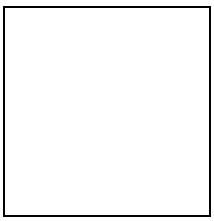
UNIVERSIDAD DE GRANADA

E.T.S. DE INGENIERÍA INFORMÁTICA



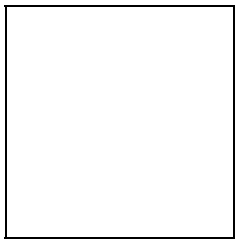
Departamento de Ciencias de la Computación
e Inteligencia Artificial

REPRESENTACIÓN DE CONOCIMIENTO ESTRUCTURADO Y SU APLICACIÓN AL DISEÑO DE SISTEMAS AUTOMÁTICOS

TESIS DOCTORAL

Carlos Javier Mantas Ruiz

Granada, Abril de 1999



REPRESENTACIÓN DE CONOCIMIENTO ESTRUCTURADO Y SU APLICACIÓN AL DISEÑO DE SISTEMAS AUTOMÁTICOS

MEMORIA QUE PRESENTA

CARLOS JAVIER MANTAS RUIZ

PARA OPTAR AL GRADO DE DOCTOR EN INFORMÁTICA

ABRIL 1999

DIRECTORES

JUAN LUIS CASTRO PEÑA **MIGUEL DELGADO CALVO-FLORES**

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

E.T.S. de INGENIERÍA INFORMÁTICA UNIVERSIDAD DE GRANADA

REPRESENTACIÓN DE CONOCIMIENTO ESTRUCTURADO Y SU APLICACIÓN AL DISEÑO DE SISTEMAS AUTOMÁTICOS

CARLOS JAVIER MANTAS RUIZ

La memoria titulada **Representación de Conocimiento Estructurado y su Aplicación al Diseño de Sistemas Automáticos**, que presenta D. Carlos Javier Mantas Ruiz para optar al grado de Doctor, ha sido realizada en el Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada bajo la dirección de los Doctores D. Juan Luis Castro Peña y D. Miguel Delgado Calvo-Flores.

Granada, Abril de 1999

El Doctorando

Los Directores

Fdo. C.J. Mantas Ruiz

Fdo. J.L. Castro Peña

Fdo. M. Delgado Calvo-Flores

“Es más importante la imaginación que el conocimiento”
Albert Einstein

ÍNDICE GENERAL

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ALGORITMOS

Capítulo 1

INTRODUCCIÓN

1.1.- Motivación

El objetivo general de nuestro trabajo consiste en presentar una metodología para modelar y/o diseñar sistemas automáticos que usan herramientas de Inteligencia Artificial, basada en el estudio de la estructura de estos sistemas, centrándonos principalmente en la forma de definir métodos de aprendizaje automático en función de componentes generales estructurados. En el resto del texto, a estos sistemas automáticos basados en Inteligencia Artificial los denominaremos **sistemas inteligentes**.

Tres hechos han motivado el desarrollo del trabajo presentado en esta tesis:

a) La evolución de los modelos de representación del conocimiento, desde aquellos primeros que sólo permitían especificar sistemas uniformes hasta los actuales que permiten identificar varios módulos en un mismo sistema, cada uno con diferente tipo de conocimiento, y una estrategia de control asociada:

   Inicialmente, se definieron modelos para representar conocimiento que sólo permitían especificar sistemas de una manera uniforme, o sea, sólo podían modelarlos usando un único esquema de representación. Los sistemas no se podían estudiar como compuestos por varios subsistemas que realizaban distintas tareas, dificultando de esta manera incluir conocimiento de control en la especificación de los mismos. Ejemplos de estos modelos son las redes semánticas [Quillian68, Raphael68] que se apoyaban en un modelo de memoria asociativa humana basada en redes, cuyos nodos eran determinados conceptos y cuyos arcos, conexiones entre nodos, eran relaciones de diversos tipos entre los conceptos que conectaban; los marcos o frames [Minsky75], que eran formas estructuradas de patrones de conceptos complejos que podrían estar en la mente humana, los cuales se organizaban en redes jerarquizadas; y los sistemas basados en reglas [Shortliffe75, Shortliffe76, Buchanan78] cuya modelización del conocimiento consistía, simplemente, en

listar las reglas condición-acción que dirigían el comportamiento de un sistema, intentando incluir conocimiento de control por medio de metareglas.

