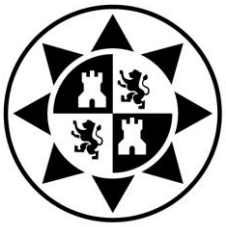


Universidad
Politécnica
de Cartagena

industriales
etsii UPCT

Desarrollo software de técnicas para el diseño automático de Redes Neuronales Artificiales con bajo coste computacional

Titulación: Ingeniero Técnico Industrial

Intensificación: Electrónica industrial

Alumno/a: Juan Manuel Muñoz Olmo

Director/a/s: Joaquín Roca González

Pedro J. García Laencina

Cartagena, 15 de Mayo de 2013

Autor	Juan Manuel Muñoz Olmo
E-mail del Autor	jmmo@alu.upct.es
Director(es)	Joaquín Roca González
E-mail del Director	Jroca.gonzalez@upct.es
Codirector(es)	Pedro José García Laencina
	pedroj.garcia@cud.upct.es
Título del PFC	Desarrollo software de técnicas para el diseño automático de redes neuronales artificiales con bajo coste computacional
Descriptores	RNAs, neural networks, ELM, low computational cost, Automatic Software development, Algorithms, Matlab.

Resumen

Las técnicas de inteligencia computacional, entre las que destacan las Redes Neuronales Artificiales (RNAs), han sido utilizadas con éxito en multitud de contextos y problemas procedentes de la ingeniería de control, la economía, la medicina, etc. Sin embargo, generalmente, las RNAs presentan diversos inconvenientes durante el proceso de optimización de los hiper-parámetros (i.e., pesos y número de neuronas), como son el elevado tiempo computacional requerido y la convergencia a soluciones sub-optimas (mínimos locales) para un determinado problema al usar técnicas de optimización basadas en gradiente.

Recientemente, el algoritmo de entrenamiento conocido como Extreme Learning Machine ha permitido solventar estos claros inconvenientes que presentan las técnicas tradicionales de entrenamiento de RNA. En este PFC se desarrollan nuevas técnicas computacionales para el diseño y la combinación automática de RNAs utilizando el novedoso y eficiente algoritmo ELM. Implementando un conjunto de funciones y herramientas software, en el entorno de programación MATLAB, que permitan diseñar automáticamente RNAs con altas prestaciones en términos predictivos y, al mismo tiempo, con bajo coste computacional.

Hay que resaltar que este PFC forma parte de un trabajo de investigación de un mayor alcance, donde se están desarrollando nuevos y eficientes sistemas y algoritmos de inteligencia computacional, que está siendo desarrollado fundamentalmente en el Centro Universitario de la Defensa (CUD) de San Javier.

Titulación	Ingeniero Técnico Industrial, Especialidad Electrónica Industrial
Intensificación	
Departamento	Tecnología Electrónica
Fecha de Presentación	Mayo de 2013

A la ciudad de Cartagena y su Gente.

Índice

Lista de Figuras

Lista de tablas

Capítulo 1

Introducción

1.1 Motivación.

Las técnicas de inteligencia computacional, entre las que destacan las Redes Neuronales Artificiales (RNAs), han sido utilizadas con éxito en multitud de contextos y problemas procedentes de la ingeniería de control, economía, medicina, etc. Así, por ejemplo, las RNAs han sido empleadas para control automático, detección de señales, estimación de variables financieras, filtros 'antispam', diagnosis médica, diseño de fármacos, etc.

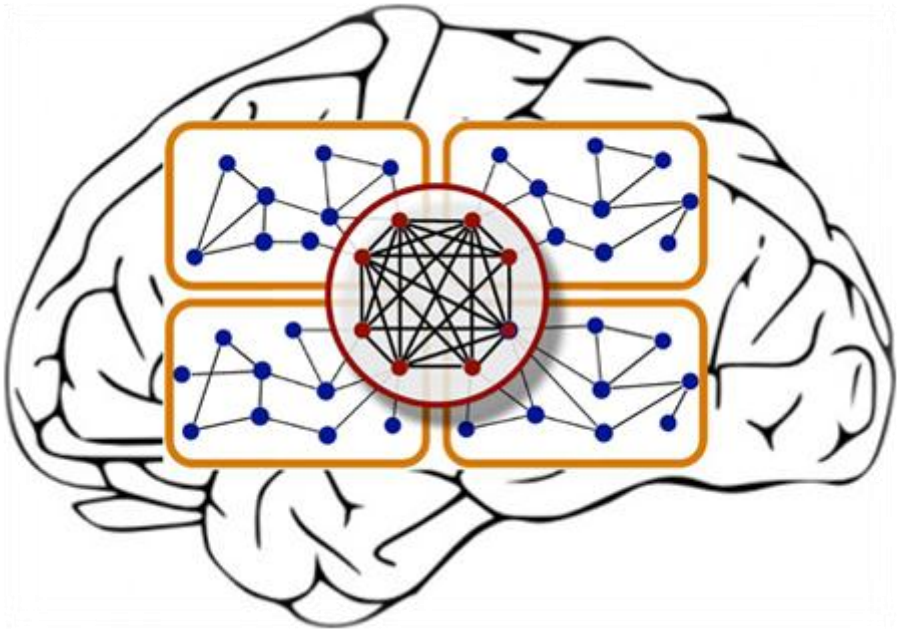


Figura 1.1 Aplicaciones de Redes Neuronales para estudio y tratamiento del ADN

Sin embargo, generalmente, las RNAs presentan diversos inconvenientes durante el proceso de optimización de los hiper-parámetros (i.e., pesos y número de neuronas), como son el elevado tiempo computacional requerido, y la convergencia a soluciones sub-optimas (mínimos locales) para un determinado problema al usar técnicas de optimización basados en gradiente [1]. Recientemente, el algoritmo de entrenamiento conocido como Extreme Learning Machine [2], [3] ha permitido solventar estos claros inconvenientes que presentan las técnicas tradicionales de entrenamiento de RNA.

1.2 Descripción del proyecto y objetivos.

En lugar de emplear una única RNA para modelar un determinado problema, un procedimiento muy recomendable es modelar dicho problema mediante un conjunto de RNAs, cuya solución viene dada por la combinación de la salida obtenida por cada una de las RNAs [4]. Siguiendo este enfoque, en este PFC se desarrollarán nuevas técnicas computacionales para el diseño y la combinación automática de RNAs utilizando el novedoso y eficiente algoritmo ELM.

Se implementarán un conjunto de funciones y herramientas software, en el entorno de programación MATLAB, que permitan diseñar automáticamente RNAs con altas prestaciones en términos predictivos y, al mismo tiempo, con bajo coste computacional, independientemente de los conocimientos previos del usuario final sobre inteligencia computacional y RNAs.

Hay que resaltar que este PFC forma parte de un trabajo de investigación de un mayor alcance, donde se están desarrollando actualmente eficientes sistemas y algoritmos de inteligencia computacional. En concreto, el trabajo de investigación está siendo desarrollado fundamentalmente en el Centro Universitario de la Defensa (CUD) de San Javier.

1.3 Desarrollo.

Este PFC intenta explicar de un modo tan claro como me sea posible, la investigación, desarrollo y conclusiones obtenidas durante varios meses de trabajo. También posee una componente académica, para que futuros alumnos puedan utilizar éste para continuar y avanzar en esta línea de investigación. Su estructura se fija según estos objetivos, de una manera en la que el autor cree que mejor se describen las diferentes partes del proceso desarrollado.

Este documento ha sido escrito de un modo tal que, pueda ser entendido por una persona que posea conocimientos de matemáticas, programación, control, y esté familiarizado con la teoría y técnicas básicas de Inteligencia computacional.

El Capítulo 2 se ocupa de la teoría básica referente a la Inteligencia Computacional y su estado del arte, prestando especial atención a las RNAs, y su fundamento teórico. A continuación en el Capítulo 3 se proporciona una breve revisión de las técnicas tradicionales disponibles para el entrenamiento de éstas últimas, que nos preparará para ir adentrándonos en un análisis exhaustivo del algoritmo ELM, y en el estudio y diseño de técnicas de combinación basadas en éste, que tomarán lugar en el Capítulo 4 y el Capítulo 5 respectivamente.

Tras este desarrollo teórico previo, procederemos en el Capítulo 6 a la implementación en MATLAB© de éstas técnicas, teniendo siempre como meta conseguir altas prestaciones manteniendo un bajo coste computacional. Comprobando la validez de nuestro código bajo distintos experimentos de verificación, abordando problemas reales de carácter multidisciplinar.

En el Capítulo 7, aportaremos las conclusiones obtenidas en este trabajo, y comentaremos las futuras líneas de investigación y desarrollo a abarcar en el futuro.

Capítulo 2

Análisis del estado del arte en Inteligencia Computacional

2.1 Inteligencia Computacional.

2.1.1 Introducción y definición.

El campo de la Inteligencia Computacional (IC) es el sucesor de la Inteligencia Artificial (IA) [5], combina los distintos elementos del aprendizaje, evolución, y lógica difusa para crear programas que exhiban, en cierto modo, algún grado de comportamiento inteligente en entornos cambiantes y complejos.

En los últimos años se ha experimentado un crecimiento teórico y práctico en este campo, con la aparición de nuevas ideas, aplicaciones y patentes [6]. Antes de adentrarnos más en materia, deberíamos reflexionar un poco sobre el concepto de Inteligencia. Ha habido multitud de intentos de llegar a una definición sobre ésta, y aún, hoy día, este tema sigue provocando un gran debate al respecto. Los diccionarios definen la Inteligencia como la habilidad para comprender, entender y aprovecharse de la experiencia, también se interpreta la inteligencia, como la capacidad para el pensamiento y la razón (especialmente a un gran nivel). Otras palabras clave que describen aspectos de la Inteligencia incluyen la creatividad, la habilidad, la conciencia, la emoción y la intuición.

La siguiente cuestión a tener en cuenta, sería si las máquinas (ordenadores, robots, etc…) pueden ser inteligentes. Este tema, es fruto aún de más controversia que la propia definición de Inteligencia. Una definición más reciente de la inteligencia artificial, proviene del consejo de Redes Neuronales del IEEE de 1996. “El estudio de cómo diseñar ordenadores que hagan cosas que la gente hace mejor”. Aunque esta definición no parezca acertada, podemos extraer que el principal objetivo de la IA es intentar imitar la capacidad de la mente humana para resolver problemas.

2.1.2 Breve reseña histórica.

Aristóteles (384-322 A.C.) fue seguramente el primero en aproximarse al concepto de IA, su objetivo fue el de explicar y codificar los estilos del razonamiento deductivo, al que se refería como *Silogismos*. Ramón LLul (1232-1315) desarrolló en el "*Ars Magna*" un optimista intento de construir una máquina con la capacidad de responder todas las preguntas. Este concepto, hoy en día, aun aproximándonos queda lejos de estar resuelto, un ejemplo sería el software SIRI© de Apple Inc. © con ejemplos de respuestas de éste, que quedan bastante lejos de las respuestas esperada a priori por el usuario. El matemático Gottfried Leibniz (1646–1716) razonó sobre la existencia de un Cálculo Filosófico, una álgebra universal que pudiese ser usada para representar todo el conocimiento (incluyendo verdades morales) en un sistema deductivo.

La primera gran contribución fue aportada por George Boole en 1854, con su desarrollo de los fundamentos de la Lógica Proposicional. En 1879, Gottlieb Frege estableció la base del Cálculo de Predicados. Ambas teorías constituyeron las primeras herramientas de la IA.

A mediados del siglo XX, Alan Turing aportó bastante a la cuestión de la definición de IA y el concepto de Vida Artificial [7]. El creía que se crearían máquinas que imitasen el cerebro humano. Pensaba firmemente, que no habría nada que un ordenador bien diseñado pudiese hacer, del mismo modo que un cerebro humano. Estos conceptos son todavía hoy en día visionarios, encontrándonos que se ha logrado modelar pequeñas partes de sistemas neuronales biológicos, y no encontrando todavía soluciones a los complejos problemas de la intuición, consciencia y emoción, que forman parte integral de la inteligencia humana.

El término IA fue acuñado por primera vez en 1956 en la Conferencia de Dartmouth, organizada por John MacCarthy, al que se considera el padre de la IA. Desde esta fecha hasta finales de los 70, hubo mucho trabajo en el modelado de neuronas cerebrales. Muy notable fue el trabajo sobre el *Perceptrón* de Rosenblatt [8], y el *Adaline* (Elemento Lineal Adaptativo) por Widrow and Hoff. [9] En 1969 [9], Minsky y Papert propiciaron un importante revés para la investigación de redes neuronales artificiales, con su trabajo *Perceptrones* [10], en el que alegaban que, basándose en su "juicio intuitivo", la extensión de perceptrones simples a perceptrones multicapa era estéril. Esto desencadenó la llamada "época oscura" en los trabajos de investigación sobre redes neuronales, hasta que a principios y mitad de la década de los ochenta Hopfield [11], Hinton [12], y Rumelhart and McLelland [13] marcaron un hito con sus publicaciones. Y propiciaron que hoy en día la IC se haya convertido en una de las áreas más grandes de investigación en todo el mundo.

2.2 Paradigmas de IC.

Vamos a proceder con una breve introducción de cada uno de los elementos que componen los paradigmas de la Inteligencia Computacional; la Lógica difusa, los Algoritmos evolutivos y las RNAs, prestando especial atención a estas últimas. Aunque hay autores que consideran a la Inteligencia de Enjambre y a los Sistemas Inmunes Artificiales como parte de la inteligencia Computacional, en este PFC al contrario, se engloba a estos últimos en el campo de la Vida Artificial, con lo que no formarán parte de este capítulo.

2.2.1 Lógica Difusa.

La teoría de conjuntos tradicional requiere de elementos que pueden ser o no, partes de un conjunto. De manera similar, la lógica binaria necesita de valores que pueden tomar los valores de 0 o 1. El razonamiento humano no se comporta, casi siempre, de una manera tan exacta. Nuestras observaciones y razonamientos suelen incluir en cierta medida una componente de incertidumbre.

La Lógica Difusa (también llamada lógica borrosa o lógica heurística) nos permite lo que se conoce un *razonamiento aproximado*. Con los *conjuntos borrosos*, un elemento pertenece a un conjunto con un cierto grado de certeza. Permitiéndonos deducir nuevos hechos a partir de hechos inciertos, con un grado de certeza asociado a cada hecho. En este sentido, nos proporcionan una alternativa para tomar decisiones basadas en un conocimiento escaso, impreciso o inexistente. Aplicándose con éxito a los sistemas de control, transmisión de engranajes y a los sistemas de frenos de vehículos, elevadores de control, electrodomésticos, control de señales de tráfico, aplicaciones de seguimiento, ranking dinámicos, y otros muchos.

2.2.2 Algoritmos Evolutivos.

Esta técnica es un proceso guiado de búsqueda aleatoria para problemas que presentan un reto debido a espacios grandes y complejos. Está inspirada en los mecanismos de evolución biológica, tales como la reproducción y la mutación; los fuertes se adaptan y los débiles mueren., que crean grandes conjuntos de soluciones, *cromosomas*, a un problema, el cuál evoluciona tras varias generaciones hasta alcanzar soluciones próximas a la óptima que solucionan suficientemente un problema.

Los *cromosomas* definen las características de los individuos de cada población, cada características es conocida como *gen*. El valor de estos genes son llamados *alelos*. Una función de adecuación evalúa a todos los individuos (soluciones al problema) y las mejores soluciones tienen una mayor oportunidad de reproducirse en la generación

siguiente. Otros métodos de reproducción incluyen operadores genéticos como el cruce y la mutación.

La computación evolutiva se ha utilizado con éxito en aplicaciones del mundo real, por ejemplo, en minería de datos, la optimización combinatoria, diagnóstico de fallas, clasificación, clustering, planificación, y la aproximación de series temporales.

2.2.3 Redes Neuronales Artificiales.

Las RNAs son sistemas implementados de tal manera que funcionen como las neuronas biológicas de los seres vivos. El cerebro humano continuamente recibe señales de entrada de muchas fuentes y las procesa a manera de crear una apropiada respuesta de salida. Nuestros cerebros cuentan con millones de neuronas que se interconectan para elaborar "Redes Neuronales". Las RNAs fueron originalmente una simulación abstracta de los sistemas nerviosos biológicos, formados por un conjunto de unidades llamadas "neuronas" o "nodos" conectadas unas con otras. Estas conexiones tienen una gran semejanza con las dendritas y los axones en los sistemas nerviosos biológicos.

2.2.3.1 La neurona biológica.

Una neurona (o célula nerviosa) es una célula biológica especial que procesa información (Figura 2.1). Está compuesta de un cuerpo celular, o soma, y dos tipos de ramas que llegan al exterior: el axón y las dendritas. El cuerpo celular tiene un núcleo que contiene información acerca de rasgos hereditarios y un plasma que sostiene el equipo molecular para producir material necesario para la neurona. Una neurona recibe señales (impulsos) de otras neuronas a través de sus dendritas (receptores) y transmite señales generadas por su cuerpo celular a lo largo del axón (transmisor), el cual eventualmente se divide en ramales y sub-ramales. En los extremos de estos ramales están las sinapsis.

Una sinapsis es una estructura elemental y una unidad funcional entre dos neuronas (un ramal axón de una neurona y una dendrita de otra). Cuando el impulso alcanza el extremo de la sinapsis, ciertos químicos llamados neurotransmisores son liberados. Éstos se difunden a través del espacio sináptico, para aumentar o inhibir, dependiendo del tipo de la sinapsis, la tendencia de la neurona receptora a emitir impulsos eléctricos. La efectividad de la sinapsis puede ser ajustada por las señales que pasan a través de ella de tal manera que las sinapsis pueden aprender de las actividades en las que participan. Esta dependencia de la historia actúa como una memoria, la cual es posiblemente responsable de la memoria humana [14].

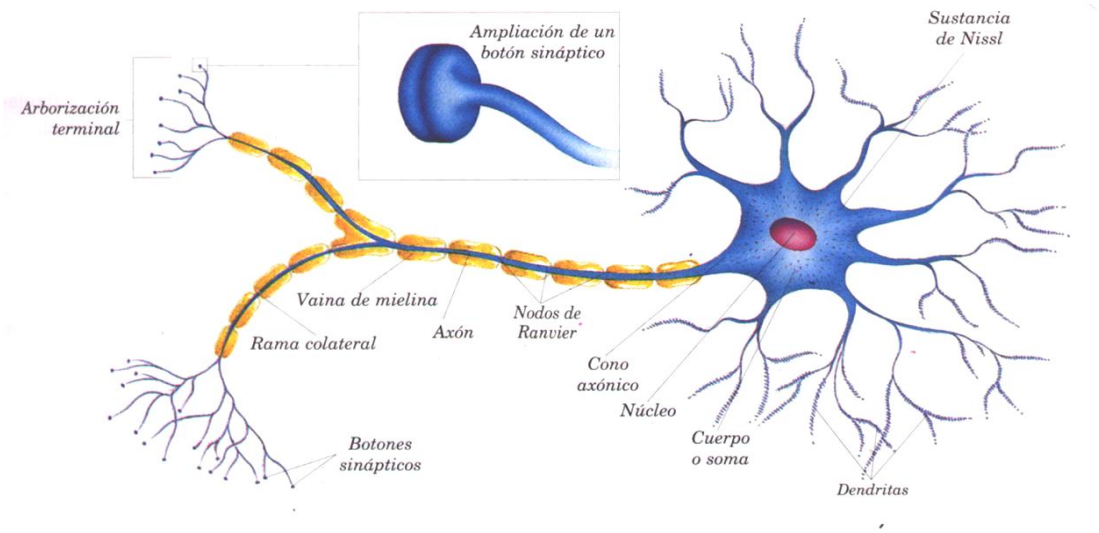


Figura 2.1 Diagrama de una neurona biológica.

La corteza cerebral contiene cerca de 10^{11} neuronas, masivamente conectadas. Cada neurona está conectada con otras 10^3 a 10^4 neuronas. En total, el cerebro humano contiene aproximadamente 10^{14} a 10^{15} interconexiones. Las neuronas se comunican a través de un corto tren de pulsos, normalmente de milisegundos de duración, a una velocidad de conmutación mucho más lenta que en circuitos electrónicos. ¿Cuándo será posible modelar completamente un cerebro humano? No de momento.

Las neuronas artificiales usadas para construir RNAs son mucho menos complejas que las encontradas en el cerebro, problemas con objetivos simples pueden ser resueltos fácilmente empleando RNAs de un tamaño moderado pero su capacidad estará limitada por la potencia y el espacio de memoria de los sistemas de computación modernos. Por lo que todavía queda un largo camino por recorrer.

2.2.3.2 La neurona artificial.

La unidad básica de una RNA es la neurona. Aunque existen varios tipos de neuronas diferentes, la más común es la de tipo McCulloch-Pitts [15]. En la siguiente figura (figura 2.2) puede verse una representación de la misma.

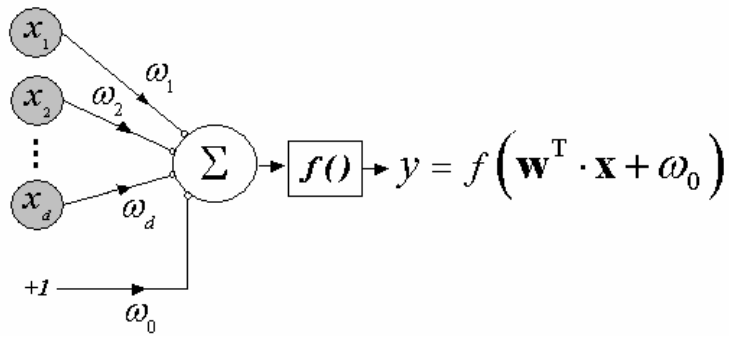


Figura 2.2 Diagrama de una neurona artificial.

Este modelo se conoce como *perceptrón* simple de McCulloch-Pitts, y es la base de la mayor parte de la arquitectura de las RNAs. Una neurona artificial es un procesador elemental, en el sentido de que procesa un vector de entrada $\boldsymbol{x} = \{x_1, x_2, ..., x_d\}$ y produce una respuesta o salida única **y**. Los elementos clave de una neurona artificial son los siguientes:

- Las entradas $\boldsymbol{x}$ que reciben los datos de otras neuronas a las que está conectada.
- Los pesos sinápticos ω_i . En una neurona artificial a las entradas que vienen de otras neuronas se les asigna un peso. Al parámetro ω_0 se le suele conocer *bias* o *sesgo*, y puede ser considerado como otro peso de una entrada extra fijada permanentemente a un valor de +1.
- Una regla de propagación. Con esas entradas y los pesos sinápticos, se realiza una suma ponderada.
- Una función de activación. El valor obtenido con la regla de propagación, se filtra a través de una función conocida como función de activación y es la que nos da la salida de la neurona. Según para lo que se desee entrenar, se suele escoger una función de activación u otra. En la Tabla 2.1 se muestran las funciones de activación más usuales.

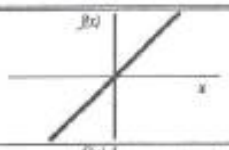
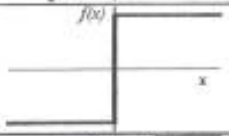
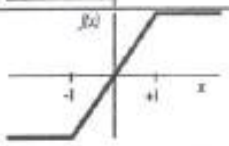
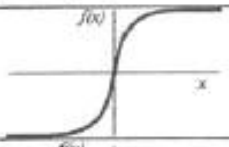
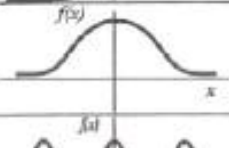
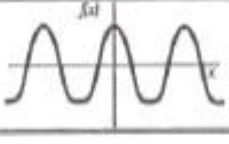
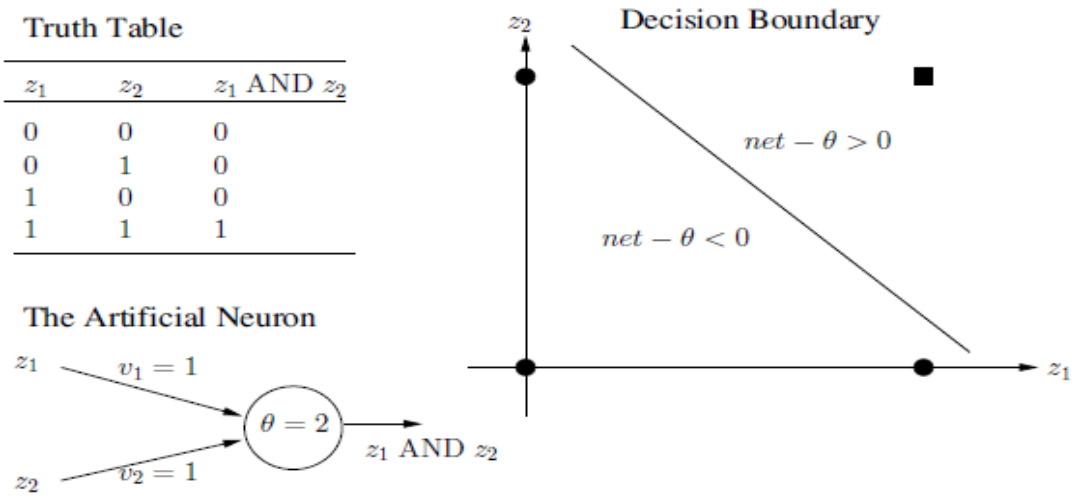
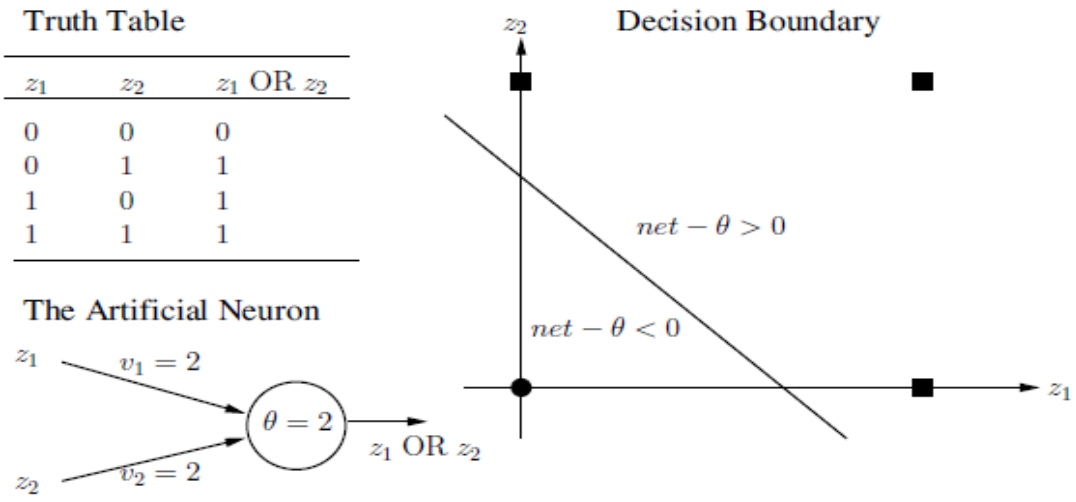
	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	$\{-1, +1\}$ $\{0, +1\}$	
Lineal a tramos	$y = \begin{cases} -1, & si\ x < -1 \\ x, & si\ +1 \le x \le -1 \\ +1, & si\ x > +1 \end{cases}$	$[-1, +1]$	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	$[0, +1]$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

Tabla 2.1 Funciones de activación habituales.

El perceptrón es capaz tan sólo de resolver funciones definidas por un hiperplano (objeto de dimensión N-1 contenida en un espacio de dimensión N) que corte un espacio de dimensión N. Ejemplos los operadores lógicos AND o OR (fig 2.3). Un ejemplo de una función que no puede ser resuelta es el operador lógico XOR. Una explicación más sencilla de un hiperplano sería, hablando en un plano de dos dimensiones, una línea que separa a los elementos existentes en dos grupos. El perceptrón sólo puede resolver una función, si todos los posibles resultados del problema pueden separarse de ésta forma (en dos secciones) es decir, que no se combinen entre sí.



(a) AND Perceptron



(b) OR Perceptron

Figura 2.3 Funciones lineales AND y OR separables por el Perceptrón

2.2.3.3 Tipos y Características.

Varios tipos diferentes de RNAs se han desarrollado, algunos ejemplos de éstos:

• RNAs de una sola capa, tales como la red de Hopfield [11].

• RNAs multicapa [15] feedforward, incluyendo, por ejemplo, las de retropropagación estándar, vínculo funcional y redes de productos unitarios, éstas serán sobre las que nos centremos en los siguientes capítulos de este PFC.

• RNAs temporales, tales como la Elman [16] y Jordan [17] redes simples recurrentes, así como de redes con retardo temporal.

• RNAs auto-organizadas como los mapas de carácter de auto-organizativos de Kohonen [18] y las de aprendizaje cuantificador vectorial.

• RNA combinada mezclando entrenamientos supervisado y no supervisado, por ejemplo, alguna redes de función de base radial [19].

Entre las ventajas que presentan las RNAs podemos encontrar las siguientes:

• Aprendizaje: Las RNA tienen la habilidad de aprender mediante una etapa que se llama etapa de aprendizaje. Esta consiste en proporcionar a la RNA datos como entrada a su vez que se le indica cuál es la salida (respuesta) esperada.

• Auto organización: Una RNA crea su propia representación de la información en su interior, descargando al usuario de esto.

• Tolerancia a fallos: Debido a que una RNA almacena la información de forma redundante, ésta puede seguir respondiendo de manera aceptable aún si se daña parcialmente.

