anteriores se refieren fundamentalmente al control de la dirección. Existe también el problema del control de la velocidad del vehículo. Aunque hay un evidente acoplamiento cinemático y dinámico con el anterior, se ha

demostrado que en diversas aplicaciones prácticas pueden realizarse desacoplamientos. En cualquier caso, el problema de planificación y control de velocidad es notablemente complejo, y suele realizarse con criterios heurísticos. Por consiguiente, la aplicación de técnicas de razonamiento aproximado resulta particularmente interesante [Kikuchi.93].

Si bien la mayor parte de los robot autónomos se basan en topologías clásicas, haciendo uso de ruedas, también se han estudiado otras posibilidades. Así, para el acceso a terrenos agrestes resulta de interés el uso de robots con patas. Éstos son capaces de moverse también en entornos de oficinas e industrias, por ser capaces de subir y bajar escaleras. En los robot con patas se ha de prestar atención al control del movimiento de las patas para conseguir un desplazamiento uniforme. Estos estudios se han basado habitualmente en el movimiento de los animales, como se puede comprobar en [Raibert 86].

El robot autónomo PILAR

El control basado en lógica borrosa de uno de estos robots autónomos puede verse en [Sanz 93], y la implementación de un prototipo de robot de seis patas se expone con cierto detalle en [Sanz 96a].

El diseño del control de este robot autónomo se ha realizado de manera totalmente jerárquica, haciendo uso del modelo ARS descrito en el capítulo anterior. El control de mayor nivel, el de navegación, implementa dos comportamientos diferentes posibles:

- a) Seguir a un objeto y detenerse a una distancia predefinida.
- b) Movimiento libre en un entorno genérico, evitando obstáculos, y tendiendo al área libre más amplia.

En el contexto de este control jerárquico, se propone además un método para la eliminación de bloqueos en el planeamiento de la trayectoria [Sanz 96a].

El prototipo de robot autónomo de seis patas ha sido bautizado PILAR (*Programmable Intelligent Legged Autonomous Robot*) (véase la Figura 11.1), estando ideado para que pueda moverse en cualquier ambiente, terreno natural, volcanes, minas, plantas nucleares, fábricas, etc.

Cada una de sus patas es controlada mediante dos servomotores eléctricos. El corazón de PILAR es un microcontrolador 68HC11; todo el sistema es alimentado mediante una batería de Níquel-Cadmio de 9.6 V. En la cabeza móvil incluye un sensor de ultrasonidos, mediante el cual mide la distancia a los obstáculos; aunque su capacidad de giro es de $\pm 80^\circ$, solamente se implementan $\pm 45^\circ$. PILAR actúa en función de los obstáculos detectados, y de las reglas borrosas (programadas en el 68HC11), que codifican su comportamiento.

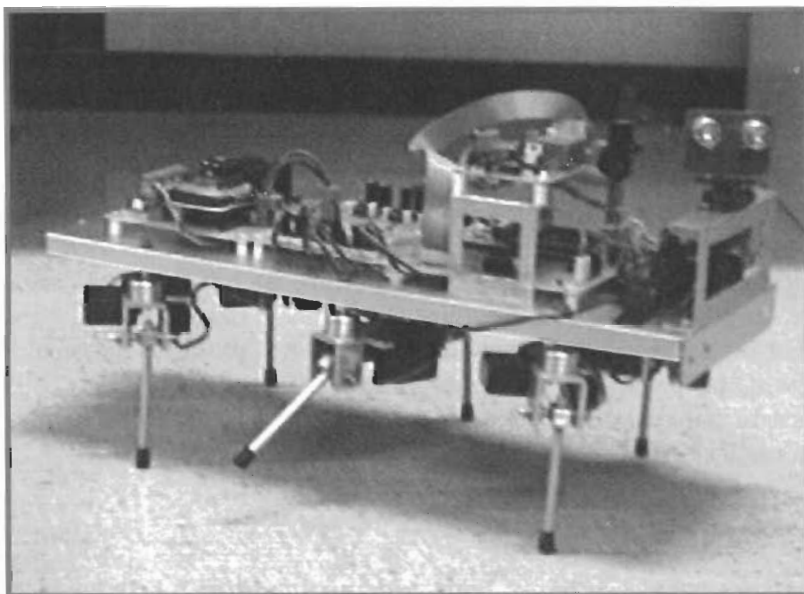


Figura 11.1. Robot autónomo hexápodo PILAR, desarrollado en la Universidad de Zaragoza

El sistema se ha diseñado modularmente, de modo que pueda ser fácilmente expandible y reconfigurable. Absolutamente autónomo, puede caminar en terrenos accidentados. De la misma manera que el modelo ARS está inspirado en el sistema nervioso de los insectos, el patrón de coordinación del movimiento de las seis patas de PILAR está basado en el de los insectos, presentando su misma secuencia temporal.

11.5 CONCLUSIÓN FINAL

La lógica borrosa tiene una historia corta, pero un rápido crecimiento debido a su capacidad de resolver problemas relacionados con la incertidumbre de la información o del conocimiento de los expertos. Además, proporciona un método formal para la expresión del conocimiento en forma entendible por los humanos. Estas cualidades le aseguran un amplio campo de aplicabilidad, y un alto interés para las aplicaciones industriales, presentes y futuras. Queremos concluir esta parte dedicada a la lógica borrosa con un principio enunciado por Terano recientemente *Cuanto más humano deba ser un sistema, más lógica borrosa contendrá.*

A modo de resumen final de nuestro trabajo, querríamos expresar la conclusión fundamental que no nos cansamos de repetir una y otra vez: no existe panacea. Para resolver problemas complejos, como los pertenecientes a un entorno industrial o a muchos otros entornos (economía y finanzas, medicina, etc.), con frecuencia la solución óptima consiste en una inteligente combinación de diversas técnicas, cada una de las cuales debe ser aplicada a aquel aspecto parcial del problema que mejor se adecue. En este sentido, tanto las redes neuronales como los sistemas borrosos se aplicarán especialmente allá donde los comportamientos no lineales sean importantes. Cuando no se posea un modelo suficientemente bueno, pero sí se disponga de un amplio conjunto de ejemplos (casos experimentales), el empleo de una red neuronal puede resultar útil, y podemos dejar que mediante un proceso de entrenamiento ella misma encuentre el modelo o características más relevantes. Sin embargo, cuando se disponga de un conjunto de reglas proporcionadas por los expertos en un determinado tema, el empleo de sistemas basados en lógica borrosa puede ser tremendamente útil. No obstante, de la combinación de ambas técnicas, y de éstas con otras más clásicas (estadísticas, tratamiento de señal, etc.), son esperables mejores resultados todavía.

APÉNDICE A

CONTENIDO DEL CD-ROM

En el CD-ROM que acompaña este texto se incluyen las **versiones de demostración** de dos de los entornos de trabajo más potentes para redes neuronales y sistemas borrosos: **NeuroSolutions** (versión 3.022), de la empresa NeuroDimension Inc. (Gainesville, Florida, EE.UU.; www.nd.com), y **fuzzyTECH** (versión 5.31c), de Inform GmbH (Aachen, Alemania; www.fuzzytech.com). En ambos casos se ofrecen versiones para **computador PC compatible con sistema operativo MS-Windows**.

Es muy importante tener en cuenta que ambos programas se ofrecen tal cual, por cortesía de ambas empresas, ni los autores del libro ni la editorial RA-MA asumen ningún tipo de responsabilidad sobre su uso o consecuencias.

Ante cualquier duda sobre el funcionamiento de ambos programas, por favor, póngase en contacto directamente con dichas empresas (www.nd.com y www.fuzzytech.com, respectivamente).

Nosotros tan sólo le proporcionamos las versiones de evaluación de dos programas que pensamos que pueden resultar muy útiles, pero nada más. Si desea adquirir las versiones plenamente operativas de ambos programas, información adicional, consejos sobre su uso, resolución de problemas que puedan surgir, etc., también deberá ponerse en contacto directamente con estas empresas (recuerde, www.nd.com y www.fuzzytech.com).

En el CD-ROM se encuentran las instrucciones de instalación, así como la explicación del funcionamiento de ambos programas, además de información adicional. Para su lectura basta con que tenga instalado en su computador PC compatible cualquier navegador habitual: ejecute desde el programa *Explorador* de MS-Windows (Win9x) el fichero **default.htm** del CD-ROM, simplemente pulsando con el ratón sobre su icono (si se tiene activada la propiedad de reproducción automática de CD se ejecutará automáticamente).

En el CD-ROM se incluye también el **entorno académico IDEA** y sus fuentes, como ejemplo de implementación de sistemas basados en lógica borrosa. Este entorno ha sido desarrollado bajo la dirección de Alfredo Sanz Molina; ante cualquier duda o problema relacionado con el entorno, por favor, póngase directamente en contacto con él (asmolina@posta.unizar.es).

Finalmente, indicaremos que en el CD pueden encontrarse también algunos artículos publicados por los autores (para profundización en algunos temas), así como las direcciones de páginas web que se proporcionan en el apéndice B.

APÉNDICE B

RECURSOS EN INTERNET

Internet es una fuente prácticamente inagotable de recursos de todo tipo. No es nuestra intención realizar una lista exhaustiva de direcciones Internet; simplemente, queremos proporcionar una serie de lugares que nos han resultado muy útiles, y desde los cuales se puede iniciar una búsqueda que llevará sin duda a muchos otros sitios interesantes. En estas páginas web pueden encontrarse artículos, cursos, manuales, informes, asociaciones, congresos, software, hardware y ejemplos de aplicaciones.

El lector debe ser consciente de que Internet está en permanente cambio, por lo que advertimos que si no puede acceder a alguna de las direcciones citadas, es probable que haya desaparecido o haya sido cambiada. Lo único que podemos garantizar es que las direcciones que se adjuntan eran plenamente operativas a fecha de septiembre del 2000.

Por otro lado, trataremos de mantener información actualizada sobre este libro y los temas aquí tratados en las siguientes páginas web:

<http://www.5campus.com/libroredes.html>

<http://www.cps.unizar.es/~te>

Téngase en cuenta que la actualización de los contenidos de estas páginas web estará siempre en función de nuestra (limitada) disponibilidad de tiempo (rogamos la comprensión del lector en este punto).

Links de redes neuronales

<ftp://ftp.sas.com/pub/neural/FAQ.html>

<http://www.cs.cmu.edu/groups>

FAQ (*Frequently Asked Questions*) de redes neuronales (actualizado) y FAQ de todo tipo (μ P, μ C, inteligencia artificial, redes neuronales, lógica borrosa, robótica, etc.).

<http://www.it.uom.gr/pdp/digital.htm>

Biblioteca Digital del PDP Lab de la Universidad de Macedonia (Grecia). Incluye múltiples recursos sobre redes neuronales, sistemas borrosos, computación evolutiva, vida artificial, etc. Muy interesante.

<http://www.msm.ele.tue.nl/research/neural/>

Redes neuronales en la Universidad Tecnológica de Eindhoven (múltiples recursos y conexiones). Muy interesante.

<http://www.ewh.ieee.org/tc/nnc/>

<http://atc.ugr.es/~bprieto/SRIG/ieee.html#home>

IEEE Neural Network Council y Grupo español de redes neuronales (Spanish RIG). Incluyen información sobre revistas, congresos, grupos de investigación y conexiones sobre redes neuronales, lógica *fuzzy* y algoritmos genéticos.

<http://www.emsl.pnl.gov:2080/proj/neuron/neural/>

Redes neuronales en el Pacific Northwest National Laboratory, Richland, Washington (recursos, hardware, software, aplicaciones, etc.).

<http://www.cis.hut.fi/research/>

Redes neuronales en la Universidad Tecnológica de Helsinki (incluye la web del grupo de Teuvo Kohonen).

<http://www.particle.kth.se/~lindsey/HardwareNNWCourse/home.html>

Curso de redes neuronales en hardware, por C.S. Lindsey, Royal Institute of Technology, Estocolmo (Suecia)

<http://www.cotec.es/cas/index.html>

Fundación COTEC, donde puede encontrarse el informe COTEC sobre oportunidades tecnológicas de las redes neuronales.

<http://www.mbfys.kun.nl/snn/Research/siena/>

La web del proyecto SIENA, con los resultados del informe.

<http://www.kcl.ac.uk/neuronet/>

NeuroNet, la red europea de excelencia en redes neuronales.

<http://www.nd.com/>

Página de NeuroDimension Inc., empresa de Florida desarrolladora del programa NeuroSolutions

Links de lógica borrosa

<http://www.fuzzytech.com/>

Web de fuzzyTECH, la empresa desarrolladora del simulador incluido en el CD. Contiene información de lógica borrosa y redes neuronales; aplicaciones ejemplos y demos.

<http://www.aptronix.com/fide/fide.htm>

Sitio de FIDE. Información y recursos de lógica borrosa y redes neuronales, con ejemplos y programas de demostración.

<http://www.ortech-engr.com/fuzzy/TilShell.html>

Página web de TILShell. Contiene información sobre lógica borrosa y referencias a otras páginas y asociaciones.

<http://www.mathworks.com/products/fuzzylogic/>

Sitio del *fuzzy logic toolbox* de Matlab. Incluye demos y manual.

http://www-europe.mathworks.com/access/helpdesk/help/pdf_doc/fuzzy/fuzzy_tb.pdf

Introducción a la lógica borrosa y manual del *fuzzy logic toolbox* de Matlab.

<http://www.abo.fi/~rfuller/ifsa.html>

Sitio de la IFSA (*International Fuzzy Systems Association*), la principal asociación de lógica borrosa. Contiene gran cantidad de recursos y referencias a otras páginas y asociaciones.

<http://www-cgi.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/areas/fuzzy/0.html>

Departamento de informática de la Universidad Carnegie Mellon. Contiene gran cantidad de recursos, FAQ y referencias a otras páginas y asociaciones.

<http://decsai.ugr.es/~herrera/fl-ga.html>

Recopilación del DECSAI de la Universidad de Granada, con referencias de lógica borrosa y algoritmos genéticos.

<http://www.eufit.org/>

Es el principal congreso europeo de lógica borrosa, y por ello una buena referencia de artículos.

<http://www.ang-physik.uni-kiel.de/pfister/fuzzy/hoefi/eufit94/aachen.html>

Diseño de sistemas jerárquicos utilizando algoritmos genéticos desordenados, y su aplicación al diseño de vehículos autónomos.

<http://www.motorola.com/pub/SPS/MCU/fuzzy/>

<http://www.motorola.com/pub/SPS/MCU/mcu11>

En las páginas de Motorola puede encontrarse información y, herramientas para lógica *fuzzy*.

<http://buffy.eecs.berkeley.edu/Research/CS/AI/>

Algunas aplicaciones de la universidad de Berkeley del departamento de Lotfi Zadeh.

BIBLIOGRAFÍA

- [Abu-Mostafa 86] Abu-Mostafa, Y.S. *Neural networks for computing?* En: *Neural networks for computing*, J. S. Denker (edt.), AIP Proceedings, 151, pp. 1-7, 1986.
- [Abu-Mostafa 87] Abu-Mostafa, Y. S., Psaltis, D. *Computadoras óptico-neuronales*. Investigación y Ciencia, 58-65, mayo 1987.
- [Abu-Mostafa 89] Abu-Mostafa, Y. S. *Information theory, complexity and neural networks*. IEEE Communications Mag., 27, 11, 25-28, 1989.
- [Adcok 92] Adcok, T. A. *Implementation of Fuzzy Logic*. Texas Instruments, 1992.
- [AI Expert 90] *Loudspeaker diagnosis*. AI Expert, 5, 6, pp. 71, junio 1990.
- [Alahakoon 00] Alahakoon, D., Halgamuge, S. K., Srinivasan, B. *Dynamic self-organizing maps with controlled growth for knowledge discovery*. IEEE Transactions on Neural Networks, 11, 3, 601-614, 2000
- [Alegre 91] Alegre, M. C. *Inteligencia artificial en el control de procesos controladores borrosos*. Mundo Electrónico, 42-49, febrero, 1991.
- [Alkon 89] Alkon, D. L. *Almacenamiento de memoria y sistemas neurales*. Investigación y Ciencia, 14-23, septiembre, 1989.
- [Altrock 93] Altrock, C von, Krause, B. *Fuzzy logic and neurofuzzy technologies in embedded automotive applications*. Fuzzy Logic '93, pp. A113-9, San Francisco, California 1993.
- [Altrock 95] Altrock, C. von. *Fuzzy Logic and Neurofuzzy Applications Explained*. Prentice-Hall, 1995.