En los últimos años están apareciendo modelos de representación del conocimiento que incluyen conceptos para especificar información a un alto nivel de abstracción. Por ejemplo, el modelo que consiste en describir un sistema por medio de "tareas genéricas" [Chandrasekaran87], siendo una "tarea genérica" un concepto que engloba a una tarea de utilidad general, un método para llevarla a cabo y la clase de conocimiento que el método necesita; el modelo basado en los métodos *role-limiting* [McDermott88] que son métodos que resuelven tareas generales en las que se especifican, al nivel de operaciones primitivas, todas las acciones y subacciones necesarias para ejecutar el método; la aproximación *Components of Expertise* [Steels90] que consiste en modelar el conocimiento de un sistema por medio de un árbol de tareas y subtareas, donde una tarea compleja se descompone en las subtareas necesarias para realizarla y, donde implícitamente se asume la elección de algún método para llevar a cabo una tarea; la metodología KADS para construir sistemas expertos [Wielinga92] que consiste en usar un lenguaje de modelización basado en una serie de términos primitivos de diferentes niveles de abstracción, con el propósito de capturar el conocimiento acerca de un dominio particular aportado por un experto; y, finalmente, el modelo *Task-Structure* [Chandrasekaran92] que es un modelo de representación general que consiste en una organización de tareas, subtareas y métodos, intentando englobar en su definición al resto de modelos de representación de conocimiento estructurado. Todos estos modelos permiten identificar tareas realizadas por un sistema en diferentes niveles de abstracción, logrando, de esta manera, especificar conocimiento de control asociado a ellas.

Podemos apreciar la tendencia hacia la utilización de modelos de representación del conocimiento que permitan visualizar la estructura del conocimiento manipulado por un sistema. Este hecho vislumbra un nuevo campo de estudio referente al análisis de las propiedades de un sistema en función de la estructura del conocimiento que maneja.

b) La evolución en el campo del aprendizaje automático, desde el interés por el estudio de los sistema monoestrategia hasta el diseño de sistemas de aprendizaje multiestrategia:

Primeramente, los sistemas de aprendizaje que aparecían eran sistemas monoestrategia, o sea, sistemas que emplean un único tipo de inferencia y un único esquema para representar su conocimiento. Por ejemplo, las redes neuronales artificiales [Lippmann87] donde el conocimiento se representa mediante una topología de red y pesos, y su mecanismo de inferencia consiste en la dinámica particular de cada neurona y la propagación de su salida; los árboles de decisión [Quinlan86] donde el conocimiento se representa mediante un árbol con atributos en sus nodos y la inferencia consiste en realizar consultas sobre el valor de un atributo en cada nodo hasta alcanzar una hoja; los algoritmos genéticos [Goldberg89, Holland75, Michalewicz96] donde el conocimiento se representa mediante una población de cromosomas y el mecanismo de inferencia consiste en realizar una serie de operaciones genéticas como son el cruce y la mutación, etc.

Actualmente, están apareciendo unos primeros trabajos para el diseño de sistemas de aprendizaje multiestrategia, o sea, sistemas que durante su fase de diseño pueden emplear distintos esquemas de aprendizaje [Michalsky93]. Por ejemplo, en el trabajo presentado en [Giordana97] el marco de trabajo de cooperación multiestrategia entre varios esquemas de aprendizaje se estructura jerárquicamente, en el nivel más alto, un sistema de aprendizaje relacional simbólico construye una teoría de clasificación usando la lógica de primer orden, a continuación, en el nivel más bajo, una estrategia conexionista o genética refina las disyunciones de la teoría de clasificación diseñada en el nivel anterior. Los sistemas multiestrategia pueden tener la capacidad de resolver un rango más amplio de problemas de aprendizaje que los sistemas monoestrategia, ya que tienen la ventaja de poder complementar estrategias de aprendizaje individuales.