• Flexibilidad: Una RNA puede manejar cambios no importantes en la información de entrada, como señales con ruido u otros cambios en la entrada (por ejemplo si la información de entrada es la imagen de un objeto, la respuesta correspondiente no sufre cambios si la imagen cambia un poco su brillo o el objeto cambia ligeramente).

• Tiempo real: La estructura de una RNA es paralela, por lo cual si esto es implementado con computadoras o en dispositivos electrónicos especiales, se pueden obtener respuestas en tiempo real.

Entre las desventajas presentadas por las RNAs es que éstas deben ser entrenadas para cada problema. Además, es necesario realizar múltiples pruebas para determinar la arquitectura adecuada, el número de capas ocultas, el número de nudos por capa, las interconexiones, la función de transformación, etc. El entrenamiento es largo y puede consumir varias horas de tiempo de computación. Las técnicas estadísticas convencionales, sin embargo sólo requieren la extracción y la normalización de una muestra de datos.

Es un argumento erróneo sostener que el tiempo de desarrollo para los modelos basados en RNAs sea más corto que el tiempo necesario para desarrollar, por ejemplo, una tabla de puntuación basada en una regresión múltiple. Los estudios donde se han constatado un tiempo de desarrollo más corto no han tenido en cuenta el tiempo requerido para la preparación de datos. Las RNAs presentan un aspecto complejo para un observador que desee realizar cambios. Para añadir nuevo conocimiento, es necesario cambiar las interacciones entre muchas unidades para que su efecto unificado sintetice este conocimiento.

2.2.3.4 Aplicaciones

Las RNAs deben ser aplicadas en situaciones en las que las técnicas tradicionales fallan en el intento de alcanzar resultados satisfactorios, o cuando un mejorado en el modelado puede conllevar una diferencia en la eficiencia de la operación del sistema, mejorando la relación coste-beneficio. Debemos considerar hacer uso de las RNAs cuando:

- El número de variables o la diversidad de los datos sean muy grandes.

- Las relaciones entre las variables sean vagamente entendibles.

- Las relaciones entre estas variables sean difíciles de detallar adecuadamente por métodos tradicionales.

- Las capturas o variables presenten semejanzas dentro de un conjunto de patrones, tal como sucede en aplicaciones de procesamiento de señales, de control, de reconocimiento de patrones, producción y reconocimiento del habla, en los negocios, en la medicina, etc.

Algunos ejemplos de aplicaciones reales:

• Predicción: en el mundo real existen muchos fenómenos de los que conocemos su comportamiento a través de una serie temporal de datos o valores. Lapedes y Farber [20] del Laboratorio de Investigación de los Álamos, han demostrado que la red backpropagation supera en un orden de magnitud a los métodos de predicción polinómicos y lineales convencionales para las series temporales caóticas.

• Modelado de Sistemas: los sistemas lineales son caracterizados por la función de transferencia que no es más que una expresión analítica entre la variable de salida y una variable independiente y sus derivadas. Las RNAs también son capaces de aprender una función de transferencia y comportarse correctamente como el sistema lineal que está modelando.

• Reconocimiento de patrones: llamado también lectura de patrones, identificación de figuras y reconocimiento de formas. Los patrones se obtienen a partir de los procesos de segmentación, extracción de características y descripción dónde cada objeto queda representado por una colección de descriptores. La clasificación de éstos es llevada a cabo por RNAs.

• Filtrado de Ruido: las RNAs también pueden ser utilizadas para eliminar el ruido de una señal. Son capaces de mantener en un alto grado las estructuras y valores de los filtros tradicionales.

• Modelos Económicos y Financieros: una de las aplicaciones más importantes del modelado y pronóstico es la creación de pronósticos económicos como por ejemplo los precios de existencias, la producción de las cosechas, el interés de las cuentas, el volumen de las ventas etc. Las RNAs están ofreciendo mejores resultados en los pronósticos financieros que los métodos convencionales.

• Servo-control: un problema difícil en el control de un complejo sistema de servomecanismo es encontrar un método de cálculo computacional aceptable para compensar las variaciones físicas que se producen en el sistema. Entre los inconvenientes destaca la imposibilidad en algunos casos de medir con exactitud las variaciones producidas y el excesivo tiempo de cálculo requerido para la obtención de la solución matemática. Existen diferentes redes neuronales que han sido entrenadas para reproducir o predecir el error que se produce en la posición final de un robot. Este error se combina con la posición deseada para proveer una posición adaptativa de corrección y mejorar la exactitud de la posición final.

Capítulo 3

Revisión de las técnicas tradicionales para el entrenamiento de RNAs

3.1 Arquitectura básica del Perceptrón Multicapa.

Desde un punto de vista matemático, se puede ver una red neuronal como un grafo *dirigido y ponderado* donde cada uno de los nodos son neuronas artificiales y los arcos que unen los nodos son las conexiones sinápticas. Al ser dirigido, los arcos son unidireccionales, es decir la información se propaga en un único sentido, desde una neurona origen a neurona destino. Por otra parte es ponderado, lo que significa que las conexiones tienen asociado un número real, un peso, que indica la importancia de esa conexión con respecto al resto de las conexiones.

Lo usual es que las neuronas se agrupen en capas de manera que una RNA está formada por varias capas de neuronas. Aunque todas las capas son conjuntos de neuronas, según la función que desempeñan, suelen recibir un nombre especifico. Las más comunes son las siguientes:

- Capa de entrada: Las neuronas de la capa de entrada, reciben los datos que se proporcionan a la RNA para que los procese.

- Capas ocultas: Estas capas introducen grados de libertad adicionales en la RNA. El número de ellas puede depender del tipo de red que estemos considerando. Este tipo de capas realiza gran parte del procesamiento.

- Capa de salida: Esta capa proporciona la respuesta de la red neuronal, y normalmente también realiza parte del procesamiento.

El número de capas en una red multicapa (MLP) se define con el número de capas de pesos, en vez de capas de neuronas. Es decir, una RNA con una capa de entrada, dos capas ocultas y una capa de salida, se dice que es una red de 3 capas, pues presenta 3 capas de pesos.

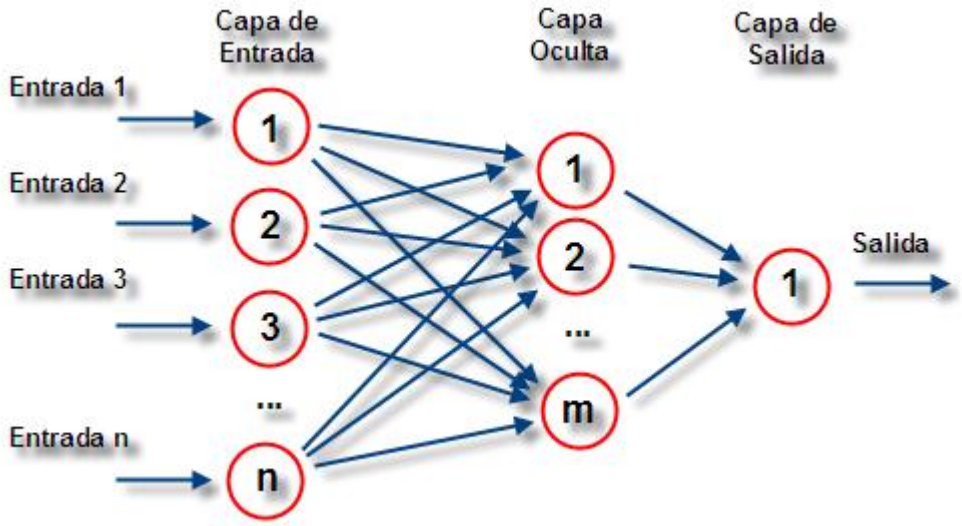


Figura 3.1 Ejemplo de arquitectura de un Perceptrón Multicapa.

En general, una red estándar de L-capas consiste de una capa de entrada, L-1 capas ocultas, y una capa de salida de neuronas todas sucesivamente conectadas (completamente o parcialmente) con conexiones de alimentación hacia adelante, pero sin enlaces entre unidades de la misma capa o enlaces retroalimentados

Si la arquitectura de la red no presenta ciclos, es decir, no se puede trazar un camino de una neurona a sí misma, la red se llama unidireccional. A partir de ahora siempre nos referiremos a redes neuronales unidireccionales (FFNN, FeedForward Neural Network). Si por el contrario las interconexiones entre neuronas y capas son recurrentes o auto-recurrentes nos encontraríamos con una FeedBack Neural Network. Siguiendo esta clasificación el otro tipo es NEOCOGNITRON topológicamente organizado en capas bidimensionales, lo que permite eliminar las variaciones geométricas o distorsiones en la información de entrada de la red.

Un MLP con una capa oculta puede formar cualquier región convexa en este espacio. Las regiones convexas se forman mediante la combinación de las regiones formadas por cada neurona. Un MLP con dos capas ocultas puede generar regiones de decisión arbitrariamente complejas [21]. El proceso de separación en clases que se lleva a cabo consiste en la partición de la región deseada en pequeños hipercubos.

Cada hipercubo requiere 2d neuronas en la primera capa oculta (siendo d el número de entradas a la red), una por cada lado del hipercubo, y otra en la segunda capa oculta, que lleva a cabo la operación lógica AND de la salida de los nodos del nivel anterior. La salida de los nodos de este segundo nivel se activará solo para las entradas de cada hipercubo. Los hipercubos se asignan a la región de decisión adecuada mediante la conexión de la salida de cada nodo del segundo nivel solo con la neurona de salida (tercera capa) correspondiente a la región de decisión en la que este comprendido el

hipercubo llevándose a cabo una operación lógica OR en cada nodo de salida. Este procedimiento se puede generalizar de manera que la forma de las regiones convexas sea arbitraria, en lugar de hipercubos.

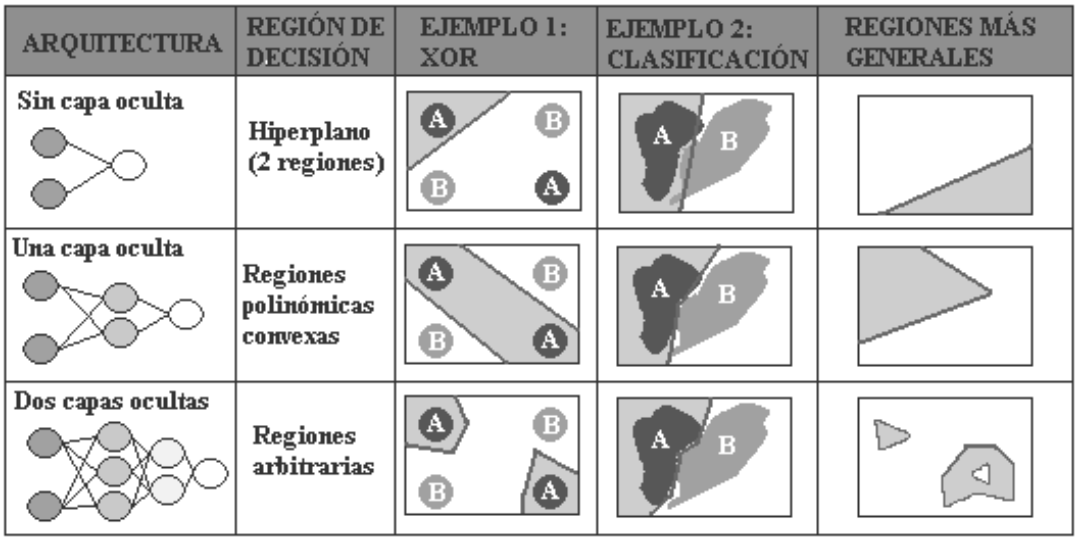


Figura 3.2 Relación de arquitecturas típicas del MLP y sus características.

3.2 Mecanismos de aprendizaje.

Es el proceso mediante el cual una red neuronal modifica sus pesos (destruye, modifica o crea conexiones) en respuesta a una información de entrada. El valor de cero para una conexión en la red, implica que esa conexión no existe. El proceso de aprendizaje finaliza, cuando los valores de los pesos dejan de cambiar, entonces decimos que la red ha aprendido.

Los criterios que definen las técnicas mediante las cuales se han de modificar los pesos de las conexiones de la red son propios de cada red, y definen lo que se conoce como regla de aprendizaje. Podemos encontrar redes que aprenden antes y durante su funcionamiento (o modo de operación), estos es aprendizaje ON LINE, y redes cuyo aprendizaje se realiza sólo antes del modo de operación, lo que se conoce como aprendizaje OFF LINE.

De acuerdo al mecanismo de aprendizaje utilizado se clasifica la regla de aprendizaje, y por tanto la red de esta manera se puede distinguir entre aprendizaje supervisado y no supervisado.

El tipo de aprendizaje se debe a que nosotros, dependiendo de la arquitectura de red empleada, podemos considerar un proceso que fuerce a una red a entregar una respuesta dada ante una entrada específica. Una respuesta particular puede o no ser especificada.

3.2.1 Supervisado.

En este caso, el proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo (supervisor), quien modifica los pesos de las conexiones, mientras la salida del conjunto difiera de la deseada para una entrada dada, buscando aproximarse a la misma. Han sido los modelos de redes más desarrolladas desde inicios de estos diseños.

Dado un nuevo patrón de entrenamiento, por ejemplo, (m+1)-ésimo, los pesos serán adaptados de la siguiente forma:

$$w_{ij}^{(m+1)} = w_{ij}^{(m)} + \Delta w_{ij}^{(m)} \tag{3.1}$$

Se puede ver un diagrama esquemático de un sistema de entrenamiento supervisado en la siguiente figura:

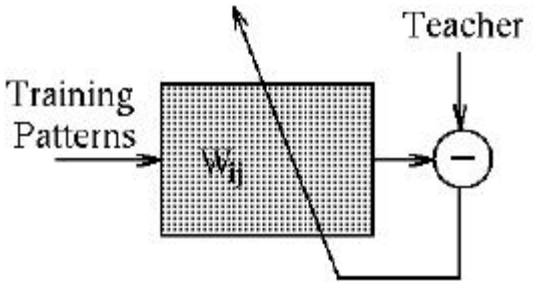


Figura 3.3 Diagrama esquemático de un sistema de entrenamiento.

3.2.2 Corrección del error.

Consiste en variar los pesos de las conexiones de la red en función del error cometido a la salida. Se definen entonces, unas reglas y algoritmos matemáticos para tal fin, los cuales, resultaran más ventajosos en la medida que nos brinden mayor información, acerca del error global a la salida de la red, y se discriminen los errores cometidos por cada unidad de proceso individualmente.

Es generalmente empleado en redes FFNN, en las que los patrones de entrenamiento están compuestos de dos partes fundamentales, un vector de entrada y un vector de salida, asociando la entrada con su correspondiente salida en cada elemento procesador denominado nodo. La manera general en que trabaja este tipo de aprendizaje es la siguiente:

Un vector de entrada es presentado en la capa de entrada, junto con un conjunto de respuestas deseadas para ese vector $\bar{d}$.Al realizar una iteración, se genera el error o discrepancia entre la respuesta deseada y la obtenida para cada nodo en la capa de salida, dicho valor de error es utilizado para actualizar el vector de pesos y, se realimenta para producir una nueva salida. Los pares de vectores de entrenamiento se van modificando hasta que el porcentaje de cambio de los pesos esté cercano a cero indicando en realidad que el error total está próximo a cero. No es necesario llegar hasta un valor de cero en el error; dependiendo de la aplicación, puede estar dentro de un rango dado. Este tipo de aprendizaje es utilizado en arquitecturas de redes neuronales artificiales que necesitan ser entrenadas fuera de línea, es decir, que se necesitan entrenar antes de poder aplicarlas.

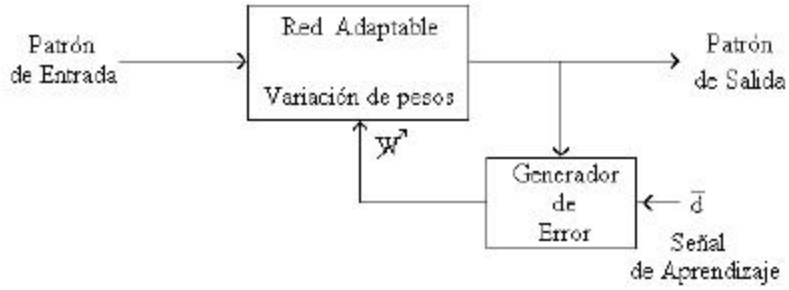


Figura 3.4 Diagrama esquemático de un sistema de corrección de error.

3.2.3 Aprendizaje por refuerzo.

Más lento que el de corrección de error, no se pretende exactitud en la salida del sistema, de modo que la función del supervisor se reduce a excitar o inhibir la actividad neuronal, mediante una señal de refuerzo que indica si la señal obtenida a la salida de la red se ajusta a la deseada (éxito = 1, fracaso = -1), y en función de ello se ajustan los pesos en base a un mecanismo de probabilidades.

3.2.4 Aprendizaje estocástico.

Consiste básicamente en realizar cambios aleatorios en los valores de los pesos de las conexiones de la red y evaluar su efecto a partir del objetivo deseado, y de distribuciones de probabilidad. Comúnmente se hace analogías con el estado energético de un sólido, si la energía del sistema es menor después de un cambio, se acepta el cambio, en caso contrario la aceptación del cambio queda condicionada a una distribución de probabilidades determinada y preestablecida. Se utiliza un procedimiento conocido como "recocido simulado" consistente, en utilizar ruido para escapar de los errores locales, facilitando la búsqueda del error global. Este ruido va siendo decrecido durante el proceso de aprendizaje. La combinación de la asignación probabilística, con el "recocido simulado", es lo que se conoce como aprendizaje estocástico.

3.2.5 No supervisado o auto-supervisado.

La red no requiere influencia externa para ajustar los pesos de las conexiones. La utilidad de estas redes, se reduce a encontrar las características, regularidades, correlaciones o categorías que se pueden establecer entre los datos que se presenten a su entrada. La interpretación de la salida de una red de este tipo, depende de su estructura, y del algoritmo de aprendizaje empleado. La red aprende a adaptarse basada en las experiencias recogidas de los patrones de entrenamiento anteriores, sin el beneficio de un maestro. El siguiente es un esquema típico de un sistema "No Supervisado":

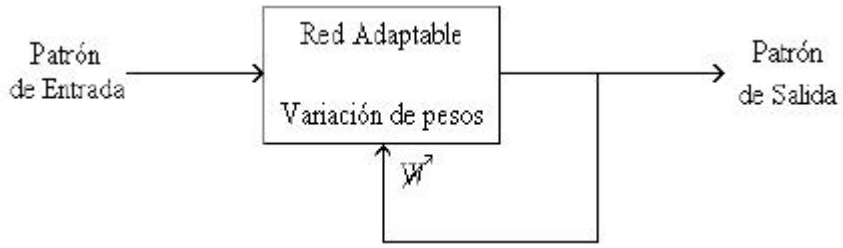


Figura 3.5 Diagrama esquemático de un sistema no supervisado.

3.2.6 Hebbiano.

Utilizado cuando se pretende medir la familiaridad, o extraer características de los datos de entrada, se basa en el siguiente postulado formulado por Donald O. Hebb en 1949 [22]: Cuando un axón de una celda A está suficientemente cerca como para conseguir excitar una celda B, y repetida o persistentemente toma parte en su activación, algún proceso de crecimiento o cambio metabólico tiene lugar en una o ambas celdas, de tal forma que la eficiencia de A, cuando la celda a activar es B aumenta. Resumiendo, el ajuste de los pesos se lleva a cabo de acuerdo con la correlación de los valores de las salidas de las dos neuronas conectadas.

3.2.7 Competitivo y cooperativo.

El trabajo de la red se orienta hacia el clustering o clasificación de los datos de entrada, se busca que el número de neuronas que se activen para una determinada entrada, sea menor que el número de neuronas de dicha entrada. De esta manera las neuronas compiten por activarse, quedando sólo una, o una por cada cierto grupo como vencedora, dejando al resto forzadas a sus valores de respuesta mínimos.

3.3 Algoritmo BackPropagation.

Tras 20 años de parón en el desarrollo de RNA, surge el algoritmo de retropropagación de los errores (BP, Backpropagation algorithm) que se ha convertido sin duda en el algoritmo más empleado a partir de que Rumelhart y McClelland popularizaran este método [13]. La idea básica, sencilla en su filosofía, requirió un desarrollo matemático fuerte para conseguir su convergencia. Se consideran los errores en las capas ocultas, que al propagarse hasta la capa de salida, producen los errores que realmente se miden. En definitiva, con este algoritmo se retropropagan los errores de la capa de salida hacia las capas anteriores, y emplear estos errores retropropagados para ajustar los pesos de las entradas de la correspondiente capa.

. Por tanto es necesario tener un conjunto de *N* patrones de entrenamiento junto con sus N salidas deseadas o *targets*. En la Figura 3.4 se describe la red multicapa sobre la que se analizará el algoritmo de retropropagación. Las entradas procedente del patrón *n*-ésimo, con *n = (1…N)* de la neurona *j*-ésima de la primera capa oculta se denotan como *yi(n)*, siendo $\omega ij(n)$ los pesos asociados a la conexión de cada uno de los pesos a la neurona *j*-ésima. La combinación lineal de las entradas en la neurona *j*-ésima ponderadas con los respectivos pesos se denota $\upsilon j(n)$ y $\phi(\cdot)$ es la función de activación empleada, generalmente la función sigmoide.

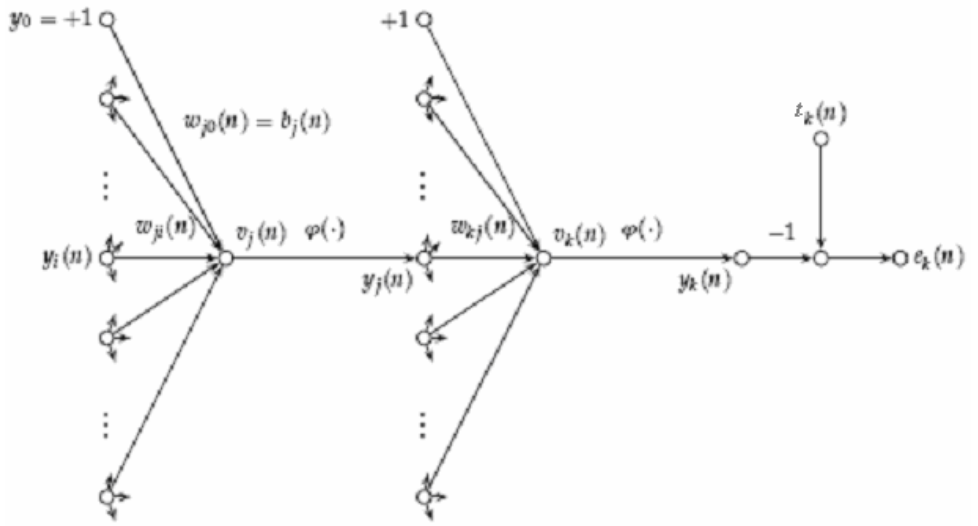


Figura 3.6 Diagrama de conexiones y pesos de un MLP con dos capas ocultas.

El algoritmo BP para MLPs es una generalización del algoritmo LMS (explicado posteriormente en el apartado (sección 3.4.3.5), pues ambos algoritmos realizan la actualización de salida) en la iteración n (esto es, la presentación del *n*-ésimo patrón de entrenamiento) por:

$$e_k(n) = t_k(n) - y_k(n) \tag{3.2}$$

El valor instantáneo del error para la neurona k es $\frac{1}{2}e_k^2(n)$ Así mismo, el valor instantáneo del error total *E(n)* se obtiene sumando todos los errores correspondientes a cada una de las neuronas de la capa de salida, esto es:

$$E(n) = \frac{1}{2}\sum_k e_k^2(n) \tag{3.3}$$

Donde *k* va desde uno hasta el número de neuronas de la capa de salida de la red. El error cuadrático promedio, teniendo en cuenta los *N* patrones *(n = 1…N),* es obtenida sumando todos los *E(n)* sobre todos los *n* y normalizando con respecto al tamaño del conjunto, de forma que

$$E_{prom} = \frac{1}{N}\sum_n E(n) \tag{3.4}$$

Tanto *E(n)* como E_{prom} están en función de los parámetros libres (pesos sinápticos y niveles de sesgo) de la red. Para un conjunto dado de entrenamiento, E_{prom} representa la función de coste como medida de rendimiento del aprendizaje. El objetivo del proceso de aprendizaje es ajustar los parámetros libres de la red para minimizar E_{prom}. Para esto, se considera un método simple de entrenamiento en el que los pesos se actualizan presentando uno a uno los patrones hasta que se termine una *época*, esto es, hasta que se presente completamente el conjunto de entrenamiento. Los ajustes a los pesos se hacen de acuerdo a los errores respectivos calculados para cada patrón presentado a la red.

Considérese una neurona *k* que está recibiendo un conjunto de señales producidas por una capa de neuronas a su izquierda (ver Figura 3.6). El algoritmo de retropropagación aplica la corrección $\Delta\omega_{kj}(n)$ al peso $\omega_{kj}(n)$, que es proporcional a la derivada parcial . Esta derivada ı $\partial E(n)/\partial\omega_{kj}(n)$ a dirección de búsqueda en el espacio de los pesos para el peso $\omega_{kj}(n)$.

La corrección $\Delta\omega_{kj}(n)$ aplicada a $\omega_{kj}(n)$, está definida por la regla delta:

$$\Delta\omega_{kj}(n) = -\eta\frac{\partial E(n)}{\partial\omega_{kj}(n)} \tag{3.5}$$

Donde η es la tasa de aprendizaje del algoritmo. El signo menos describe el descenso del gradiente en el espacio de pesos (es decir, busca una dirección que el cambio de peso reduzca el valor de *E(n)*. Finalmente se tiene que

$$\Delta\omega_{kj}(n) = \eta \cdot \delta_k(n) \cdot y_j(n) \tag{3.6}$$

Donde el gradiente local δ_k (n) está definido por

$$\delta_k(n) = e_k(n)\varphi_k'(\upsilon_k(n)) \tag{3.7}$$

Donde la prima en φ_k' $(\upsilon_k(n))$ denota diferenciación.

De acuerdo con las dos anteriores ecuaciones existe un factor involucrado que es la señal de error $e_k(n)$. En este contexto, se pueden identificar dos casos distintos, dependiendo donde este localizada la neurona. En el caso descrito, la neurona k pertenece a la capa de salida, por lo tanto se puede determinar el gradiente local de una manera simple, ya que se tiene una respuesta deseada $d_k(n)$ y $e_k(n)$, que se puede calcular fácilmente.

El segundo caso se presenta cuando una neurona *j* está localizada en una capa oculta de la red, cuando no se ha especificado una respuesta deseada para esa neurona. De acuerdo a esto, la señal de error para una neurona oculta se determina recursivamente en términos de las señales de error de todas las neuronas con las cuales esa neurona oculta está directamente conectada. Considérese la situación descrita en la Figura 3.6. Se redefine el gradiente local $\delta_j(n)$ para la neurona oculta como

$$\delta_j(n) = -\frac{\partial E(n)}{\partial y_j(n)}\varphi_j'(\upsilon_j(n)) \tag{3.8}$$

Finalmente se obtiene la fórmula de retropropagación para el gradiente local $\delta_j(n)$ así:

$$\delta_j(n) = \varphi_j'(\upsilon_j(n))\sum_k \delta_k(n)\omega_{kj}(n) \tag{3.9}$$

De esta forma la corrección aplicada $\Delta\omega_{kj}(n)$ al peso que conecta la neurona *i* con la neurona *j* está definido por la regla delta:

$$\Delta\omega_{ji}(n) = -\eta \cdot \delta_j(n) \cdot y_i(n) \tag{3.10}$$

En la aplicación del algoritmo BP se distinguen dos pasos distintos. El primer paso es referido como el *paso hacia adelante*, y el segundo como *paso hacia atrás*. En el *paso*

hacia adelante los pesos se mantienen inalterables y las salidas de los nodos de la red se calculan neurona por neurona. Esta fase de cómputo comienza en la primera capa oculta presentándole el vector de entrada, calculando las respectivas salidas y pasando esta información hacia las demás capas ocultas (si existen); la fase termina cuando se calcula la señal de error (comparar la salida de la neurona con su respuesta deseada) para cada neurona en la capa de salida.

El *paso hacia atrás*, por otra parte, comienza en la capa de salida pasando las señales de error hacia la izquierda a través de la red, capa por capa, y recursivamente calculando el δ (gradiente local) para cada neurona. Este proceso recursivo permite que los pesos sinápticos de la red sufran cambios de acuerdo con la regla delta (ecuación 3.10). Para una neurona en la capa de salida su gradiente local se calcula con la ecuación 3.7. Por consiguiente, esta se usa para calcular los cambios en los pesos que alimentan la capa de salida. Dados estos δs se usa la ecuación 3.9 para calcular los gradientes de todas las neuronas de la penúltima capa y así aplicar las correcciones a los respectivos pesos. Este cómputo recursivo se realiza capa por capa hasta ajustar los pesos de toda la red. La figura siguiente muestra la representación como grafo de la ecuación 3.9, asumiendo que la capa de salida consta de m_L neuronas.

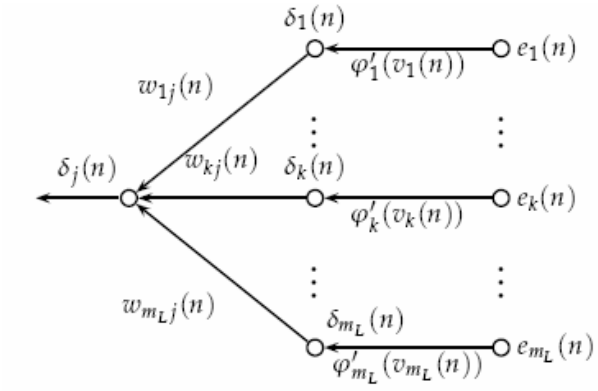


Figura 3.7 Representación de la propagación hacia atrás de los errores en una neurona.