- [Amit 89] Amit, D. J. *Modelling the Brain Function: The World of Attractor Neural Networks*. Cambridge University Press, 1.989.
- [Aracil 89] Aracil, J., Ollero, A., García Cerezo, A. *Stability indices for the global analysis of expert control systems*. IEEE Trans. on System, Man and Cybernetics, vol. 19, nº 5, septiembre 1989.
- [Arbib 98] Arbib, M. A. (ed.). *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1998.
- [Armstrong 93] Armstrong, J. R., Gray, F.G. *Structured Logic Design with VHDL*. Prentice-Hall, 1993.
- [Atiya 99] Atiya, A. F., El-Shoura, S. M., Shaheen, S. I., El-Sherif, M. *A comparison between neural network forecasting techniques - Case study: River flow forecasting*. IEEE Transactions on Neural Networks, 10, 2, pp. 402-, 2000.
- [Atlas 89] Atlas, L. E., Suzuki, Y. *Digital systems for artificial neural networks*. IEEE Circuits and Devices Mag., 20-24, nov. 1989.
- [Avellana 95] Avellana Tarrats, N. *Neurocomputador de propósito general: Asignación óptima de recursos*. Tesis Doctoral, Universidad Autónoma de Barcelona, Diciembre 1995.
- [Ayache 89] Ayache, N., Faugeras, O.D. *Maintaining representations of the environment of a mobile robot*. pp. 205-220, 1989.
- [Baker 89] Baker, T., Hammerstrom, D. *Characterization of artificial neural networks algorithms*. Int. Symp. on Circuits and Systems ISCAS'89, pp. 89-81, 1989.
- [Barnden 98] Barnden, J. A. *Artificial intelligence and neural networks*. En [Arbib 98], parte III, pp. 98-102, 1998.
- [Barron 93] Barron, J. J. *Putting Fuzzy Logic into focus: When dealing with ambiguous data, desktop fuzzy-logic applications deliver precise results*. EDN, pp. 111-118, abril 1993.
- [Barto 83] Barto, A. G., Sutton, R., Anderson, C. W. *Neuron-like adaptive elements that can solve difficult learning control problems*. IEEE Trans. on Systems, Man and Cybernetics, 13, 5, 835-846, 1983.
- [Bauer 92] Bauer, H. U., Pawelzik, K.R. *Quantifying the neighborhood preservation of self-organizing feature maps*. IEEE Trans. on Neural Networks, 3, 4, 570-579, 1992.
- [Baum 89] Baum, E. B., Haussler, D. *What size net gives valid generalization?* Neural Computation, 1, 151-160, 1989.

- [Beer 90] Beer, R. D. *Intelligence as Adaptive Behaviour, An Experiment in Computational Neuroethology*. Academic Press, 1990.
- [Benítez 97] Benítez, J., Castro, J., Requena, I. *Are neural networks black boxes?*. IEEE Tran. on Neural Networks, 8, 1156-1164, 1997.
- [Bentley 74] Bentley, D., Ronald, R. *The neurobiology of cricket song*. Scientific American, 231, 2, pp. 34-44, 1974.
- [Bibyk 90] Bibyk, S., Ismail, M. *Neural network building blocks for analog MOS VLSI*. En Toumazou, C. y otros, *Analogue IC Design: The Current-Mode Approach*, P. Peregrinus Ltd., 597-615, 1990.
- [Bishop 94] Bishop, C. M. *Neural networks and their applications*. Rev. Sci. Instrum., 65, 6, pp. 1803-1832, 1994.
- [Bishop 95] Bishop, C. M. *Neural networks for Patter Recognition*. Oxford University Press, 1995.
- [Blayo 91] Blayo, F., Demartines, P. *Data analysis: How to compare Kohonen neural networks to other techniques?*. Proc. Int. Workshop on Artificial Neural Networks (IWANN91), 469-476, Granada 1991.
- [Borshevich 93] Borshevich, V., Mustyatsa, A. V., Oleinik, W. L. *Fuzzy spectral analysis of heart's rhythms*. 5th IFSA World Congress, pp. 561-563, 1993.
- [Botros 94] Botros, N. M., Abdul-Aziz, M. *Hardware implementation of an artificial neural network using field programmable gate arrays*. IEEE Trans. on Industrial Electronics, vol. 41, n° 6, 665-7, 1994.
- [Bout 89] Bout, D. E. van der, Miller, T. K. *A digital architecture employing stochasticism for the simulation of Hopfield neural nets*. IEEE Trans. on Circuits and Systems, 36, 5, 732-8, 1989.
- [Boverie 93] Boverie, S., Demaya, B., Lequellec, J. M. *Performance evaluation of fuzzy control through an international benchmark*. Fifth IFSA World Congress, pp. 941-944, 1993.
- [Braae 79] Braae, M., Rutherford, D. A. *Selection of parameters for a fuzzy logic controller*. Fuzzy Set and Systems, 1979.
- [Branko 91] Branko, S. *Neural and Intelligent Systems Integration; Fifth and Sixth Generation Integrated Reasoning Information System*. Wiley-Interscience Publication, pp. 664, 1991.
- [Brodie 78] Brodie, S. E., Knight, B. W., Ratliff, F. *The response of the Limulus retina to moving stimuli: A prediction by Fourier synthesis*. Journal of General Physiology, 72, pp. 129-154, 162-166, 1978.

- [Brubaker 92] Brubaker, D., Cedric, S. *Fuzzy-logic system solves control problem*. EDN, vol. 18, pp. 121-127, junio 1992.
- [Brubaker 93] Brubaker, D. I. *Everything you always wanted to know about fuzzy logic*. EDN, pp. 103-106, marzo 1993.
- [Bruck 90] Bruck, J. *On the convergence properties of the Hopfield model*. Proc. of the IEEE, pp. 1579-1585, octubre 1990.
- [Bruijn 93] Bruijn, C. de, van der Wal, A. J. *Fine-Tuning of a PID controller by fuzzy logic*. Fuzzy Logic '93, A211-5, San Francisco, 1993.
- [Buckley 93] Buckley, J. J., Yoichi H. *Fuzzy controllers and fuzzy expert systems as hybrid neural nets*. 5th IFSA World Congress, 70-72, 1993.
- [Burr 93] Burr, J. B. *Digital neurochip design*. En: [Przytula 93], 1993
- [Cabestany 93] Cabestany, J., Castillo, F., Moreno, M. *Redes Neuronales: realizaciones en CI*. Mundo Electrónico, 239, pp. 24-31, 1993.
- [Cárdenas 93] Cárdenas, E., Castillo, J. C., Cerdón. *Estudio comparativo de sistemas de inferencia y métodos de defuzzyficación aplicados al control difuso del péndulo invertido*. III Congreso Español de Tecnologías y Lógica Fuzzy, 259-266, Santiago de Compostela, 1993.
- [Carpenter 88] Carpenter, G., Grossberg, S. *The ART of Adaptive Pattern Recognition by a self-organizing neural network*. IEEE Computer, 77-88, marzo 1988.
- [Casale 93] Casale, J. F., Watterson, J. W. *Journal of Forensic Sciences*. March, 1993.
- [Castro 98] Castro, A., J. Pino, A. López Molinero, J. Pérez, B. Martín del Brío. *Roman glazed and Islamic ceramics characterization by artificial neural networks of their chemical and mineralogical analysis*. 26th Computer Applications in Archaeology CAA'98, Barcelona, 24-28, marzo 1998
- [Chapman 66] Chapman, R. A. *The repetitive responses of isolated axons from the crab Carcinus Maenas*. Journal of Exp. Biology, 45, 1966.
- [Cherkasski 91] Cherkasski, V., Lari-Najafi, H. *Constrained topological mapping for nonparametric regression analysis*. Neural Networks, 4, 27-40, 1991.
- [Cherkasski 94] Cherkasski, V., Friedman, J. H., Wechsler, H. *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*. NATO ASI Series F, vol. 136, Springer-Verlag, 1994.

- [Churchland 90] Churchland, P. M., Smith Churchland, P. *¿Podría pensar una máquina?* Investigación y Ciencia, 18-24, marzo, 1990.
- [Cios 91] Cios, K. J., Shin, I., Goodenday, L. S. *Using fuzzy set to diagnose coronary artery stenosis*. Computer, pp. 57-63, marzo 1991.
- [Clarkson 95] Clarkson, M. *Eyes, Ears and Brains on a chip*. Byte, febrero, pp. 91-96, 1995.
- [Cohen 83] Cohen, M., Grossberg, S. *Absolute stability of global pattern formation and parallel memory storage by competitive neural networks*. IEEE Trans. Syst., Man and Cyb., 13, 815-825, 1983.
- [Combettes 93] Combettes, P. L. *The foundations of set theoretic estimation*. Proceedings of the IEEE, vol. 81, nº 2, febrero 1993.
- [Connections 93] *Connections. News Letter of the IEEE Neural Networks Council*. septiembre, 1993.
- [Conner 93] Conner, D. *Fuzzy-logic control system*. EDN, pp. 77-88, 1993.
- [Conrad 92] Conrad, M. *Molecular computing paradigms*. IEEE Computer, pp. 6-9, diciembre 1992.
- [COTEC 98] Fundación COTEC para la innovación tecnológica. *Redes Neuronales*. Documentos COTEC sobre oportunidades tecnológicas, nº 13, 1998 (<http://www.cotec.es/cas/index.html>)
- [Cottrell 87] Cottrell, M., Fort, J. C. *Etude d'un algorithme d'auto-organisation*. Annales de l'Institut Henri Poincaré, 23, 1, pp. 1-20, 1987.
- [Cottrell 94] Cottrell, M., Fort, J. C., Pagès, G. *Two or three things that we know about the Kohonen algorithm*. Proc. of the European Symp. on Artificial Neural Networks ESANN'94, abril 1994.
- [Cox 92] Cox, C. E., Ekkerhard W. *GANGLION-A fast field-programmable gate array implementation of a connectionist classifier*. IEEE Journal of Solid-State Circuits, vol. 27, nº 3, 288-299, 1992.
- [Cox 93a] Cox, E. *Adaptive fuzzy systems*. IEEE Spectrum, pp. 27-31, February, 1993a.
- [Cox 93b] Cox, E. *A Model-free trainable system for the analysis of financial time-series data*. Fuzzy Logic '93, San Francisco, California, pp. A124-5, 1993b.
- [Croall 92] Croall, I. F., Mason, J. P (eds.). *Industrial Applications of Neural Networks*. Springer-Verlag, 1992.
- [Daihee 94] Daihee, Park, Kandel, A., Langholz, G. *Genetic-based new fuzzy reasoning models with application to fuzzy control*. Proceedings of the IEEE, vol. 24, nº. 1, pp. 39-47, enero 1994.

- [DARPA 88] DARPA *Neural Networks Study, October 1987-February 1988*. AFCEA International Press, 1988.
- [DeFelipe 99] DeFelipe, J. *Cajal y la corteza cerebral*. Investigación y Ciencia, pp. 36-38, octubre 1999.
- [Delbrück 00] Delbrück, T. *Silicon retina for autofocus..* Actas del IEEE International Symposium on Circuits and Systems ISCAS'2000. Ginebra, Suiza, pp. IV-393-7. IEEE Press, 2000.
- [Demartines 92] Demartines, P., Blayo, F. *Kohonen self-organizing maps: Is the normalization necessary?*. Complex Systems, 6, 105-123, 1992.
- [Demartines 94] Demartines, P. *Analyse de données par réseaux de neurones auto-organisants* PhD Thesis, Institut National Polytechnique de Grenoble, 1994.
- [Demuth 98] Demuth, H., Beale, M. *MatLab Neural Networks Toolbox User's Guide*. The MathWorks Inc., 1998. Disponible en <http://www.mathworks.com>
- [Denker 87] Denker, J. S., Schwartz, D., Wittner, B., Solla, S., Howard, R., Jackel, L., Hopfield, J. J. *Large automatic learning, rule extraction and generalization*. Complex Systems, 1, 877-922, 1987.
- [Deschamps 94] Deschamps, J. P. *Diseño de Circuitos Integrados de Aplicación Específica*. Editorial Paraninfo, 1994.
- [DeSieno 88] DeSieno, D. *Adding a conscience to competitive learning*. Proc. Int. Conf. on Neural Networks, pp. 117-124. IEEE Press, 1988.
- [Diederich 87] Diederich, S., Oppen, M. *Learning patterns in spin-glass networks by local learning rules*. Physical Review Letters, 58, 9, 949-952, 1987.
- [Domany 91] Domany, E., van Hemmen, J. L., Schulten, K. (eds.). *Models of Neural Networks*. Springer-Verlag, 1991
- [Driankov 93] Driankov, D., Hellendoom, H., Reinfrank, M. *An Introduction to Fuzzy Control*. Springer-Verlag, 1993.
- [Dualibe 00] Dualibe, C., Jespers, P., Verleysen, M. *A 5.26 Mflips Programmable Analogue Fuzzy Logic Controller in a Standard CMOS 2.4μ Technology*. Actas del IEEE Int. Symp. on Circuits and Systems ISCAS'2000. Ginebra, Suiza. IEEE Press, 2000
- [Duranton 96] Duranton, M. *L-neuro 2.3: a VLSI for image processing and neural networks*. 5th International Conference on Microelectronics for Neural Networks and Fuzzy Systems, MicroNeuro'96. Lausanne, Suiza, pp. 157-160, 1996.

- [Eberhart 90] Eberhart, R. C., Dobbins, R. W. *Neural Network PC Tools. A Practical Guide*. Academic Press, San Diego, 1.990
- [Eichfeld 93] Eichfeld, H., Künemund, T. *A fuzzy controller chip for complex real-time applications*. 5th IFSA World Congress, 1390-3, 1993.
- [Enciclopedia 82] *Gran Enciclopedia Aragonesa*. Voz Ramón y Cajal, Santiago. Tomo X 1982 (<http://www.aragob.es/pre/cido/cajal.htm>).
- [Erwin 92a] Erwin, E., Obermayer, K., Schulten, K. *Self-organizing maps: Stationary states, metastability and convergence rate*. Biological Cybernetics, 67, 35-45, 1992.
- [Erwin 92b] Erwin, E., Obermayer, K., Schulten, K. *Self-organizing maps: Ordering, convergence properties and energy functions*. Biological Cybernetics, 67, 47-55, 1992.
- [Fahlman 88] Fahlman, S. *An empirical study of learning speed in back-propagation networks*. Tech. Rep. CMU-CS-88-162, junio 1988.
- [Fahlman 90] Fahlman, S., Lebiere, C. *The Cascade Correlation learning architecture*. Technical Report CMU-CS-90-100, Carnegie-Mellon University, 1990.
- [Fang 00] Fang., W. C. *A smart vision system-on-a-chip design based on programmable neural processor integrated with active pixel sensor*. IEEE International Symposium on Circuits and Systems ISCAS'2000. Ginebra, Suiza, pp. II.128-131. IEEE Press, 2000.
- [Farhat 89] Farhat, N. H. *Optoelectronic neural networks and learning machines*. IEEE Circuits and Devices Mag., 32-41, Sept. 1989.
- [Farmer 90] Farmer, J. D. *A Rosetta Stone for Connectionism*. Physica D, 42, pp. 153-157, 1990.
- [Favata 91] Favata, F., Walker, R. *A study of the application of Kohonen-type neural networks to the travelling salesman problem*. Biological Cybernetics, 64, 463-468, 1991.
- [Ferguson 92] Ferguson, R. B. *Chemical process optimization utilizing neural network systems*. SICHEM '92, Seoul, Korea. Ed. Erlbaum, Hillsdale, N.J., 1992.
- [Ferrán 91] Ferrán, E. A., Ferrara, P. *Topological maps of protein sequences*. Biological Cybernetics, 65, 451-458, 1991.
- [Fiesler 94] Fiesler, E. *Neural network clasification and formalization*. En: *Computer Standards and Interfaces*, vol. 16, special issue on Neural Networks Standards, J. Fulcher (edt.), Elsevier, 1994.