Podemos observar una tendencia, en el campo de la Inteligencia Artificial, hacia el estudio de los sistemas de aprendizaje multiestrategia debido a las ventajas que presentan este tipo de sistemas. Nosotros pensamos que el estudio de los sistemas inteligentes en función de su estructura, significaría un avance en esta nueva línea de investigación del aprendizaje automático, ya que posibilitaría definir, buscando ciertas propiedades estructurales, nuevos métodos de aprendizaje en función de componentes abstractos. Estos componentes se podrían instanciar con subsistemas que usen diferentes esquemas de aprendizaje, dando lugar a la definición de sistemas de aprendizaje multiestrategia. En definitiva, el estudio de sistemas inteligentes en base a su estructura puede permitir definir una metodología para diseñar sistemas de aprendizaje multiestrategia.

c) La presentación de numerosos sistemas inteligentes que se estudian usando un único módulo para representar su conocimiento, aunque su definición implique la aparición de varias tareas obedeciendo a una estrategia de ejecución global, cada una con su propio esquema de representación del conocimiento:

   Se están presentando sistemas estructurados, o sea, sistemas constituidos por varios componentes, cada uno con su propio mecanismo de inferencia y su propio esquema de representación del conocimiento, obedeciendo, todos ellos, a una estrategia global de ejecución, que resuelven diversos problemas y que se estudian de un modo uniforme. Algunos de ellos tienen una estructura equivalente, diferenciándose solamente en los elementos utilizados para instanciar sus componentes. Por ejemplo, sistemas para realizar control [Raju93, Chiu94, Sugeno93, Sun94, Yager94], para hacer razonamiento [Lukaszewicz90, Castro98a], para llevar a cabo tareas de aproximación de funciones [Fahlman90, Takagi91, Takagi92, Chiang94], para robótica [Saffiotti93, Saffiotti95, Saffiotti97, Jacobs91], para realizar clasificación [Sirat90, Park90, Alché94], etc.

   Estos sistemas pueden estudiarse como compuestos por varios subsistemas que realizan tareas particulares y que son dirigidos por una estrategia de control global (como veremos más adelante), sin embargo, se estudian usando un único módulo de representación de conocimiento y sin especificar su conocimiento de control subyacente. Además, algunos de ellos presentan características estructurales comunes lo que permitiría su estudio conjunto, sin embargo, se estudian de manera independiente. Nosotros pensamos que estas dos carencias en la presentación de sistemas inteligentes estructurados (falta de especificación del conocimiento de control del sistema y ausencia de estudio conjunto de sistemas con estructura equivalente), se deben a la falta del uso de herramientas de descripción estructurales. Podría ser interesante definir un nuevo concepto que permita estudiar los sistemas inteligentes estructurados presentados en la literatura, especificando la estrategia de control de sus módulos y agrupando, en un mismo esquema, a aquellos sistemas con características estructurales comunes.

Analizando los hechos anteriores, apreciamos la necesidad de una herramienta, fácil de utilizar y que esté en la línea de la evolución actual de los modelos de

representación del conocimiento, que permita estudiar los sistemas inteligentes y analizar sus propiedades en función de su estructura. Esta herramienta deberá posibilitar la definición de nuevos métodos de aprendizaje automático, basados en componentes genéricos estructurados, que podrían ser instanciados con diferentes esquemas de aprendizaje en función de las características del problema a resolver. Así, se presentaría una nueva metodología para definir sistemas de aprendizaje multiestrategia, lo cual avanzaría un poco más en el campo de estudio reciente de la Inteligencia Artificial referente al *aprendizaje multiestrategia*. Un primer paso para conseguir esto, consistiría en definir un nuevo modelo de representación del conocimiento que permita representar la estructura de un sistema. Su uso posibilitaría:

- Representar una gran variedad de sistemas inteligentes.