3.3.1 Resumen del Algoritmo de Retropropagación.

En el modo de entrenamiento secuencial o estocástico [25], la actualización de pesos se efectúa después de la presentación de cada patrón de entrenamiento. Para ser específicos, considérese una época consistente en N patrones de entrenamiento organizados en el orden $\{(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)\}$ (pares entrada-respuesta deseada).

El primer patrón se presenta a la red, y se realiza la secuencia de operaciones hacia adelante y hacia atrás, descrita anteriormente, y como resultado se tiene un cierto ajuste

de pesos. Luego se presenta el segundo patrón de igual forma y así sucesivamente hasta el último patrón, momento en el cual se concluye una *época*.

Para este modo de operación secuencial, el algoritmo realiza ciclos a través de la muestra de entrenamiento $(x_N , t_N)^N_{n-1}$de la siguiente manera [23]:

1. *Inicialización.* Obtener los pesos sinápticos de una distribución de probabilidad uniforme, cuya varianza y desviación estándar de los campos locales inducidos de las neuronas se encuentren en la transición entre las partes lineales y saturadas de la función de activación sigmoide.

2. *Presentaciones de los ejemplos de entrenamiento*. Presentar una época de ejemplos de entrenamiento a la red. Realizar los pasos hacia adelante y hacia atrás, descritos en los puntos 3 y 4, respectivamente.

3. *Propagación hacia adelante*. Para un ejemplo de entrenamiento (x_n ,d_n), calcular las sumas ponderadas. La suma ponderada $\upsilon_j(n)$ para la neurona *j* en la capa *l* es:

$$\upsilon_j^{(l)}(n) = \sum_{i=0}^{m} \omega_{ji}^{(l)}(n) \cdot y_i^{(l-1)}(n) \tag{3.11}$$

Donde $y_i^{(l-1)}(n)$ es la salida de neurona i en la capa anterior l-1 en la iteración n, $\omega_{ji}^{(l)}(n)$ es el peso de neurona *j* en la capa *l* que es alimentada por la neurona *i* en la capa *l*-1, y *m* es la cantidad de neuronas de la capa anterior. En este caso se tienen en cuenta las entradas para los sesgos. Asumiendo el uso de una función sigmoide, la salida de la neurona *j* en la capa *l* es:

$$y_j^{(l)}(n) = \varphi_j\left(\upsilon_j^{(l)}(n)\right) \tag{3.12}$$

Si la neurona *j* está en la capa de entrada (esto es, l=0) entonces

$$y_j^{(0)}(n) = x_j^n \tag{3.13}$$

donde x_j^n es el *j*-ésimo elemento del vector de entrada X_n. Si la neurona *j* está en la capa de salida (esto es *l*=*L*, donde *L* es el índice de la última capa de la red), entonces el error se calcula así:

$$e_j(n) = t_j(n) - y_j^{(L)}(n) \tag{3.14}$$

donde $d_j(n)$ es el *j*-ésimo elemento del vector de respuestas deseadas d_n

4. *Propagación hacia atrás*. Calcular los grandiantes locales de la red, definidos por

$$\delta_j^{(l)}(n) = \begin{cases} \text{Para la neurona } j \text{ en la capa de salida } L: \\ e_j^{(L)}(n)\cdot\varphi_j'\left(\upsilon_j^{(L)}(n)\right) \\ \text{Para la neurona } j \text{ en la capa oculta } l: \\ \varphi_j'\left(\upsilon_j^{(l)}(n)\right)\sum_k \delta_k^{(l+1)}(n)\cdot\omega_{kj}^{(l+1)}(n) \end{cases} \tag{3.15}$$

El ajuste en los pesos de la red de la capa l se realiza de acuerdo a la regla delta generalizada

$$\omega_{ji}^{(l)}(n+1) = \omega_{ji}^{(l)}(n) - \eta\cdot\delta_j^{(l)}(n)\cdot y_i^{(l-1)}(n) \tag{3.16}$$

donde η es la tasa de aprendizaje. Escoger un valor adecuado influye en la convergencia del algoritmo, por lo que en general se toman valores de pequeños, pero esto significa que tendrá que hacer un gran número de iteraciones. Si la tasa de aprendizaje es grande, el resultado son cambios drásticos en los pesos, avanzando muy rápidamente por la superficie de error, con el riesgo de estar oscilando alrededor del mínimo. Este efecto se puede contrarrestar con tasa adaptativa o con la adición del término de momento.

5. *Iteración*. Iterar los cómputos hacia adelante y hacia atrás de los puntos 3 y 4 presentando nuevas épocas de ejemplos de entrenamiento a la red hasta que el criterio de terminación se cumpla. El orden de presentación de los ejemplos de entrenamiento debería ser aleatorio de una época a otra. Los parámetros de momento y de tasa de aprendizaje son habitualmente ajustados (usualmente decrementados) cuando el número de iteraciones se incrementa.

3.3.2 Problemas y variantes del algoritmo de retropropagación.

Como puede verse en la literatura relativa a RNAs, se ha escrito mucho acerca de los problemas y excelencias de los perceptrones multicapa, así como sobre sus aplicaciones. Surgen muchas variantes para aplicaciones concretas, o para resolver algunos de los problemas. A continuación se incluye un resumen de los problemas más importantes de los MLPs y variantes del algoritmo BP.

3.3.2.1 Problemas de los MLPs y algoritmo BP.

- Número de capas y de neuronas: No hay procedimientos claros para decidir el mejor número de capas y/o neuronas en el MLP. Hay publicaciones sobre cotas mínimas y máximas, generalmente para problemas de clasificación, en función del número de patrones de entrenamiento, pero en la práctica no se utilizan. Generalmente se determina

de forma intuitiva y/o experimental, pero sin duda la experiencia del investigador sirve de gran ayuda.

- Conjunto de entrenamiento. Se puede decir casi lo mismo que en el apartado anterior. Lo que está claro, es que si se pretende un buen aprendizaje del dominio del problema, o una buena generalización de los patrones, estos deben cubrir todo el espectro del dominio. El número de patrones a emplear dependen del problema y la experiencia del investigador ayuda. Se suele determinar casi siempre (a veces solo se dispone de patrones concretos) de forma experimental.

- Preparación de los datos. En general, los datos no se usan directamente en la forma en que se obtienen o los proporciona el problema. Casi siempre se adaptan los datos reales al modelo usado. Por ejemplo, las salidas hay que normalizarlas en [0,1] si se usa activación sigmoide, porque si no, el error nunca podrá reducirse de forma satisfactoria. También es habitual normalizar las entradas, aun cuando no sea necesario. Obviamente habrá un post- procesamiento de resultados, para hacerlos válidos en la realidad del problema..

- Velocidad de convergencia. El algoritmo de retropropagación es lento y se han propuesto muchas modificaciones y variantes para mejorar la velocidad del entrenamiento. Algunas se reflejan en el apartado de variantes

- Mínimos locales. Algo importante que se debe tener en cuenta es la posibilidad de existencia de mínimos locales en la superficie de error (ver Figura 3.8).El algoritmo de retropropagación con descenso de gradiente no garantiza que se alcance un mínimo global, una vez que se alcance un mínimo, el algoritmo se detiene, quedando a veces "atrapado" en un mínimo local, del que no puede salir, y por tanto el aprendizaje no se hace bien. Una forma de obviar esto es realizar el aprendizaje varias veces, partiendo de pesos diferentes cada vez, y seleccionar la ejecución que mejor resuelve el problema. Algunas de las variantes buscan usar otros procesos de optimización no lineal más seguros que el de gradiente descendente.

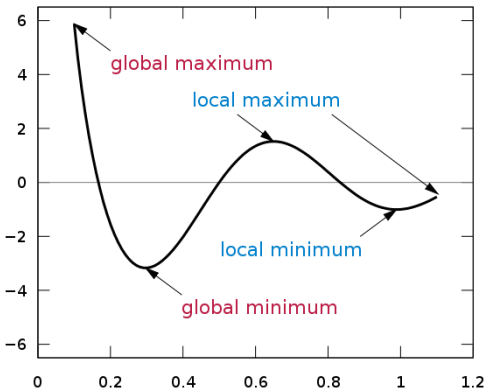


Figura 3.8 Función de error E en función de los pesos ω

• Sobreentrenamiento. Es el problema de que se aprende muy bien los ejemplos de entrenamiento, pero cuando se usa el MLP entrenado, con muestras que no ha aprendido, no es capaz de dar buenas respuestas. Se dice entonces que la red no generaliza bien. Se puede evitar esto seleccionando muy bien el conjunto de entrenamiento y parando el entrenamiento cuando el error es suficientemente pequeño pero no excesivamente pequeño.

• Saturación. Se puede producir cuando las salidas esperadas en cada neurona de salida son 0 o 1. Al pretender acercar las salidas de la red a estos valores, nos ponemos en las zonas donde la función de activación tiene tangente de pendiente casi cero y entonces no se produce apenas modificación de los pesos y por tanto del error. La forma de resolverlo es cambiar los valores de salidas esperadas a 0.1 y 0.9. Esto puede requerir adaptación de los datos reales. También se produce si los pesos se hacen muy grandes, porque los resultados de la red ya no varían.

• Estabilidad y robustez. Un tema interesante, sobre todo para las implementaciones en hardware de la RNA entrenada, es la estabilidad del aprendizaje sobre variaciones en los pesos (y/o las entradas). Hay medidas de estabilidad estadística, que se pueden usar para seleccionar el MLP entrenada más estable de entre varias ejecuciones. También hay métodos que tienen en cuenta estas medidas en el entrenamiento de la red, aunque son más lentos.

3.3.2.2 Variantes del algoritmo de retropropagación.

Han surgido muchas variantes del algoritmo de retropropagación, en general intentando disminuir el tiempo de computación, o para resolver alguno de los problemas del algoritmo. La cantidad de estas variantes hace casi imposible reflejarlas todas, por lo que indicaremos algunas de las más destacadas.

• Tasa de aprendizaje adaptativa. Con el algoritmo de retropropagación basado en descenso de gradiente, la tasa de aprendizaje η se mantiene constante durante el entrenamiento. Como ya se ha comentado más arriba, el algoritmo es muy sensible al valor de η. Para evitarlo, se emplea η variable durante el proceso de entrenamiento. Pretende mantener un paso de aprendizaje elevado pero garantizando un aprendizaje estable, sin oscilaciones. Primero, se calcula la salida inicial de la red y el valor del error. En cada *época* nuevos pesos son calculados usando la nueva tasa, y se calcula de nuevo la salida de la red y el nuevo error. Si el nuevo error excede el de la anterior época en un valor mayor que una relación predefinida (p.e. 1.04), entonces los pesos de la época actual son descartados y la tasa de aprendizaje se reduce multiplicando por un valor constante (p.e. 0.7). En el otro caso, si el nuevo error es menor que el anterior, el valor de la tasa de aprendizaje es incrementada. (p.e. 1.05).

• Retropropagación con término de momento. Para evitar oscilaciones en torno a un mínimo y para conseguir una mayor velocidad de convergencia, se suele añadir un término de momento a la fórmula de actualización de los pesos por descenso de gradiente. De tal manera que la ecuación de actualización de pesos queda:

$$\omega_{ji}^{(l)}(n+1)=\omega_{ji}^{(l)}(n)-\eta\cdot\delta_{j}^{(l)}(n)\cdot y_{i}^{(l-1)}(n)+\mu\cdot\Delta\omega_{ji}^{(l)}(n-1) \qquad (3.17)$$

Siendo

$$\Delta\omega_{ji}^{(l)}(n-1)=\omega_{ji}^{(l)}(n)-\omega_{ji}^{(l)}(n-1) \qquad (3.18)$$

donde μ es el parámetro de momento y está comprendido entre 0 y 1. Presenta el inconveniente de que se introduce un segundo parámetro μ cuyo valor debe ser escogido.

• Método del gradiente conjugado. Este algoritmo pertenece a los métodos de búsqueda de línea, y pretende minimizar el número de iteraciones necesarias para alcanzar el mínimo de la superficie de error buscando la dirección y magnitud óptimas de actualización de los pesos, de manera que, para superficies cuadráticas se demuestra que se alcanza la solución en d pasos, siendo d la dimensión del espacio de los pesos.

El algoritmo básico BP ajusta los pesos en la dirección negativa a la del gradiente, pues está es la dirección en la cual decrece la función de error. Sin embargo, el uso del vector gradiente en cada iteración como dirección de búsqueda del mínimo global no es la mejor opción. La solución está en escoger sucesivas direcciones de búsqueda tal que, en cada paso del algoritmo, el gradiente no cambie en esa dirección y sea ortogonal al anterior, de tal manera que se dice que las direcciones de actualización sucesivas son conjugadas. El método del gradiente conjugado, sin aplicarlo al algoritmo de retropropagación, se rige por las siguientes ecuaciones.

$$\boldsymbol{\omega}(n+1)=\boldsymbol{\omega}(n)+\alpha(n)\cdot\mathbf{d}(n) \qquad (3.19)$$

$$\mathbf{d}(k)=-\mathbf{g}(n+1)+\beta(n)\cdot\mathbf{d}(n) \qquad (3.20)$$

$$\beta(n)=\begin{cases}\dfrac{\mathbf{g}(n+1)^{T}\cdot\left[\mathbf{g}(n+1)-\mathbf{g}(n)\right]}{\mathbf{g}(n)^{T}\cdot\mathbf{g}(n)} & \text{(Polak-Ribiere)}\\[2ex] \dfrac{\mathbf{g}(n+1)^{T}\cdot\mathbf{g}(n+1)}{\mathbf{g}(n)^{T}\cdot\mathbf{g}(n)} & \text{(Fletcher-Reeves)}\end{cases} \qquad (3.21)$$

Donde $\alpha(n)$ se obtiene por un procedimiento de búsqueda en linea y $g(n)$ es el gradiente de la función de error en la iteración n.

• Métodos de Quasi-Newton. La idea del algoritmo de Newton es evitar el cálculo directo de la matriz Hessiana (pues implica un alto coste computacional) en la obtención de la dirección de Newton: $\boldsymbol{d=-H^{-1}\cdot\nabla E}$. Para ello calcula **H** en cada paso del algoritmo tomando únicamente información de primer orden, es decir, empleando tan sólo el valor de los pesos y del gradiente. Además, dicho algoritmo garantiza en cada paso que el Hessiano sea definido positivo (evitando que converja a un máximo de la función de error). A continuación se explica el proceso que dicho algoritmo realiza:

1. En primer lugar se parte de una aproximación de $\boldsymbol{H^{-1}: G = I.}$

2. Se actualizan los pesos según la siguiente expresión:
$$\boldsymbol{\omega}(n)=\boldsymbol{\omega}(n)-\lambda(n)\cdot\mathbf{G}(n)\cdot\nabla\mathbf{E}(n) \quad (3.22)$$
En dicha actualización se emplea una búsqueda en línea para la determinación del parámetro λ en cada paso. Esto es lo que garantiza que no se produzca la inestabilidad del algoritmo.

3. A continuación se actualiza la matriz **G** en función de la información de primer orden (simplificando de esta manera considerablemente el cálculo de $\boldsymbol{H^{-1}}$). G es actualizada de la forma
$$\mathbf{G}(n+1)=\mathbf{G}(n)+F\left[\mathbf{G}(n),\boldsymbol{\omega}(n+1),\boldsymbol{\omega}(n),\nabla E(n+1),\nabla E(n)\right] \quad (3.23)$$

4. **G**(*n*) debe satisfacer la "*Condición Quasi-Newton*" o "*Condición Secante*". Dicha condición es la que garantiza precisamente que la matriz Hessiana quede definida positiva en cada paso del algoritmo. Esta condición es la siguiente
$$\boldsymbol{\omega}(n+1)-\boldsymbol{\omega}(n)=-\mathbf{H}^{-1}\cdot\left(\nabla E(\boldsymbol{\omega}(n+1))-\nabla E(\boldsymbol{\omega}(n))\right) \quad (3.24)$$
Esta condición se satisface para dos puntos cercanos al mínimo. Una fórmula de la estima G propuesta que satisface la condición de Quasi-Newton es la propuesta por *Broyden-Fletcher-Goldfard- Shannon* (BFGS):
$$\mathbf{G}(n+1)=\mathbf{G}(n)-\frac{\mathbf{p}\cdot\mathbf{p}^T}{\mathbf{p}^T\cdot\mathbf{v}}-\frac{\left(\mathbf{G}(n)\cdot\mathbf{v}\right)\cdot\mathbf{v}^T\cdot\mathbf{G}(n)}{\mathbf{v}^T\cdot\mathbf{G}(n)\cdot\mathbf{v}}+\left(\mathbf{v}^T\cdot\mathbf{G}(n)\cdot\mathbf{v}\right)\cdot\mathbf{u}\cdot\mathbf{u}^T \quad (3.25)$$
donde
$$\mathbf{p}=\boldsymbol{\omega}(n+1)-\boldsymbol{\omega}(n) \quad (3.26)$$
$$\mathbf{v}=\nabla E(n+1)-\nabla E(n) \quad (3.27)$$
$$\mathbf{u}=\frac{\mathbf{p}}{\mathbf{p}^T\cdot\mathbf{v}}-\frac{\mathbf{G}(n)\cdot\mathbf{v}}{\mathbf{v}^T\cdot\mathbf{G}(n)\cdot\mathbf{v}} \quad (3.28)$$

Si ponemos **G** en la condición de Quasi-Newton anterior veríamos que ésta queda satisfecha. Vemos además que **G**(n +1) es función de los pesos y del gradiente, esto es, de la información de primer orden, así como de **G** en la etapa k-ésima. Se evita de esta manera calcular la matriz **H** de manera directa

Por último, si la superficie de error es cuadrática y aplicamos el algoritmo Quasi-Newton, se demuestra que dicho algoritmo converge al mínimo en d pasos (con d número de dimensiones del espacio de pesos) y **G** converge a **H**, siempre y cuando λ sea calculada mediante una búsqueda en línea exacta.

3.4 Redes de Funciones de Base Radial.

3.4.1 Arquitectura de las redes RBF.

La arquitectura genérica de una red RBF consta de tres capas de neuronas con conexiones de tipo FFNN como puede verse en la Figura 3.9.

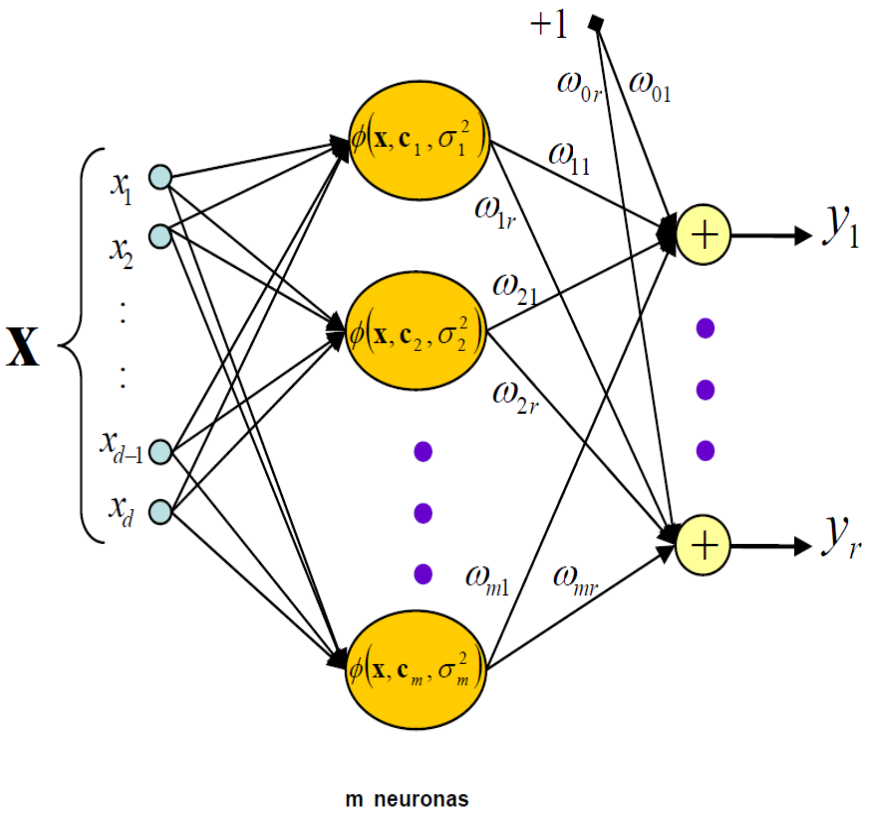


Figura 3.9 Arquitectura típica de una red RBF

Capa Entrada. La primera capa está formada por las neuronas de entrada, representando cada una de estas neuronas cada una de las características que componen el vector o patrón de entrada $x = \{x_1 , x_2 ,..., x_d \}$, es decir, la neurona i-ésima de la capa de entrada está asociada a la característica i-ésima x_i del patrón de entrada x. Esta primera capa no realiza ningún procesado de los datos de entrada, pasándolos tal cual a la siguiente capa; es por esta razón por la que algunos autores consideran a estas redes como de dos capas, sólo considerando aquellas capas que realizan algún tipo de procesado de los datos. La capa de entrada simplemente realiza una función de simple transmisión de información.

Capa Oculta. La siguiente es la capa oculta, que al contrario que en otros tipos de redes, es única. Cada una de las neuronas de la capa oculta está asociada a un vector sináptico llamado comúnmente *centroide*. Al contrario de lo que suele ser habitual en otras redes, en vez de calcular la suma ponderada de las entradas (producto escalar del vector de entrada y de pesos sinápticos) y aplicarla una cierta función de activación, calcula la distancia euclídea entre el vector de las entradas $x = \{x_1 , x_2 ,..., x_d \}$ y el *centroide* de cada una de las neuronas ocultas, y a esa distancia se le aplica una *función de base radial* que realiza una transformación no lineal, y cuya salida pasara a la siguiente capa. La función más utilizada es la gaussiana, aunque existen otras funciones que se pueden utilizar. La respuesta de las neuronas de la capa oculta es *localizada*, pues en contra de lo que ocurre en otras redes, las neuronas de la capa oculta sólo responden con una intensidad apreciable en el caso de que el vector de entrada presentado y el *centroide* pertenezcan a una zona próxima en el espacio de entrada.

Capa Salida. Finalmente, las neuronas de la capa de salida, realizan una combinación lineal de las funciones de base radial, es decir, las salidas proporcionadas por las neuronas de la capa oculta son multiplicadas por el peso de su conexión y sumadas todas ellas, para a continuación aplicar el resultado a una función de activación como por ejemplo la lineal (generalmente) o la sigmoide.

3.4.2 Modelo genérico de una red RBF.

Para una red neuronal de base radial con *d* neuronas en la capa de entrada, *m* neuronas en la capa oculta, *r* neuronas en la capa de salida y salida lineal, la fórmula que describe la salida de la neurona *k*-ésima de la capa oculta, y_k, viene dada por la siguiente expresión:

$$y_k(\mathbf{x}) = \sum_{j=1}^{m} \omega_{kj} \cdot \phi_j(\mathbf{x}) + \omega_{k0} \qquad k = 1...r \tag{3.29}$$

Donde ω_{kj} representa el peso asociado a la conexión que une la neurona *j*-ésima de la capa oculta con la neurona *k*-ésima de la capa de salida, ω_{k0} es el **umbral** asociado a la neurona de salida *k*-ésima, x es la entrada de la red y $(\phi_j)_{j=1...m}$ son las **funciones de base radial o de activación** de cada uno de las neuronas de la capa oculta.

Las FBR se expresan normalmente como traslaciones de una función prototipo de la siguiente forma:

$$\phi_j(\mathbf{x}) = \phi\left(\frac{\|\mathbf{x}-\mathbf{c}_j\|}{d_j}\right) \tag{3.30}$$

donde c_j es el centro asociado a la neurona *j*-ésima de la capa oculta, d_j es la desviación, anchura o factor de escala para la distancia euclídea entre el vector de entrada x y el centroide, $\|x - c_j\|$.

Como función prototipo se suele emplear una gaussiana de la forma:

$$\phi(r) = e^{-\frac{r^2}{2}} \tag{3.31}$$

donde r es la distancia entre el centroide y el vector de entrada(ver Figura 3.10).

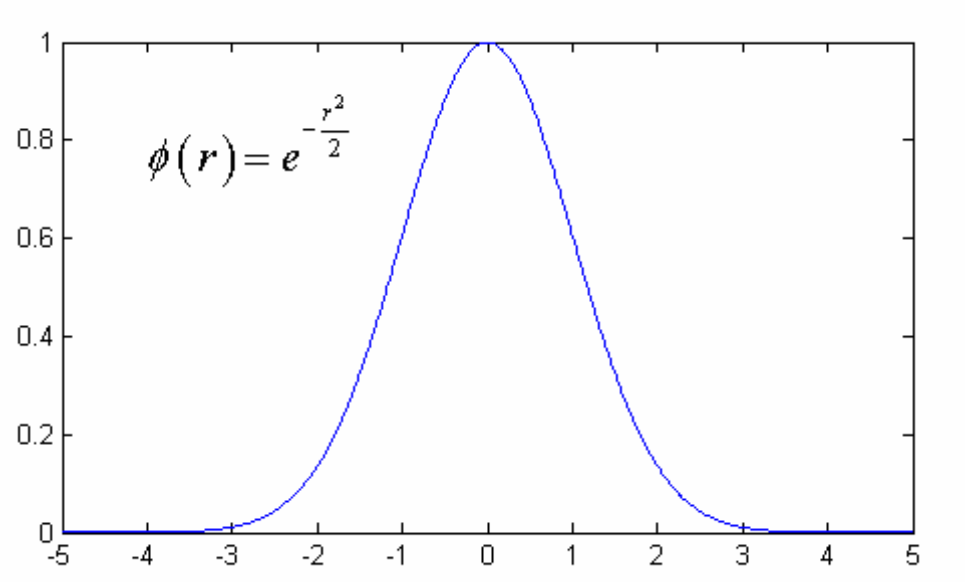


Figura 3.10 Función de base radial gaussiana

Aunque también se pueden emplear otro tipo de funciones, como se presenta en la siguiente tabla:

Nombre	Expresión	Parámetros
Gaussiana	$\phi(r) = e^{-\frac{r^2}{2\sigma^2}}$	Con parámetro de normalización $\sigma > 0$
Cauchy	$\phi(r) = \frac{1}{1+\frac{r^2}{\sigma^2}}$	Con parámetro de normalización $\sigma > 0$
Multi-Cuadraticas	$\phi(r) = \left(r^2 + \sigma^2\right)^{\frac{1}{2}}$	Con parámetro de normalización $\sigma > 0$
Multi-Cuadráticas Generalizadas	$\phi(r) = \left(r^2 + \sigma^2\right)^{\beta}$	Con parámetro de normalización $\sigma > 0 \quad 0 < \beta < 1$
Cúbica	$\phi(r) = r^3$	Con parámetro de normalización $\sigma > 0$

Nota: En caso de tener exponente negativo en las funciones multi-cuadráticas, se obtienen las llamadas funciones multi-cuadráticas inversas y multi-cuadráticas generalizadas inversas.

Tabla 3.1 Relación de funciones de base radial

La mayoría de estas funciones tienen similar dependencia radial, caracterizándose porque alcanzan un máximo en torno al origen ($r = 0$) y decrecen hacia cero $\phi(r) \rightarrow 0$ cuando $r \rightarrow \infty$. Por tanto, estas funciones se caracterizan porque la respuesta es localizada. Se dice que además son *aproximadores universales* ya que pueden aproximar arbitrariamente bien cualquier función continua y acotada (Poggio & Girosi, [24]).

También se han usado FBRs con una dependencia distinta, como por ejemplo la multicuadrática, cuya expresión se encuentra en la tabla anterior, pero en el que la raíz está en el numerador. En este caso la respuesta no es localizada.

El parámetro de normalización σ es un factor de escala que nos da la anchura de la función. Viene a ser como el radio de influencia de la neurona en el espacio de las entradas. En algunos casos se pueden escoger diferentes valores de σ para cada función de cada neurona, con lo que estas funciones $\phi_j(r)$ dejan de tener simetría radial y adquieren formas elipsoidales.

Como ya se ha comentado, la salida de la red es generalmente lineal aunque también existen otras posibilidades como la función de activación de forma sigmoidea (Figura 3.11)

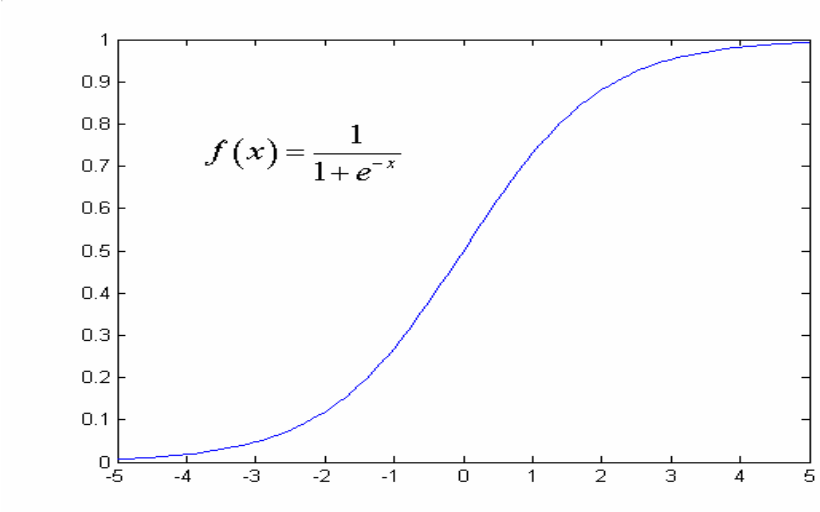


Figura 3.11 Función sigmoide

El principal problema de este tipo de redes se produce con la alta dimensionalidad del espacio de entrada, es decir, cuando el número de neuronas en la capa de entrada es muy alto, se produce el problema llamado *curse of dimensionality*. Este problema provoca que el número de neuronas en la capa oculta, necesarias para obtener una buena generalización, crezca de forma exponencial.