- [Figueiredo 99] Figueiredo, M., Gomide, F. *Design of fuzzy systems using neural networks*. IEEE Trans. on Neural Networks, 10, 4, 815-827, 1999.
- [Flam 92] Flam, F. *Neural nets: A new way to catch elusive particles*. Science, Vol. 256, pp. 1282-1283, 1992.
- [Flexer 95] Flexer, A. *Connectionists and statisticians, friends or foes?*. Proc. Int. Work. on Artificial Neural Networks, IWANN95, pp. 454-461, Torremolinos (España), junio, 1995.
- [Fogelman 98] Fogelman-Soulié, F. *Applications of neural networks*. En [Arbib 98], parte III, pp. 94-98, 1998.
- [Fort 88] Fort, J. C. *Solving a combinatorial problem via self-organization process: An application of the Kohonen Algorithm to the TSP*. Biological Cybernetics, 59, 33-40, 1988.
- [Freedman 94] Freedman, D. H. *A romance blossoms between gray matter and silicon*. Science, vol. 265, 889-890, 1994.
- [Freeman 92] Freeman, F. A., Skapura, D.M. *Neural Networks: Algorithms, Applications and Programming Techniques*. Addison-Wesley, 1992. (*Redes Neuronales. Algoritmos, Aplicaciones y Técnicas de Programación*. Addison-Wesley/Díaz de Santos, 1993).
- [Fritzke 93] Fritzke, B. *Kohonen feature maps and growing cell structures- A performance comparison*. En: *Neural Information Processing Systems 5*, C.L. Giles, S.J. Hanson, J. D. Cowan (eds.), Morgan Kaufmann, San Mateo, CA, 1993.
- [Fritzke 95] Fritzke, B. *Growing grid: A self-organizing network with constant neighbourhood and adaptive strength*. Neural Processing Letters, 2, 5, 9-13, 1995.
- [Fu 90] Fu, L. M. *Analysis of the dimensionality of neural networks for pattern recognition*. Pattern Recognition, 23, 10, 1131-40, 1990.
- [Fukuda 93] Fukuda, T., Ishigami, H., Shibata, T. *Structure optimization of fuzzy neural network by genetic algorithm*. Fifth IFSA World Congress, pp. 964-967, 1993.
- [Fukushima 75] Fukushima, K. *Cognitron: A self-organizing multilayered neural network*. Biological Cybernetics, 20, 121-136, 1975.
- [Fukushima 80] Fukushima, K. *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*. Biological Cybernetics, 36, 193-202, 1980.
- [Fukushima 83] Fukushima, K., Miyake, S., Ito, T. *Neocognitron: A neural network model for a mechanism visual of pattern recognition*. IEEE Trans. on System, Man and Cyb., 13, 826-834, 1983.

- [Funahashi 89] Funahashi, K. I. *On the approximate realization of continuous mappings by neural networks*. Neural Networks, 2, 183-192, 1989.
- [Fuochi 92] Fuochi, A. *Neural networks: No zealots yet but progress being made*. Comput. Can. 18, 2, 16, 1992.
- [Furuhashi 93] Furuhashi, T., Hasegawa, T., Horikawa, S-I. *An adaptive fuzzy controller using fuzzy neural networks*. Fifth IFSA World Congress, pp. 769-772, 1993.
- [Furukawa 93] Furukawa, M., Yamakawa, T. *The advanced method to optimize cross-detecting lines for a fuzzy neuron*. 5th IFSA World Congress, 66-69, 1993.
- [Gabrielli 99] Gabrielli, A., Gandolfi, E. *A fast digital fuzzy processor*. IEEE Micro, Vol. 9, nº 1, 1999.
- [Gachet 93] Gachet, D., Salichs, M. A., Puente, E. A. *Experiments with a distributed neural network controller for an autonomous mobil robot*. Fifth IFSA World Congress, 1993.
- [Gallant 93] Gallant, S. I. *Neural Network Learning and Expert Systems*. MIT Press, 1993.
- [García 95] García Villares, J. L., Blasco, J., Martín del Brío, B., Domínguez, J. A., Barquillas, J., Ramírez, I., Medrano, N. *Short-term electric load-forecasting using ANN. Part II: Multilayer perceptron for hourly electric-demand forecasting*. 14th IASTED Int. Conf. on Modelling, Identification and Control, Igsl, Austria, febrero 1995.
- [García Cerezo 87] García Cerezo, A. *Aplicaciones del Razonamiento Aproximado en el Control y Supervisión de Procesos*. Tesis Doctoral, Universidad de Santiago de Compostela, 1987.
- [García Cerezo 91] García Cerezo, A. *Aplicaciones actuales de la lógica borrosa*. Automática y Control, vol. 216, pp. 113-119, septiembre 1991.
- [Gardner 88] Gardner, E. *The space of interactions in neural networks models*. Journal of Physics A, 21, 257, 270, 1988.
- [Gedeon 95] Gedeon, T. D., Wong, P. M., Harris, D. *Balancing the bias and variance: Network topology and pattern set reduction techniques*. Proc. Int. Work. on Artificial Neural Networks, IWANN95, pp. 550-8, Torremolinos (España), junio, 1995.
- [Gerousis 00] Gerousis, C., y otros. *Modeling nanoelectronic CNN cells: CMOS, SETs and QCAs*. IEEE Int. Symposium on Circuits and Systems ISCAS'2000. Ginebra, Suiza, pp. I-274. IEEE Press, 2000.
- [Geschwind 79] Geschwind, N. *Specializations of the human brain*. Scientific American, 158-168, septiembre 1979.

- [Girosi 93] Girosi, F. *Regularization theory, radial basic functions and networks*. En [NATO 93].
- [Glorennec 91] Glorennec, P. Y. *Adaptative fuzzy control*. IFSA World Congress, pp. 33-36, 1991.
- [Goldberg 89] Goldberg, D. E., Korb, B., Deb, K. *Messy genetic algorithms: Motivation, analysis, and first result*. Complex Systems, vol. 3, pp. 493-530, 1989.
- [González 92] Gonzalez, J., Hurtado, P., Ollero, A. *Construcción de mapas locales en entornos no-estructurados utilizando un scanner láser radial*. Mundo Electrónico, pp. 9, 1992.
- [Goser 90] Goser, K., Ramacher, U., Rückert, U. (eds). *Proc of the 1st Int. Conf. Microelectronics for Neuronal Networks*, University of Dortmund, junio, 1990.
- [Gowan 93] Gowan, W. A. *Optical character recognition using fuzzy Logic*. Fuzzy Logic'93, pp. M333-19, San Francisco, California, 1993.
- [Graf 88] Graf, H. P., Jackel, L. D., Hubbard, W. E. *VLSI implementation of a neural network model*. IEEE Computer, marzo, 41-49, 1988.
- [Graf 89] Graf, H. P., Jackel, L. D. *Analog electronic neural networks circuits*. IEEE Circuits and Devices Mag., 44-49, 55, julio 1989.
- [Graf 94] Graf, H. P., Reyneri, L. M. (eds). *Proc of the 4th Int. Conf. on Microelectronics for Neuronal Networks and Fuzzy Systems, MICRONEURO94*, Turin, Italy, IEEE Press, septiembre, 1994
- [Graham 88] Graham, B. P., Newell, R. B. *Fuzzy identification and control of a liquid level ring*. Fuzzy Sets and Systems, 26, 255-273, 1988.
- [Granado 99] Granado, B., Lacassagne, L., Garda, P. *Can general purpose microprocessors simulate neural networks in real-time?* Actas del International Work-Conference on Artificial and Natural Neural Networks, IWANN'99, Alicante (España), Junio 1999. LNCS Vol. 1606 y 1607, pp. 21-29, Springer-Verlag, 1999.
- [Grantner 93] Grantner, J., Patyra, M. *VLSI Implementation of fuzzy logic, finite state machines*. Fifth IFSA World Congress, 781-4, 1993.
- [Grossberg 76a] Grossberg, S. *Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors*. Biological Cybernetics, 23, 121-134, 1976.
- [Grossberg 76b] Grossberg, S. *Adaptive pattern classification and universal recoding: II. Feedback, expectation, olfaction, illusions*. Biological Cybernetics, 23, 187-202, 1976.

- [Grossberg 82] Grossberg, S. *Studies of Mind and Brain*. Ed. Reidel, Boston, MA, 1982.
- [Grossberg 87] Grossberg, S. *The Adaptive Brain*. North-Holland, 1987.
- [Gutfreund 92] Gutfreund, H., Toulouse, G. *The physics of neural networks*. Tech. Rep., 1992.
- [Hall 92] Hall, C. *Neural net technology: Ready for prime time?* IEEE Expert, pp. 2-4, Diciembre, 1992.
- [Hammerstrom 90] Hammerstrom, D. *A highly parallel digital architecture for neural network emulation*. En: *VLSI for Artificial Intelligence*, Oxford Univ. (UK), 1990.
- [Hammerstrom 91] Hammerstrom, D., Nguyen, N. *An implementation of Kohonen's self-organizing map on the Adaptive Solutions neurocomputer*. En: Kohonen, T. et al (eds.) *Artificial Neural Networks* (Proc. of ICANN'91), Vol. I, pp 712-720, Nort-Holland, 1991.
- [Hammerstrom 93a] Hammerstrom, D. *Neural Networks at work*. IEEE Spectrum, pp. 26-32, junio 1993.
- [Hammerstrom 93b] Hammerstrom, D. *Working with Neural Networks*. IEEE Spectrum, pp. 46-53, julio 1993.
- [Hartman 90] Hartman, E., Keeler, J. D., Kowalski, J. M. *Layered neural networks with Gaussian hidden units as universal approximators*. Neural Computation, 2, 2, 210-215, 1990.
- [Hayashi 91] Hayashi, S. *Autotuning fuzzy PI controller*. IFSA World Congress, Vol Engineering, pp. 41-44, 1991.
- [Haykin 99] Haykin, S. *Neural Networks. A Comprehensive Foundation*. 2ª edición. Prentice-Hall, 1994, 1999.
- [Hebb 49] Hebb, D. *The Organization of Behaviour*. Wiley, 1949.
- [Hecht-Nielsen 87] Hecht-Nielsen, R. *Kolmogorov's mapping neural networks existence theorem*. Proc. Int. Conf. on Neural Networks, III, pp. 11-13, 1987.
- [Hecht-Nielsen 90] Hecht-Nielsen, R. *Neurocomputing*. Addison Wesley, 1990.
- [Heim 91] Heim, P., Hochet, B., Vittoz, E. *Generation of learning neighbourhood in Kohonen feature maps by means of simple nonlinear network*. Electronics Letters, vol. 27, nº 3, 1991.
- [Hellenthal 93] Hellenthal, B. *Neurofuzzy technologies in european home appliances*. Fuzzy Logic '93, A111-6, San Francisco (EEUU) 1993.

- [Hèrault 94] Hèrault, J., Jutten, C. *La memoria de las redes neuromiméticas*. Mundo Científico, 150, 14, 884-891, 1994.
- [Hernández 96] Hernández, S., Medrano, N., Martín del Brío, B. *Implementing a reconfigurable neural coprocessor by means of FPGAs*. XI Sem. Diseño de Circuitos Integrados y Sistemas DCIS'96. Sitges, 1996.
- [Herrera 93a] Herrera, F., Lozano, M., Verdegay, J. L. *Tuning Fuzzy Logic Controllers by Genetic Algorithms*. Informe Técnico, Dept. de Ciencias de la Computación e IA, Universidad de Granada, 1993.
- [Herrera 93b] Herrera, F., Lozano, M., Verdegay, J. L. *Un algoritmo genético para el ajuste de controladores difusos*. III Congreso Español de Tecnologías y Lógica Fuzzy, 251-258, Santiago, 1993.
- [Herrera 96] Herrera, F., Verdegay, J. L. *Genetic Algorithms and Soft Computing*. Physica-Verlang, 1996.
- [Hertz 91] Hertz, J., Krogh, A., Palmer, R.. *Introduction to the theory of Neural Computation*. Addison-Wesley, 1991.
- [Heskes 93] Heskes, T. M., Kappen, B. *Error potentials for self-organization*. IEEE Int. Conf. on Neural Networks, 1219-1223, 1993.
- [Hinton 86] Hinton, G. E, Sejnowski, T. J. *Learning and relearning in Boltzmann Machines*. En: [Rumelhart 86a], pp. 282-317, 1986.
- [Hodgin 52] Hodgin, A. L., Huxley, A. F. *A quantitative description of the membrane current and its application to conduction and excitation in nerve*. Journal of Physiology. 117. 500-544, 1952.
- [Höhfeld 93] Höhfeld, H., Schürmann. *The roles of neural networks and fuzzy logic in process optimization*. Siemens Review, fall, 9-13, 1993.
- [Holland 75] Holland, J. H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.
- [Holland 92] Holland, J. H. *Adaptation in Natural and Artificial Systems*. (2ª ed.) MIT Press, Cambridge, MA, 1992.
- [Hollstien 71] Hollstien, R. B. *Artificial Genetic Adaptation in Computer Control Systems*. Doctoral dis., Univ. of Michigan, *Disertations Abstracts Intenational*, 32(3), 1971, 1510B, Microfilm nº 71-23, 773.
- [Hopcroft 84] Hopcroft, J. E. *Máquinas de Turing*. Investigación y Ciencia, 1984.
- [Hopfield 82] Hopfield, J. J. *Neural networks and physical systems with emergent collective computational abilities*. Proc. of the National Academy of Sciences, 79, pp. 2554-2558, 1982.

- [Hopfield 84] Hopfield, J. J. *Neurons with graded response have collective computational properties like those of two-state neurons*. Proc. of the National Academy of Sciences, 81, pp. 3088-3092, 1984.
- [Hopfield 85] Hopfield, J. J., Tank, D. W. *Neural computation of decisions in optimization problems*. Biological Cybern., 52, 141-152, 1985.
- [Hopfield 86] Hopfield, J. J., Tank, D. W. *Computing with neural circuits: A model*. Science, vol. 233, pp. 625-633, 1986.
- [Horgan 94] Horgan, J. *Marvin L. Minsky: el genio de la inteligencia artificial*. Investigación y Ciencia, 28-29, febrero, 1994.
- [Horikawa 90] Horikawa, S-I. *A fuzzy controller using neural network and its capability to learn expert's control rules*. Proc. of IIZUKA'90, Fukuoka, 1990.
- [Hornik 89] Hornik, K., Stichcombe, M., White, H. *Multilayer feedforward networks are universal approximators*. Neural Networks, 2, 359-366, 1989.
- [Hrycej 92] Hrycej, T. *Modular Learning in Neural Networks*. John Wiley and Sons, 1992.
- [Huertas 93] Huertas, J. L., Sánchez-Solano, S., Baturone, I. *Building blocks for current-mode implementation of VLSI Fuzzy micro-controllers*. Fifth IFSA World Congress, pp. 929-932, 1993.
- [Hush 92a] Hush, D. R, Horne, B. G. *An overview of neural networks. Part I: Static Networks*. Informática y Automática, 25, 1, 19-36, 1992.
- [Hush 92b] Hush, D. R, Horne, B. G. *An overview of neural networks. Part II: Dynamic Networks*. Informática y Automática, 25, 2, 17-32, 1992.
- [Hush 93] Hush, D. R, Horne, B. G. *Progress in supervised neural networks. What's new since Lippmann?*. IEEE Signal Proc. Mag., 8-38, enero 1993.
- [Ichihashi 91] Ichihashi, H. *Iterative fuzzy modelling and hierarchical network*. IFSA World Congress, 49-52, 1991.
- [IEEE 00] Número especial sobre *Neural Networks for Data Mining and Knowledge Discovery*, de la revista IEEE Transactions on Neural Networks, mayo 2000.
- [IEEE 92] IEEE Computer, número sobre *Molecular Computing*, noviembre 1992.
- [IEEE 96] Número especial *Toward an Artificial Eye* de la revista IEEE Spectrum Magazine. Editado por el IEEE, mayo, 1996.