- Agrupar en unas pocas clases generales a una gran cantidad de sistemas inteligentes conocidos, gracias a sus características estructurales comunes.

- Estudiar en conjunto a los sistemas inteligentes, lo que implicará su utilización de una manera más fácil y cómoda, al conocerlos mejor.

- Definir nuevos sistemas a partir de componentes abstractas, que se podrían particularizar con herramientas concretas en función del problema que se quiera resolver.

Estas características nos permiten vislumbrar una nueva sistemática para identificar un sistema inteligente que resuelva un problema, la cual consiste en:

1. consultar el estudio de sistemas inteligentes en base a su estructura para elegir un sistema cuyas propiedades generales, heredadas de su estructura, sean las más indicadas y, a continuación,

2. añadir al sistema las propiedades particulares, derivadas de la instanciación de los componentes generales de su estructura, necesarias para resolver completamente el problema.

Esta metodología contrasta con las dificultades actuales a las que hay que enfrentarse a la hora de identificar un sistema inteligente para resolver un problema, debido a:

∗ el aprovechamiento de los trabajos realizados anteriormente es difícil, ya que se presentan de manera dispersa y muy orientados a la resolución de un problema particular y,

∗ la necesidad de seleccionar todas las propiedades óptimas para el sistema de una vez, tanto las generales, heredadas de la estructura del sistema, como las particulares derivadas de las herramientas concretas usadas en los componentes del sistema en el momento de su utilización.

1.2.- Objetivos

El propósito de esta tesis es doble:

- por una parte pretendemos estudiar diversos sistemas inteligentes en función de su estructura con el objeto de disponer de alguna información acerca de ellos a la hora de resolver un problema y

- por otra parte, perseguimos definir nuevos métodos de aprendizaje automático a un nivel alto de abstracción que identifiquen sistemas inteligentes estructurados con unas propiedades generales concretas y, cuyos componentes se puedan instanciar de varias formas, implicando cada una de ellas propiedades particulares distintas para el sistema identificado.

Lograr estos propósitos conlleva una serie de subobjetivos:

a) Definir un modelo para representar conocimiento estructurado, o sea, un modelo que sólo especifique la estructura de un sistema, olvidándonos de las características particulares de sus componentes, de tal forma que el modelo pueda representar fácilmente gran cantidad de sistemas inteligentes y logre agrupar en una misma representación a varios sistemas.

b) Estudiar las propiedades de un sistema inteligente derivadas de su estructura y, a partir de este estudio, apreciar las ventajas e inconvenientes del sistema y de su procedimiento de diseño.

c) Utilizar el estudio anterior para realizar un análisis conjunto de varios sistemas inteligentes en función de las propiedades generales derivadas de su estructura.

d) Definir un nuevo método de aprendizaje automático a un nivel abstracto, que simule la estructura del aprendizaje humano y que logre identificar modelos de sistemas con buenos resultados de aproximación.

e) Presentar varias instanciaciones del método de aprendizaje automático estructurado definido a nivel abstracto, que aporte a los modelos de sistema identificados, diferentes propiedades particulares en función de las herramientas concretas usadas en la instanciación.

1.3.-Descripción por capítulos

En el capítulo 2, después de apreciar la estructura de ciertas formas de comportamiento humano y de estudiar diferentes sistemas inteligentes estructurados, presentamos un modelo para representar conocimiento estructurado, denominado MORSE. Este modelo permite especificar la estructura de algunos tipos de comportamiento humano y la de varios sistemas inteligentes que resuelven distintas tareas, consiguiendo estudiar conjuntamente a algunos sistemas con características estructurales comunes. Para concluir este capítulo, presentamos un análisis de sistemas inteligentes en función de las características particulares de los modelos MORSE que los representan. Este análisis permite apreciar las propiedades generales de un sistema inteligente derivadas de su estructura.