Obviamente, y debido en gran parte a este problema, dependiendo del número de neuronas en la capa oculta se conseguirán resultados muy diferentes. La determinación de las neuronas necesarias para resolver un problema suele ser un factor crítico, aunque no el único, a la hora de conseguir una buena generalización de la red.

Normalmente la determinación de la topología de la red (número de neuronas de la capa oculta), se hace de forma artesanal, es decir, el investigador utilizando su experiencia, determina el número de neuronas a utilizar, si bien el método de prueba y error es uno de los más utilizados cuando no se dispone de esta experiencia.

3.4.3 Aprendizaje en redes de funciones de base radial.

El aprendizaje de las redes RBF se caracteriza porque en general, y al contrario que otros tipos de redes es un aprendizaje por etapas, es decir, que se realiza por separado para cada una de las dos capas de procesamiento (oculta y de salida).

Los métodos de aprendizaje para las redes RBF deben de dar respuesta a varios puntos que determinan las características de dicho aprendizaje. Esencialmente los algoritmos deben determinar lo siguiente:

- Número de neuronas de la capa oculta.
- Selección de centroides en la capa oculta.
- Ajuste de anchuras en la capa oculta.
- Entrenamiento de la capa de salida.

3.4.3.1 Aprendizaje basado en agrupamiento.

A este tipo de aprendizaje basado en métodos de agrupamiento se le suele denominar como *Auto-organización de centros*. Muy resumidamente se puede decir que su funcionamiento consiste en localizar los m centroides representativos del espacio de entrada.

Como hemos comentado en los párrafos anteriores, una cuestión importante es la elección del número de neuronas ocultas. Cada neurona, cubre una parte del espacio de entrada, de modo que la elección correcta del número de neuronas se basa en que cubran suficientemente (para una aplicación dada) dicho espacio.

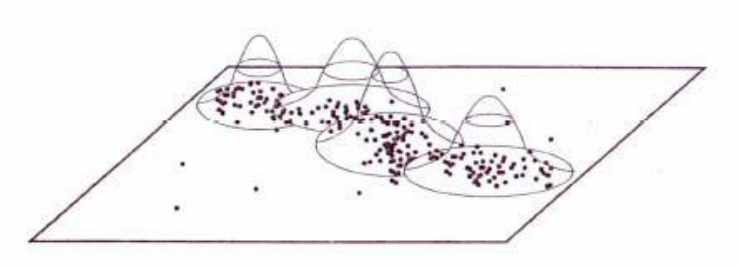


Figura 3.12 Distribución de neuronas en el espacio de entrada

Sin embargo, es usual trabajar con espacios de entrada de muchas variables, lo que provoca que el número de nodos necesario crezca exponencialmente al hacerlo la dimensión. Al final, habrá que llegar a un compromiso entre el número de neuronas seleccionado, el error que se alcanza en el ajuste de los patrones de aprendizaje y la capacidad de generalización, ya que puede aparecer sobreajuste al tomar neuronas ocultas.

Respecto a la selección de los valores de los centroides de la capa oculta, existen varios algoritmos para realizar esta tarea basados en *clustering*, buscando que los centroides estén asociados a zonas en el espacio de entrada donde hay una mayor densidad de patrones de aprendizaje. Entre dichos algoritmos destacan el algoritmo clásico k-medias, y los algoritmos de aprendizaje competitivo propuestos por Kohonen.

3.4.3.2 Algoritmo FSCL.

Este algoritmo para el cálculo de centroides, llamado *Frequecy Sensitive Competitive Learning*, para el cálculo de los centroides. Es un algoritmo de aprendizaje competitivo no supervisado. En un algoritmo competitivo, solo se actualiza un centroide en cada momento, compitiendo los centroides entre sí para resultar "vencedor" ante un patrón de entrada determinado, por lo que es necesario determinar el criterio de selección del centroide ganador, que normalmente es la distancia del centroide a la entrada, así el centroide más próximo al patrón de entrada es el vencedor, actualizando su valor convenientemente. Además por ser no supervisado, solo se dispone de los patrones de entrada y no de las salidas deseadas.

El algoritmo FSCL consigue equiparar la capacidad de victoria entre todos los centroides, evitando así el fenómeno de los "*centroides muertos*", que son aquellos que no se actualizan por encontrarse muy alejados de los datos y no resultan vencedores. La manera de conseguirlo es teniendo en cuenta durante la selección del centroide vencedor el número de veces que cada uno de los centroides resulta "*vencedor*", es decir, cada vez que un centroide resulta vencedor, se le penaliza multiplicando la distancia del centroide al patrón de entrada por el número de veces que dicho centroide resulta vencedor. Con ello se consigue que la probabilidad de victoria de cada uno de los centroides sea aproximadamente idéntica, e igual a $1/m$ siendo m el número de centroides. A continuación se describe el algoritmo que se ha implementado:

1. Se escoge el número de centroides m y el número de iteraciones, y se inicializan. Una manera de inicializarlos es situar los centroides iniciales en torno a la media de la distribución de los datos de entrada.

2. Inicializar el valor del parámetro de actualización μ, y el número de veces que resulta vencedor cada centroide $v = \{v_1 , v_2 ,..., v_m\}$ a la unidad.

3. Escoger un patrón de entrada x aleatoriamente del conjunto total de N patrones $X = \{x_1 , x_2 ,..., x_N \}$, y calcular las distancias d_j de dicho patrón a los distintos centroides c_j

4. Aplicar el criterio de penalización, es decir, multiplicar el número de victorias v_j de cada centroide por la distancia obtenida d_j , y tomar k según

$$k = \arg\min_j \left\{ v_j \cdot \left\| \mathbf{x} - \mathbf{c}_j \right\|^2 \right\} \tag{3.32}$$

5. Actualizar el centroide k-ésimo según la ecuación siguiente:

$$\mathbf{c}_k = \mathbf{c}_k + \mu \cdot \left(\mathbf{x} - \mathbf{c}_k \right) \tag{3.33}$$

6. Volver al paso 3 hasta que el número de actualizaciones sea igual a N.

7. Actualizar el parámetro de actualización según:

$$\mu = \alpha \cdot \left(e^{-i \cdot \beta} - e^{-N \cdot \beta} \right) \quad donde$$
$$\alpha = \frac{\eta}{\left(e^{-\beta} - e^{-N \cdot \beta} \right)} \quad con \quad \eta = 0.3 \quad \beta = 0.0005 \tag{3.34}$$

8. Volver al paso 2 hasta acabar el número de iteraciones, entonces el proceso finaliza.

3.4.3.3 Aprendizaje por cuantificación vectorial (LVQ).

La *cuantificación vectorial* es un método clásico que produce una aproximación de una *fdp p(x)* del vector de entrada x usando un número finito de vectores mí, que los llamaremos a partir de ahora centros, con i *=1,2,..., k*. Una vez escogido los centros, la aproximación de x implica encontrar el centro m_c más cercano a x. Se pretende que con la distribución óptima de los centros se minimice el error *E* dado por

$$E = \int \left\| \mathbf{x} - \mathbf{m}_c \right\| \cdot p(\mathbf{x}) d\mathbf{x} \tag{3.35}$$

donde dx es el volumen diferencial en el espacio de x y el índice c es el índice de aquel centro más cercano a x. Se dice que el centro m_c es el "vencedor". No existe una solución cerrada para la óptima situación de los centros.

Kohonen propuso una serie de procedimientos iterativos, conocidos como LVQs, que resolvían el problema. Cada uno de ellos mueve los centros intentando conseguir la mejor clasificación de un conjunto de entrenamiento dado usando la regla 1-nn. Los patrones son analizados uno a uno, es decir, hay tantas iteraciones como número de patrones del conjunto de entrenamiento, y los centros son actualizados después de que cada iteración.

El método original *LVQ1* emplea la siguiente regla de actualización. El centro más cercano a al patrón de entrada x, mc, es actualizado por

$$\mathbf{m}_c \leftarrow \mathbf{m}_c + \alpha(t) \cdot \left[\mathbf{x} - \mathbf{m}_c \right] \; si \; \boldsymbol{x} \; es \; clasificado \; correctamente \; por \; \boldsymbol{m_c}$$
$$\mathbf{m}_c \leftarrow \mathbf{m}_c - \alpha(t) \cdot \left[\mathbf{x} - \mathbf{m}_c \right] \; si \; \boldsymbol{x} \; es \; clasificado \; incorrectamente \tag{3.36}$$

mientras que el resto de centros no son actualizados. Inicialmente para $\alpha(t)$ se escoge un valor pequeño y es reducido linealmente hacia cero, conforme aumenta el número de iteraciones. El efecto de esta regla de adaptación es mover el centro hacia las muestras más cercanas de su misma clase, y alejarlo de las otras clases.

Es un algoritmo de aprendizaje competitivo no supervisado. En un algoritmo competitivo, La anterior regla de adaptación fue modificada con LVQ2. Asumiendo que dos centros m_i *y* m_j, que pertenecen a clases diferentes, son los más cercanos a un determinado patrón de entrada x, se define una ventana simétrica, de anchura mayor que cero, entre ambos centros, de tal manera que m_i y m_j solo serán actualizados si x cae dentro de dicha ventana (ver Figura 3.13). LVQ2 se basa en la idea de aproximar las fronteras de decisión hacia la frontera ideal de Bayes.

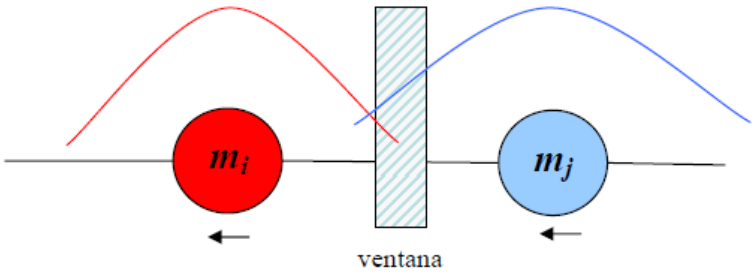


Figura 3.13 Representación de la ventana empleada en LVQ2

Si *Ci* es la clase del centro más cercano, pero x pertenece a *Cj≠ Ci,* donde *Cj* es la clase del segundo del centro más cercano y además x tiene que "caer" dentro de la "ventana", entonces:.

$$\mathbf{m}_i \leftarrow \mathbf{m}_i - \alpha(t)\cdot\left[\mathbf{x}-\mathbf{m}_i\right]$$
$$\mathbf{m}_j \leftarrow \mathbf{m}_j + \alpha(t)\cdot\left[\mathbf{x}-\mathbf{m}_j\right] \qquad (3.37)$$

en todos los demás casos, no hay actualización:

$$\mathbf{m}_k \leftarrow \mathbf{m}_k$$

El tamaño óptimo de la ventana debe ser determinado experimentalmente y depende del número de muestras.

Un desarrollo de este algoritmo modificando el anterior sería el LVQ3, cuyas ecuaciones mostramos a continuación.

Si m_i *y* m_j son los centros más cercanos a la muestra x, y x y m_j pertenecen a la misma clase *Cj*, mientras que mi pertenece a *Cj≠ Ci* y además x tiene que "caer" dentro de la "ventana", entonces:

$$\mathbf{m}_i \leftarrow \mathbf{m}_i - \alpha(t)\cdot[\mathbf{x}-\mathbf{m}_i]$$
$$\mathbf{m}_j \leftarrow \mathbf{m}_j + \alpha(t)\cdot[\mathbf{x}-\mathbf{m}_j] \qquad (3.38)$$

y su x, m_i *y* m_j pertenecen a la misma clase:

$$\mathbf{m}_k \leftarrow \mathbf{m}_k + \varepsilon\cdot\alpha(t)\cdot[\mathbf{x}-\mathbf{m}_k]$$

en todos los demás casos, no hay actualización:

$$\mathbf{m}_k \leftarrow \mathbf{m}_k$$

El valor óptimo de ε depende del tamaño de la ventana, siendo menor conforme decrece el tamaño de ventana.

3.4.3.4 Ajuste de parámetro de normalización σ.

Hasta ahora sólo se han ajustado las posiciones de los centroides en el espacio de los patrones de entrada. Sin embargo, se hace necesario además ajustar los valores de los parámetros de normalización o desviaciones σ_j de cada neurona. Estos valores son importantes porque determinan el área o zona de influencia de cada neurona de la capa oculta. Cuanto mayor sea la desviación asociada a una neurona, mayor será el campo de acción de esa neurona, de tal forma que una neurona da mayores valores de salida para un patrón que está cerca de esa neurona (centroide), y tanto más cuanto más grande es la desviación. Lo que ocurre es que por ejemplo, una neurona puede dar valores próximos a cero para un determinado patrón (de ahí el carácter local de las neuronas), pero si se aumenta la anchura o desviación asociada a la neurona, entonces la salida patrón puede variar enormemente.

Los valores de estos parámetros deberán ser tales que se cubra bien el espacio de entrada. Una posible forma de conseguir esto será haciendo que el radio **O** de una neurona fuese igual a la máxima distancia del conjunto de los patrones asociados a él, con respecto al centroide. Esta forma, presenta el inconveniente de polarizar los pesos en torno a los patrones de entrenamiento (y por tanto se puede obtener una mala generalización) y además en el caso de que el conjunto de patrones asociados a una neurona estuviese formado por un único patrón, este parámetro sería nulo y solo daría salida significativa en el caso de que un patrón coincidiese con el centroide.

En la práctica se recurre a fijar ese valor de una forma heurística, utilizándose diversas técnicas de las que destacamos:

- Tomar como valor de σ_j , un valor fijo para todos los casos e igual

$$\sigma^2 = \frac{d^2}{2m} \qquad (3.39)$$

siendo d la máxima distancia entre centroides. Este método es simulado en este proyecto.

• Tomar como valor de O un promedio de las distancias del centroide con respecto al resto de los centroides. Existen varias posibilidades:

1) Media geométrica de la distancia euclídea del centroide j-ésimo a los dos centroides más cercanos

$$\sigma_j = \sqrt{d_{j,1} \cdot d_{j,2}} \tag{3.40}$$

donde $d_{j,1}$ y $d_{j,2}$ son las distancias a los dos centros más cercanos.

2) Estimación a partir de los patrones por cada clase

$$\sigma_j^2 = \frac{1}{\#C_j} \sum_{\mathbf{x} \in C_j} \left\| \mathbf{x} - \mathbf{c}_j \right\|^2 \tag{3.41}$$

donde C_j es la clase a la que pertenece cada centroide c_j . Pero antes es necesario establecer la pertenencia de cada centroide a una determinada clase, para ello se asigna a cada centroide los patrones que estén a la menor distancia, y la clase C_j es aquella que sea mayoritaria dentro del grupo de patrones asignados a cada centroide c_j. Este método también se ha empleado en el proyecto.

Las desviaciones pueden influir mucho en la *capacidad de generalización* de una red neuronal u otra. Para ver esto con mayor claridad, piénsese primero en obtener unos errores de aprendizaje lo más bajos posibles para lo cual lo más sencillo es tener tantas neuronas como patrones de entrada; si los valores de las anchuras son altos, las neuronas tendrán poco carácter local con lo que una neurona dará valores altos para patrones que estén alejados de ella, interfiriendo en la salida que sobre ese patrón de una neurona que esté más cerca de ese patrón, y aumentando el error de aprendizaje. Si las anchuras son pequeñas, posiblemente las salidas que las neuronas den para un patrón alejado serán próximas a 0, dando valores altos sólo para los patrones que estén más cerca de esa neurona y el error de aprendizaje disminuya.

El problema es que cuando se entrena la red hay que atender también a los *errores de validación* o *test*, y este error es el que determina la capacidad de generalización de la red, que no es más que la reacción de la red ante nuevos datos no aprendidos, es decir que para valores de entrada no aprendidos, la salida de la red ha de ser lo más próxima a la salida real. Si las desviaciones no son lo suficientemente grandes como para atender a esos nuevos patrones, las neuronas devolverán valores muy pequeños, y la salida de la

red no se va a adecuar a los valores reales. En definitiva, entender que la desviación asignada a una neurona es algo muy importante, y que con desviaciones pequeñas, el error de entrenamiento puede ser pequeño pero el de test puede ser muy grande, con lo que hay que buscar que ambos errores sean parecidos y lo más bajos posibles.

3.4.3.5 Entrenamiento de la capa de salida.

Tras acabar el entrenamiento de los nodos ocultos, se procede a entrenar las neuronas que conforman la capa de salida. Para ello se usaran las distintas salidas $\phi_1,\phi_2,\ldots,\phi_m$ que produce la capa de nodos ocultos para los patrones de entrada x. Estas salidas son en general función de los valores de entrada, los centroides obtenidos en el entrenamiento de la capa oculta y los radios σ. Se pueden emplear distintos métodos de optimización como algoritmos de descenso de gradiente, de gradiente conjugado, de Newton, de Quasi-Newton… Un ejemplo es el algoritmo LMS Least Mean Squares, también conocido como algoritmo de Widrow-Hoff o regla delta (ya que trata de minimizar una diferencia entre el valor observado y el deseado en la salida de la red). Pertenece a la familia de los algoritmos de gradiente estocástico. Con el término"estocástico" se pretende distinguir este algoritmo del *Steepest Descent*, que utiliza un gradiente determinista para el cálculo de los pesos. Una característica importante del LMS es su simplicidad.

Dado el conjunto $O= \{o_1 , o_2 ,..., o_m\}$, siendo $(o_n)_{n=1..N}$ la salida de cada neurona de la capa oculta debida a cada patrón de entrada $(X_n)_{n=1..N}$, y sus N salidas deseadas $T= \{t_1 , t_2 ,..., t_m\}$, este algoritmo pretende encontrar los pesos w que minimizan la función de error cuadrático medio dado por:

$$E = \frac{1}{2}\sum_{n=1}^{N} \varepsilon_n^2 \quad siendo\ \varepsilon_n = y\left(\mathbf{o}^n;\mathbf{w}\right) - t^n \tag{3.42}$$

Siendo y la salida lineal de la red,

$$y = \mathbf{w}^T \cdot \mathbf{o} = \sum_i w_i \cdot o_i \tag{3.43}$$

La función de error es una función matemática definida en el espacio de pesos multidimensional para un conjunto de patrones dados. Es una superficie que tendrá muchos mínimos (globales y locales), y el algoritmo va a buscar el punto en el espacio de pesos donde se encuentra el mínimo global de esta superficie. Aunque la superficie de error es desconocida, el método LMS consigue obtener información local de dicha superficie a través del gradiente. Con esta información se decide qué dirección tomar para llegar hasta el mínimo global de dicha superficie.

Basándose en el método del gradiente decreciente, se obtiene una regla para modificar los pesos de tal manera que hallamos un nuevo punto en el espacio de pesos más próximo al punto mínimo. Es decir, las modificaciones en los pesos son proporcionales al gradiente decreciente de la función error:

$$\Delta w_j = -\alpha \frac{\partial E}{\partial w_j} \tag{3.44}$$

Por tanto, si se deriva la función error con respecto a los pesos y si se aplica la regla de la cadena para el cálculo de dicha derivada, se obtiene la siguiente ecuación de actualización de los pesos (versión bloque):

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta\mathbf{w}(k) = \mathbf{w}(k) + \frac{\alpha}{2} \cdot \sum_{n=1}^{N} \left(t^n - y\left(\mathbf{o}^n; \mathbf{w}(k)\right)\right) \cdot \mathbf{o}^n \tag{3.45}$$

El valor del parámetro α tiene una gran influencia sobre el entrenamiento. Si α es demasiado grande, la convergencia es posible que no se produzca, debido a que se darán "saltos" en torno al mínimo sin alcanzarlo. Si α es demasiado pequeño, alcanzaremos la convergencia, pero la etapa de aprendizaje será más larga.

En cuanto al momento en el que se debe detener el entrenamiento, depende de los requerimientos del sistema. El entrenamiento se detiene cuando el error observado es menor que el admisible. Se suele tomar el error cuadrático medio como la magnitud que determina el instante en el que el algoritmo ha convergido.

3.5 Máquinas de vectores soporte.

La formulación de las máquinas de vectores soporte (*Support Vector Machine*, SVM) se basa en el principio de minimización estructural del riesgo, que ha demostrado ser superior al principio de minimización del riesgo empírico, que es el empleado por muchas de las redes neuronales convencionales [25], [26].

3.5.1 Minimización del error.

Para una determinada tarea de aprendizaje, con una cantidad finita de datos de entrenamiento, el mejor rendimiento al generalizar se conseguirá si se equilibra la balanza entre la *exactitud* obtenida sobre ese conjunto de entrenamiento y la *capacidad de generalización* de la máquina, esto es, la habilidad cualquier conjunto de entrenamiento sin error.

Sea un conjunto de N patrones junto con sus salidas deseadas $\{(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)\}$. Para simplificar las fórmulas siguientes, la salida esperada t_n será +1 o -1.

Ahora supongamos que existe una distribución de probabilidad desconocida $P(x, t)$ de la que se han tomado esas muestras. Esto es más general que asignar un t fijo a cada x. Con esta distribución, las etiquetas t_n se asignan según una distribución de probabilidad condicional en. x_i . Más adelante, sin embargo, se asumirá un t fijo para un x dado.

Supongamos que tenemos una máquina para aprender la transformación $x_i \rightarrow t_i$. La máquina está definida por $x \rightarrow f(x,\zeta)$ donde ζ es un conjunto de parámetros ajustables (por ejemplo, en una red neuronal los pesos y los umbrales). Una elección particular de ζ genera lo que llamamos una *máquina entrenada*.

La esperanza del error de evaluación para una máquina entrenada, llamada también *riesgo real, riesgo esperado* o simplemente *riesgo*, es

$$R(\zeta) = \int \frac{1}{2} \left| t - f(\mathbf{x}, \zeta) \right| dP(\mathbf{x}, t) \tag{3.46}$$

sin embargo, $P(x, t)$ es desconocida normalmente.

El *riesgo empírico* es el error medio medido sobre el conjunto de entrenamiento:

$$R_{emp}(\zeta) = \frac{1}{2} \sum_{i=1}^{N} \left| t_i - f(\mathbf{x}_i, \zeta) \right| \tag{3.47}$$

Nótese que aquí no aparece ninguna distribución de probabilidad. A la cantidad

$$\frac{1}{2} \left| t_i - f(\mathbf{x}_i, \zeta) \right| \tag{3.48}$$

se le denomina perdida. Ahora si se escoge un ρ tal que $0 \leq \rho \leq 1$. Entonces, la siguiente cota se cumple con probabilidad 1- ρ:

$$R(\zeta) \leq R_{emp}(\zeta) + \sqrt{\frac{h\left(\log\left(2N/h\right)+1\right) - \log\left(\rho/4\right)}{N}} \tag{3.49}$$

donde h es un número entero no negativo llamado dimensión de *Vapnik-Chervonenkis* (VC) y es una medida de la idea de capacidad de generalización mencionada anteriormente. A la parte derecha de la desigualdad se le llama *cota del riesgo* y al segundo término de la cota del riesgo se le llama *confianza de VC*.

Algunas notas sobre esta cota:

- es independiente de $P(x, t)$,
- normalmente no es posible calcular la parte izquierda de la desigualdad,

- si se conociera h, se podría calcular la parte derecha. De esta forma, dadas diferentes máquinas de aprendizaje (familias de funciones $f(x,\zeta)$ y eligiendo un valor de ρ lo suficientemente pequeño, si seleccionamos la máquina que minimice la parte derecha de la desigualdad, estaremos eligiendo la máquina que da la cota superior más pequeña del riesgo real.

3.5.2 La dimensión de Vapnik-Chervonenkis.

La dimensión VC es una propiedad de un conjunto de funciones $\{f(\zeta)\}$ y puede definirse para distintas clases de funciones f. Aquí nos centraremos en funciones de la forma $f(x,\zeta) \in \{+1,-1\}\ \forall\ x, \zeta$

Si un conjunto de N patrones puede etiquetarse de 2^N formas distintas, y para cada etiquetado puede encontrarse un elemento del conjunto $\{f(\zeta)\}$ que asigne correctamente estas etiquetas, diremos que el conjunto de patrones es roto por ese conjunto de funciones. La dimensión VC del conjunto de funciones $\{f(\zeta)\}$ se define como el máximo número de patrones de entrenamiento que pueden ser rotos por $\{\{f(\zeta)\}$. Nótese que si la dimensión VC es h, entonces existe como mínimo un conjunto de patrones que puede romperse, pero en general no todos los conjuntos de h patrones podrán romperse.

Supongamos que el espacio en que viven los datos es $\mathbf{R}^2$ y que $\{f(\zeta)\}$ está formado por líneas rectas orientadas. Aunque es posible encontrar tres puntos que pueden romperse con este conjunto de funciones (Figura 3.14), es imposible encontrar cuatro. Por lo tanto la dimensión VC del conjunto de líneas rectas orientadas en $\mathbf{R}^2$ es siempre 3. Nótese de paso que, aunque los puntos de la Figura 3.14 pueden romperse, no todos los conjuntos de tres puntos $\mathbf{R}^2$ en pueden romperse. Por ejemplo, si los tres puntos son colineales, es imposible romperlos con líneas orientadas.

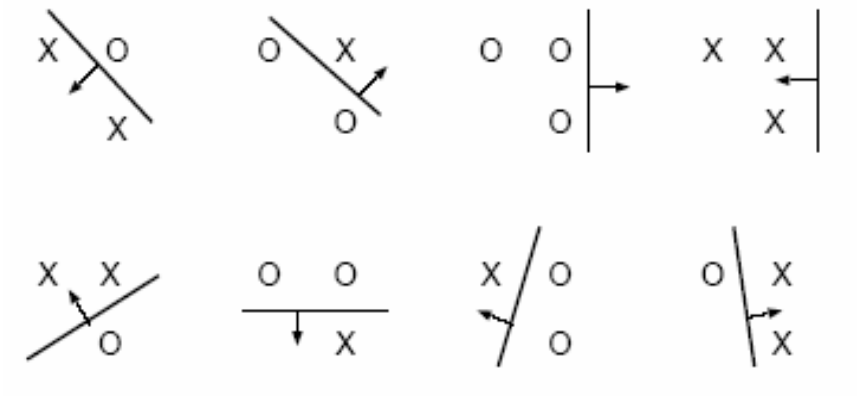


Figura 3.14 Tres puntos en $\mathbf{R}^2$ separados por líneas rectas

El resultado anterior se generaliza en el siguiente teorema y corolario:

• *Teorema*. Consideremos un conjunto de patrones de $\mathbf{R}^n$ y tomemos cualquiera de ellos como origen. Los *N* patrones pueden romperse con un hiperplano orientado si, y solo si, los vectores de posición de los restantes patrones son linealmente independientes.

• *Corolario*. La dimensión VC del conjunto de hiperplanos orientados en $\mathbf{R}^n$ es $n+1$, ya que siempre pueden encontrarse n patrones linealmente independientes (dado un origen cualquiera), pero nunca $n+1$.

La dimensión VC formaliza la idea la capacidad de un conjunto de funciones. Intuitivamente cabe esperar que máquinas con muchos parámetros tengan una dimensión VC alta y que máquinas con pocos parámetros su dimensión sea baja. Aunque, esto suele ser así, hay un contraejemplo: una maquina con un único parámetro y con dimensión VC infinita (puede romper cualquier conjunto de independientemente de *N*)

$$f(x,\zeta) = \theta\left(sen(\zeta x)\right), \quad x,\zeta \in \mathbb{R} \tag{3.50}$$

Donde θ es la función escalón definida como

$$\theta(x) = \begin{cases} +1 & x > 0 \\ -1 & x \le 0 \end{cases} \tag{3.51}$$

La dimensión VC formaliza Como curiosidad, aunque con esta función se puede romper un número arbitrario de puntos, es posible encontrar cuatro puntos que no pueden ser rotos (Figura 3.15).

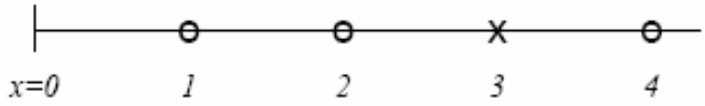


Figura 3.15 Cuatro puntos no separables a pesar de tener dimensión VC infinita

La Figura 3.16 muestra la evolución con un nivel de confianza del 95% (ρ = 0.05) de la confianza VC al aumentar la dimensión VC. Como puede verse, la confianza es monótonamente creciente con h. Aunque la gráfica está generada para un valor de N = 10000, lo anterior es cierto para cualquier valor de N.