- [IEEE 97] Número especial sobre *Everyday Applications of Neural Networks* de la revista IEEE Tran. of Neural Networks. IEEE, julio, 1997.
- [IEEE 99] Número especial sobre *Computational Intelligence* de la revista Proceedings of the IEEE. Editado por el IEEE, septiembre 1999.
- [IEEE 99b] Número especial sobre *Vapnik-Chervonenkis Learning Theory and It's Applications*, IEEE Transactions on Neural Networks, 1999.
- [IEEE 99c] Número especial sobre *Pulse Coupled Neural Networks* de la revista IEEE Transactions on Neural Networks, mayo 1999.
- [Ienne 93] Ienne, P. *Architectures for Neurocomputers: Review and performance evaluation*. Technical Report. Swiss Federal Institute of Technology, Laussane, enero 1993.
- [Iokibe 93] Iokibe, T., Sakawa, M. *Electric energy demand supervisory and control methods using fuzzy logic*. Fifth IFSA World Congress, 1378-1381, 1993.
- [ISCAS 2000] Actas del 2000 *IEEE International Symposium on Circuits and Systems ISCAS'2000*. Ginebra, Suiza. IEEE Press, mayo 2000.
- [Ishizuka 93] Ishizuka, O., Masuda, T., Tang, Z. *A high-speed fuzzy processor using bipolar technology*. 5th IFSA World Congress, 933-6, 1993.
- [Jabri 96] Jabri, M. A. y otros. *Adaptive Analog VLSI Neural Systems*. Chapman & Hall, 1996.
- [James 98] James, J. V. *Tracking the right clues with exploratory data analysis*. IEEE Spectrum, pp. 58-65, julio 1998.
- [Jang 92] Jang J. S. R. *ANFIS: Adactative-Network-Based Fuzzy Inference System*. IEEE Trans. Syst., Man and Cybern., 23 665-685, 1992.
- [Jang 93a] Jang, J. S. R. Sun, C. T. *Functional equivalence between radial basic function networks and fuzzy inference systems*. IEEE Trans. Neural Networks, vol. 4, n° 1, enero 1993.
- [Jang 93b] Jang, J. S. R. *Self-learning fuzzy controllers based on Temporal Back-Propagation*. Department of Computer Science and Artificial Intelligence, Berkeley, 1993.
- [Jang 95] Jang, J.S.R. *Neuro-fuzzy modeling and control*. Proc. of the IEEE, march 1995.
- [Jang 97] Jang, J.S.R., C.T. Sun, E. Mizutani. *Neuro-Fuzzy and Soft Computing*. Prentice-Hall, 1997.
- [Jihong 93a] Jihong, Lee. *A fuzzy controller using fuzzy relations on input variables*. Fifth IFSA World Congress, pp. 895-898, 1993.

- [Jihong 93b] Jihong, Lee, Chae Seog. *Two supplementary methods for PI-type fuzzy logic controllers*. 5th IFSA World Congress, 891-4, 1993.
- [Jutten 95] Jutten, C. *Learning in evolutive neural architectures: An ill-posed problem?* Proc. Int. Work. on Artificial Neural Networks, IWANN95, pp. 361-374, Torremolinos (España), junio 1995.
- [Kandel 76] Kandel, R. E., Freeman, W.H. *Cellular basis of behavior: An introduction to behavioral neurobiology*. 1976.
- [Kandel 79] Kandel, R. E. *Microsistemas de neuronas*. En: *El Cerebro*, Ed. Investigación y Ciencia, 39-49, 1979.
- [Kandel 92] Kandel, E. R., Hawkins, R. D. *Bases biológicas del aprendizaje y de la individualidad*. Investigación y Ciencia, 58, noviembre 1992.
- [Kandel 99] Kandel, E. R., Schwartz, T. H., Jessel, T. M. *Principles of Neural Science*. 4ª edición, McGraw-Hill, 1999.
- [Kangas 94] Kangas, J. *On the analysis of pattern sequences by self-organizing maps*. Tesis doctoral, Univ. Tec. de Helsinki, mayo 1994.
- [Katai 93] Katai, O., Ida, M., Sawargi, T. *Fuzzy control as self-organizing constraint-oriented problem solving*. Fifth IFSA World Congress, pp. 887-890, 1993.
- [Katashiro 93] Katashiro, T. *A fuzzy microprocessor for real-time control applications*. Fifth IFSA World Congress, pp. 1394-1397, 1993.
- [Kawaji 93] Kawaji, S., Matsunaga, N. *Obstacle avoidance algorithm for vehicle using fuzzy inferences*. Fifth IFSA World Congress, pp. 1246-9, 1993.
- [Kestelyn 90] Kestelyn, J. *Neural net for quality control*. AI Expert 5, 10, 71, octubre 1990.
- [Khan 93a] Khan, E. *Neural network based algorithms for rule evaluations & defuzzification in fuzzy logic design*. IEEE World Congress on Neural Networks 93, julio 1993.
- [Khan 93b] Khan, E., Venkatapuram, P. *Neufuz: Neural network based fuzzy logic desing algorithms*. FUZZ-IEEE93, vol. 1, pp. 647-654, San Francisco, California, abril, 1993.
- [Khotanzad 00] Khotanzad, A., Elragal, H., Lu, T.L. *Combination of artificial neural network forecasters for prediction of natural gas consumption*. IEEE Tran. on Neural Networks, 11, 2, 464-, 2000.
- [Kikuchi 93] Kikuchi, S., Chakroborty, P. *Modeling of car-following behavior: a fuzzy control system*. Fuzzy Logic'93, pp. M233-11, San Francisco, California, 1993.

- [King 89] King, A. G. *Nissan patents Fuzzy Logic ABS gearbox*. Automotive Electronic News, julio 1989.
- [Klein 95] Klein, R. *FPGAs take on specialized DSP functions*. Electronic Design, 12, junio, pp. 144-153, 1995.
- [Koch 96] Koch, C., Mathur, B. *Neuromorphic vision chips*. IEEE Spectrum, pp. 38-46, mayo 1996.
- [Kohonen 00] Kohonen, T., Kaski, S., Lagus, K.K., Salojärvi, J., Paatero, V., Saarela, A. *Organization of a massive document collection*. IEEE Transactions on Neural Networks, 11, 3, 574-585, 2000
- [Kohonen 82a] Kohonen, T. *Self-organized formation of topologically correct feature maps*. Biological Cybernetics, 43, 59-69, 1982.
- [Kohonen 82b] Kohonen, T. *Analysis of a simple self-organizing process*. Biological Cybernetics, 44, 135-140, 1982.
- [Kohonen 88a] Kohonen, T., Barna, G., Chrisley, R. *Statistical pattern recognition with neural networks: benchmarking studies*. Proc. IEEE Int. Conf. on Neural Networks, I-61-68, San Diego, 1988.
- [Kohonen 88b] Kohonen, T. *The neural phonetic typewriter*. IEEE Computer Magazine, pp. 11-22, marzo 1988.
- [Kohonen 89] Kohonen, T. *Self-Organization and Associative Memory*. 3th edition, Springer-Verlag, 1989.
- [Kohonen 90] Kohonen, T. *The Self-Organizing Map*. Proc. of the IEEE, 78, 9, pp. 1464-1480, 1990.
- [Kohonen 91a] Kohonen, T., Mäkisara, K., Simula, O., Kangas, J. (eds.). *Artificial Neural Networks* (Proc. of ICANN'91), Vol. I y II, Elsevier (Nort-Holland), 1.991
- [Kohonen 91b] Kohonen, T. *Self-organizing maps: optimization approaches*. En: [Kohonen 91a], vol. 2, pp. 981-990, 1991.
- [Kohonen 93a] Kohonen, T. *Things you haven't heard about the Self-Organizing Map*. IEEE 1993 Int. Conf. on Neural Networks, San Francisco, pp. 1147-1156, 1993.
- [Kohonen 93b] Kohonen, T. *Physiological interpretation of the self-organizing map algorithm*. Neural Networks, 6, 7, pp. 895-905, 1993.
- [Kohonen 95] Kohonen, T. *Self-Organizing Maps*. Springer-Verlag, 1995.
- [Kolinummi 97] Kolinummi, P., Hämäläinen, T., Kaski, K. *Improving ANN processing with a dedicated hardware systems*. IEEE Circuits and Devices Magazine, marzo, 19-27, 1997

- [Kosko 88] Kosko, B. *Feedback stability and unsupervised learning*. Proc. of the IEEE Int. Conf on Neural Networks, I, pp. 141-152, 1988.
- [Kosko 92] Kosko, B. *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*. Prentice-Hall, 1992.
- [Kosko 92a] Kosko, B. *Neural Networks and Fuzzy Systems*. Prent.-Hall 1992.
- [Kosko 92b] Kosko, B. *Neural Networks for Signal Processing*. Prentice-Hall, 1992.
- [Kosko 93] Kosko, B., Isaka, S. *Lógica Borrosa*. Investigación y Ciencia, pp. 60-65, septiembre 1993.
- [Krishanapuram 92] Krishanapuram, R., Joonwhoan, L. *Fuzzy-set-based hierarchical networks for information fusion in computer vision*. Neural Networks, vol. 5, pp. 335-350, 1992.
- [Kröse 93] Kröse, B. J. A., van der Smagt, P. P. *An introduction to Neural Networks*, University of Amsterdam, 1.993.
- [Kung 93a] Kung, S. Y. *Digital Neural Networks*. Prentice Hall, 1.993.
- [Kung 93b] Kung, S.Y., Chou, W.H. *Mapping neural networks onto VLSI array processors*. En [Przytula 93], 1993.
- [Kurzweil 99] Kurzweil, R. *La Era de las Máquinas Espirituales*. Planeta, 1999.
- [Lande 00] Lande, T. S., Marienborg, J.T., Berg, Y. *Neuromorphic cochlea implants..* IEEE International Symposium on Circuits and Systems ISCAS'2000. Ginebra, Suiza, pp. IV-401-5. IEEE Press, 2000.
- [Lande 98] Lande, T. S. (editor). *Neuromorphic Engineering*. Kluwer, 1998.
- [Lange 94] Lange, E., Nitta, Y., Kyuma, K. *Optical neural chips*. IEEE Micro Magazine, diciembre, 29-41, 1994.
- [Lapedes 87] Lapedes, A. S., Faber, R. M. *Nonlinear signal processing using neural networks: Prediction and system modelling*. Los Álamos, technical report LA-UR-87-2662, 1987.
- [Lawman 95] Lawman, G. *Configuring FPGAs over a processor bus*. Xilinx Application Notes. Julio, 1995.
- [LeCun 98] LeCun, Y., Bottou, L., Orr, G.B., Müller, K.R. *Efficient BackProp*. En [Orr 98], pp. 9-50, 1998.
- [Lee 91] Lee, S., Kil, R. M. *A Gaussian potential function network with hierarchically self-organizing learning*. Neural Networks, 4, 2, 207-214, 1991.
- [Lee 92] Lee, S. *Supervised learning with Gaussian potentials*. En [Kosko 92b], pp. 189-227, 1992.

- [Lee Jang 93] Lee Jang, Gyu, Chung Hakyoung. *Optimal path planning mobile robot*. Fifth IFSA World Congress, pp. 1258-1261, 1993.
- [Legg 92] Legg, G. *Special tools and chips make fuzzy logic simple*. EDN, vol. 6, pp. 68-76, julio 1992.
- [Lehman 93] Lehman, T. *A cascadable chip set for ANNs with on chip back-propagation*. Proc. of the 3rd. Int. Conf. on Microelectronics for Neural Nets MICRONEURO'93, 149-158. Edinburgh, 1993.
- [Lembessis 93] Lembessis, E., Tanscheit, R. *Rule-base size reduction techniques in a learning fuzzy controller*. Fifth IFSA World Congress, pp. 761-764, 1993.
- [Li 00] Li., H. X., Chen, C. L. P. *The equivalence between fuzzy logic systems and feedforward neural networks*. IEEE Transactions on Neural Networks, 11, 2, pp. 356-365, 2000.
- [Lin 96] Lin, C. T., Lee, C. S. G. *Neural Fuzzy Systems*. Prentice-Hall 1996.
- [Liñán 99] Liñán, G., Foldesy, P., Espejo, S., Domínguez, R., Rodríguez-Vázquez, A. *A 0.5 μ m CMOS 10⁶ transistors analog programmable array processor for real-time image processing*. 1999 European Solid-State Circuits Conference, pp- 358-361, 1999.
- [Lindh 93] Lindh, L., Müller-Glaser, K, Rauch, H, Stanischewski, F. *A real-time kernel. Rapid prototyping with VHDL and FPGAs*. 1993.
- [Lindsey 98a] Lindsey, C. S. *Neural Networks in Hardware: Architectures, Products and Applications*. Lección on-line, marzo 1998. (<http://www.particle.kth.se/~lindsey/HardwareNNWCourse/home.html>)
- [Lindsey 98b] Lindsey, C. S. *Neural Network Hardware*. Lección on-line, noviembre 1998 (<http://www1.cern.ch/NeuralNets/nnwInHepHard.html>)
- [Linsker 88] Linsker, R. *Self-organization in a perceptual network*. IEEE Computer, pp. 105-117, marzo 1988.
- [Lippman 87] Lippmann R.P. *An introduction to computing with neural nets*. IEEE ASSP Magazine, abril 4-22, 1987.
- [Lipsett 90] Lipsett, R., Schaefer, C. Ussery, C. *VHDL: Hardware Description and Design*. Kluwer, 1990.
- [Li-Xin 93] Li-Xin. *Adaptative Fuzzy Systems and Control*. Prentice, 1994.
- [Lo 91] Lo, Z. P., Bavarian, B. *On the rate of convergence in topology preserving neural networks*. Biological Cyber., 65, 55-63, 1991.
- [Lo 93] Lo, Z. P., Yu, Y., Bavarian, B. *Analysis of the convergence properties of topology preserving neural networks*. IEEE Trans. on Neural Networks, 4, 2, pp. 207-220, 1993.