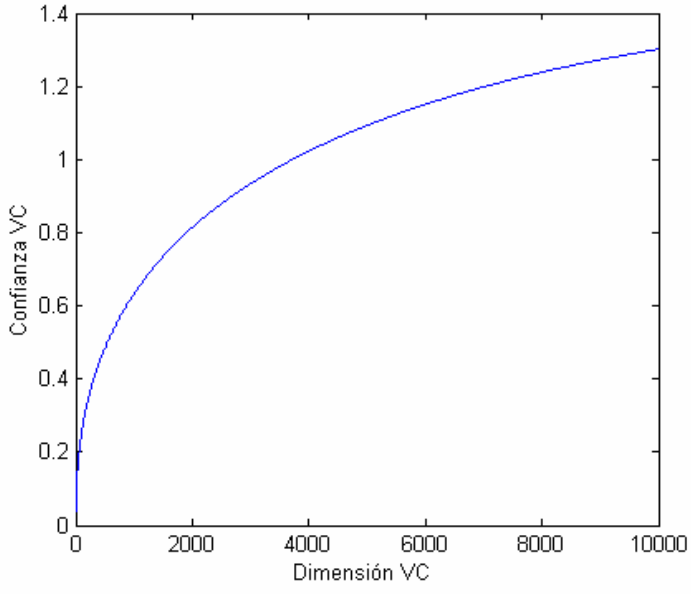


Figura 3.16 Confianza VC respecto a la dimensión VC

Por lo tanto, si tenemos un conjunto de máquinas de aprendizaje cuyo riesgo empírico es cero, nos quedaremos con aquella de menor dimensión VC, es decir, aquella que minimiza la cota del riesgo.

De cualquier forma, es importante tener en cuenta que la ecuación 3.49 da (con una determinada probabilidad) una cota superior del riesgo real, y que esto no impide que una determinada máquina con el mismo R_{emp} y cuyo conjunto de funciones asociado presente una dimensión VC mayor tenga mejor rendimiento. Un ejemplo de sistema de este tipo que da un buen rendimiento pese a tener dimensión VC infinita es el de un clasificador basado en los k vecinos más cercanos con $k = 1$. Este conjunto de funciones tiene dimensión VC infinita y riesgo empírico cero, ya que cualquier conjunto de patrones etiquetados arbitrariamente será aprendido correctamente por el algoritmo. En este caso la cota no da ninguna información. Con todo, los clasificadores basados en el vecino más cercano funcionan bien. Así, una capacidad infinita no implica un rendimiento pobre.

La confianza VC de la ecuación 3.45 depende de la clase de funciones escogidas, mientras que el *riesgo empírico* y el *riesgo ideal* dependen de la función particular elegida por el algoritmo de entrenamiento.

El objetivo es encontrar un subconjunto del conjunto de funciones que minimice la *cota de riesgo*. Para ello se divide la clase completa de funciones en subconjuntos anidados. Para cada conjunto se debería poder calcular h o, al menos, establecer una cota de su valor. La *minimización estructural del riesgo* consiste en encontrar el subconjunto de funciones que minimiza la cota de error actual. Entonces se toma aquella máquina entrenada de la serie con menor valor para la suma del riesgo empírico y la confianza VC.

3.5.3 Máquinas de vectores soporte lineales.

Comenzaremos con máquinas lineales entrenadas con datos linealmente separables (el caso general, máquinas no lineales entrenadas con datos no separables, se resuelve con un problema de programación cuadrática muy similar).

Consideremos el conjunto de entrenamiento

$$\{(x_1 , t_1), (x_2 , t_2),...,(x_N , t_N)\}\ t_i \in \{+1,-1\}\ x_i \in \mathbf{R}^d \tag{3.52}$$

y supongamos que existe un hiperplano que separa los puntos de ambas clases. Se verifica que los puntos x sobre el hiperplano satisfacen la ecuación $\omega^T x + b = 0$ donde el vector ω es normal al hiperplano, $|b|/\|w\|$ es la distancia perpendicular del hiperplano al origen y $\|w\|$ es la norma euclídea de ω.

Sean d_+ *(*d_-*)* la distancia más corta al hiperplano de separación al patrón perteneciente a la clase +1 (perteneciente a la clase -1, respectivamente) más cercano. Definamos el *margen* del hiperplano como la suma $d_+ + d_-$. En el caso linealmente separable, el algoritmo buscara simplemente el hiperplano con mayor margen. A continuación se formula este concepto.

Supongamos que todos los datos de entrenamiento satisfacen

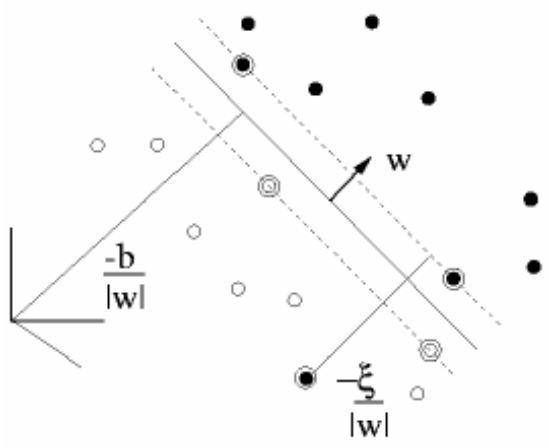
$$\omega^T x + b \geq +1 \text{ para } t_i = +1 \tag{3.53a}$$

$$\omega^T x + b \leq -1 \text{ para } t_i = -1 \tag{3.53b}$$

esto es

$$t_i\ (\omega^T x + b) - 1 \geq 0\ \ \forall i \tag{3.54}$$

Ahora consideremos los patrones para los que se cumple la igualdad en la ecuación 3.53a. Estos patrones están sobre el hiperplano H_1: $\omega^T x + b = +1$ con normal ω y distancia al origen $|1-b|/\|w\|$. De forma similar, para el hiperplano H_2: $\omega^T x + b = -1$, la distancia al origen es $|-1-b|/\|w\|$. Por tanto, $d_+ = d_- = 1/\|w\|$ y el margen es simplemente $2/\|w\|$. Nótese que H_1 y H_2 son paralelos (tienen la misma normal) y que no hay patrones de entrenamiento entre ellos. Podemos en definitiva, encontrar el par de hiperplanos que dan el máximo margen minimizando la función de coste $1/2\|w\|^2$ con las restricciones de la ecuación 3.54.



Nota: Los vectores soporte están rodeados por un circulo

Figura 3.17 Separación mediante hiperplanos lineales para el caso separable.

Ahora pasaremos a una formulación lagrangiana del problema. Hay dos razones importantes para hacer esto:

• la primera es que las restricciones de la ecuación 3.54 se sustituirán por restricciones sobre los *multiplicadores de Lagrange*, que serán más fáciles de manejar,

• la segunda es que con esta reformulación del problema, los patrones de entrenamiento solo aparecen en forma de productos escalares entre vectores. Esta propiedad es crucial para poder generalizar el procedimiento al caso no lineal.

Por lo tanto, introduzcamos N multiplicadores de Lagrange que denotaremos por $\alpha_1,\alpha_2,\ldots,\alpha_N$, uno para cada una de las restricciones de la ecuación 3.54. La regla es que para restricciones de la forma $c_i \geq 0$ las ecuaciones que las definen se multiplican por multiplicadores de Lagrange positivos y se restan de la función objetivo para formar el lagrangiano. En el caso de restricciones de la forma $c_i = 0$, los multiplicadores de Lagrange no tienen restricciones. Lo anterior da el lagrangiano

$$L_P = \frac{1}{2}\|\boldsymbol{\omega}\|^2 - \sum_{i=1}^{N} \alpha_i t_i \left(\boldsymbol{\omega}^T \mathbf{x}_i + b\right) + \sum_{i=1}^{N} \alpha_i \qquad (3.55)$$

A continuación se debe minimizar L_P con respecto a ω, b y a la vez exigir que las derivadas de L_P con respecto a todos los α_i se anulen, todo sujeto a las restricciones $\alpha_i \geq 0$ (restricciones C_1). Puede demostrarse que se trata de un problema de programación cuadrática convexo. Esto quiere decir que podemos resolver de forma equivalente el siguiente problema dual: maximizar L_P sujeto a la restricción de que el gradiente de L_P con respecto a ω y b se anule y sujeto también a la restricción de que $\alpha_i \geq 0$ (restricciones C_2). Esta formulación particular del problema se conoce como *dual de Wolfe* y presenta la propiedad de que el máximo de L_P con las restricciones C_2 ocurre en los mismos valores de ω, b y α que el mínimo de C_1.

Al requerir que se anule el gradiente L_P con respecto a ω, b, obtenemos las condiciones:

$$\boldsymbol{\omega} = \sum_i \alpha_i t_i \mathbf{x}_i \tag{3.56}$$

$$\sum_i \alpha_i t_i = 0 \tag{3.57}$$

Ya que estas restricciones aparecen como igualdades, podemos sustituirlas en la ecuación 3.55 para obtener

$$L_D = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j t_i t_j \mathbf{x}_i \mathbf{x}_j \tag{3.58}$$

La solución se obtiene minimizando L_P o maximizando L_D y viene dado por la ecuación 3.55.

Nótese que hay un multiplicador de Lagrange α_i para cada patrón de entrenamiento. Tras obtener una solución, aquellos puntos para los que $\alpha_i \geq 0$ se denominan *vectores soporte* y yacen sobre los hiperplanos H_1 y H_2. El resto de patrones tienen $\alpha_i=0$ y satisfacen la ecuación 3.54, quizá con la igualdad. Con estas máquinas, los vectores soporte son los elementos críticos del conjunto entrenamiento: son los más cercanos a la frontera de decisión y si el resto de patrones no se consideraran en un nuevo entrenamiento, el algoritmo encontraría el mismo hiperplano de separación.

ω está determinado explícitamente por el algoritmo de entrenamiento, pero no es el caso del umbral b, aunque su obtención es inmediata: basta tomar en la ecuación 3.54 cualquier i para el que $\alpha_i \neq 0$ y despejar b; por ejemplo, si $t_i=1$ (vector positivo):

$$b=1-\omega^T x_i \tag{3.59}$$

En cualquier caso, siempre es más seguro tomar el valor medio de b que resulte de todas las ecuaciones.

En el caso no linealmente separable nos interesa poder relajar las restricciones de las ecuaciones 3.56, pero únicamente cuando sea necesario, es decir, nos interesa añadir un nuevo coste a la función de objetivo. Una posible forma de hacerlo es introduciendo variables débiles ζ_i, $i=1,\ldots,N$ en las restricciones para convertirlas en:

$$\omega^T x + b \geq +1 - \zeta_i \quad para\ t_i = +1 \tag{3.60a}$$

$$\omega^T x + b \leq -1 + \zeta_i \quad para\ t_i = -1 \tag{3.60b}$$

$$\zeta_i \geq 0 \quad \forall i$$

Con esto, para que una muestra quede mal clasificada, el correspondiente ζ_i debe ser mayor que la unidad. La suma $\sum_i \zeta_i$ es, por tanto, una cota superior en el número de errores sobre el entrenamiento. Una forma de añadir el coste de los errores a la función

objetivo es intentar minimizar $1/2\|w\|^2 + \gamma(\sum_i \zeta_i)$ donde γ es un parámetro ajustable, un valor grande de γ equivale a una mayor penalización de los errores.

Este problema es un problema de programación cuadrática con la propiedad de que ni los ζ_i ni sus multiplicadores de Lagrange asociados, aparecen en el problema *dual de Wolfe*, ecuación 3.58, sujeto ahora por las restricciones

$$0 \leq \alpha_i \leq \zeta_i. \tag{3.61}$$

$$\sum_i \alpha_i t_i = 0 \tag{3.62}$$

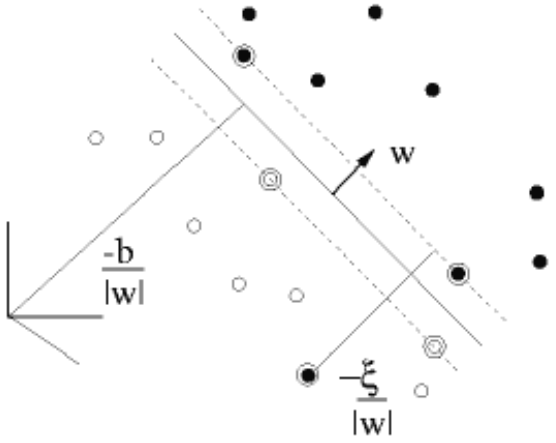
La solución viene dada nuevamente por:

$$\boldsymbol{\omega} = \sum_{i=1}^{N^S} \alpha_i t_i \mathbf{s}_i \tag{3.63}$$

Donde ahora N^S es el número de vectores soporte y s_i es el i-ésimo vector soporte. Por lo tanto, la única diferencia con el caso del hiperplano óptimo es que los valores de α_i están ahora acotados superiormente por γ. La Figura 3.18 ilustra gráficamente algunas de las ideas anteriores.

Aunque no se demostrará, el cálculo del umbral b sigue de nuevo la expresión $b=1-\omega^T x_i$ con $0 \leq \alpha_i \leq \gamma$. Al igual que en el caso linealmente separable, es más adecuado tomar la media sobre todos los vectores soporte.

En cuanto al parámetro γ, conforme $\gamma \rightarrow 0$, la solución converge a la obtenida con el hiperplano óptimo (sobre el problema no separable). Por otro lado conforme $\gamma \rightarrow \infty$, la solución converge hacia una en la que domina el término de maximización del margen y se hace énfasis en minimizar el error de clasificación.



Nota: Los vectores soporte están rodeados por un circulo

Figura 3.18 Separación mediante hiperplanos lineales para el caso no separable.

3.5.4 Máquinas de vectores soporte no lineales.

Si se realiza una transformación del espacio de entrada a otro espacio de alta dimensionalidad en el que los datos son separables linealmente. En primer lugar, observemos que la única forma en que aparecen los datos en las ecuaciones 3.58, 361, 362 como productos escalares $x_i \cdot x_j$. Supongamos entonces que llevamos los datos a un nuevo espacio euclídeo H, denominado espacio de Hilbert, (posiblemente de *dimensión infinita*) mediante una transformación $\aleph$ (ver la Figura 3.19) tal que

$$\aleph:\mathbb{R}^m \rightarrow H \tag{3.64}$$

Con esto, el algoritmo de entrenamiento solo dependerá de los datos a través de productos escalares en *H*, es decir, de funciones de la forma $\aleph(x_i)\cdot\aleph(x_j)$. Si existiera una función núcleo (*kernel*) *K* tal que $K(x_i,x_j) = \aleph(x_i)\cdot\aleph(x_j)$, podríamos usar únicamente en el algoritmo de entrenamiento sin tener que conocer explícitamente $\aleph$.

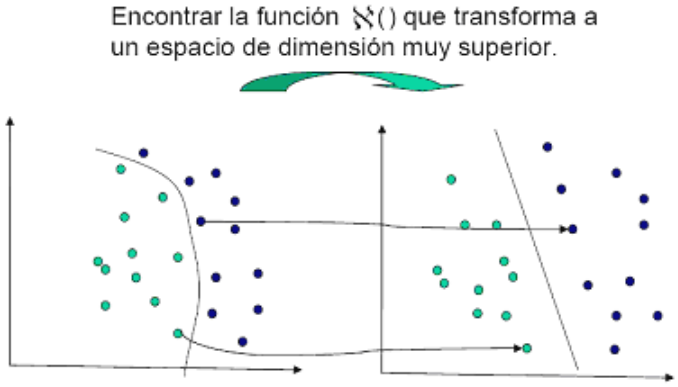


Figura 3.19 Transformación del espacio de entrada a un espacio de dimensión muy superior

.

Un ejemplo de función núcleo es

$$K(\mathbf{x}_i,\mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2\sigma^2}\right) \tag{3.65}$$

en este ejemplo H es de dimensión infinita y no sería sencillo trabajar explícitamente con $\aleph$. Sin embargo, si sustituimos $x_i \cdot x_j$. por $K(x_i,x_j)$ a lo largo de todo el algoritmo de entrenamiento, obtendremos una SVM que vive en un espacio de dimensión infinita y que opera en prácticamente la misma cantidad de tiempo. La separación sigue siendo lineal, pero en un espacio diferente. En la fase de evaluación, la SVM se usa calculando el producto escalar de un patrón de entrenamiento dado *x* con ω y calculando el signo de

$$f(\mathbf{x}) = \sum_{i=1}^{N^S} \alpha_i t_i \aleph(\mathbf{s}_i)\cdot\aleph(\mathbf{x}) + b = \sum_{i=1}^{N^S} \alpha_i t_i K(\mathbf{s}_i,\mathbf{x}) + b \tag{3.66}$$

donde sí son los vectores soporte. De nuevo, por lo tanto, podemos evitar el cálculo directo de $\aleph(x)$.

Capítulo 4
Análisis del algoritmo Extreme Learning Machine (ELM)

Las Extreme Learning Machine (ELM) propuestas por Huang [27], [2] utilizan una arquitectura de Single Layer Feedforward Neural Network. (SLFN) [28]. Aleatoriamente elige los pesos de entrada y analíticamente determina los pesos de salida de la SLFN. Posee mayor capacidad de generalización gracias a su rápida velocidad de aprendizaje a diferencia de otros tipos de RNAs, requiere de una menor intervención humana y su potencia es mil veces superior a otras técnicas tradicionales [29]. Contando con la ventaja de la automatización de la selección de todos los parámetros de la red, lo que evita estas intervenciones humanas triviales, convirtiéndose en una eficiente herramienta en aplicaciones online y de tiempo real, y se puede utilizar para muchas funciones de activación no lineales y funciones de kernel.

4.1 Una nota sobre SLFN.

Las Single Layer Feedforward Neural Network con L nodos ocultos, pueden ser representadas como la representación matemática de una SFLN incorporando nodos ocultos RBF y nodos aditivos en las siguientes ecuaciónes

$$f_L(x) = \sum_{i=1}^{L} \beta_i G(a_i, b_i, x), \qquad x \in R^n, \qquad a_i \in R^n \tag{4.1}$$

donde a_i y b_i son los parámetros de aprendizaje de los nodos ocultos y β_i los pesos conectando el i-enésimo nodo al nodo de salida. $G(a_i,b_i,x)$ es la salida del i-enésimo nodo oculto con respecto a la entrada x. Para los nodos ocultos aditivos la función de activación $g(x) : R \rightarrow R$ (p.e., sigmoide y umbral) , $G(a_i,b_i,x)$ es dada por

$$G(a_i, b_i, x) = g(a_i.x + b_i), \qquad b_i \in R \tag{4.2}$$

donde a_i es el vector de peso conectando la capa de entrada al i-enésimo nodo y b_i es el sesgo de los i-enésimo nodo. a_i,x denota el producto vectorial entre a_i y x en R^n.

Para el nodo oculto RBF con función de activación $g(x): R \rightarrow R$ (p.e., Gausiana), $G(a_i,b_i,x)$ está dada por la ecuación

$$G(a_i, b_i, x) = g(b_i\|x - a_i\|), \qquad b_i \in R^+ \tag{4.3}$$

Donde a_i y b_i son los centros y el *factor de impacto* del i-enésimo nodo RBF, R^+ indica que está definida en el conjunto de valores reales positivos. Una red RBF por tanto es un caso especial de SLFN con RBF nodos en su capa oculta.

Para N, muestras arbitrarias $(x_i, t_i) \in R^n$ x R^m. Aquí x_i, es un vector n x 1 de entrada y t_i, es un vector m x 1 objetivo. Si un SLFN con L nodos ocultos pueden aproximar estas N muestras con error cero, esto implica que existen unos β_i, a_i y x tal que

$$f_L(x_j) = \sum_{i=1}^{L} \beta_i G(a_i, b_i, x_j), \qquad j = 1, ..., N. \tag{4.4}$$

esta última expresión puede compactarse como

$$H\beta = T \tag{4.5}$$

donde

$$H(\tilde{a}, \tilde{b}, \tilde{x}) = \begin{bmatrix} G(a_1, b_1, x_1) & \cdots & G(a_L, b_L, x_1) \\ & \cdots & \\ G(a_1, b_1, x_N) & \cdots & G(a_L, b_L, x_N) \end{bmatrix}_{N \times L} \tag{4.6}$$

con $\tilde{a} = a_1, \cdots, a_L;\ \tilde{b} = b_1, \cdots, b_L;\ \tilde{x} = x_1, \cdots, x_N.$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{L \times m} \quad and \quad T = \begin{bmatrix} t_1^T \\ \vdots \\ t_N^T \end{bmatrix}_{N \times m} \tag{4.7}$$

H es la matriz de la capa oculta de salida de la SFLN con i-enésima columna de H siendo el i-enésimo nodo oculto de salida respecto a las entradas $x_1, x_1, ., x_N$.

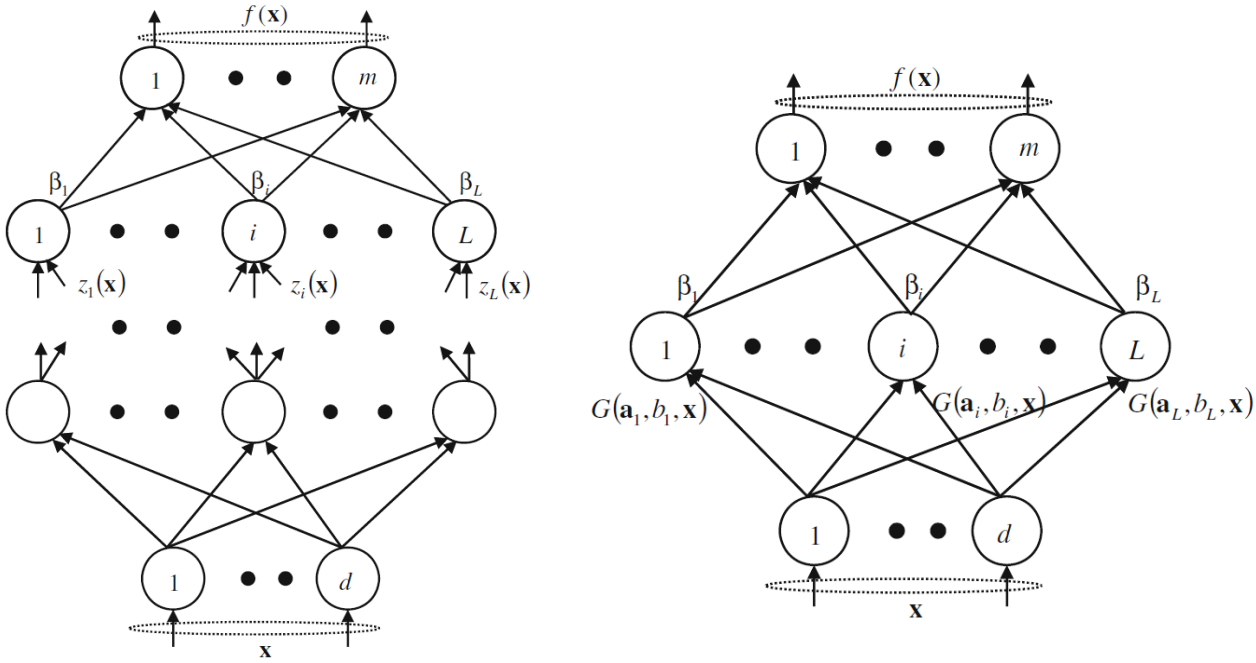


Figura 4.1 Comparación entre una MLP con N capas ocultas y una SLFN.

4.2 Principios de ELM.

Una ELM diseñada como una SLFN con L neuronas ocultas pueden aprender L muestras distintas con error cero. Incluso si el número de neuronas ocultas (L) < el número de muestras distintas (N), ELM puede todavía asignar parámetros aleatorios a los nodos ocultos y calcular los pesos de salida usando la pseudo-inversa de H proporcionando sólo un pequeño error $\varepsilon > 0$. Los parámetros de los nodos ocultos de ELM a_i y b_i (pesos de entrada y *sesgos* o centros y *factores de impacto*). No necesitan ser "sintonizados" durante el entrenamiento y pueden ser asignados simplemente con valores aleatorios. Los siguientes teoremas concluirán en esta afirmación.

4.2.1 Teorema 1.

Descrito por Liang [30], dada una SLFN con L nodos ocultos (aditivos o RBF) y una función de activación $g(x)$ infinitamente diferenciable en cualquier intervalo de un espacio R dado. Entonces, para L distintos vectores de entrada $\{x_i | x_i \varepsilon R^n, i = 1, \ldots, L\}$ y unos $\{(a_i, b_i)\}_{i=1}^{L}$ aleatoriamente generados por cualquier fdp continua, respectivamente, la matriz de salida de la capa oculta es invertible con probabilidad 1, la matriz de salida de la capa oculta H de la SLFN es invertible y $\|H\beta - T\| = 0$.

4.2.2 Teorema 2.

También enunciado por Liang [30], dado cualquier pequeño valor positivo de $\varepsilon > 0$ y una función de activación $g(x)$: $R \rightarrow R$ que es infinitamente diferenciable en cualquier intervalo, entonces existe $L \leq N$ tal que para un N arbitrarios vectores de entrada distintos, $\{x_i | x_i \varepsilon R^n, i = 1, \ldots, L\}$, y unos $\{(a_i, b_i)\}_{i=1}^{L}$ aleatoriamente generados por cualquier fdp continua $\|H_{N\times L}\beta_{L\times m} - T_{N\times m}\| < \varepsilon$ con probabilidad igual a uno.

Ya que los parámetros de los nodos ocultos de una ELM no necesitan ser sintonizados durante el entrenamiento, y que son simplemente asignados con valores aleatorios, la ecuación 4.5 se transforma en un sistema lineal y los pesos de salida pueden ser estimados de la siguiente forma.

$$\beta = H^{\dagger}T \tag{4.8}$$

Donde $H^{\dagger}$ es la inversa generalizada de Moore-Penrose [31] de la matriz H de capa de salida y puede ser calculada usando varios métodos incluyendo el método de Proyección Ortogonal, el método de Ortogonalidad, el método Iterativo, el de Descomposición de Valor Singular (SVD) [31] etc.

La capacidad de aproximación universal de ELM ha sido analizada en [32] en un método incremental, y se ha demostrado que una única SLFN con nodos RBF o aditivos generados aleatoriamente con funciones de activación continuas definidas a trozos puede universalmente aproximar a cualquier función objetivo en cualquier subespacio compacto del espacio Euclideo R^n.

.

4.2.3 Teorema 3.

Enunciado por Huang [32], dada una función continua no constante y acotada por partes $g: R \rightarrow R$ para nodos aditivos o cualquier función continua integrable por partes $g: R \rightarrow R$ y $\int_R g(x)dx \neq 0$ para nodos RBF, para una función continua objetivo f y cualquier función generada aleatoriamente $\{g_n\}, lim_{n\rightarrow\infty} \|f - f_n\| = 0$ mantiene una probabilidad de uno si

$$\beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2} \tag{4.9}$$

El Algoritmo Incremental (también llamado I-ELM) para SFLN y TFLN (Two Hidden Layer FeedForward Neural Network) la cual incrementa el número de neuronas ocultas uno-a-uno hasta que el error se vuelve menor que una constante predefinida ε.

ELM Incremental Convexo (CI-ELM) [33] es otra extensión de ELM, donde los pesos de salida de los nodos existentes son recalculados basándose en el concepto de *Optimización Convexa* de Barron [34], cuando un nuevo nodo oculto es añadido aleatoriamente usando $f_n = (1 - \beta_n) f_{n-1} + \beta_n g_n$, donde $0 \leq \beta_n \leq 1$.

.

4.2.4 Teorema 4.

Dada una función continua no constante y acotada por partes $g: R \rightarrow R$, si el espacio vectorial generado $\{G(x,a_i,b_i) : (a,b) \in R^d \text{ x } R\}$es denso en L^2, entonces para cualquier función objetivo continua f y cualquier secuencia $\{ g_n(x)=G(x,a_i,b_i) \}$ generada aleatoriamente basada en cualquier distribución muestreada continua, $lim_{n\rightarrow \inf} \|f - ((1 - \beta_n) f_{n-1} + \beta_n g_n)\| = 0$ se mantiene con probabilidad uno si

$$\beta_n = \frac{\langle e_{n-1}, g_n - f_{n-1} \rangle}{\|g_n - f_{n-1}\|} \tag{4.10}$$

Donde $\{G(x,a,b)$ es la salida de los nodos ocultos.

Basado en el teorema anterior, el peso de salida para los nuevos nodos ocultos añadidos es $\beta_L = (E.[E-(F-H_L)]^T)/([E-(F-H_L)].[E-(F-H_L)]^T) = (\sum_{p=1}^{N} e(p)[e(p)-(f(p)-h(p))])/(\sum_{p=1}^{N}[e(p)-(f(p)-h(p))])$,los pesos de salida de los nodos ocultos existentes son recalculados como $\beta_i = (1-\beta_L)\beta_i$, y el error residual después de añadir el nuevo nodo oculto *L* es $E = (1-\beta_L)E+\beta_L(F-H_L)$ donde la estimación basada en el entrenamiento $H = [h(1),\cdots,h(N)]^T$ es el vector activación del nuevo nodo para todos las muestras de entrenamiento *N*, $E = [e(1),\cdots,e(N)]^T$ es el vector residual antes de que el nuevo nodo oculto sea añadido y $F = [t_1,\cdots,t_N]^T$ es el vector objetivo.

.

4.2.5 Teorema 5.

Este teorema establece al ELM como un aproximador universal para cualquier tipo de nodo oculto computacional. Dada cualquier función continua no constante y acotada por partes $g: R \rightarrow R$, si el espacio vectorial generado $\{G(x, a_i, b_i) : (a,b) \in R^d \text{ x } R\}$es denso en L^2, entonces para cualquier función objetivo continua f y cualquier secuencia $\{g_n(x)=G(x,a_i,b_i)\}$ generada aleatoriamente basada en cualquier distribución muestreada continua, $lim_{n\rightarrow\infty}\|f-f_n\| = 0$ se mantiene con probabilidad uno si los parámetros de salida son determinados por LMS para minimizar $\|f(x) - \sum_{i=1}^{n}\beta_i g_i(x)\|$.

4.2.6 Teorema 6.

Dada una SLFN con cualesquiera nodos ocultos no costantes y definidos por partes $G(x,a,b)$ si el espacio vectorial generado $\{G(x,a_i,b_i) : (a,b) \in R^d \text{ x } R\}$es denso en L^2, entonces para cualquier función objetivo continua f y cualquier función secuencia generada aleatoriamente $\{g_n\}$y cualquier entero positivo *k*, $lim_{n\rightarrow\infty}\|f-f_n^*\| = 0$ mantendrá la probabilidad a uno si

$$\beta_n = \frac{\left\langle e_{n-1}^*, g_n^* \right\rangle}{\|g_n^*\|^2}, \tag{4.11}$$

donde $f_n^* = \sum_{i=1}^{n}\beta_i^* g_i^*$, $e_n^* = f - f_n^*$ y $g_n^* = \{g_i | min_{(n-1)k+1\leq i\leq nk}\|(f-f_{n-1}^*)-\beta_n g_i\|\}$.

Los algoritmos basados en descenso del gradiente no pueden directamente entrenar RNAs con funciones umbral si estas son no diferenciables. Por lo tanto la mayor parte de la literatura utiliza la función sigmoidal como una aproximación a las funciones de umbral. El siguiente lema 1, y los teoremas 7,8 por Huang [32] proponen el uso de

funciones de umbral para ELM.

Lema 1: Una SLFN con N neuronas ocultas con la función de activación $g(x) = 1/(1 + e^{-\lambda x})$ y con pesos de entrada elegidos aleatoriamente y sesgos ocultos que pueden aprender N distintas observaciones con cualquier pequeño error arbitrario.

.

4.2.7 Teorema 7.

Para una SLFN con *N* neuronas ocultas con una función de activación $g(x) = 1/(1 + e^{-\lambda x})$ para la capa oculta, dada cualquier constante $\varepsilon > 0$, siempre existe un entero $L \leq N$ tal que una SLFN con *L* neuronas ocultas y con pesos de entrada elegidos aleatoriamente y sesgos ocultos pueden aprender N observaciones con un entrenador del error menor que ε.

.

4.2.8 Teorema 8.

Suponiendo que una función de activación umbral $h(x) = 1_{x>0} + 0_{x<0}$ es usada en la capa oculta. Dada una constante distinta de cero $\varepsilon > 0$, siempre existe un entero $L \leq N$ tal que una SLFN con *L* neuronas ocultas y con pesos de entrada elegidos aleatoriamente y sesgos ocultos pueden aprender N observaciones con un entrenador del error menor que ε.

.

4.2.9 Teorema 9.

También llamado *Teorema de Convergencia* [37], estable que para un conjunto dado de muestras de entrenamiento distintas $\{(x_i, t_i)\}_{i=1}^{N}$, dado un valor arbitrario y positivo ε, entonces existe un entero positivo k tal que $E(H_k) = min \|H_k\beta_k - T\| < \epsilon$

4.3 Algoritmo ELM.

La esencia de ELM es por tanto:

1. La capa oculta de ELM necesitan no necesita ser sintonizada iterativamente.
2. De acuerdo con la teoría de FFNN, tanto el error de aprendizaje $\|H\beta - T\|$, y la norma de los pesos $\|\beta\|$ necesitan ser minimizados.
3. La asignación de características de la capa oculta necesita satisfacer la condición de aproximación universal.

De acuerdo a los teoremas 2 y 5, los nodos ocultos pueden ser generados aleatoriamente, los únicos parámetros desconocidos en SLFN son los vectores de los pesos de salida β_i

entre la capa oculta y la capa de salida, que pueden ser resueltos por un simple LMS directamente.

4.3.1 ELM Básico.

Dado un conjunto de entrenamiento $\aleph = \{(x_i,t_i)|x_i \in R^d,\ t_i \in R^m,\ i=1,\ldots,N\}$, una función $G(a_i,b_i,x)$, y un número de nodos ocultos L.

Paso 1, Generar aleatoriamente los parámetros de los nodos ocultos $(a_i,b_i),i=1,\ldots,L$.
Paso 2, Calcular la matriz de salida H de la capa oculta.
Paso 3, Calcular el vector de pesos de salida β:

$$\beta = H^{\dagger}T \tag{4.12}$$

.

4.3.2 Asignación aleatoria de características de la capa oculta basada en ELM.

El método de proyección ortogonal puede ser usado solamente cuando H^TH es no singular y $H^{\dagger} = (H^TH)^{-1}H^T$. Debido al uso de búsqueda e iteraciones, el método de ortogonalización y el método iterativo presentan limitaciones. Las mayorías de las implementaciones del ELM usan SVD para calcular la inversa generalizada de Moore-Penrose de la matriz H, ya que puede usarse en todas las situaciones. ELM es por lo tanto un método de aprendizaje por conjuntos.

De acuerdo a la *teoría de la regresión en cresta* [38], fue sugerido [39] [40], que un valor $1/\lambda$ positivo fuera añadido a la diagonal de H^TH o HH^T de cálculo de los pesos de salida β. La solución resultante es estable y tiende a tener un mejor rendimiento en generalización.

$$\boldsymbol{\beta} = \mathbf{H}^T\left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T} \tag{4.13}$$

y la correspondiente función de salida del ELM es:

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} = \mathbf{h}(\mathbf{x})\mathbf{H}^T\left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T} \tag{4.14}$$

O tenemos

$$\boldsymbol{\beta} = \left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} \tag{4.15}$$

y la correspondiente función de salida del ELM es

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} = \mathbf{h}(\mathbf{x})\left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} \tag{4.16}$$

Por lo que podemos minimizar la expresión $\|H\beta-T\|^2 + \lambda\|\beta\|^2$, que es el principal objetivo del ELM, como hemos comentado antes.

Por lo que el algoritmo ELM puede ser reescrito como:

Dado un conjunto de entrenamiento $\aleph = \{(x_i,t_i)|x_i \in R^d,\ t_i \in R^m,\ i=1,\ldots,N\}$, una función $G(a_i,b_i,x)$, y un número de nodos ocultos L.

Paso 1, Generar aleatoriamente los parámetros de los nodos ocultos $(a_i,b_i),i=1,\ldots,L$.

Paso 2, Calcular la matriz de salida H de la capa oculta.

Paso 3, Calcular el vector de pesos de salida β:

$$\boldsymbol{\beta} = \mathbf{H}^T\left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T} \tag{4.13}$$

O

$$\boldsymbol{\beta} = \left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} \tag{4.15}$$

En estas implementaciones, la condición del número de nodos ocultos puede ser leve, ya que no dependen estrechamente el número de muestras de entrenamiento N. Funciona tanto para el casos $L < N$ y el $L \geq N$. Difiere del teorema de la interpolación que requiere $L \leq N$ (Teorema 2), pero consistente con el teorema de aproximación universal (Teorema 5). La Figura 4.2 muestra una clasificación por contornos obtenida por ELM para un caso de clase binaria. La fórmula 4.13 es usada si el número de nodos ocultos L es mayor que el número de muestras. Toh y Deng [40] [41] estudiaron tal mejora regularización de SLFN del tipo de aditivo sigmoide. Deng y Man [41] [42] se centraron en obtener la solución analítica 4.16 basada en métodos de optimización. Toh propuso una tasa total de error correspondiente basada en la solución multi-clase de ELM. (TER-ELM). Miche [43] estudió el ELM con una cascada de 2 faltas reguladas.

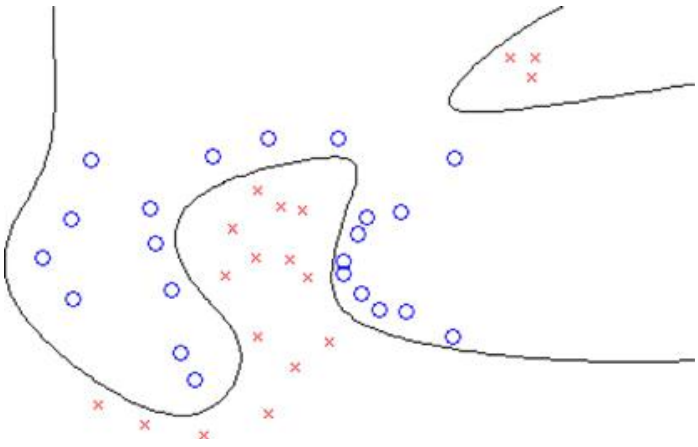


Figura 4.2 Clasificación por contornos obtenida por ELM para un caso de clase binaria: $L = 10^3$ y $\lambda=10^5$

.

4.3.3 Núcleo basado en ELM.

Huang [39] demostró que el algoritmo simple unificado de ELM puede ser obtenido para casos de clasificación de regresión, binarios y multi-etiqueta, sin embargo, tienen que ser manejados separadamente por SVMs y sus variantes. Si el mapeado de características de la capa oculta $h(x)$ es desconocida por el usuario, se puede definir una matriz de núcleos para ELM como:

$$\mathbf{\Omega}_{ELM} = \mathbf{H}\mathbf{H}^T : \Omega_{ELM\,i,j} = \mathbf{h}(\mathbf{x}_i) \cdot \mathbf{h}(\mathbf{x}_j) = K(\mathbf{x}_i,\mathbf{x}_j) \tag{4.17}$$

Entonces la función de salida del ELM (4.16) se puede compactar como:

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\mathbf{H}^T\left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T} = \begin{bmatrix} K(\mathbf{x},\mathbf{x}_1) \\ \vdots \\ K(\mathbf{x},\mathbf{x}_N) \end{bmatrix}^T \left(\frac{\mathbf{I}}{\lambda} + \mathbf{\Omega}_{ELM}\right)^{-1}\mathbf{T} \tag{4.18}$$

En esta implementación específica del núcleo de ELM, el mapeado de características de la capa oculta $h(x)$ no necesita ser conocida por el usuario, en vez de su correspondiente kernel $k(u,v)$ (p.e. $k(u,v)= exp(-\gamma\|u - v\|^2)$) que si es proporcionado a los usuarios. Tampoco ha de ser dada el número de nodos ocultos L. Por lo tanto, el algoritmo ELM puede ser reescrito para el caso del núcleo del siguiente modo:

Dado un conjunto de entrenamiento $\aleph = \{(x_i,t_i)|x_i \in R^d,\ t_i \in R^m,\ i=1,\ldots,N\}$, con núcleo $k(u,v)$: Calcula la función de salida:

$$f(\mathbf{x}) = \begin{bmatrix} K(\mathbf{x},\mathbf{x}_1) \\ \vdots \\ K(\mathbf{x},\mathbf{x}_N) \end{bmatrix}^T \left(\frac{\mathbf{I}}{\lambda} + \mathbf{\Omega}_{ELM}\right)^{-1}\mathbf{T} \tag{4.19}$$

Se puede ver que el núcleo basado en el algoritmo ELM puede ser implementado en sencillo paso de aprendizaje. Frenay y Verleysen [44] [45] estudiaron la implementación del núcleo de ELM si $h(x)$ es conocido por el usuario, definiendo el núcleo de ELM como:

$$K(\mathbf{u},\mathbf{v}) = \lim_{L\to+\infty}\frac{1}{L}\mathbf{h}(\mathbf{u}) \cdot \mathbf{h}(\mathbf{v}) \tag{4.20}$$

Un parámetro de núcleo insensible se puede obtener de forma analítica para SVMs para regresión, lo que reduce significativamente la complejidad computacional. Se conjetura que el núcleo Fre'nay y Verleysen de ELM [45] puede funcionar para SVM y sus variantes, así como en (4.16).

4.4 Determinación automática de arquitectura neuronal en ELM

.

El algoritmo ELM proporciona un procedimiento analítico de entrenamiento a una SLFN con arquitectura fija (p.e., de tamaño predefinido de capa oculta, *M*) y asignación aleatoria de parámetros ocultos. Sin embargo, el tamaño óptimo de la red es generalmente desconocido y por tanto, se hace necesario una tediosa experimentación para encontrar una buen tamaño. Esto significa que una búsqueda exhaustiva de *M* se hace por medio de un *procedimiento de validación cruzada* [46], que puede implicar un excesivo coste computacional en grandes conjuntos de datos. La investigación en la determinación automática de la arquitectura neuronal se pueden agrupar en dos categorías principales: los algoritmos constructivos (o crecimiento) [47-49] y los algoritmos poda [50-52]. En general, un método constructivo comienza con un entrenamiento con una arquitectura de tamaño relativamente pequeño, añadiendo neuronas ocultas durante el entrenamiento cuando la SLFN alcanza la convergencia, finalmente, obteniendo una red de tamaño óptimo para un criterio dado [48]. La principal ventaja de este enfoque es que no requiere un conocimiento previo acerca de la complejidad de la red para la tarea objetivo.

4.4.1 Incremental ELM.

Incremental ELM (I-ELM) es un método iterativo donde los nodos ocultos son incrementados uno a uno. Cuando un *k*-enesimo nodo oculto es añadido, el peso β_k asociado a este nuevo nodo y al nodo de salida debería estimar el peso basado en el conjunto de entrenamiento [47].

$$\beta_k = \frac{\mathbf{E} \cdot \tilde{\mathbf{H}}^T}{\tilde{\mathbf{H}} \cdot \tilde{\mathbf{H}}^T} = \frac{\sum_{p=1}^{N} e_p h_p}{\sum_{p=1}^{N} h_p^2} \tag{4.21}$$

Donde h_p es la activación del nuevo nodo para la entrada de la *p*-enésima muestra de entrenamiento y e_p es el correspondiente error residual antes de que este nuevo nodo fuera añadido. $\hat{H} = [h_1, \ldots, h_N]^T$ denota el vector de activación del nuevo nodo para todas las *N* muestras de entrenamiento, y $E = [e_1, \ldots, e_M]^T$ es el vector de error residual antes de que el nuevo nodo sea añadido. En aplicaciones reales, conseguir un cero de error aproximado no fuese lo deseado ya que implicaría añadir infinitos nodos a la red, resultando en un sobreajuste del conjunto de entrenamiento. Debido a esto, se hace necesario fijar un número máximo de nodos ocultos M_{max} y un criterio de error de convergencia ε. Con lo que podemos resumir el algoritmo I-ELM de la siguiente forma:

Dado un conjunto de entrenamiento $\aleph = \{(x_i,t_i)|x_i \in R^d,\ t_i \in R^m,\ i=1,\ldots,N\}$, una función de activación $f(.)$, un número de nodos ocultos M_{max}, y una precisión esperada de aprendizaje ε.

Paso 1, Inicialización, estableciendo M=0 y el error residual $E=t$ donde $t=[t_1,\ldots,t_N]^T$

Paso 2, Paso de aprendizaje:

Paso 3, Mientras $M < M_{max}$ y $E > \varepsilon$ se:

Paso 4, Incrementa en uno el número de nodos ocultos *M*, *M=M+1;*

Paso 5, Asignamos aleatoriamente los pesos de entrada w_m y los sesgos b_m

Paso 6, Calculamos el peso de salida β_M para el nuevo nodo

Paso 7, Calculamos el error residual después de añadir el nuevo nodo oculto

$$\mathbf{E}_M = \mathbf{E}_{M-1} - \beta_M \cdot \tilde{H}_M \tag{4.22}$$

Este método presenta como desventaja el ser demasiado sensible a la inicialización aleatoria de los pesos para los nuevos nodos, lo que podría producir una solución de un mínimo local para unos inadecuados pesos de inicialización. Antes del aprendizaje no existen nodos ocultos en la SFLN y el error residual es inicialmente establecido con el mismo valor que el valor objetivo esperado del conjunto de entrenamiento. El aprendizaje finalizará cuando *M* exceda el valor predeterminado M_{max} o el error residual sea menor que ε.

Después Huang [35] encontró con el EI-ELM (Enhanced Random search based incremental Learning machine) que algunos nodos ocultos en las redes juegan un papel menor en la salida de la red lo que aumenta la complejidad del sistema.

Así, en EI-ELM, en cada paso de aprendizaje varios nodos ocultos se generan aleatoriamente y entre ellos el nodo oculto que conduce al mayor error residual decreciente se añadirá a la red existente. El peso de salida se calcula como en I-ELM.

4.4.2 Técnicas de poda para ELM.

Esta técnica empieza entrenando una SLFN totalmente conectada que es mayor de lo necesario, y va removiendo el exceso de componentes de ésta alcanzando una red de tamaño óptimo. Por lo tanto, en un diseño de poda, los nodos o pesos se van eliminando de acuerdo a su importancia. En general, una SLFN más grande posee la ventaja de una mayor tasa de convergencia aunque necesite de un mayor tiempo computacional.

Numerosos algoritmos han sido usados para podar redes neuronales, por ejemplo algunos métodos de poda borran un solo peso de una vez. Otros métodos consideran interacciones entre pesos y más de un peso cada vez, lo que permite reducir el tamaño de la red significativamente, sin perder la precisión obtenida.

Para obtener modelos de EML más compactos, varios métodos de poda automática han sido propuestos [53-58]. Generalmente, en un método de poda los pesos de entrada $\{\mathbf{w}_i, b_i\}_{i=1}^{M}$ de los nodos ocultos son asignados aleatoriamente. Basándose en los parámetros asignados, las respuestas de los nodos ocultos (representados como la matriz H) son obtenidas. Junto con los datos objetivo, $\{\mathbf{t}_j\}_{j=1}^{N}$ y *H*, se calcula la relevancia de cada nodo oculto contribuyendo al verdadero valor objetivo. Según esta relevancia los nodos son podados o se mantienen. La noción de relevancia es definida por alguna medida estadística que es diferente para cada algoritmo de poda. Principalmente dos diferentes enfoques de métodos de poda han sido propuestos: P-ELM (Pruned-ELM) [53] y OP.ELM (Optimally Pruneed ELM) [54] [58].

4.4.2.1 P-ELM

P-ELM calcula la relevancia estadística de los nodos ocultos usando dos criterios: Chi cuadrado (χ^2) y medidas de Ganancia de Información [59]. En este caso, y dado un T que es una tarea de decisión (p.e. datos discretos), las respuestas de los nodos ocultos son discretizadas basándose en el método de discretización de Fayyad´s [60] antes de que los dos criterios estadísticos sean usados.

El criterio de Chi-cuadrado (χ^2) medio las diferencias entre las distribuciones de variables categóricas (p.e. la salida variable objetivo y discretizada para cada neurona), mientras que el criterio IG mide la reducción en la incertidumbre sobre el valor de la clase objetivo, dada la respuesta discretizada de cada nodo oculto.

Dado un conjunto de entrenamiento $\aleph = \{(x_i,t_i)|x_i \in R^d, t_i \in R^m, i=1,\ldots,N\}$, una función de activación *f(*sigmoide o gausiana*)*, un número inicial de nodos ocultos *M*, mayor de lo necesario, y un conjunto de umbrales *q* $\{\gamma_s\}_{s=1}^{q}$.

Paso 1, Se divide $\aleph$, en dos subconjuntos no superpuestos para el entrenamiento (muestras N_T) y la validación (muestras N_V). con $N=N_T+N_V$

Paso 2, Se realiza una asignación aleatoria de $\{\mathbf{w}_i, b_i\}_{i=1}^{M}$ pesos de entrada

Paso 3, Se calcula la matriz *H* de capas ocultas de salida usando un subconjunto de aprendizaje.

Paso 4, Se calcula la relevancia estadística r_i para las neuronas ocultas usando χ^2 o Ganancia de Información.

Paso 5, Se ordenan las neuronas ocultas en orden descendente según $r_{i,}$ que es la medida de relevancia para cada neurona.

Paso 6, A partir de *s=1* hasta que tome el valor de *q*

Paso 7, Encontramos el subconjunto de entrenamiento de unidades ocultas, con cardinalidad $S_{s,}$ que satisface el umbral γ_S , $(r_k < \gamma_S, k=1,2,\ldots, S_s)$.

Paso 8, Obtenemos el error de clasificación en el conjunto de validación: $E_{V.}$

Paso 9, Calculamos el AIC (Akaike's information criterion)= $2N_V \log\left(\frac{E_V^2}{N_V}\right) + S_s$
Paso 10, Al llegar *s=q, se selecciona la red con* M^* *nodos de acuerdo al mínimo (AIC)*
Paso 11, Se calcula la matriz de salida de capas ocultas H^* usando $\aleph$.
Paso 12, Se calcula matriz de pesos de salida $B^* = (H^*)^{\dagger}T$

Como inconvenientes, presenta que el valor umbral elegido es demasiado sensible a la correcta selección del subconjunto de validación. Esta afinación sensible se basa en la definición de los subconjuntos de validación y aumenta significativamente el coste computacional debido al hecho de que varias repeticiones deben hacerse. Se debe también señalar que el subconjunto de validación conserva las muestras de datos de entrenamiento, y por lo que hay menos muestras disponibles durante el proceso de aprendizaje, que puede ser un inconveniente en algunos problemas definidos por pequeños conjuntos de datos. Además, como se explicó anteriormente, P-ELM sólo se puede aplicar en problemas de clasificación.

4.4.2.2 OP-ELM

A diferencia del método de poda anterior, OP-ELM [54] [58] se puede aplicar tanto a la clasificación y tareas de regresión. Por otra parte, en lugar de utilizar un subconjunto de validación, este procedimiento utiliza un eficiente criterio de Leave-One-Out (LOO) para la selección de unidades ocultas. Además, OP-ELM utiliza el algoritmo MRSR (Multiresponse Regresión Sparse) para la medición de la relevancia de las unidades ocultas [61], éste es en esencia una extensión de LARS (Least Angle Regression) utilizado para clasificar en vez de para decidir. La principal ventaja de MRSR es que se realiza una clasificación exacta de las neuronas en la capa oculta. Como se muestra en [58], OP-ELM ofrece unos modelos extremadamente rápidos y precisos, y logra , más o menos, el mismo nivel de precisión que el de otros métodos de aprendizaje automático conocidos, tales como MLPs , SVMs o Procesos Gausianos (GP). A continuación, describiremos las tres etapas principales del método OP-ELM, que se resumen en la figura 4.3.

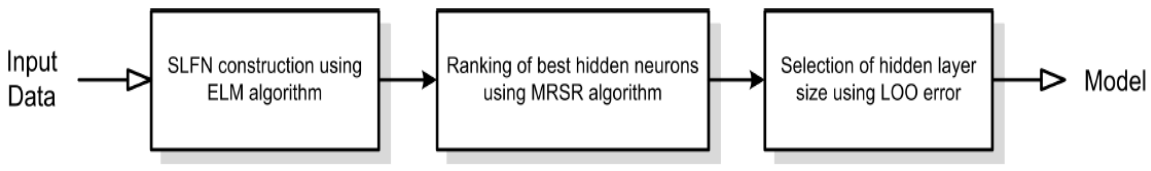


Figura 4.3 Las 3 fases del Algoritmo OP-ELM

El primer paso es inicializar el número de neuronas *M, M>>n*. A diferencia del ELM simple, OP-ELM aproxima tres tipos de funciones (sigmoides, gausianas y lineales) poseyendo mayor robustez y capacidad de generalización. Para las funciones sigmoidales, los pesos son asignados aleatoriamente siguiendo una distribución uniforme que cubre todo el rango de los datos de entrada (previamente normalizada a cero media y unidad de

varianza). Considerando que, los núcleos gaussianos tienen sus centros tomados al azar a partir del conjunto de datos y de diámetros comprendidos entre el 20% y el 80% percentil de la distancia de la distribución de los datos de entrada.

El segundo paso, consistiría en aplicar el algoritmo MRSR para aplicar la clasificación de las neuronas ocultas de acuerdo a su precisión. Descrito como:

Para los *N x M* capas ocultas de la matriz de salida H, cada columna h_i (vector de *N* salidas para la *i*-enésima neurona oculta) es añadida una a una al modelo en sucesivos pasos del algoritmo. De este modo, para un paso k (k,1,…,M) el modelo se define

$$\tilde{\mathbf{O}}^{k}_{N\times m} = \mathbf{H}_{N\times M}\mathbf{B}^{k}_{M\times m} \tag{4.23}$$

Donde $\acute{O}^k$ y B^k son, respectivamente, el modelo de matriz de salida y la matriz de pesos de salida para el *k*-enésimo paso. B^k tiene k filas no nulas en el *k*-enésimo paso del algoritmo MRSR. Con cada nuevo paso una nueva fila no nula es añadida en B^k y una nueva columna (h_k) es introducida al modelo. Es importante tener en cuenta que con MRSR se obtiene una clasificación exacta para problemas lineales. Ya que en un esquema de ELM la salida es lineal con respecto a las unidades ocultas inicializadas al azar, la clasificación obtenida de las neuronas por MRSR será exacta.

Una vez que se haya obtenido la clasificación de las neuronas y que *H* se ordenó en consecuencia, las mejores unidades ocultas para el modelo ELM (*M** Nodos ocultos) tienen que ser elegidas. Métodos basados en *validación cruzada* OFFLINE se pueden utilizar para este tipo de tarea. Uno de los enfoques es la validación aleatoria de los subconjuntos (como se hace en P-ELM): se divide aleatoriamente el conjunto de datos en datos de entrenamiento y de validación, por cada uno de esas divisiones, el modelo se ajusta a los datos de entrenamiento, y la exactitud de la predicción se evalúa a través de los datos de validación. En este caso, los resultados varían si el análisis se repitió con diferentes divisiones aleatorias. Una solución mejor es utilizar LOO: que utiliza una sola observación del conjunto de entrenamiento original como validación de datos, y las observaciones restantes como el conjunto de entrenamiento. Esto se repite de tal manera que cada observación en el conjunto de muestras se usa una vez para la validación, y debido a esto, todas las muestras se utilizan para entrenamiento y validación. Considerando que, el principal inconveniente con el método LOO es que puede ser muy costoso, en términos de cómputo, cuando el conjunto de datos posee un gran número de muestras. Sin embargo, el error LOO puede calcularse de una manera exacta para modelos lineales mediante el uso de PRESS (Prediction Sum of Squares) [63].

Considerando que $\hat{\mathbf{B}}_{(j)}$es la solución de mínimos cuadrados del error cuando la j-enésima muestra (j-enésima fila de H) es omitida.

$$\hat{\mathbf{B}}_{(j)} = \mathbf{H}^{\dagger}_{(j)}\mathbf{T}_{(j)}, \tag{4.24}$$

donde $H_{(j)}$ y $T_{(j)}$, son respectivamente, H y T sin su j-enésima fila. Entonces, el error LOO es dado por

$$\varepsilon_{PRESS} = \frac{1}{N}\sum_{j=1}^{N} \|\mathbf{h}_j \hat{\mathbf{B}}_{(j)} - \mathbf{T}_{(j)}\|^2 \tag{4.25}$$

siendo h_j es la j-enésima fila de H. Al seleccionar usando el criterio de error LOO, el ε_{PRESS} para cada modelo es calculado, y se escoge el que tenga un menor valor de éste. Si usáramos la expresión 4.25, sería demasiado ineficiente debido al alto coste computacional. Por lo que usaremos la siguiente expresión:

$$\varepsilon_{PRESS} = \frac{1}{N}\|\Gamma(\mathbf{I}-\mathbf{P})^{-1}(\mathbf{P}-\mathbf{I})\|_{\mathcal{F}}^2 \tag{4.26}$$

con $P=HH^{\dagger}$, I la matriz identidad y donde $\Gamma(A)$ mantiene sólo los elementos de la diagonal de la matriz cuadrada A. $\|A\|_F$ indica la matriz de Frobenius, la normal de A.

Para redes del tipo OP-ELM, el estadístico PRESS es iterativamente calculado añadiendo un nodo oculto (previamente clasificado en *H*) al modelo. El modelo con M^* neuronas ocultas obtiene el menor estadístico PRESS que es considerado el óptimo.

$$\varepsilon_{PRESS}^{M^*} < \varepsilon_{PRESS}^{k}, \forall k \in (1,2,\ldots,M). \tag{4.27}$$

Nótese que este procedimiento presenta una ventaja clara respecto P-ELM, ya que no requiere seleccionar un umbral de relevancia fijo. Esta clasificación tiene dos efectos positivos: convergencia más rápida y modelos más pequeños ($M^*<M$). Por último, ofrecemos un resumen del algoritmo OP-ELM.

Dado un conjunto de entrenamiento $\aleph = \{(x_i,t_i)|x_i \in R^d,\ t_i \in R^m,\ i=1,\ldots,N\}$, una mezcla de funciones de activación *f(*sigmoide, gausiana y lineal*)*, y un gran número inicial de nodos ocultos *M*,

Paso 1, Se asignan aleatoriamente los pesos de entrada $\{\mathbf{w}_i, b_i\}_{i=1}^{M}$

Paso 2, Calcular la matriz de salida H de las capas ocultas H usando X y los pesos de entrada.

Paso 3, Clasificar las salidas ocultas usando el algoritmo MRSR, p.e. H es clasificado, y establecida H^0 como una matriz vacia.

Paso 4, Desde un valor de $k=1$ hasta que valga N,

Paso 5, Se añade el *k*-enésimo nodo al modelo $\rightarrow H^k = [H^{k-1}, h_k]$,siendo h_k la k-enésima columna de H.

Paso 6, Se calcula el error LOO (ε_k^{PRESS}) usando la ecuación 4.26 con H^k.

Paso 7, Cuando $k=N$, se selecciona el tamaño de la red M^* de acuerdo a 4.27.