- [López 93] López Piñero, J. M. *Cajal y la estructura histológica del sistema nervioso*. Investigación y Ciencia, febrero, 6-13, 1993.
- [Lupo 89] Lupo, J. C. *Defense applications of neural networks*. IEEE Communications Magazine, 82-88, noviembre, 1989.
- [Luttrell 89a] Luttrell, S. P. *Self-organization: A derivation from first principles of a class of learning algorithms*. Proc. of the Int. Joint Conf. on Neural Networks, II, 495-8, Washington D.C., 1989.
- [Luttrell 89b] Luttrell, S. P. *Hierarchical self-organizing networks*. Proc. of 1st IEE Conf. of Artificial Neural Networks, pp. 2-6, Londres 1989.
- [Luttrell 90] Luttrell, S. P. *Derivation of a class of training algorithms*. IEEE Trans. on Neural Networks, 1, 2, pp. 229-232, 1990.
- [LVQ_PACK 95] *LVQ_PACK. The Learning Vector Quantization Program Package*, V. 3.1. Helsinki University of Technology, Finlandia, 1995. Disponible en <http://www.cis.hut.fi/research/som-research/>
- [Maeder 89] Maeder, R. *Programing in Mathematica*. Addison-Wesley, 1989.
- [Maeng 93] Maeng, Jun Kim, Kang Geuntaek. *Desing of fuzzy controller based on fuzzy model for container crane system*. Fifth IFSA World Congress, pp. 1250-1253, 1993.
- [Mahowald 91a] Mahowald, M., Doublas, R. *A silicon neuron*. Nature, 354, 515-6, 1991.
- [Mahowald 91b] Mahowald, M., Mead, C. *La retina de silicio*. Investigación y Ciencia, pp. 42-49, julio, 1991.
- [Mamdani 74] Mamdani, E. H. *Application of fuzzy control algothims for control of simple dinamic plant*. Proc IEE, 121, 12, 1585-8, 1974.
- [Mamdani 75] Mamdani, E. H., Odtenggaard, J.J., Lembessis, E. *An experiment in linguistic sysnthesis with a fuzzy logic controller*. Int J. Man. Machine Studies, vol. 7, pp. 1-13, 1975.
- [Mamdani 83] Mamdani, E. H., Odtenggaard, J. J., Lembessis, E. *Use of fuzzy logic for implementing rule-based control of industrial processes*. Advances in Fuzzy Sets, Possibility Theory and Applications (Wang eds) Plenum Press, 1983.
- [Mandic 85] Mandic, N. J., Scharf, E.M., Mamdani, E. H. *Practical aplication of a heuristic fuzzy rule-based controller to the dynamic control of a robot arm*. IEE Procedings-G, Vol 132, 4, 190-203, 1985.
- [Marchesi 93] Marchesi, M., G. Orlandi, G., Piazza, F., Uncini, A. *Fast neural networks without multipliers*. IEEE Trans. on Neural Networks, 4, 1, 53-62, 1993.

- [Marijuán 92] Marijuan, P. C. *La acumulación social del conocimiento: una perspectiva interdisciplinar*. Jornadas de Organización del Conocimiento, Zaragoza, noviembre 1992.
- [Marijuán 99] Marijuán, P. C. Conferencia Internacional *Cajal y la Consciencia*. Zaragoza (España), noviembre de 1999 (<http://cajal.unizar.es/>).
- [Marks 93] Marks II, R. J. *Intelligence: Computational versus artificial*. IEEE Trans. on Neural Networks, 4, 5, 737-739, 1993.
- [Marose 90] Marose, R. AI Expert, mayo 1990.
- [Martín del Brío 93a] Martín del Brío, B., Serrano Cinca, C. *Self-organizing neural networks for analysis and representation of data: Some financial cases*. Neural Computing and Applications, 1, 193-206, 1993.
- [Martín del Brío 93b] Martín del Brío, B., Barquillas Pueyo, J. *Modelo de Red Neuronal de Kohonen orientado a la realización hardware*. Actas de la XXIV Reunión Bienal de la Real Sociedad Española de Física, Jaca (España), pp. EL4-, septiembre 1993
- [Martín del Brío 94a] Martín del Brío, B., Blasco Alberto, J. *A digital SIMD architecture for self-organizing maps*. 6th Microcomputer School, Brno (Czech Rep.), 263-268, septiembre 1994.
- [Martín del Brío 94b] Martín del Brío, B. *Procesamiento Neuronal con Mapas Auto-organizados: Arquitecturas Digitales*. Tesis Doctoral, Universidad de Zaragoza, 1994.
- [Martín del Brío 95a] Martín del Brío, B., N. Medrano, I. Ramírez, J. A. Domínguez, J. Barquillas, J. Blasco, J. García. *Short-term electric power load-forecasting using artificial neural networks. Part I: Self-organizing networks for classification of day-types*. 14th IASTED Int. Conf. on Modelling, Identif. and Control, Igsl, Austria, febrero 1995.
- [Martín del Brío 95b] Martín del Brío, B., Medrano, N., J. Blasco, J. *Feature map architectures for pattern recognition: Techniques for automatic region selection*. ICANGA 95, Alès (Francia), 1995. Actas Artificial Neural Nets and Genetic Algorithms. D.W. Pearson, Steele, N.C., Albretch, R.F. (editors), 124-127, Springer, 1995
- [Martín del Brío 95c] Martín del Brío, B., Serrano Cinca, C. *Self-organizing neural networks for analysis an representation of data: The financial state of Spanish companies*. En: *Neural Networks in the Capital Markets*, P. A. Refenes (Ed.), Wiley and Sons, 341-357, 1995.
- [Martín del Brío 96] Martín del Brío, B. *A dot product neuron for hardware implementation of competitive networks*. IEEE Transactions on Neural Networks, 7, 2, 529-532, 1996.

- [Martín del Brío 98] Martín del Brío, B., Medrano Marqués, N., Hernández, S. *A low-cost neuroprocessor board for emulating de SOFM Neural Model*. 5th IEEE International Conference on Electronics, Circuits and Systems, ICECS'98, Lisboa (Portugal), septiembre 1998
- [Martín del Brío 99] Martín del Brío, B. *Sistemas Electrónicos basados en Microprocesadores y Microcontroladores*. Editorial Pressas Universitarias, Zaragoza, 1999. ISBN: 84-7733-516-8.
- [Martinetz 93] Martinetz, T., Schulten, K. *A neural network with Hebbian-like adaptation rules learning visuomotor coordination of a PUMA robot*. IEEE Int. Conf. on Neural Net., San Francisco, 820-2, 1993.
- [Masaki 90] Masaki, A., Hirai, Y., Yamada, M. *Neural networks in CMOS: A case study*. Circuits and Devices Mag., 6, 4, 13-17, julio 1990.
- [Masters 93] Masters, T. *Practical Neural Networks Recipes in C++*. Academic Press, 1993.
- [Mauduit 92] Mauduit, N., Duranton, M., Gobert, J. *Lneuro 1.0: A piece of hardware LEGO for building neural network systems*. IEEE Transaction on Neural Networks, 3, 3, pp 414-422, 1992.
- [May 94] May, G. S. *Manufacturing integrated circuits: The neural way*. IEEE Spectrum, 47-51, Sep. 1994.
- [McClelland 86] McClelland, J. L., Rumelhart, D.E. (eds.). *Parallel Distributed Processing. Vol 2: Psychological and biological models*. MIT Press, 1986.
- [McClelland 88] McClelland, J. L., Rumelhart, D.E.. *Explorations in Parallel Distributed Processing*, The MIT Press, 1988.
- [McCord 91] McCord, N., Illingworth W. T. *A Practical Guide to Neural Nets*, Addison-Wesley Publishing, 1991.
- [McCulloch 43] McCulloch, W. S., Pitts, W. *A logical calculus of the ideas immanent in nervous activity*. Bulletin of Mathematical Biophysics, 5, 115-133, 1943.
- [Mead 89a] Mead, C. *Analog VLSI and Neural Systems*. Addison-Wesl 1989.
- [Mead 89b] Mead, C., Ismail, M (eds.). *Analog VLSI Implementation of Neural Systems*. Kluwer Academic Pub., 1989.
- [Medrano 00] Medrano Marqués, N., Martín del Brío, B. *Computer voice interface using the mouse port*. XVI Design of Integrated Circuits and Systems Conference, DCIS'2000. Montpellier (Francia), noviembre 2000.

- [Medrano 98] Medrano Marqués, N. *Nuevas Técnicas Hardware y Software para Análisis de Datos con Redes Neuronales*. Tesis Doctoral, Universidad de Zaragoza, mayo 1998
- [Medrano 99] Medrano Marqués, N., Martín del Brío, B. *Topology preservation in SOFM: An Euclidean versus Manhattan distance comparison*. International Work-Conference on Artificial and Natural Neural Networks IWANN'99. Alicante, junio 1999
- [Melton 92] Melton, M. S., Phan, T., Reeves, D.S., Ven den Bout, D.E. *The TInMANN VLSI Chip*. IEEE Trans. on Neural Networks, 3, 3, 375-384, 1992.
- [MicroNeuro 96] Actas del *5th International Conference on Microelectronics for Neural Networks and Fuzzy Systems, MicroNeuro'96*. Lausanne, Suiza. IEEE Press, 1996.
- [Minho 93] Minho, L., Soo-Young, L. *Neuro-fuzzy identifiers and controllers for fuzzy systems*. Fifth IFSA World Congress, 93, 77-80, 1993.
- [Minsky 69] Minsky, M., Papert, S. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.
- [Moody 90] Moody, J. E., Darken, C. J. *Fast learning networks of locally-tuned processing units*. Neural Computation, 1, 2, 281-294, 1989.
- [Moody 92] Moody, J. E. *The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems*. En: *Advances in Neural Information Processing Systems 4*, Morgan Kaufmann Pub., 1992.
- [Moravec 88] Moravec, H. *Mind Children. The Future of Robot and Human Intelligence*. Harvard University Press, 1988.
- [Morgan 90] Morgan, N. (edt.). *Artificial Neural Networks: Electronic Implementations*, IEEE Computer Society Press, 1.990.
- [Müller 90] Müller, B., Reinhardt, J. *Neural Networks. An Introduction*, Springer-Verlag, 1990.
- [Muller 92] Muller, U. A., Baumle, B., Kohler, P., Gunzinger, A., Guggenbuhl, W. *Achieving supercomputer performance for neural net simulation with an array of digital signal processors*. IEEE Micro, 55-65, octubre 1992.
- [Murray 89] Murray, A. F. *Pulse arithmetic in VLSI neural networks*. IEEE Micro, 9, 6, pp. 64-74, 1989.
- [Murtagh 90] Murtagh, F (arranged by). *Neural Networks for Statistical and Economic Data* (Proc. of the Workshop, Dublin, diciembre 1990), Munotec Systems, 1990

- [Myers 93] Myers, D. J., Murray, A. F (eds.). *Proc of the 3rd Int. Conf. Microelectronics for Neural Networks*. Univ. de Edinburgo, abril 1993.
- [Nass 92] Nass, R. *Fuzzy logic finally gains acceptance in the U.S.* EDN, vol. 25, pp. 37-40, junio 1992.
- [Nasution 94] Nasution, S.H. *Fuzzy critical path method*. Proceedings of the IEEE, vol. 24, n° 1, pp. 48-57, enero 1994.
- [NATO 93] NATO Advanced Study Institute. *NATO ASI From Statistics to Neural Networks*. Les Arcs (France), Junio, 1993 (de próxima publicación por Springer-Verlag [Cherkaski 94]).
- [Nelson 99] Nelson, T., Kerridge, B. *Neural networks evaluates noise in auto tape decks*. Test & Measurement Europe, 33-37, abril, 1999.
- [NeuroNet 95] *Manifesto for Artificial Neural Networks*. Publicado por NeuroNet, ISBN 0 952663406, <http://www.kcl.ac.uk/neuronet/>. King's College London, 1995
- [Nguyen 92] Nguyen, D., y Widrow, B. *The truck backer-upper: An example of self-learning in neural networks*. Proc. Int. Joint Conf. on Neural Networks. Vol II. Erlbaum, Hillsdale, N.J. 357-363, 1992.
- [Nieto 89] Nieto Sampedro, M. *Plasticidad sináptica*. Investigación y Ciencia, 40-49, febrero 1989.
- [Nikkei 89] Nikkei Business. *(All in Japan.) Tunnel drilling system uses fuzzy logic*. NIKKEI Business Publications, Tokyo, noviembre 1989.
- [Nomura 91] Nomura, H., Hayashi I., Wakami, N. *A self-tuning method of fuzzy control by descent methods*. IFSA World Congress, 155-158, 1991.
- [Oja 82] Oja, E. *A simplified neuron model as principal component analyzer*. Journal of Mathematical Biology, 15, 267-273, 1982.
- [Omondi 94] Omondi, A. *Computer Arithmetic Systems, Algorithms, Architecture and Implementations*. Prentice-Hall, 1994.
- [Onodera 93] Onodera, H., Takeshita, K., Tamaru, K. *Hardware architecture for Kohonen network*. IEICE Transactions on Electronics (Japan), Vol. E76-C, n° 7, 1159-1166, 1993.
- [Orr 98] Orr, G.B., Müller, K.R. *Neural Networks: Tricks of the Trade*. Springer-Verlag, 1998.
- [Pankove 90] Pankove, J., Radehaus, C., Wagner, K. *Winner-Take-All neural net with memory*. Electronics Letters, vol. 26, n° 6. 1990.
- [Passino 98] Passino, K., Yurkovich, S. *Fuzzy Control*. Addison-Wesley, 1998.

- [Patyra 93] Patyra, M., Lemaitre, L., Mlynek, D. *Current mirror-based approach to the integration of CMOS*. Fifth IFSA World Congress 1993, pp. 785-788, 1993.
- [Pérez 89] Pérez Vicente, C.J. *Memory melting in neural networks with discrete synapses*. Phys. Rev. A, 39, 4303, 1989.
- [Pérez 91] Pérez Vicente, C.J., Carrabina, J., Garrido, F., Valderrama, E. *Learning algorithms for feed-forward neural networks with discrete synapses*. Proc. of the Int. Work. on Art. Neural Networks, Granada (Spain), pp. 144-152, 1991.
- [Perfetti 90] Perfetti, R. *Normalisation in neural networks using parallel automatic level control*. Electronics Letters. 26, nº 2, 82-3, 1990.
- [Perry 91] Perry, D. *VHDL*. Ed. McGraw-Hill, 1991.
- [Personnaz 86] Personnaz, L., Guyon, I., Dreyfus, G. *Collective computational properties of neural networks: New learning mechanisms*. Physical Review A, 34, 5, pp. 4217-4228, 1986.
- [Pina 89] Pina V. *Estudio empírico de la crisis bancaria*. Revista Española de Financiación y Contabilidad, vol. XVIII, 58, 309-338, 1989.
- [Pino 93] Pino, B. del, Pelayo, F. J., Prieto, A. *Implementación VLSI de redes neuronales artificiales*. En: Burón, A.M., y otros (edit.), *Microelectrónica 92*, 135-196, Servicio de Publicaciones de la Universidad de Cantabria, 1993.
- [Poggio 90] Poggio, T., Girosi, F. *Networks for approximation and learning*. Proc. of the IEEE, Sept., 1481-1497, 1990.
- [Poppe 95] Poppe, T., Obradovic, D., Schlang, M. *Neural networks: reducing energy and raw materials requirements*. Siemens Review, otoño, 24-27, 1995.
- [Prechelt 98] Prechelt, L. *Early Stopping -But When?* En [Orr 98], 55-70, 1998.
- [Principe 00] Principe, J. C., Euliano, N. R., Lefebvre, W. C. *Neural and Adaptive Systems. Fundamentals Through Simulations*. John Wiley 2000.
- [Przytula 93] Przytula, K. W., Prasanna, V. K (eds.). *Parallel Digital Implementations of Neural Networks*. Prentice-Hall, 1993
- [Raibert 86] Raibert, M. H. *Running with symmetry*. The International Journal of Robotics Research, Vol 5, No. 4, pp. 45-61, 1986.
- [Ramacher 91a] Ramacher, U., Rückert, U., Nossek, J. A (eds.). *Proc of the 2nd Int. Conf. Microelectronics for Neural Networks*. Kyrill & Method Verlag, Munich, octubre 1991.

- [Ramacher 91b] Ramacher, U., Rückert, U. (eds.). *VLSI Design of Neural Networks*, Kluwer Academic Publishers, 1991.
- [Ramacher 94] Ramacher, U. Neurocomputers: Toward a new generation processors. *Siemens Review*, 3, 26-29, 1994.
- [Ramón y Cajal 1899] Ramón y Cajal, S. *Textura del Sistema Nervioso del Hombre y de los vertebrados*. N. Moya, Madrid, 1899-1904.
- [Ramón y Cajal 1999] Ramón y Cajal, S. *Texture of the Nervous System of Man and Vertebrates*, Springer-Verlag, Viena, Nueva York, 1999.
- [Refenes 93] Refenes A. N. (edt.). *Proc. 1st International Workshop on Neural Networks in the Capital Markets*. London (UK), noviembre 1993.
- [Refenes 95] Refenes A. N. (edt.). *Neural Networks in the Capital Markets*. John Wiley & Sons, 1995.
- [Reilly 82] Reilly, D. L., Cooper, L.N., Elbaum, C. *A neural model for category learning*. *Biological Cybernetics*, 45, 35-41, 1992.
- [Reyero 95] Reyero, R., Nicolás, F. *Sistemas de Control Basados en Lógica Borrosa*. OMROM Electronics-IKERLAN, 1995.
- [Reyes Mozos 93] Reyes de los Mozos, M., Valderrama, E., Arguelles, J. *FCSO: A fuzzy compensated crystal oscillator*. Fifth IFSA World Congress, pp. 842-844, 1993.
- [Reyneri 99] Reyneri, L. M. *Unification of neural and wavelet networks and fuzzy systems*. *IEEE Tran. on Neural Networks*, 10, 4, 801-814, 1999
- [Ritter 86] Ritter, H., Schulten, K. *On the stationary state of Kohonen's self-organizing sensory mapping*. *Biological Cybernetics*, 54, 99-106, 1986.
- [Ritter 88] Ritter, H., Schulten, K. *Convergence properties of Kohonen's topology conserving maps: Fluctuations, stability, and dimension selection*. *Biological Cybernetics*, 60, 59-71, 1988.
- [Ritter 88b] Ritter, H., Schulten, K. *Kohonen self-organizing maps: Exploring their computational capabilities*. *Proc. of the IEEE Int. Conf. on Neural Networks*, 109-116, San Diego, julio 1988.
- [Ritter 89] Ritter, H., Kohonen, T. *Self-organizing semantic maps*. *Biological Cybernetics*, 61, 241-254, 1989.
- [Ritter 91a] Ritter, H., Martinetz, T., Schulten, K. *Neural Computation and Self-Organizing Maps*, Addison-Wesley, 1991.
- [Ritter 91b] Ritter H., Obermayer K., Schulten K., Rubner, J. *Self-organizing maps and adaptive filters*. En: Domany E., van Hemmen J. L.,

- Schulten K(eds). *Models of Neural Networks*. Springer-Verlag, Berlín, Heidelberg, Nueva York, pp. 281-306, 1991.
- [Ritter 91c] Ritter, H. *Asymptotic level density for a class of vector quantization processes*. IEEE Trans. on Neural Networks, 2, 1, 173-174, 1991.
- [Rodrigues 91] Rodrigues, J. S., Almeida, L. B. *Improving the learning speed in topological maps of patterns*. En: *Neural Networks. Advances and Applications*, Gelenbe, E. (edt.), North-Holland Elsevier, 63-78, 1991.
- [Roermund 00] Roermund, A., Hoekstra, J. *From nanotechnology to nanoelectronic systems, from SETs to neural nets*. IEEE Int. Symp. on Circuits and Systems ISCAS'2000. Ginebra, I-8-12. IEEE Press, 2000.
- [Rosenblatt 62] Rosenblatt, F. *Principles of Neurodynamics*. Spartan Books, Nueva York, 1962.
- [Roska 00] Roska, T., Rodríguez-Vázquez, A. *Review of CMOS implementations of the CNN universal machine-type visual microprocessor*. IEEE Int. Symp. on Circuits and Systems ISCAS'2000. Ginebra, Suiza, pp. II.120-3. IEEE Press, 2000.
- [Rovira 89] Rovira, M. Mundo Electrónico, noviembre, pp. 81, 1989.
- [Rubner 90] Rubner, J., Schulten, K. *Development of feature detectors by self-organization*. Biological Cybernetics, 62, 193-199, 1990.
- [Ruiz 93] Ruiz, A., Gutiérrez, J. A *VLSI-CMOS programmable membership function Circuit*. Fifth IFSA World Congress, 977-980, 1993.
- [Rumelhart 86a] Rumelhart, D. E., McClelland, J.L. (eds.). *Parallel Distributed Processing. Vol 1: Foundations*. MIT Press, 1986.
- [Rumelhart 86b] Rumelhart D. E., Hinton G.E., Williams R.J. *Learning representations by backpropagating errors*. Nature, 323, 533-6, 1986.
- [Rumelhart 86c] Rumelhart, D. E., Zipser, D. *Feature discovery by competitive learning*. En: [Rumelhart 86a], pp. 151-193, 1986.
- [Rüping 93] Rüping, S., Rückert, U., Gosser, K. *Hardware desing for self-organizing feature maps with binary input vectors*. Proc IWANN'93, Sitges, 488-493, 1993.
- [Ruspini 93] Ruspini, E. H., Saffiotti, A., Konolige, K. *A fuzzy controller for flakey, an autonomous mobile robot*. Fuzzy Logic '93, pp. A213-10, San Francisco, California, 1993.

- [Sae-Hie 93] Sae-Hie, Park, Yong-Ho, K., Choi, Y-K. *Self-organization of fuzzy rule base using genetic algorithm*. Fifth IFSA World Congress, pp. 881-886, 1993.
- [Samardzija 91] Samardzija, N., Waterland, R.L. *A neural network for computing eigenvectors and eigenvalues*. Biological Cybernetics 65, 211-214, 1991.
- [Sammon 69] Sammon, J. W Jr. *A nonlinear mapping for data structure analysis*. IEEE Trans. on Computers, C-18, 5, 401-409, 1969.
- [Sánchez 93] Sánchez, L., Tuya, J., Alvarez, J. C. *Extracción automática de reglas en un sistemas de control difuso*. III Congreso Español de Tecnologías y Lógica Fuzzy, pp. 243-250, Santiago de Compostela, 1993.
- [Sanger 89] Sanger, T. *Optimal unsupervised learning in a single-layer linear feedforward neural network*. Neural Networks, 2, 459-473, 1989.
- [Santos 96] Santos, M., de la Cruz, J. M., Domingo, S. *Influence of the information processing in fuzzy logic controllers*. IPMU'96, pp. 81-86, 1996.
- [Sanz 93] Sanz Molina, A. *Control fuzzy de sistemas de respuesta autonoma*. III Congreso Español de Tecnologías y Lógica Fuzzy, pp. 267-278, Santiago de Compostela, 1993.
- [Sanz 94] Sanz Molina, A. *Entorno para el Desarrollo de ANN y Circuitos Fuzzy*. Tesis Doctoral, Universidad de Zaragoza, 1994.
- [Sanz 96] Sanz Molina, A. *An object oriented envelopement for complex neuro fuzzy controller*. IPMU'96, pp. 514-518, 1996.
- [Sarle 94] Sarle, W. S. *Neural Networks and Statistical Models*. Proceedings of the 19th annual SAS Users Group International Conference, Cary, NC (SAS Institute), pp 1538-1550, 1994.
- [Sasaki 93] Sasaki, M., Ueno, F., Inoue, T. *An 8-bit resolution 140 kFLIPS fuzzy microprocessor*. 5th IFSA World Congress, 921-924, 1993.
- [Scharf 85] Scharf, E. M. *FuzzylLogic could redefine robotic control*. Robotic, pp. 11-13, February, 1985.
- [Schwartz 92] Schwartz, E. I. and Treece, J. B. *Smart programs go to work: How applied-intelligence software makes decisions for the real world*. Bussines Week, 2 de marzo, 97-105, 1992
- [Schwartz 92b] Schwartz, E. I. *Where neural networks are already at work: Putting AI to working the markets*. Bussines Week, 2, noviembre, 136-137, 1992.

- [Scientific 79] *The Brain. Scientific American* (monográfico), sept., 1979.
- [Scientific 92] *Mind and Brain. Scientific American* (monográfico), sept, 1992.
- [Sensory 00] Sensory Inc., Sunnyvale, California, <http://www.sensoryinc.com/>. Puede encontrarse información actualizada sobre sus productos (RSC164, 200, 264, 364 y otros).
- [Seok 93] Seok, Kim Yoo, Gyu Lee Jang. *Motion planing of an autonomous mobil robot in flexible manufacturing systems*. Fifth IFSA World Congress pp. 1254-1257, 1993.
- [Serrano 93] Serrano Cinca, C., Martín del Brío, B. *Predicción de la quiebra bancaria mediante el empleo de Redes Neuronales Artificiales*. Revista Española de Financiación y Contabilidad, vol. XXII, nº 74, 153-176. Madrid, 1993.
- [Setiono 00] Setiono, R. *Extracting M-of-N rules from trained neural networks*. IEEE Transactions on Neural Networks, 11, 2, pp. 512-9, 2000.
- [Shandle 93] Shandle, J. *Neural Networks are ready for prime time*. Electronic Design, 41, 4, 51-58, 1993.
- [Shann 93] Shann, J. J., Fu, H. C. *A fuzzy neural network for knowlwdge learning*. Fifth IFSA World Congress, pp. 151-154, 1993.
- [Shao 88] Shao, S. *Fuzzy self-organizing controller and its application for dynamic process*. Fuzzy Set and Systems, 26, 2, 151-164, 1988.
- [Shea 89] Shea, P.M., y Lin, V. *Detection of explosives in checked airline baggage using an artificial neural system*. Proc. Int. Joint Conf. on Neural Networks, Washington D.C., vol. II, 31-4, 1989
- [Shenoi 93] Shenoi, S., Ashenayi, K. *Hardware implementation of an Autonomous fuzzy controller*. Fifth IFSA World Congress, pp. 834-837, 1993.
- [Shepherd 97] Shepherd, G. M. *The Synaptic Organization of the Brain*. 4ª edición, Oxford University Press, 1997.
- [Sibigtroth 93a] Sibigtroth, J. M. *Fuzzy inference algorithms for 8-Bit MCUs*. Fuzzy Logic'93, pp. A123-19, San Francisco, California, 1993a.
- [Sibigtroth 93b] Sibigtroth, J. M., Mazuelos, D. *Basic traning: fuzzy logic for 8-Bit MCUs*. Fuzzy Logic'93, pp. T11-27; 1993b.
- [SIENA 96] Informe del proyecto ESPRIT 9811 *Stimulation Initiative for European Neural Applications SIENA*., 1996. Disponible en [http://www.mbfys.kun.nl/snn/ Research/siena/index.html](http://www.mbfys.kun.nl/snn/Research/siena/index.html)
- [Simpson 89] Simpson, P. K. *Artificial Neural Systems*, Pergamon Press, 1989.

- [Simpson 92] Simpson, P.K. *Foundations of neural networks*. En: Sánchez-Sinencio, E., Lau, C. *Artificial Neural Networks. Paradigms, Applications and Hardware*. IEEE Press, 3-25, 1992.
- [SOM_PACK 92] SOM_PACK. *The Self-Organizing Map Program Package*. Helsinki University of Technology, Finland, noviembre 1992.
- [Song Han 93] Song Han, Il. *Analogue-digital Hybrid circuit for an adaptive fuzzy network*. Fifth IFSA World Congress, pp. 838-841; San Francisco, California, 1993.
- [Soto 93] Soto, E., Mandado, E., Fariña, F. *Lenguajes de descripción hardware (I) y (II)*. Mundo Electrónico, 236, 34-8, y 237, 32-6, 1993.
- [Specht 91] Specht, D. F. *A general regression neural network*. IEEE Transactions on Neural Networks, 2, 6, 568-576, 1991.
- [Spiegel 88] Spiegel M. R. *Statistics*. McGraw Hill, Nueva York, 1988.
- [Sugeno 85a] Sugeno, J., Nishida, M. *Fuzzy control of model car*. Fuzzy Set and Systems, vol. 26, nº 2, pp. 151-164, 1985.
- [Sugeno 85b] Sugeno, J. *Industrial applications of fuzzy control*. Elsevier, 1985.
- [Sugeno 93a] Sugeno, J., Park, G.K. *An approach to linguistic instruction based learning and its application to helicopter flight control*. Fifth IFSA World Congress, pp. 1082-1085, 1993a.
- [Sugeno 93b] Sugeno, J., Griffin, M.F., Bastian, A. *Fuzzy hierarchical control of an unmanned helicopter*. Fifth IFSA World Congress, pp. 179-182, 1993b.
- [Sung-Kwun 93] Sung-Kwun, Oh, Park Jong-jin. *The optimal tuning algorithm for fuzzy controller*. Fifth IFSA World Congress, 830-3, 1993.
- [Szátmari 00] Szátmari, I., Zarándy, A., Földesy, P., Kék, L. *An analogic CNN engine board with the 64x64 Analog CNN-UM chip*. Actas del IEEE International Symposium on Circuits and Systems ISCAS'2000. Ginebra, Suiza, pp. II.124-7. IEEE Press, 2000.
- [Sztandera 93] Sztandera, L., Cios, K.J. *Decision making in a fuzzy environment. generated by a neural network architecture*. Fifth IFSA World Congress, pp. 73-76, 1993.
- [Taecheon 93] Taecheon, A., Sungkvun, O., Woo, K. *Automatic generation of fuzzy rules using the fuzzy-neural network*. Fifth IFSA World Congress, pp. 1181-1185, 1993.
- [Takagi 83] Takagi, H., Sugeno, J. *Derivation of control rules form human operator's control action*. Proc of the IFAC Sysmp. on Fuzzy

- Information, Knowledge Representation and Decision Analysis, pp. 55-60, julio 1983.
- [Takagi 90] Takagi, H. *Fusion Technology of Fuzzy Theory and Neural Networks*. Proc. of IIZUKA'90, Fukuoka, 1990.
- [Tanaka 93a] Tanaka, K., Sano, M. *Phase compensation of fuzzy control systems and realization of neuro-fuzzy compensators*. Fifth IFSA World Congress, pp. 845-848, 1993a.
- [Tanaka 93b] Tanaka, K. *Balancing speed, precision, and flexibility accelerating complex rule bases in hardware*. Fifth IFSA World Congress, pp. 937-940, 1993b.
- [Tanenbaum 90] Tanenbaum, A.S. *Structured Computer Organization*. 2nd edition. Prentice-Hall, 1990.
- [Tanscheit 88] Tanscheit, R., Scharf, E.M. *Experiments with the use of a rule based self-organising controller for robotic applications*. Fuzzy Set and Systems, vol. 26, n° 2, pp. 195-214, 1988.
- [Tavan 90] Tavan, P., Grubmüller, H., Kühnel, H. *Self-organization of associative memory and pattern classification: Recurrent signal processing on topological feature maps*. Biological Cybernetics, 64, 95-105, 1990.
- [Tavernier 94] Tavernier, C. *Circuitos Lógicos Programables*. Paraninfo, 1994.
- [Terano 92] Terano, T., Asai, K., Sugeno, J. *Fuzzy Systems Theory and its Applications*. Academic Press Inc., 1992.
- [Thimm 95] Thimm, G., Fiesler, E. *Neural network initialization*. Proc. Int. Work. on Artificial Neural Networks, IWANN95, pp. 534-542, Torremolinos, junio, 1995.
- [Thiran 92] Thiran, P., Hasler, M. *Quantization effects in Kohonen networks*. Congrès Européen de mathématiques sur les aspects théoriques des reseaux des neurones, París, julio 1992.
- [Thiran 93] Thiran, P. *Self-organization of a Kohonen network with quantized weights and an arbitrary one-dimensional stimuli distribution*. European Symp. on Neural Net. ESANN'93, Bruselas, abril 1993.
- [Thiran 94] Thiran, P., Peiris, V., Heim, P., Hochet, B. *Quantization effects in digitally behaving circuit implementations of Kohonen networks*. IEEE Trans. on Neural Networks, 5, 3, 450-458, 1994.
- [Tou 74] Tou, J. T., González, R. C. *Pattern Recognition Principles*. Addison-Wesley Pub. Comp., 1974.