Paso 8, Calcular los pesos de la matriz de salida: $B^* = (H^*)^{\dagger} T$

Con el fin de evitar la inversión de la matriz y varios productos matriciales en el cálculo, otras obras utilizan diferentes criterios de selección de variables, tales como la Criterio de Información de Hannan-Quinn [57] o el Test Delta [56].

Capítulo 5

Estudio y diseño de las técnicas de combinación de RNAs basadas en ELM

Hasta ahora, hemos analizado varios métodos de ELMs para diseñar y entrenar varias SLFN. Como se explicó anteriormente, cada modelo ELM se construye a partir de una generación aleatoria de los pesos w y, después de esto, los β son calculados directamente mediante el uso de la matriz inversa generalizada de Moore-Penrose. Así, cada diferente inicialización aleatoria de los pesos w genera un modelo diferente. En la práctica, teniendo en cuenta un problema objetivo, el usuario generalmente tiene que realizar varias pruebas (es decir, generar un conjunto de redes diferentes) y seleccionar el "mejor" modelo para los datos objetivo (es decir, descartar las redes restantes).

Es ampliamente conocido que el rendimiento de una SLFN puede mejorarse mediante la combinación de varias soluciones para la misma tarea [63-65]. En lugar de generar una población de L modelos y usar únicamente el "mejor" modelo, una combinación de estas L redes aprovecharía, en lugar de ignorar, la información contenida en los modelos redundantes. Tal combinación de modelos se denomina *comités* o *conjuntos*. Hay dos aspectos principales que surgen cuando se considera el enfoque de *comité*: primero, la creación de *L* modelos que se combinan en un conjunto, y segundo, el método por el cual las salidas de estos miembros se combinan [63].

En el primer lugar, hay que tener en cuenta que cuando las *L* redes entrenadas poseen la misma capacidad de generalización, no encontraríamos ninguna mejora de rendimiento entre las salidas de los modelos individuales y el obtenido por el conjunto de ellas. Con el fin de que los modelos generalicen de forma diferente, deben tener diferentes condiciones iniciales antes de su generación (por ejemplo, mediante la variación de los pesos iniciales o de su topología) o distintos conjuntos de datos de entrada. Con respecto a la segunda cuestión, la forma más sencilla de construir un comité es la media o promedio ponderado de las predicciones de un conjunto de modelos individuales, pero otros esquemas de combinación son también factibles. Por ejemplo, se podrían ordenar las L redes entrenadas según los rendimiento obtenidos y, a continuación, el promedio de los mejores $L/2$ modelos [67].

La solución más sencilla es la media de todos los ELMs

$$\hat{o}_j = \frac{1}{L}\sum_{l=1}^{L} o_{lj} \tag{5.1}$$

donde $\hat{o}_j$ indica la salida del conjunto para el j-enénismo vector x_j, y o_{lj} es la salida del l-enésimo modelo individual de ELM para x_j.

La media puede no ser la mejor elección en la práctica, ya que trata a cada miembro por igual, sin importar cuál de los miembros realiza una aportación mayor a la generalización final. Otras soluciones como la mostrada en la figura 5.1 pueden presentar una mejor aproximación para nuestro problema

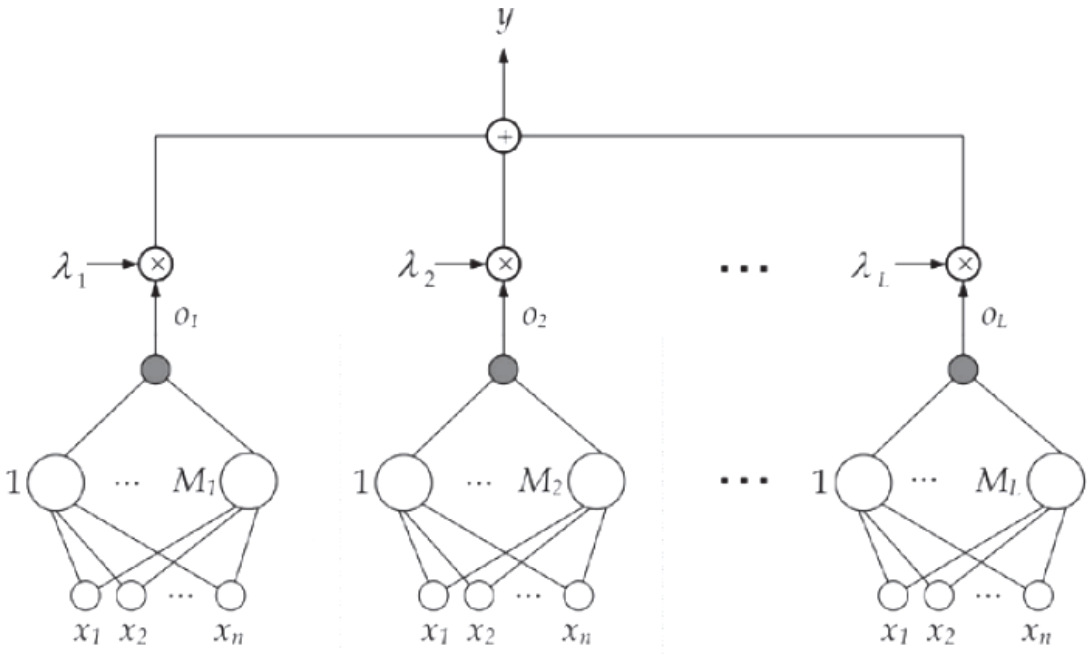


Figura 5.1 Combinación lineal de múltiples ELMs.

5.1 Método de conjuntos basado en índice de productos excluyentes (PIEx).

La primera propuesta para conjunto ELM esquemas ha sido el Método de Conjuntos basados en Índice de Productos Excluyentes (PIEx) [63].

$$G = \frac{1}{LN}\sum_{l=1}^{L}\left(\sum_{j=1}^{N} e_{lj}^2 - \sum_{j=1}^{N} a_{lj}^2\right) \tag{5.2}$$

donde e_{lj} es el error medida entre el l-enésimo modelo de salida y el valor objetivo ($e_{lj=\ tj}$ - $o_{lj)}$ y alj= $\hat{o}_j$ - o_{lj} es la diversidad entre el l-enésimo modelo individual y el conjunto de salida. Ya que el primer término del sumatorio dentro de los paréntesis de 5.2 es constante para cada red individual después de su entrenamiento, el rendimiento del conjunto de redes está determinado por el segundo término. Por lo tanto, a mayor

diversidad, mayor capacidad de generalización obtendrá el conjunto [60]. Chen propuso un procedimiento de selección sencillo para comités, una SLFN con un gran error de entrenamiento y poca diversidad, debería ser descartada, mientras que una red con error de entrenamiento pequeño y gran diversidad puede ayudar a aumentar el rendimiento del conjunto [63]. En concreto, el método PIEx mide la diversidad ρ_l entre los modelos individuales y el sistema del conjunto usando el coeficiente de correlación de Pearson. Con ρ_l y el error cuadrático medio de cada modelo ELM (MSE_l), definimos el índice de producto como:

$$P_l = MSE_l \rho_l \tag{5.3}$$

Y, por lo tanto, a mayor MSEl y mayor ρl conseguimos un mayor Pl. Este método descarta un modelo ELM del conjunto si su Pl es mayor que el valor medio de todos los P_l, y la salida final se calcula con el promedio de las redes elegidas. De acuerdo a los resultados obtenidos en varios problemas de regresión, un comité de redes ELM basado en el método PIEx supera al promedio simple en todas las simulaciones [63]. Sin embargo, este procedimiento utiliza el mismo tamaño de capa oculta para todos los miembros de la comisión que pueden afectar la capacidad de aprendizaje y de generalización del sistema conjunto.

Dado un conjunto de entrenamiento $\aleph = \{(x_i,t_i)|x_i \in R^d,\ t_i \in R^m,\ i=1,\ldots,N\}$, una función de activación *f(.)*, un número de nodos ocultos *M* y un número de redes *L*.

Paso 1, Se entrenan las *L* redes ELM con *M* neuronas usando el algoritmo estándar ELM, con diferentes pesos de inicialización.

Paso 2, Se calcula la salida del conjunto $\hat{o}$ por simple media.

Paso 3, Desde un valor de *l*=1 hasta que este valga *L* haremos:

Paso 4, Medimos la diversidad ρ_l entre el *l*-enésimo modelo de ELM y el conjunto $\hat{o}$.

Paso 5, Medimos el error cuadrático medio MSE de la *l*-enésima red MSE_l.

Paso 6, Calculamos el Índice de Producto: $P_l = MSE_l \rho_l$

Paso 7, Cuando *l*=*L* calculamos la media del valor P_l: $\bar{P}$

Paso 8, Seleccionamos el conjunto de los L^* Modelos ELM que satisfacen que $P_l < \bar{P}$

Paso 9, El conjunto $\hat{o}$ es la media de redes de salida para los L^* modelos ELM seleccionados.

5.2 Método de conjuntos adaptativos.

Van Heeswijk propuso un sistema de conjunto cuya salida es una combinación lineal ponderada de los resultados de los modelos ELM individuales [64-67]:

$$\hat{o}_j = \frac{1}{L}\sum_{l=1}^{L} v_l o_{lj} \tag{5.4}$$

donde v_l es el conjunto de pesos de la red ELM *l*-enésima, que determina su contribución a la predicción del conjunto. Para su construcción, un número de esquemas

de ELM de complejidad variada son generados, cada uno de los cuales está entrenado individualmente utilizando el algoritmo estándar de ELM. Estas redes tienen número aleatorio de neuronas ocultas, número aleatorio de características entrada y diferentes inicializaciones de los pesos de $(\mathbf{v} = v_l{}_{l=1}^{L})$ entrada. Después de que esta fase de entrenamiento se lleve a cabo, los pesos del conjunto, , se adaptan con el fin de minimizar el error de predicción.

En [64], un procedimiento iterativo para actualizar v se ha introducido con el fin de resolver problemas de predicción de series temporales. En particular, cuando una red individual no tiene un buen rendimiento de predicción (con respecto a los modelos restantes) su peso en el conjunto es disminuido, y viceversa [64]. Para cada paso temporal (τ), la predicción de cada red se compara primero con el valor objetivo, $e_l(\tau) = t_j - o_l(\tau)$, si el error cuadrático $e_l(\tau)^2$ del *l*-enésimo modelo es mayor que el error cuadrático medio de todos los modelos en el paso temporal (τ), entonces el correspondiente peso v_l se reduce con una tasa fija de cambio y viceversa. Recientemente, Van Heeswijk ha extendido ésta, utilizando una combinación lineal con restricciones de positividad en los pesos del conjunto [68]. Cada red está previamente entrenada usando el algoritmo ELM considerando un número muy grande de neuronas y diferentes subconjuntos aleatorios de características de entrada. Entonces, el tamaño óptimo de capa oculta se determina usando el clásico Criterio de Información Bayesiana (BIC):

$$BIC = N \times \log\left(\frac{E}{N}\right) + M \times \log N \tag{5.5}$$

donde *E* denota el error cuadrático medio, *M* es el tamaño de la capa oculta y *N* es el número de vectores de entrenamiento. Un total de *L* esquemas de ELM (generalmente *L*>100) son obtenidos junto con sus salidas. Conjuntos de validación predefinidos son usados con los datos objetivo para resolver el sistema de combinación lineal 5.4 con las limitaciones positivas en v_l, es decir, $v_l > 0$.

Dado un conjunto de entrenamiento $\aleph = \{(x_i,t_i) | x_i \in R^d,\ t_i \in R^m,\ i=1,\ldots,N\}$, una función de activación *f(.)*, un número máximo de nodos ocultos M_{max} y un número de redes *L*.

Paso 1, Se divide el conjunto $\aleph$ en conjuntos de entrenamiento y de validación.

Paso 2, Se entrenan las *L* redes ELM con un tamaño aleatorio de capa oculta (h_l<M_{max}) y diferentes subconjuntos aleatorios de las características de entrada.

Paso 3, Para cada modelo, el tamaño de su capa oculta es determinado por minimización de 5.5 (BIC) del conjunto de validación.

Paso 4, Se resuelve la combinación lineal restringida dada por la ecuación 5.4 en el conjunto de validación.

Este procedimiento puede ser muy costoso en términos computacionales. Para resolver este inconveniente, en [68] se aplica el algoritmo de ELM usando cálculo GP (Procesamiento Gráfico), en lugar de la implementación de CPU estándar, con el fin de reducir el tiempo de entrenamiento.

Al contrario que el método PIEX, este procedimiento de comité tiene más diversidad e independencia en los modelos ELM individuales porque éstos son entrenados con diferentes tamaños de capas ocultas y subconjuntos de características de entrada. Sin embargo, esta técnica es bastante sensible a la correcta elección del conjunto de validación y sus resultados varían si el análisis es repetido con diferentes divisiones aleatorias del conjunto.

5.3 Combinación óptima de modelos ELM.

Como ya se ha mencionado en este PFC, en el método OP-ELM, es posible obtener una óptima combinación lineal usando las variables de entrada clasificadas por el algoritmo MRSR y el estadístico PRESS. Además, la técnica de OP-ELM ofrece un diseño rápido, robusto y automático de las redes ELM. Debido a esto, una mejora directa de la técnica anterior de conjunto es construir L redes aleatorias utilizando el método de OP-ELM, y a continuación, combinarlas de manera óptima utilizando la misma metodología. Esta propuesta se ha introducido recientemente en [69] y se conoce como OC-ELM (Combinación óptima de las máquinas de Extreme Learning). Por lo tanto, este método resuelve el problema de la siguiente combinación lineal:

$$\sum_{l=1}^{L} v_l o_{lj} = t_j, \quad j = 1,\ldots,N, \tag{5.6}$$

o_{lj} es la salida del t-enésimo modelo individual de ELM para x_j, que ha sido entrenado usando OP-ELM. El método OC-ELM extiende OP-ELM mediante la combinación de las estimaciones de las L^* redes (con $L^*<L$), que han sido elegidas, automáticamente y de manera óptima, como de aquellas que proporcionen una mejor predicción por conjuntos para los datos objetivo. Se realiza ordenando las diferentes ELMs (5.6) con el algoritmo MRSR [61] y, a continuación, seleccionando las L^* redes que minimizan el error LOO, que es calculado por el estadístico PRESS [64].

.

5.3.1 Construcción inicial de redes ELM.

La primera fase de este proceso empieza construyendo *L* diferentes SLFNs usando el algoritmo OP-ELM, vamos a utilizar tres diferentes alternativas [70]:

1. OC-ELM, todas las SLFNs son entrenadas usando las *n* variables de entrada que definen los "dataset". En este caso, la diversidad viene dada por la naturaleza de las diferentes redes (que poseen diferentes tamaños de capa oculta) obtenidas con el OP-ELM para cada peso de entrada de inicialización.
   .
2. La selección de características es un enfoque muy útil para descartar entradas irrelevantes, que podrían ser perjudiciales para el modelado de los datos objetivo. Por lo que la siguiente propuesta tendremos esto en cuenta, al obtener un subconjunto de variables de entrada a través de una selección de características. En ésta, las características son añadidas secuencialmente a un "conjunto candidato" vacío hasta que la adición de más características no disminuya el erro LOO. Después de *L* repeticiones del algoritmo OP-ELM se promedia el error LOO en cada subconjunto de características. Cuando se alcanza la convergencia, se obtienen las *L* diferentes SLFNs entrenadas mediante OP-ELM con los mejores subconjuntos de entradas. Esta técnica es conocida como OC-OPELM-FFS (Combinación Optima de OP-ELMs basada en Selección de características de avance).

3. Con el fin de que un *comité* supere el rendimiento de las redes individuales, es fundamental que exista suficiente diversidad entre los *L* modelos. Para aumentar la diversidad, esta tercera alternativa considera subconjuntos aleatorios de diferentes funciones durante el entrenamiento OP-ELM de cada red. Así, se obtienen *L* diferentes SLFNs compuestas de un número diferente de unidades ocultas y características de entrada. En este caso, puede ser necesario utilizar un conjunto bastante grande de redes SLFN candidatas con subconjuntos de entrada aleatoria, porque modelos inexactos puede obtenerse a partir de subconjuntos de características determinadas. Debido a esto, los "datasets" de alta dimensión requieren un mayor número de redes aleatorias. Tenga en cuenta que estos modelos inadecuados con (es decir, subconjuntos de características inapropiadas al azar) se omiten automáticamente en la fase de poda siguiente. Esta alternativa se denomina OC-OPELMs-RFS (Combinación óptima de OP-ELM basada en subconjuntos de características azar).

Podemos ver estas construcciones gráficamente en la siguiente figura:

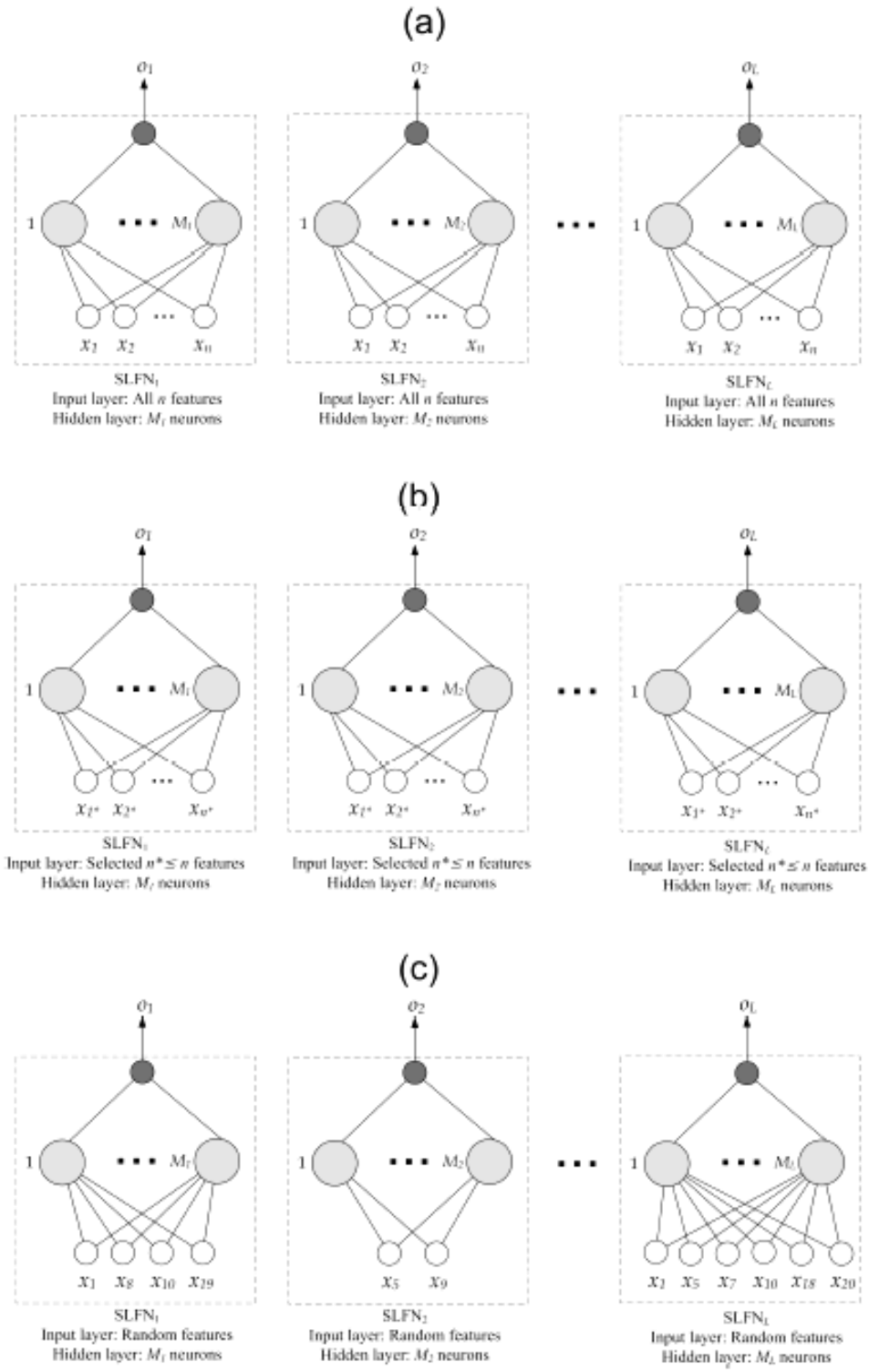


Figura 5.2 Construcción inicial de *L* SFLNs para ensamblado de conjuntos: (a) OC-OPELM; (b) OC-OPELM-FFS; y (c) OC-OPELM-RFS.

.

5.3.2 Diseño de conjuntos.

Una vez que las diferentes *L* ELMs han sido entrenadas, sus salidas de red son previamente clasificadas utilizando el método MRSR y la matriz de salida de la red por lo tanto, ordenada. Entonces, de acuerdo con este trabajo, los mejores *L* modelos se seleccionan para minimizar el error LOO, que puede medirse de una manera exacta mediante la estadística PRESS. Los modelos inadecuados e inútiles se descartan del conjunto. Este conjunto poda reduce las necesidades de almacenamiento, velocidades de la etapa de operación y tiene el potencial de mejorar el rendimiento de la arquitectura SLFN. Presentando el sistema de conjunto obtenido un rendimiento óptimo.

Capítulo 6
Implementación y validación en problemas reales de carácter multidisciplinar

Para la implementación de los algoritmos, se ha trabajado utilizando como base la OP-ELM Toolbox de MATLAB© desarrollada por Miche, Sorjamaa y Lendasse [55], modificando éste código para desarrollar las distintas técnicas de Combinación Óptima de Conjuntos presentadas anteriormente.

No se aportan en este PFC capturas del código desarrollado, ya que éste forma parte de un trabajo de investigación de un mayor alcance, donde se están desarrollando nuevos y eficientes sistemas y algoritmos de inteligencia computacional, desarrollado fundamentalmente en el Centro Universitario de la Defensa (CUD) de San Javier. Para cualquier información al respecto, por favor contacten con D. Pedro García Laencina (pedroj.garcia@cud.upct.es). Por lo que en este capítulo, nos centraremos en proporcionar una evaluación detallada y sistemática de todos estos métodos, teniendo como objetivo mostrar mediante resultados experimentales el rendimiento de éstos.

En primer lugar, se utilizará un "ejemplo de juguete" para ilustrar el rendimiento del algoritmo de ELM estándar. En este caso, el método ELM es comparado con el algoritmo estándar de BP para entrenamiento de SLFNs. En particular, los MLPs son entrenados utilizando el método del Gradiente Conjugado Escalado (SGC), que es un algoritmo computacionalmente eficiente y ampliamente utilizado para entrenar las redes neuronales artificiales. Además, también evaluaremos el entrenamiento de SLFN con ELM y BP en problemas de regresión del mundo real. Estas primeras simulaciones se realizan para un tamaño predefinido de capa oculta.

Tras mostrar las ventajas de ELM, analizaremos el diseño automático de una FFNN utilizando el método de OP-ELM, simulando tres conocidas tareas de regresión. Finalmente, los modelos ELM se combinan en un conjunto utilizando el enfoque OC-ELM, lo que da una solución directa y automática para conjuntos de ELM. Todos los experimentos han sido procesados en la misma estación que constaba con 4 GB de memoria RAM y un procesador de 2,67 GHz con MATLAB© 7.11 (R2010b).

6.1 Primera evaluación, "Ejemplo de Juguete": ELM vs BP-MLP.

En primer lugar, un conjunto de 1000 muestras es generado a partir de una suma de senos. Esto nos proporciona un conjunto de datos unidimensional llamado *Sinusoidal*, y que está dividido en dos subconjuntos con igual número de muestras (500 casos) para entrenamiento y fases de test. Con el fin de comparar los enfoques de entrenamiento BP y ELM, a ambos se les asigna el mismo número de neuronas ocultas (de 1 a 30 neuronas) con función de activación del tipo sigmoide. Para cada tamaño de capa oculta, se realizan 30 simulaciones (es decir, 30 inicializaciones de pesos diferentes) para ambos métodos. En el caso del entrenamiento MLP, se llevan a cabo 500 iteraciones de entrenamiento. La Figura 6.1 muestra la evolución del tiempo de entrenamiento (en segundos) respecto al tamaño de la capa oculta (neuronas) para ambos métodos de entrenamiento: MLP-línea azul con cuadrados y ELM-línea roja con círculos. Nótese que este gráfico utiliza una escala logarítmica en el eje vertical. Y se puede observar en esta figura que el algoritmo ELM presenta un comportamiento casi 100 veces más rápido que el entrenamiento BP para MLP, sin tener en cuenta que un entorno de programación C podría ejecutarse mucho más rápido que en MATLAB©. Es bastante obvio darse cuenta que el tiempo de entrenamiento aumenta a medida que el tamaño de la capa oculta es mayor.

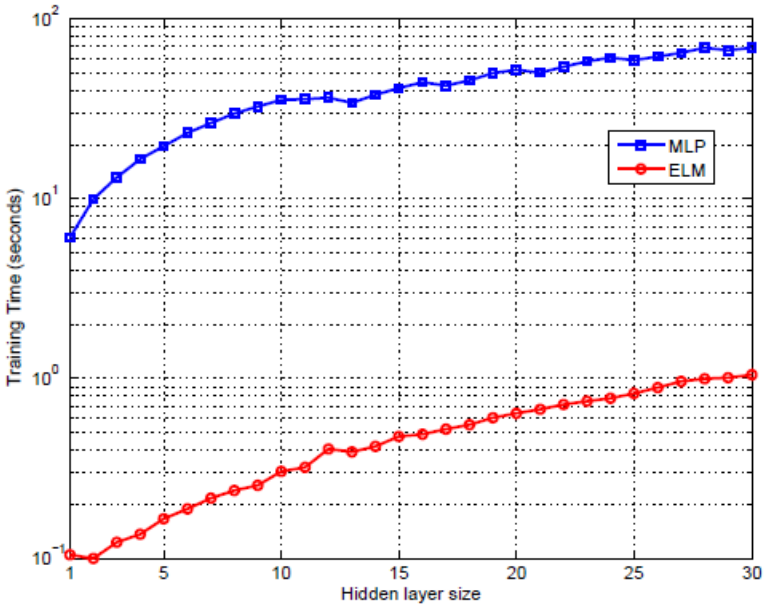


Figura 6.1 Evolución del tiempo de entrenamiento (s) respecto al tamaño de la capa oculta (neuronas).

A continuación, en la Figura 6.2 se muestra la raíz del error cuadrático medio (RMSE) en el conjunto de test respecto al diferente número de neuronas para el entrenamiento MLP basado en gradiente y para el método ELM. Los resultados son el resultado de un promedio de 30 simulaciones. Gracias a esta figura, se puede observar que el entrenamiento MLP estándar se comporta mejor que el algoritmo ELM cuando el tamaño de la capa oculta es pequeño. Esto es debido al hecho de que los pesos de entrada no se actualizan en el método ELM y, por lo tanto, este procedimiento es muy sensible a la inicialización de los pesos de entrada con un menor número de nodos ocultos. Esto significa que ELM generalmente necesita de un mayor número de neuronas en la capa oculta, debido a la selección aleatoria de los parámetros de entrada de red, esto puede implicar una limitación en las implementaciones de hardware para aplicaciones reales, pero que puede ser resuelta utilizando un diseño de poda eficiente.

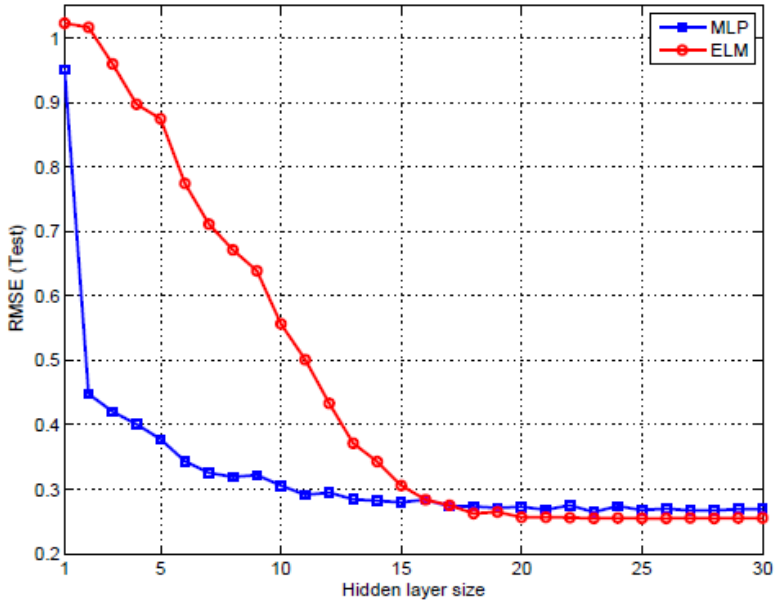


Figura 6.2 Evolución del RMSE respecto al tamaño de la capa oculta (neuronas).

Es necesario señalar que, en general, los "métodos de poda" funcionan mejor que los "métodos de crecimiento" para redes ELM. Esto es debido a que el algoritmo de ELM requiere de un número suficientemente grande de neuronas y, por lo tanto, modelos pequeños ELM conducen a un bajo ajuste con los datos de destino. La generalización de rendimiento obtenida por el algoritmo ELM es muy similar (incluso mejor) a la obtenida mediante MLP estándar, cuando el número de neuronas ocultas es mayor que 15, y, por supuesto, requiriendo de un menor tiempo de entrenamiento. Con el fin de proporcionar una visión general, en la Figura 6.3 contamos con varios ejemplos de los modelos obtenidos (MLP en línea azul discontinua y ELM en línea roja continua) que ajustan los puntos de los datos de entrada para diferentes números de neuronas (2, 8, 15 y 25).

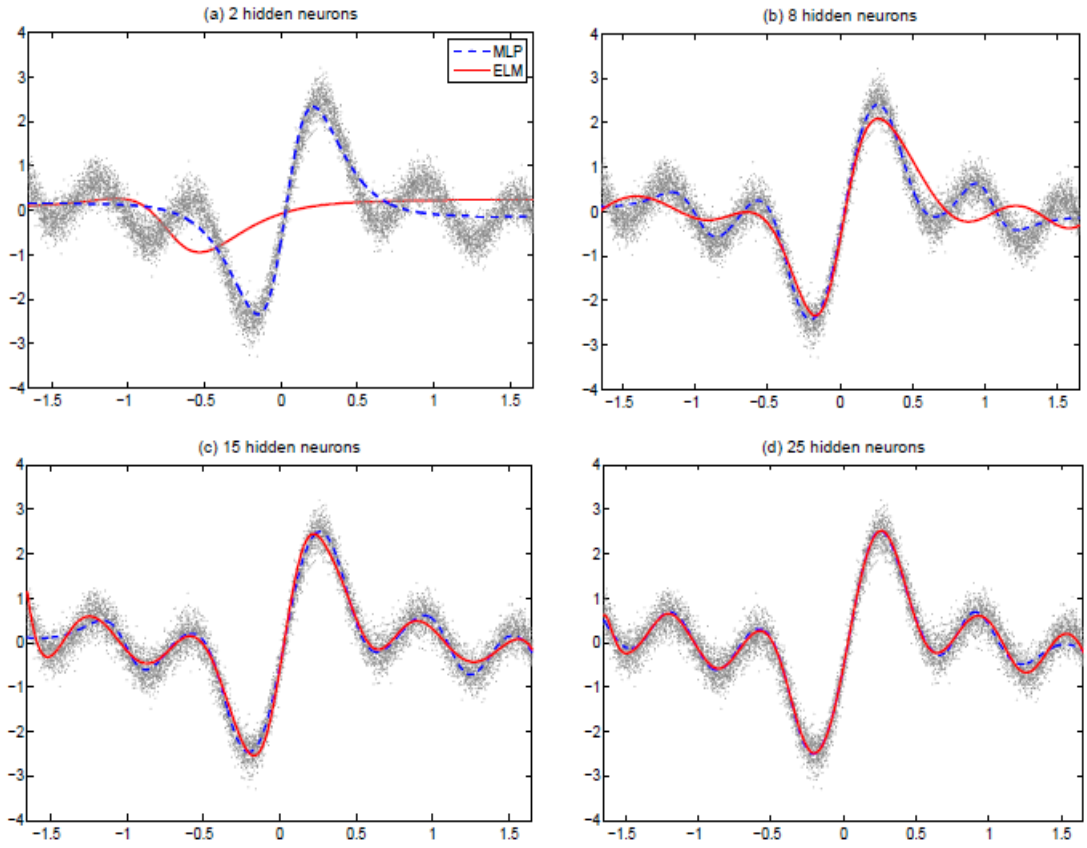


Figura 6.3 Ejemplo de resultados de entrenamiento usando MLP y ELM en el problema Sinusoidal para diferentes tamaños de capa oculta (2, 8, 15 y 25).

Se puede observar en esta figura que las redes más pequeñas (2 y 8 neuronas ocultas) entrenadas con el método MLP estándar se ajustan claramente mejor que las ELM. De hecho, el ajuste global del modelo ELM con 2 neuronas ocultas es muy impreciso. Considerando que, cuando el tamaño de la capa oculta es lo suficientemente grande (15 y 25 neuronas), los modelos ELM y las redes MLP, se ajustan muy bien a los datos.

6.2 Evaluación de problemas reales de carácter multi-disciplinar mediante ELM.

Dispondremos de 3 conjuntos de datos para afrontar 1 problema artificial y 2 problemas reales, resumidos en la Tabla 6.1; *Friedman* y *Elevators*, provienen de la web del Dr. Luis Torgo [71], y el *Tecator* ha sido obtenido del archivo StatLib de Carnegie Mellon [72]. Estos problemas de regresión se han seleccionado tratando de cubrir diferentes complejidades, tales como la existencia de grandes conjuntos de datos o datos de alta dimensión.

Conjunto de Datos	Muestras de entrenamiento	Muestras de Test	*n*
Friedman	27179	13589	10
Elevators	8752	7847	18
Tecator	129	43	100

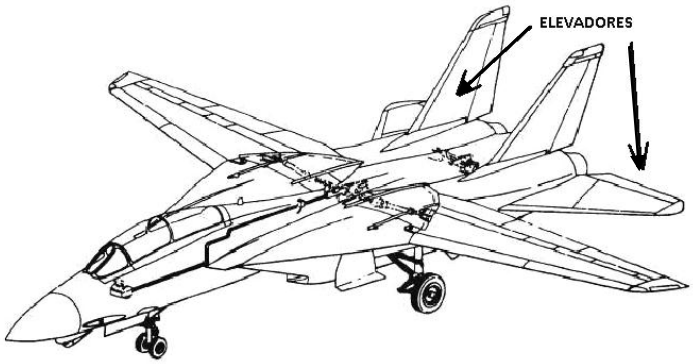
Tabla 6.1 Conjuntos de datos de Regresión usados en los experimentos.

a) Problema de *Friedman*. Este conjunto de datos artificiales representa a una sencilla función aditiva con un componente de ruido aleatorio. Las muestras son generadas a partir de diez atributos, uniformemente distribuidos entre [0,1] de forma independiente entre ellos. La variable objetivo se obtiene con:

$$f(x) = 10\sin(\Pi x_1 x_2) + 20(x_3 - 0{,}5)^2 + 10x_4 + 5x_5 + \sigma, \qquad (6.1)$$

donde σ es el ruido aleatorio gaussiano de media cero y desviación típica la unidad. Su conjunto de entrenamiento posee 27.179 muestras y el conjunto de test se compone de otras 13.589.

b) Problema *Elevators*. Tiene como fin, controlar la estabilización de un avión F16, la variable objetivo está relacionada con la acción tomada en los elevadores en el sistema de estabilización de la aeronave. Posee 16.559 muestras, 8.752 pertenecientes al conjunto de entrenamiento y otras 7.847 del conjunto de test. Con 18 atributos (0 nominales y 18 continuos).



. **Figura 6.4** Esquema descriptivo de los elevadores de estabilización del F16.

c) Problema *Tecator*. Su tarea es predecir el contenido de grasa de una muestra de carne sobre la base de absorción espectroscópica del infrarrojo cercano. Cada muestra contiene información sobre carne pura finamente picada con diferentes contenidos de humedad, grasas y proteínas. Dispone de 100 atributos diferentes, y contiene 240 muestras divididas en 5 conjuntos de datos; Entrenamiento 129, Monitorización 43, Pruebas 43, Extrapolación Grasa 8, Extrapolación 17.

Ahora, abordaremos el análisis de estos problemas utilizando el algoritmo estándar ELM, un método de poda de redes ELM (OP-ELM) y un método para Combinación Óptima de Conjuntos de redes ELM (OC-ELM). Se debe resaltar que tanto OP-ELM y OC-ELM proporcionan un diseño rápido, directo y automático, y métodos de entrenamiento para SLFN individuales y conjuntos de redes. Observe que el método OC-OPELM hace uso de los modelos *L* mismos previamente formados con OP-ELM. En los experimentos, todos los métodos utilizan los subconjuntos de aprendizaje y test original, excepto el algoritmo de ELM estándar que utiliza 10-veces *CV* (Coeficiente de Variación) para seleccionar el tamaño de la capa oculta. Por lo tanto, el tiempo de formación de ELM incluye el procedimiento de 10-veces *CV*.

La Tabla 6.2 muestra los resultados obtenidos en los experimentos realizados con los cinco métodos anteriores basados en ELM. Para cada conjunto de datos y método, esta tabla muestra el tiempo de entrenamiento computacional, los resultados de RMSE de los *L* ensayos y el tamaño del modelo. En concreto, $L = 50$ repeticiones se han realizado diferentes en todos los problemas, a excepción de *Tecator* porque este conjunto de datos de alta dimensión requiere más ensayos ($L = 100$) con el fin de que OC-OPELMs-FFS obtenga diferentes redes aleatorias con suficiente diversidad de características de entrada. Nótese que la tercera columna de esta tabla para ELM y ELM-OP muestra sus resultados obtenidos del test RMSE (media y desviación estándar de los *L* ensayos) para la mejor SLFN según el procedimiento *CV*. Mientras que los demás valores de RSME para los métodos restantes provienen de los diferentes $L^* < L$ conjuntos de redes. En la última columna, se muestra el tamaño de la capa oculta para los métodos ELM y OP ELM, mientras que para el resto de los métodos basados en conjuntos es el número de redes elegido.

Según estos resultados (ver Tabla 6.2), y como se ha mostrado también en [58], OP-ELM supera a ELM, salvo en el caso *Elevators*, gracias al uso de los modelos más pequeños (tamaños más compactos de capas ocultas) y menor tiempo de entrenamiento. En cuanto a los métodos basados en conjuntos, todos ellos ofrecen mejores capacidades de generalización que el ELM y OP-ELM en todos los problemas. Es digno de mención, que estas ventajas son obtenidas con un incremento despreciable del tiempo de entrenamiento requerido por OP-ELM, a excepción de OC-OPELM-FFS que se basa en una función incremental de selección de características de avance y, por lo tanto, necesita de un mayor esfuerzo computacional. En todas las simulaciones, los métodos OC-OPELM-FFS y OC-OPELM-RFS proporcionan mejores resultados, que los métodos de conjuntos basados en el espacio original de características de entrada. Además, el uso de subconjuntos de características aleatorias es beneficioso para la construcción de conjuntos. Este enfoque puede ser mejor que realizar una selección previa de características de avance, a excepción de los conjuntos de datos *Elevators* y *Tecator*. Por ejemplo, en *Tecator*, el espacio de datos de entrada se redujo con éxito de cien a sólo

cinco características. Mientras que para el conjunto de datos *Friedman*, aunque OC-OPELM-FFS elige las características relevantes (que son conocidas previamente por la definición de este conjunto artificial de datos), este método consigue peores resultados de rendimiento que OC-OPELM-RFS, que utiliza subconjuntos de características aleatorias (incluyendo características relevantes e irrelevantes). Como era de esperar, OC-OPELM-RFS requiere de más redes para construir el sistema de conjuntos que OC-OPELM-FFS. Así que, en general, se recomienda que para explotar la diversidad en el conjunto de redes ELM variar el espacio de datos de entrada y, también, la función de selección (antes de la construcción del conjunto), lo que es claramente útil para conjuntos de datos de alta dimensión.

Conjunto de datos	Método de Aprendizaje	RMSE	Tiempo de entrenamiento	Tamaño del modelo
Friedman	ELM	$2.67 \pm 2.8e^{-2}$	$7.87e^{+4}$	139
	OP-ELM	***$2.64 \pm 9.2e^{-3}$***	***$2.18e^{+3}$***	***99***
	LSC-OPELM	2.38	$2.26e^{+3}$	50
	OC-OPELM	2.38	$2.20e^{+3}$	30
	OC-OPELM-FFS	1.44	$5.47e^{+4}$	***16***
	OC-OPELM-RFS	***1.28***	***$2.02e^{+3}$***	33
Elevators	ELM	***$3,59e^{-3} \pm 8.5e^{-5}$***	$3.77e^{+4}$	145
	OP-ELM	$3.65e^{-3} \pm 3.6e^{-5}$	***$1.96e^{+3}$***	***114***
	LSC-OPELM	$2.91e^{-3}$	$2.01e^{+3}$	50
	OC-OPELM	$2.80e^{-3}$	$1.99e^{+3}$	33
	OC-OPELM-FFS	***$2.27e^{-3}$***	$8.92e^{+4}$	***25***
	OC-OPELM-RFS	***$2.27e^{-3}$***	***$1.81e^{+3}$***	40
Tecator	ELM	$4.88 \pm 8.7e^{-1}$	$8.25e^{+3}$	76
	OP-ELM	***$3.57 \pm 1.2e^{-1}$***	***$5.34e^{+2}$***	***67***
	LSC-OPELM	2.45	$5.40e^{+2}$	100
	OC-OPELM	2.28	$5.42e^{+2}$	43
	OC-OPELM-FFS	***1.28***	$1.07e^{+4}$	***15***
	OC-OPELM-RFS	2.04	***$5.28e^{+2}$***	29

Tabla 6.2 Resultados experimentales de los tres conjuntos de regresión.

Capítulo 7

Conclusiones y trabajos futuros

7.1 Conclusiones.

El algoritmo ELM y sus recientes extensiones, como el método de OP-ELM, proporcionan algoritmos de aprendizaje simples, rápidos y robustos para SLFNs con nodos ocultos aleatorios. El proceso de entrenamiento de un modelo ELM puede ser de varios órdenes de magnitud más rápido que los algoritmos de aprendizaje tradicionales para FFNN, mientras que mantiene una comparable capacidad de generalización y de aproximación, o incluso superar éstas. Este PFC presenta eficaces procedimientos de conjuntos basados en ELM con el fin de explotar la diversidad entre los diferentes SLFNs del sistema. Los métodos de conjuntos descartan los modelos individuales imprecisos para el conjunto según la base de la metodología OP-ELM y, por lo tanto, consiguiendo una combinación óptima de las redes.

A diferencia de otros métodos de conjuntos basados en ELM, hay que señalar que cada red se construye automáticamente sin fijar un tamaño predefinido para la capa oculta. Además, las redes ELM se han construido con diferentes espacios de entrada que incluyen: el espacio original de características de entrada, el subconjunto de entrada obtenido gracias a la función de selección de avance y los subconjuntos de características aleatorias.

Los resultados experimentales demuestran que un enfoque basado en conjuntos de modelos ELM supera a un modelo ELM individual en términos de capacidad de generalización. Los resultados también indican las ventajas del método de ELM respecto al entrenamiento MLP estándar. Además, las técnicas de poda y conjuntos se han analizado con el fin de obtener mejores resultados de rendimiento sin aumentar el tiempo computacional La diversidad del conjunto se aumenta variando el espacio de características de entrada lo que produce una mejora del rendimiento. Esto los convierte en una herramienta valiosa para su implementación en aplicaciones reales que requieren de un tiempo de entrenamiento rápido con un buen rendimiento.

7.2 Trabajos futuros.

Dentro del entorno de investigación del Centro Universitario de la Defensa de San Javier, actualmente se presentan en el horizonte varias líneas de investigación y trabajos, que están siendo desarrollados destinados a la verificación de los resultados obtenidos en este PFC. En primer lugar parece obligado abarcar una validación de las nuevas metodologías presentadas en este PFC para abordar problemas de clasificación, después de haber quedado demostrado su excelente rendimiento para problemas de regresión. Viendo los resultados obtenidos se hace nesaria la utilización de técnicas de edición de datos con el fin de disponer de distintos subconjuntos de datos que permitan incrementar la diversidad en el modelo de conjunto de redes ELM. E investigar la extensión de estas metodologías en problemas multi-tarea, donde el aprendizaje simultáneo de la tarea principal con varias tareas relacionadas mejora el modelado de dicha tarea.

Bibliografía

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

[2] G.-B. Huang, Q.-Y. Zhu, C.-k. Siew, "*Extreme Learning Machine: Theory and Applications*", Neurocomputing, 70(1-3), pp. 489-501, (2006).

[3] Y. Miche, P. Bias, C. Jutten, O. Simula, A. Lendasse, "*A methodology for Building Regression Models using Extreme Learning Machine*", 16th European Symposium in Artificial Neural Networks, pp. 247-252, Bruges, Belgium, (2008).

[4] A. Sharkey. *Combining artificial neural nets: ensemble and modular multi-net systems*, Springer (1999).

[5] Andries P. Engelbrech. *"Computational Intelligence An Introduction"*, University of Pretoria. Wiley (2007).

[6] David A. Elizondo and Stephen G. Matthews. *Recent Patents on Computational Intelligence.*, Centre for Computational Intelligence, School of Computing, De Montfort University, Leicester (2009).

[7] A.M. Turing. *Computing Machinery and Intelligence*. Mind, 59:433–460, (1950).

[8] Rosenblatt, Frank. *The Perceptron a perceiving and recognizing automaton*. Report 85-460-1, Cornell Aeronautical Laboratory. (1957).

[9] B. Widrow. *ADALINE and MADALINE* – 1963, Plenary Speech. In Proceedings of the First IEEE International Joint Conference on Neural Networks, volume 1, pages 148–158, (1987).

[10] Marvin Minsky, Seymour Papert, "*Perceptrons*", MIT Press, (1969, Enlarged edition, (1988).

[11] Hopfield, J. J. *Neural networks and physical systems with emergent collective computational abilities*. Proceedings of the National Academy of Sciences, USA, 79:2554–2558. (1982).

[12] Hinton, G. E. and Sejnowski, T. J. *Optimal perceptual inference*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Washington, DC. (1983).

[13] Rumelhart, D. E. and McClelland, J. L. *Parallel distributed processing: Explorations*

in the microstructure of cognition. Volume I. Cambridge, MA: MIT Press (1986).

[14] A.K. Jain, J. Mao, and K.M. Mohiuddin. "*Artificial Neural Networks: A Tutorial*". IEEE Computer Magazine, 29(3):31.44, (1996).

[15] W.S. McCulloch and W. Pitts. "*A logical calculus of the ideas immanent innervous activity*". Bulletin of Mathematical Biophysics, 5:115.133, (1943).

[16] Jeffrey L. Elman. *Finding structure in time*. Cognitive Science Volume 14, Issue 2, Pages 179-211, (1990)

[17] Jordan, M.I.. *Serial order: A parallel distributed processing approach* (Tech. Rep. No. 8604). San Diego: University of California, Institute for Cognitive Science. (1986)

[18] Kohonen, Teuvo. "*Self-Organized Formation of Topologically Correct Feature Maps*". Biological Cybernetics 43 (1): pp. 59–69. (1982).

[19] Hardy, R.L., *Theory and applications of the multiquadric-biharmonic method, 20 years of Discovery*, 1968 1988, Comp. math Applic. Vol 19, no. 8/9, pp. 163. (1990)

[20] Alan S. Lapedes, Robert M. Farber: How Neural Nets Work. NIPS: 442-456. (1987)

[21] Martin, Edward A., Richard P. Lippmann, and Douglas B. Paul, *Two-Stage Discriminant Analysis for Improved Isolated-Word Recognition*, Proceedings IEEE International Conference on Acoustics Speech and Signal Processing, pp. 709–712, (1987).

[22] D.O. Hebb: The Organization of Behavior, Wiley: New York; (1949).

[23] Simon Haykin. "*Neural Networks: A Comprehensive Foundation*". Prentice Hall, Upper Saddle River, N.J., (1999).

[24] F. Girosi and T. Poggio "*Networks and the Best Approximation Property*," Biological cybernetics, vol. 63, pp. 169-176, (1990)

[25] Burges, Cristopher J.C., "*A tutorial on support vector machines for pattern recognition*",http://svm.research.bell-labs.com/papers/tutorial_web_page.ps.gz. (1998)

[26] Gunn, Steve, "*Support vector machines for classification and* regression ", Image Speech & Intelligent Systems Group, University of Southampton, may (1998),

[27] Guang-Bin Huang, Qin-Yu Zhu, Chee-Kheong Siew, *Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks*, International Joint Conference on Neural Networks, Vol. 2 pp: 985-990, (2004)

[28] Annema, A.J. and Hoen, K. and Wallinga, H, "*Precision requirements for single-layer feedforward neural networks*", In: Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems, pp: 145-151, (1994).

[29] R. Rajesh, J. Siva Prakash. "Extreme Learning Machines - A Review and State-of-the-art" International Journal of wisdom based Computing, Vol. 1 pp 35-48 , (2011)

[30] Nan-Ying Liang, Guang-Bin Huang, Hai-Jun Rong, P. Saratchandran, N. Sundararajan, *A Fast and Accurate On-line Sequential Learning Algorithm for Feedforward Networks,* IEEE Transactions on Neural Networks, Vol. 17, No. 6, pp: 1411-1423, (2006).

[31] Rampal Singh, S. Balasundaram, "*Application of Extreme Learning Machine Method for Time Series Analysis"*, International Journal of Intelligent Technology, Vol.2, No.4, pp: 256-262, (2007).

[32] Guang-Bin Huang, Lei Chen, Chee-Kheong Siew, *Universal Approximation Using Incremental Constructive Feedforward Networks with Random Hidden Nodes*, IEEE Transactions on Neural Networks, Vol. 17, No. 4, pp: 879-892, (2006).

[33] Guang-Bin Huang, Lei Chen, Convex Incremental Extreme Learning Machine, Neurocomputing, Vol. 70, pp: 3056-3062, (2007).

[34] A.R. Barron, "*Universal approximation bounds for superpositions of a sigmoidal function*", IEEE Trans. Inf. Theory 39 (3) 930-945. (1993).

[35] Guang-Bin Huang, Lei Chen, *Enhanced Random Search Based Incremental Extreme Learning Machine*,Neurocomputing, Vol. 71, pp: 3460-3468, (2008).

[36] Guang-Bin Huang, Qin-Yu Zhu K. Z. Mao, Chee-Kheong Siew, P. Saratchandran, N. Sundararajan, *Can Threshold Networks Be Trained Directly?*, IEEE Transactions on Circuits and Systems-II, Vol. 53, No. 3, pp: 187-191, (2006).

[37] Guorui Feng, Guang-Bin Huang, Qingping Lin, Robert Gay, *Error Minimized Extreme Learning Machine with Growth of Hidden Nodes and Incremental Learning*, IEEE Transactions on Neural Networks, (2009).

[38] Hoerl AE, Kennard RW *Ridge regression: biased estimation for nonorthogonal problems*. Technometrics 12(1):55–67 (1970)

[39] Huang G-B, Zhou H, Ding X, Zhang R Extreme learning machine for regression and multi-class classification. IEEE Trans Pattern Anal Mach Intell. (2010)

[40] Toh K-A (2008) *Deterministic neural classification*. Neural Comput 20(6):1565–1595. (2008)

[41] Deng W, Zheng Q, Chen L *Regularized extreme learning machine*. In: IEEE symposium on computational intelligence and data mining (CIDM2009), 30 March 2009–2 April 2009, pp 389–395. (2009)

[42] Man Z, Lee K, Wang D, Cao Z, Miao C *A new robust training algorithm for a class of single-hidden layer feedforward neural networks*. Neurocomputing. (2011)

[43] Miche Y, van Heeswijk M, Bas P, Simula O, Lendasse A TROP-ELM: *a double-regularized elm using lars and tikhonov regularization*. Neurocomputing (in press) (2011)

[44] Fre´nay B, Verleysen M *Using SVMs with randomized feature spaces: an extreme learning approach*. In: Proceedings of the 18th European symposium on artificial neural networks (ESANN), Bruges, Belgium, 28–30 Apr 2010, pp 315–320 (2010)

[45] Fre´nay B, Verleysen M (2011) *Parameter-insensitive kernel in extreme learning for non-linear support vector regression*. Neurocomputing (in press) (2011)

[46] S. Haykin. *Neural Networks: A Comprehensive Foundation* (2nd Edition). Prentice Hall, July (1998)

[47] T. Ash. Dynamic node creation in backpropagation networks. In International Joint Conference on Neural Networks, volume 2, page 623, June 1989.

[48] T.-Y. Kwok and D.-Y. Yeung. Constructive neural networks: some practical considerations. In Neural Networks, 1994. IEEE World Congress on Computational Intelligence.,1994 IEEE International Conference on, volume 1, pages 198–203, (1994).

[49] S.-K. Sharma. *Constructive neural networks: A review*. International Journal of Engineering Science and Technology, 2(12):7847–7855, (2010).

[50] J. Sietsma and R. Dow. *Creating artificial neural networks that generalize. Neural networks*, 4:67–79, (1991).

[51] M. Pelillo and A. Fanelli. *A method of pruning layered feed-forward neural networks*. In J. Mira, J. Cabestany, and A. Prieto, editors, New Trends in Neural Computation, Volume 686 of Lecture Notes in Computer Science, pages 278–283. Springer Berlin / Heidelberg, (1993)

[52] R. Reed. *Pruning algorithms - a review*. IEEE Transactions on Neural Networks, 7:740–747, (1993).

[53] H.-J. Rong, Y.-S. Ong, A.-H. Tan, and Z. Zhu. *A fast pruned-extreme learning machine for classification problem*. Neurocomputing, 72(1-3):359–366, (2008).

[54] Y. Miche, P. Bas, C. Jutten, O. Simula, and A. Lendasse. *A methodology for building regression models using extreme learning machine: OP-ELM*. In Proceedings of the European Symposium on Artificial Neural Networks (ESANN), pages 247–252, (2008).

[55] Y. Miche, A. Sorjamaa, and A. Lendasse. *OP-ELM: Theory, experiments and a toolbox*. In Proceedings of the International Conference on Artificial Neural Networks (ICANN), LNCS 5163, pages 145–154, (2008).

[56] F. Mateo and A. Lendasse. *A variable selection approach based on the delta test for extreme learning machine models*. In Proceedings of the European Symposium on Time Series Prediction (ESTP), pages 57–66, September (2008).

[57] Y. Miche and A. Lendasse. *A faster model selection criterion for OP-ELM and OPKNN: Hannan-quinn criterion*. In Proceeding of the European Symposium on Artificial Neural Networks (ESANN), pages 177–182, April 22-24 (2009)

[58] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse. *OP-ELM: Optimally Pruned Extreme Learning Machine. Neural Networks*, IEEE Transactions on, 21(1):158–162, December (2009).

[59] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, (1991).

[60] U. Fayyad and K. Irani. *Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning*. In Proceedings of the International Joint Conference on Uncertainty in Artificial Intelligence, pages 1022–1027, (1993).

[61] T. Similä and J. Tikka. *Multiresponse sparse regression with application to multidimensional scaling*. In Proceedings of the 15th International Conference on Artificial Neural Networks: Formal Models and Their Applications - ICANN 2005, LNCS 3697, pages 97–102, (2005).

[62] A. Bartoli. *On computing the prediction sum of squares statistic in linear least squares problems with multiple parameter or measurement sets*. International Journal on Computer Vision, 85:133–142, November (2009).

[63] L. K. Hansen and P. Salamon. *Neural network ensembles*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 12(10):993–1001, (1990).

[64] Ch. J. Merz and M. J. Pazzani. *Combining neural network regression estimates with regularized linear weights*. In Advances in Neural Information Processing Systems 9 (NIPS), pages 564–570, (1996).

[65] Z. Zhou. *Ensembling neural networks: Many could be better than all. Artificial Intelligence*, 137(1-2):239–263, (2002).

[66] H. Chen, H. Chen, X. Nian, and P. Liu. *Ensembling extreme learning machines*. In 4th International Symposium on Neural Networks, ISNN, pages 1069–1076, (2007).

[67] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, (2006).

[68] M. van Heeswijk, Y. Miche, E. Oja, and A. Lendasse. *Solving large regression problems using an ensemble of GPU-accelerated ELMs*. In ESANN2010: 18th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, pages 309–314, (2010).

[69] P. J. García-Laencina and J.-L. Sancho-Gómez. *Diseño óptimo de comité de redes*

neuronales. Jornadas de introducción a la investigación de la UPCT, 2(1):1–3, (2009).

[70] P. J. García-Laencina. *Improving Predictions Using Linear Combination of Multiple Extreme Learning Machines*. Information Technology and Control, Vol 42, No.1 (2013).

[71] Luís Torgo Regression Data Sets.
http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html

[72] Libstat Data Sets.-Carnegie Mellon University. http://lib.stat.cmu.edu/

Glosario

RNA: Red Neuronal Artificial.
ANN: Artificial Neural Networks.
ELM: Extreme Learning Machine Algorithm.
PFC: Proyecto Fin de Carrera.
IC: Inteligencia Computacional.
IA: Inteligencia Artificial.
Adaline: Adaptative Linear Algorithm.
FFNN: FeedForward Neural Network.
MLP: Multi-Layer Perceptron.
BP: Backpropagation.
LMS: Least Mean Square.
RBF: Radial Basis Function.
FBR: Función Base Radial.
FSCL: Frequecy Sensitive Competitive Learning.
fdp: Función de distribución de probabilidad.
LVQ: Learning Vector Quantization.
SVM: Maquinas de Vectores Soporte.
VC: Vapnik-Chervonenkis.
SLFN: Single Layer Feedforward Neural Network.
SVD: Descomposición de Valor Singular.
I-ELM: Incremental Extreme Machine Learning.
TFLN: Two Hidden Layer FeedForward Neural Network.
EI-ELM: Enhanced Random search based Incremental Learning Machine.
P-ELM: Pruned ELM.
OP-ELM: Optimally Pruneed ELM.
AIC: Akaike's information criterion.
LOO: Leave-One-Out.
MRSR: MultiResponse Sparse Regression.
GP: Procesos Gausianos.
LARS: Least Angle Regression.
PRESS: Prediction Sum of Squares.
PIEx: Product Index based Excluding.
MSE: Mean Square Error.

BIC: Criterio de Información Bayesiano clásico.

CPU: Central Processing Unit.

GP: Procesamiento Gráfico.

OC-ELM: Combinación óptima de las máquinas de Extreme Learning Machine.

OC-OPELM-FFS: Combinación Optima de OP-ELMs basada en Selección de Características de Avance.

OC-OPELM-RFS: (Combinación Óptima de OP-ELM basada en subconjuntos de Características Azar).

SGC: Gradiente Conjugado Escalado.

CV: Coeficiente de Variación.

RMSE: Raíz del Error Cuadrático Medio.