- [Toxa 91] *Redes Neuronales Artificiales*. XIII Escuela de Verano de Informática, organizada por la AEIA, Isla de A Toxa, julio, 1991.
- [Travis 94] Travis, J. *Glia: The brain's other cells*. Science, 266, 970-1, 1994.
- [Treleaven 89] Treleaven, P., Pacheco, M., Velasco, M. *VLSI architectures for neural networks*. IEEE Micro, Dec., pp. 8-27, 1989.
- [Trigueiros 91] Trigueiros D., Berry R. H. *Neural Networks and the automatic selection of financial ratios*. Tech. Report, INESC, Portugal, 1991.
- [Tryba 89] Tryba, V., Metzen, S., Gosser, K. *Designing basic integrated circuits by self-organizing feature maps*. Proc. of the Int. Workshop on Neural Networks and their Applications, NeuroNîmes'89, 225-235, noviembre 1991
- [Tryba 91] Tryba, V., Gosser, K. *Self-organizing feature maps for process control in chemistry*. En: [Kohonen 91a], pp. 847-852, 1991
- [Tsukamoto 79] Tsukamoto, Y. *An approach to fuzzy reasoning method*. En: Gupta M., Rammohan K., Yager R.R. (eds.), *Advances in Fuzzy Set Theory and Applications*, pp. 137-149, North-Holland, 1979.
- [Tsukimoto 00] Tsukimoto, H. *Extracting rules from trained neural networks*. IEEE Transactions on Neural Networks, 11, 2, pp. 377-389, 2000.
- [Ungering 93] Ungering, A. P. *Architecture of a PDM VLSI fuzzy logic controller with an explicit rule base*. Fifth IFSA World Congress, pp. 1386-1389, 1993.
- [Vapnik 99a] Vapnik, V. M. *Guest Editorial: Vapnik-Chervonenkis Learning Theory and It's Applications*. IEEE Transactions on Neural Networks, 10, 5, pp. 985-7, 1999
- [Vapnik 99b] Vapnik, V. M. *An overview of Statistical Learning Theory*. IEEE Transactions on Neural Networks, 10, 5, pp. 988-999, 1999
- [Varfis 92] Varfis, A., Versino, C. *Selecting reliable Kohonen maps for data analysis*. En: *Artificial Neural Networks 2*, I. Aleksander, J. Taylor (Editors), pp 1583-1586, Elsevier, 1992.
- [Vemuri 88] Vemuri, V. *Artificial neural networks: An introduction*. En: *Artificial Neural Networks: Theoretical Concepts*. IEEE Computer Press, 1-12, 1988.
- [Vesanto 99] Vesanto, J. *SOM-based data visualization methods*. Intelligent Data Analysis, 3, 111-126, 1999
- [Vesanto 00] Vesanto, J., Alhoniemi, E. *Clustering of the self-organizing map*. IEEE Transactions on Neural Networks, 11, 3, 586-600, 2000

- [Viredaz 93] Viredaz, M. A. *MANTRA I: A SIMD processor array for neural computation*. En: Proc. of the Euro-ARCH'93 Conf., 1993.
- [Viredaz 94] Viredaz, M. A. *Design and Analysis of a Systolic Array for Neural Computation*. Tesis Doctoral, EPFL, Lausanne, 1994.
- [Vittoz 90] Vittoz, E., Oguey, H., Maher, M.A. *Analog storage of adjustable synaptic weights*. Proceeding of the first International Workshop on Microelectronic for Neural Networks, 69-79, 1990.
- [Vogl 88] Vogl, T. P., Mangis, J. K., Rigler, A. K., Zink, W. T., Alkon, D. L. *Accelerating the convergence of the back-propagation method*. Biological Cybernetics, 59, 257-263, 1988.
- [von der Malsburg 73] von der Malsburg, C. *Self-organization of orientated sensitive cells in the striate cortex*. Kybernetik, 14, 85-100, 1973.
- [Wang 93] Wang, J. *12-Bit fuzzy computational acceleration (FCA) core*. Fuzzy Logic '93, pp. M332-7, San Francisco, California, 1993.
- [Wang 94] Wang, Li-Xing. *Adaptative Fuzzy Systems and Control*. Prentice Hall, 1994.
- [Warwick 95] Warwick, K. *An overview of neural networks in control applications*. En: *Neural Networks for Robotic Control*, M. Zalzal, Prentice-Hall, pp. 1-25, 1.995.
- [Wasserman 89] Wasserman, P. D. *Neural Computing. Theory and Practice*. Van Nostrand Reinhold, 1.989.
- [Wasserman 93] Wasserman, P. D. *Advanced Methods in Neural Computing*. Van Nostrand Reinhold, 1.993.
- [Weigend 91] Weigend, A. S., Rumelhart, D. E. *Generalization through minimal networks with application to forecasting*. En: Keramidas, E. (edit.) *INTERFACE'91-23rd Symp. on the Interface: Computer Science and Statistics*, pp. 362-370, Interface Foundation, 1991.
- [Weigend 93] Weigen, A., S. *Book Review: Hertz-Krogh-Palmer, Introduction to the Theory of Neural Computation*. En la revista Artificial Intelligence, 1993.
- [Weinstein 92] Weinstein, J. N. y otros. *Neural computing in cancer drug development: Predicting mechanism of action*. Science, vol. 258, 16, 447-451, 1992.
- [Werbos 74] Werbos, P. J. *Beyond Regression: New Tools for Prediction and Analysis in Behavioral Sciences*. Doctoral Dissertation, Appyied Mathematics, Harvard University, noviembre 1974.

- [Werbos 90] Werbos, P. J. *Backpropagation through time: What it does and how to do it*. Proc. of the IEEE, oct., pp. 1550-1560, 1990.
- [Werbos 98] Werbos, P. *Neural networks combating fragmentation*. IEEE Spectrum Magazine, 76-77, enero 1998.
- [White 88] White, H. *Economic prediction using Neural Networks: The case of IBM daily stock returns*. Proc of the IEEE Int Conf on Neural Networks, vol 2, pp 451-459, San Diego, 1988.
- [White 89a] White, H. *Learning in artificial neural networks: A statistical perspective*. Neural Computation, 1, 425-464, 1989.
- [White 89b] White, H. *Neural network learning and statistics*. AI Expert, 48-52, Dec. 1989.
- [Widrow 60] Widrow, B., Hoff, M.E. *Adaptive switching circuits*. 1960 IRE WESCON Convention Record, 4, pp. 96-104, 1960.
- [Widrow 88] Widrow, B., Winter, R. *Neural nets for adaptive filtering and adaptive pattern recognition*. IEEE Computer, marzo, 25-39, 1988.
- [Widrow 90] Widrow, B., Lehr, M. A. *30 years of adaptive neural networks: Perceptron, Madaline and Backpropagation*. Proc. IEEE, 78, 9, pp. 1415-1442, 1990.
- [Widrow 94] Widrow, B., Rumelhart, D. E., Lehr, M. E. *Neural networks: Applications in industry, business and science*. Communications of the ACM, 37, 3, 93-105, 1994.
- [Willaeys 80] Willaeys, D. *Optimal control of fuzzy self-organizing visual tracking controller*. Fuzzy Sets and Possibility Theory, 1980.
- [Williams 92] Williams, R. J. *Object-oriented methods transform real-time programing*. Computer Design, pp. 101-118, septiembre 1992.
- [Williams 93] Williams, T. *Benchmark suites help evaluate fuzzy logic performance*. Computer Design, pp. 42-46, enero 1993.
- [Wilson 88] Wilson, G. V., Pawley, G. S. *On the stability of the traveling salesman problem algorithm of Hopfield and Tank*. Biological Cybernetics, 58, pp. 63-, 1988.
- [Wirtyh 80] Wirtyh, N., Álvarez Rodríguez, A., Cuenca Bartolomé, J. *Algoritmos + Estructuras de datos = Programas*. Ediciones del Castillo, Madrid, pp. 382, 1980.
- [Wolfram 91] Wolfram, S. *Mathematica. A System for Doing Mathematics by Computer*. Addison-Wesley, vol. 2, pp. 961, 1991.
- [Wright 90] Wright, M. *Neural networks tackle real-world problems*. EDN, noviembre 8, pp. 79-90, 1990.

- [Wright 91a] Wright, M. *Neural networks IC architectures define suitable applications*. EDN, julio 4, pp 84-90, 1991.
- [Wright 91b] Wright, M. *Designing neural networks commands skill and savvy*. EDN, diciembre 5, pp 86-94, 1991.
- [Xie 91] Xie, Y., Jabri, M. A. *Analysis of effects of quantisation in multilayer neural networks using statical model*. Electronics Letters, vol. 27, n° 13. 1991.
- [Xilinx 95] Xilinx. *FPGAs as reconfigurable processing elements*. XCELL, num. 16, pp. 23-29. 1^{er} cuatrimestre, 1995.
- [Xilinx 96] Xilinx. *The programmable logic data book, 1996*.
- [Zadeh 65] Zadeh, L. A. *Fuzzy Sets*. Information & Control, 8, 338-353, 1965.
- [Zadeh 73] Zadeh, L. A. *Outline of a new approach to the analisys of complex systems and decision processes*. IEEE: Trans. Syst., Man, Cybern., Vol 3, pp. 28-44, enero, 1973.
- [Zadeh 88] Zadeh, L. A. *Fuzzy Logic*. IEEE Computer Magazine, 83-93, abril 1988.
- [Zadeh 92] Zadeh, L. A. Prólogo del libro [Kosko 92a].
- [Zadeh 94] Zadeh, L. A. *Fuzzy logic, neural networks and soft computing*. Communications of the ACM, 3, 3, 77-84, marzo 1994.
- [Zadeh 99] Zadeh, L. A. *From computing with numbers to computing with words-From manipulation of measurements to manipulation of perceptions*. IEEE Trans. on Circ. and Syst. 45, 1, 105-119, 1999.
- [Zeungnan 91] Zeungnan, B., Jongcheol, V. *Fuzzy self-organizing visual tracking controller*. IFSA World Congress, pp. 21-24, 1991.
- [Zhang 93] Zhang, L. *Fuzzy-based acupuncture diagnosis system*. Fifth IFSA World Congress, pp. 553-556, 1993.
- [Zhao 93] Zhao, Z. Y., Tomizuka, M., Isaka, S. *Fuzzy gain scheduling of PID controllers*. Proceedings of the IEEE, 23, 5, 1392-8, 1993.
- [Zhao 94] Zhao, Z. Y., Ravishankar, C. V., BeMent, S. *Coping with limited on-board memory and communication bandwidth in mobile-robot systems*. Proc. of the IEEE, 24, 1, 58-72, enero 1994.
- [Zurada 92a] Zurada, J. M. *Artificial Neural Systems*. West Pub. Comp., 1992.
- [Zurada 92b] Zurada, J. M. *Analog implementation of neural networks*. IEEE Circuits and Devices Magazine, sept., 36-41, 1992.

ÍNDICE ALFABÉTICO

6805, 320
68HC05, 320
68HC11, 320, 322, 335, 346, 347
68HC12, 335, 337
68HC16, 335
8051, 317-, 322
80x86, 177, 239

A

ABAM, 123
Abu-Mostafa, 32-
aceleradores, 35, 165, 334-
adalina, 55-
ADALINE, véase *adalina*
adaptabilidad, 12
adatron, 55
agrupamiento (*clustering*), 86, 227
agrupamiento (*k-means*), 155, 255
agrupamiento borroso, 255
ajuste al ruido, 73-
algoritmo auto-organizado jerárquico, 155
algoritmos constructivos, véase *arquitecturas evolutivas*
algoritmos genéticos, 35, 218, 221, 284-5, 287-

algoritmos genéticos (selección en), 289-290
algoritmos genéticos desordenados, 295, 299-300
ALPHA (DEC), 179
Amari, 123
Amit, 123
amplificador de transconductancia, 194
amplificador operacional, 193-
amplificadores, 145-8
análisis de componentes principales, 86, 227
análisis discriminante, 78, 227
análisis exploratorio de datos, 87
ANFIS, 284, 313
ANS, xxv, 3
aplicaciones, 36-7, 79-, 89, 100-6, 140-5, 148-151, 228-, 228-234, 234-8, 341-, 344-5
aprendizaje, 9
aprendizaje BP, 66-, 211, 283, 285-
aprendizaje BP en sistemas borrosos, 285-7
aprendizaje en serie (*on line*), 69, 211
aprendizaje en sistemas borroso, véase *sistemas borrosos adaptativos*
aprendizaje hebbiano, 42-, 87
aprendizaje híbrido, 28

aprendizaje no supervisado (auto-organizado), 27-8, 77-
 aprendizaje por corrección de errores, 52
 aprendizaje por lotes (*batch*), 68-, 211
 aprendizaje reforzado (premio-castigo), 28
 aprendizaje supervisado, 27, 41-
 aproximación estocástica, 58-
 aproximación funcional, 64-
 arquitecturas de redes neuronales, 21, 22
 arquitecturas evolutivas, 34-5, 76, 219
 arquitecturas fractales, 183
 arquitecturas orientadas a bus, 180
 arquitecturas reconfigurables, 184-5
 arquitecturas sistólicas, 181-182
 ARS, 326-
 ART, 87, 160, 213
 ASIC, 168
 ASSOM, 99
 asociador lineal, 42-, 55
 atractor, véase *estado estable*
 autoorganización, xxx, xxxii, 85-
 autoorganización (análisis), 108-
 axón, 5-

B

backpropagation, véase *BP*
 Balboa-860, 167
 BAM, 123, 160
 base de reglas, 261-2
 Baum, 74
bias, 18-9
 borrosidad y probabilidad, 268
 borrosificador (*fuzzifier*), 264-5
 borrosificador no singleton, 265
 borrosificador singleton, 264
 BP, 41, 51, 63-, 66-
 BP (variantes), 70-
 BSB, 160

C

capa oculta, 22

capa, 21-2
 capacidad de la red de Hopfield, 136, 141-2
 Carpenter, 87
Cascade Correlation, 35
 CCD, 190-1
 cerebro, xxvi-xxvii, xxix-, 3-, 10
 cibernética, xxiii-, xxv
 CICS, 168
Cinco Días (periódico), 229
 clasificación de ANS, 30-
 cláusula, 300
 CLB, 184
clustering, véase *agrupamiento*
 CMAC, 160
 CNAPS (Adaptive Solutions Inc.), 36, 201-202
 CNN, 199-200
 co-norma triangular, 256
 cóclea electrónica, 199
 codificación en los genes, 294
 codificación en un algoritmo genético desordenado, 300-
 cognitrón, 160
 complemento de un conjunto borroso, 256
 composición borrosa, 258-9
 composición Sup-Star, 258, 263
 computabilidad neuronal, 32-
 computador, xxv, 10
 computadores reconfigurables, 185
 conexión excitatoria, 7, 22
 conexión inhibitoria, 7, 22
 conexiones entre capas, 22
 conjunto α -corte, 249
 conjunto normalizado, 249
 conjunto singleton, 249, 250
 conjunto soportado, 249
 conjuntos borrosos, 244, 248
 contrapropagación (red de), 100, 160
 controladores borrosos, 276-
 controladores basados en modelado borroso, 278, 280

controladores borrosos auto-organizados, 278, 279-280
 controladores borrosos con auto-aprendizaje, 278, 280
 controladores borrosos directos con optimización, 278-
 controladores borrosos directos sin optimización, 276-
 controladores borrosos híbridos, 281
 coprocesador neuronal, 167
 correlación, 112, 117
 córtex somato-sensorial, 89
 CPS y CUPS, 172
 CPU, xxvii-
 Cray, 166
 crisis bancaria española, 82-4, 104-6
 cuantificación de los pesos, 111
 cuantificación vectorial, 87, 106-8
 CubiCalc, 323-4
 cuenca de atracción, 135
 chips borrosos, 334-, 338
 chips neuronales, 36, 169-

D

DARPA, 169-170, 205
 decaimiento de pesos, 75
 demanda de consumo eléctrico, 234-8
 dendrita, 5-
 densidad de probabilidad, 91-2, 101-, 110
 desarrollo de Fourier, 66
 desborrosificador (*defuzzifier*), 265
 desborrosificador por centro de área, 265
 desborrosificador por máximo, 265
 desborrosificador por media de centros, 265
 descenso por el gradiente, 57, 59-
 dilema plasticidad-estabilidad, 61
 dilema varianza-sesgo, 76, 214
 dimensión VC, 77
 dinámica asíncrona, 25
 dinámica de la red de Hopfield, 125, 127
 dinámica estocástica, 25

dinámica síncrona, 25
 dinámicas, 25, 34
 disparo (patrones), 9
 dispositivo de umbral, 19-20
 dispositivos de inferencia borrosa, 262-4
 distancia de Hamming, 113
 distancia de Mahalanobis, 16, 113
 distancia de Manhattan, 16, 112, 116
 distancia entre conjuntos borrosos, 255
 distancia euclídea, 16, 112, 115
 Domany, 123
 dominios de Voronoi, 91
 DSP, 167, 172-3

E

EDA, 326-
 EEG, 210
 embebimiento de un patrón, 139
 emulación, 35, 162, 166-, 203-
 emuladores borrosos, 335-
 encéfalo, 5
 energía de Lyapunov, véase *función de Lyapunov*
 ENIAC, xxiv
 entorno de desarrollo (concepto), 311
 entornos de desarrollo de sistemas borrosos, 307, 316-
 entornos de tipo matemático, 311-
 entropía borrosa, 255
 error de aprendizaje-test, 71, 74
 erupciones solares, 210
 escalado, véase *preprocesamiento*
 escalas multidimensionales, 78, 227
 esquema, 299-300
 estabilidad, 28-, 122-3
 estadística, 76-, 225-7
 estado estable, 122, 128-
 estados espurios, 135
 ETANN 8170NX (Intel), 197-8, 205
 expansión señal-ruido, 45, 134-5
 extracción de reglas, 227, 287

F

F-15 (avión de combate), 208
 FAM, 261-2, 273
 Farmer, 33-4
 fenotipo, 289
 FIDE, 318-321
 FIS, véase *sistemas de control borroso*
 FLC, véase *sistemas de control borroso*
 Flynn (clasificación de), 177
 FPGA, 168, 184, 203-
 Frabetti, xxxiv
 Fukushima, 87, 88
 función coste, 57-
 función de activación, 14, 16-7
 función de Heaviside, véase *función escalón*
 función de idoneidad, 297, 300
 función de inclusión gaussiana, 267-8
 función de inclusión, véase *función de pertenencia*
 función de Lyapunov, 29, 92, 109
 función de pertenencia (inclusión), 244, 248, 249-253
 función de salida, 14, 18
 función de transferencia, 16
 función energía de la red, 129-, 148
 función error, véase *función coste*
 función escalón, 48
 función trapezoidal, 250
 función vecindad, 91, 96-
 funciones de base radial, véase *RBF*
 funciones T, S, π , 250-3
 fusión de tecnologías, 342-3, 348-9
Fuzzy, véase *borroso*
 FuzzyTECH, 316-7, 351

G

gen, 289
 generalización, 28, 71-
 generalización-memorización, 71
 genoma, 288
 genotipo, 289

glia, 9
 Golgi, 3
 gradientes conjugados, 70
 gradientes conjugados escalados, 70
 Graf, 201
 Grey Walter, xxiii
 GRNN, 158, 160, 213
 Grossberg, 88, 124
 grupo neuronal, 21
 Grupo PDP, 12-3, 51, 63

H

HDL, 168
 Hebb, véase *regla de*
 Hertz-Krogh-Palmer, 88, 123
 Hetch-Nielsen, 65, 100
 Hinton, 63
 hipermapa, 100
 Hodgkin y Huxley, 37-
 Hopfield, 47, 121, 123
 Hopkins, xxiii
 Hornik, 65

I

i860 (Intel), 167, 173
 IDEA, 325
 IEEE *Neural Network Society*, 13
 implicación borrosa, 260-1, 263
 individuo, 288
 inferencia borrosa, 257- (véase también *dispositivos*)
 inicialización de los pesos, 216
instar (modelo), 100
 inteligencia artificial (IA), xxiv, xxviii, xxxi, 223
 inteligencia computacional, xv, xix, xxvi, xxxi, xxxiv, 6, 34-35, 78, 224, véase también *soft computing*
 intersección de conjuntos borrosos, 256
 iones, 5-

K

Kohonen, 85-, 87-8, 88-

L

LCA, 185

lenguajes de programación, 308

Levenberg-Marquardt, 70

ley de Dale, 7

leyes de Kirchhoff, 193

leyes de Morgan, 256

LIFE, 247

Linsker, 87

Lippmann, 41, 52

Lneuro (Philips), 200-1

lógica borrosa, 243-, 246

lógica de pulsos, 185-6

Luttrell, 108

LVQ, 100, 104, 213

Lyapunov, véase *función de Lyapunov*

M

MAC, 172, 195

MacClelland, 12, 63

MADALINE, 63

maldición de la dimensionalidad, 75

Malsburg (van der), 87, 88

Mamdani, 246, 342

mapa fonético, 103

mapas autoorganizados, 85, 88-, 213

mapas autoorganizados (algoritmo), 92-9

mapas autoorganizados (modelos de aprendizaje), 113-, 120

mapas autoorganizados que crecen, 99

mapas de Kohonen, véase *mapas autoorganizados*

mapas de rasgos, véase *mapas autoorganizados*

mapping, 28, 64

máquina de Boltzmann, 142-5

máquina del campo medio, 145

máquina von Neumann, xv, xxiii-, 32, 341

MathCAD, 315

Mathematica, 313-5

MATLAB, 312-3

matriz hessiana, 70

matriz pseudoinversa, 46

McCarthy, xxiv

McCulloch, 13

Mead, 196-7, 207

mecanismo de consciencia, 99

medidas borrosas, 254-5

medidas de similitud, 111

memoria asociativa borrosa, véase *FAM*

memoria asociativa, 23, 86, 127-

memoria autoasociativa, 23, 128, 129

memoria de acceso directo, 127

memoria distribuida, 12

memoria heteroasociativa, 37, 128

métrica de Minkowski, 112

microcontroladores, 305, 334-

MICRONEURO (congresos), 208

microprocesador, xxvi-

microprocesadores borrosos, 335-6

MIMD, 177

MIMO y MISO, 262

Minsky y Papert, 42, 50

Minsky, xxiv, xxv

MLP, véase *perceptrón multicapa*

modelo de Hopfield continuo

(analógico), 145-,

modelo de Hopfield discreto, 123-, 134

modelos competitivos, 87

modelos evolutivos, 34-35

modelos híbridos, 86, 121, 151-

modificadores concentración y dilatación, 256-7

modo aprendizaje, 26-, 28-, 34

modo recuerdo (ejecución), 26-, 28-, 34

Modus Ponens Generalizado, 259-

Modus Tolens Generalizado, 259-

monitorización de procesos, 104-

MS-DOS, 308

multilayer perceptron, véase *perceptrón multicapa*
 Müller y Reinhardt, 123
 Murray, 201

N

NAND, 20, 32-3
 Nanotecnología, 163
 Neocognitrón, 87, 160
 nervio, 5-
 Neuro-WSI (Hitachi), 186
 neurocomputadores, 35, 169-, 176-
 neurochips, véase *chips neuronales*
 neurona (biológica), 3-, 37-40
 neurona (modelo dinámico), 39-
 neurona (modelo eléctrico), 37-40
 neurona (modelo estático), 40
 neurona artificial (formal), 13-, 37-40
 neurona de Hopfield, 124
 neurona de Kohonen (modelos), 111-
 neurona de orden superior, 16
 neurona de silicio, 208
 neurona estándar, 18-
 neurona estocástica, 25
 neurona Ising, 15, 125
 neurona McCulloch-Pitts, 15, 20, 48
 neurona NAND, 20, 49-
 neurona oculta, 25
 neurona post-sináptica, 15
 neurona Potts, 15
 neurona pre-sináptica, 15
 neurona sigma-pi, 16
 neurona sigmoidea, 21
 neurona todo-nada, 19
 NeuroSolutions (simulador), 164, 221, 351
 neurotransmisores, 6-
 norma triangular, 256
 número de patrones, 74

O

objetivos (*target*), 43-

Oja, 119
 operaciones entre conjuntos borrosos, 255-, 257
 operadores genéticos, 290, 295-
 operador genético corta y empalma, 303
 operador genético cruzamiento, 296-7
 operador genético mutación, 296-7, 303
 optimización, 148-
 OR exclusivo, véase *XOR*
 ordenadores personales, 308

P

Parker, 63
 partición completa, 254
 particiones borrosas, 254
 PCA, véase *análisis de componentes principales*
 PE, 177-
 péndulo inverso, 271-6
 perceptrón multicapa, 41, 51, 63-
 perceptrón simple, 47-
 período refractario, 6
 peso sináptico, 13, 15
 picos epilépticos, 210
 PILAR (robot autónomo), 347-8
 plasticidad sináptica, 9
 población, 288
 podado de pesos, 75
 potencial de acción, 6-
 potencial post-sináptico, 15
 PowerPC (Motorola-IBM), 179
 preprocesamiento, 216-7
 principio de extensión, 258
 problema del viajante de comercio (TSP), 149-
 problemas mal condicionados, 76
 procesador elemental, 13
 procesamiento paralelo, 11
 programación estructurada, 309
 programación orientada a objeto, 309
 programas comerciales, 164, 220-223
 programas de libre distribución, 221-3
 pseudoinversa, 46, 137

PU, 171, 177-
punto fijo, véase *estado estable*
puntos de cruce, 249

Q

QuickProp, 71
quiebra bancaria, 82-84

R

Ramón y Cajal, 4
razonamiento directo e inverso, 259-260
RBF, 78, 121, 151-, 213
realimentación, véase *redes realimentadas*
reconocimiento de caracteres, 142, 230
red de potenciales gaussianos, 158
red neuronal, 22, 23-
red virtual, 173
redes auto-organizadas, 85-
redes competitivas, véase *modelos competitivos*
redes de regresión generalizada, véase *GRNN*
redes monocapa, 22-3
redes multicapa, 22-3
redes neuronales, xv-, xxxiii-, 1-
redes no supervisadas hebbianas, 87
redes PCA, 87, 119
redes realimentadas (*feedback*), 23, 25, 121-
redes recurrentes, véase *redes realimentadas*
redes supervisadas, 41-
redes unidireccionales (*feedforward*), 23, 25, 41-
regiones de decisión, 49, 50, 52
regla de aprendizaje, 44
regla de Hebb, 44-, 132-
regla de la pseudoinversa, 45-, 137-
regla de los mínimos cuadrados, véase *regla LMS*
regla de propagación, 14, 15

regla de Widrow-Hoff, véase *regla LMS*
regla del perceptrón, 52-
regla delta generalizada, véase BP
regla delta, véase *regla LMS*
regla LMS, 47, 56-, 140, 157
reglas borrosas, 261-2
reglas heurísticas, 245
reglas IF-THEN, 262, 273
reglas tipo Mamdani, 262
reglas tipo Sugeno, 262
regresión lineal, 216
regresión, 78, 227
regularización, 77
relación borrosa, 258
retina de silicio, 198-
retropropagación, véase BP
RISC, 167, 179
ritmo de aprendizaje, 53, 96, 218
robots autónomos, 343-
Rosenblatt, 47, 55
RSC-164 (Sensory Circuits Inc.), 231
Rumelhart, 12, 63, 88

S

salida de un dispositivo de inferencia borrosa. 264
Sammon (*mapping* de), 99
selección automática de las dimensiones de los rasgos, 110
señales nerviosas, 6
separabilidad lineal, 48-
Shannon, xxiv
SIENA, 229
SIMD, 177
similitud borrosa, 255
simulación de redes neuronales, 35, 162, 163-6, 220
simulación de sistemas borrosos, véase *entornos de desarrollo*
simulated annealing, véase *templado simulado*
sinapsis, 5-
SISD, 177

sistema conexionista, 12, 33-4
 sistema nervioso, 3-
 sistema neuronal, 12
 sistemas borrosos, xv-, xxiii-, 209, 218,
 225, 241-
 sistemas borrosos (historia), 246-7
 sistemas borrosos (interés de), 342-
 sistemas borrosos (opciones usuales en
 el desarrollo), 266-8
 sistemas borrosos adaptativos, 283-, 287
 sistemas de control borroso (ejemplo),
 271-6
 sistemas de control borroso, 244, 269-,
 véase también *controladores
 borrosos*
 sistemas de inferencia borrosa, véase
sistemas de control borroso
 sistemas dinámicos, 33-4, 122
 sistemas expertos conexionistas, 37
 Sistemas Expertos, xxv, 223
 sistemas neuro-borrosos, 283-, 304, 338,
 341 (véase también *sistemas borrosos
 adaptativos*)
 sistemas operativos, 307
 SNAP, 167
 sobreajuste, 72
 sobreaprendizaje, 72
 sobreespecificación, 302
 SOFM (variantes), 99-
 SOFM, véase *mapas auto-organizados*
Soft computing, xv, xix, xx, xxvi, xxxi,
 223, 339-340, véase también
inteligencia computacional
 soma, 5-
 SPARC, 179
 spin glasses, véase *vidrios de spin*
 subespecificación, 301
 suma ponderada, 15
 SuperSAB, 71
 SVM (*support vector machines*), 77, 160
 Synapse (neurocomputador de Siemens),
 36, 167, 173, 201-2

T

T-conorma, véase *co-norma triangular*
 T-norma, véase *norma triangular*
 Tanimoto (medida de), 113
 templado simulado, 144, 151, 218
 teorema de Cohen-Grossberg, 30
 teorema de convergencia (Bruck), 130-1
 teorema de Funahashi, 167
 teorema de Greville, 47, 138
 teorema de Kolmogorov, 65
 teoremas de estabilidad, 28-, 122-3
 teoría clásica de conjuntos, 256
 teoría estadística de aprendizaje, 77
 término de momento, 70
 TILShell, 320-2
 TMS320 (Texas Instruments), 173, 317,
 334
 TOTEM PCI (neurocomputador), 167,
 203, 208
 transformada de Gabor, 66
 Turing, xv, xxiv, xxvii, 32

U

umbral de disparo, 7-
 umbral, 18-9
 unicidad neuronal, 328
 unión de conjuntos borrosos, 256
 Universidad de Zaragoza, 82-84, 104-
 116, 203-6, 234-8, 322-, 345-8
 universo de discurso, 248

V

validación cruzada, 72, 215
 variable lingüística, 253
 vecindad, véase *función vecindad*
vector quantization, véase *cuantificación
 vectorial*
 vidrios de spin, 126
 VLSI, 35, 124, 159, 162-, 174-
 VQP, 99

W

wavelets, 66
WEBSOM, 106
Werbos, 51
White, 226-7
Widrow, 55
Wiener, xxiii
winner-take-all, 87
WSI, 159

X

XOR, 20-1, 50-, 79-81

Z

Zadeh, xviii, xix-, xxi-, xxvi, xxxii, 246,
247
ZISC036 (IBM), 202