En el capítulo 3 presentamos un método de aprendizaje automático denominado MEGAS, el cual está inspirado en el aprendizaje del ser humano y en el estudio de la estructura de varios sistemas inteligentes realizado en el capítulo anterior. MEGAS es un método de aprendizaje definido a partir de componentes generales que, básicamente, consiste en obtener una solución aproximada a un problema y, a continuación, afinarla por zonas. Explicamos las ventajas de su uso cuando sus componentes son reglas difusas o redes neuronales. Al final del capítulo presentamos varios experimentos de una instanciación de MEGAS donde sus componentes son redes neuronales, comparando su ejecución con la de sistemas no estructurados que usan la misma herramienta.

En el capítulo 4, ante la necesidad presentada en algunos problemas, de tener que realizar varias ejecuciones seguidas de MEGAS para lograr identificar el modelo de un sistema con una exactitud requerida, respondemos a la cuestión de cómo realizar varias iteraciones del método MEGAS. Resolvemos esta cuestión aprovechando el análisis de varios sistemas inteligentes en función de su estructura, realizado en el

capítulo 2: evitamos los problemas de un tipo de sistema estructurado (sistemas basados en el modelo de árbol) por medio de las ventajas de otro (sistemas basados en el modelo de red neuronal) y viceversa. El resultado de este capítulo es un método de aprendizaje automático denominado SEPARATE, el cual también se define de manera genérica (a partir de componentes generales) y cuya utilidad comentamos cuidadosamente con la ayuda de la experimentación.

En el capítulo 5 planteamos el problema de cómo identificar el modelo de un sistema mediante MEGAS (instanciación de componentes), enfocado a la creación de un sistema que tenga la propiedad particular referente a una fácil comprensión de su funcionamiento. Para resolverlo hacemos uso de los sistemas basados en reglas difusas [Klir95], de los árboles de decisión [Quinlan86] y de las redes neuronales artificiales [Lippmann87, Benítez97]. El resultado es un método de aprendizaje automático multiestrategia que identifica sistemas con buenos resultados en aproximación y que poseen una interpretación lingüística de su funcionamiento fácilmente comprensible (*sistemas híbridos para razonamiento aproximado*). Con este capítulo mostramos lo cómodo que es modificar un modelo, definido a nivel de estructura, para conseguir unas características particulares de funcionamiento, en este caso, interpretación lingüística de su procesamiento.

La tesis termina con una presentación de conclusiones y la formulación de algunas líneas de investigación planteadas como trabajo futuro.

Capítulo 2

MORSE: UN MODELO GENERAL PARA REPRESENTAR CONOCIMIENTO ESTRUCTURADO

2.1.- Introducción

En un sistema inteligente que procesa una entrada para producir una salida, podemos apreciar dos tipos de conocimiento: declarativo y procedural [Anderson83, Damasio90, Keil89, Sun94b]. En el conocimiento procedural suelen aparecer varias divisiones de acciones en subacciones para lograr un objetivo, formando una estructura organizada de tareas. En este capítulo, enfocamos el estudio de sistemas sobre esta estructura contenida en su conocimiento procedural.

Para ilustrar la estructura en el conocimiento procedural manipulado por un sistema usaremos al ser humano. El comportamiento humano no se puede representar de una manera uniforme ya que el ser humano suele realizar distintas subacciones en un cierto orden para lograr un objetivo. Por ejemplo, ¿cómo lee un científico un artículo?. Lo primero que hace es echar un vistazo general a todas las páginas del artículo: lee el resumen, mira las figuras, lee el título de las secciones, etc. Con esta acción consigue conocimiento general acerca del artículo: de qué trata, dónde están los puntos de mayor interés, cuál es su esquema, etc. Tras realizar esta primera tarea, el científico puede dedicarse a leer con más detenimiento los puntos más interesantes y puede relacionar con más facilidad la información extraída debido a que conoce el esquema general del artículo. Podemos observar que:

- El científico ha realizado una serie de acciones en un cierto orden: vistazo general al artículo seguido por una lectura cuidadosa de partes importantes.
- Ha sido necesario conocer el resultado de la acción “vistazo general” para poder realizar las acciones “lectura cuidadosa” siguientes, ya que hacía falta conocer los puntos más interesantes del artículo para ser capaz de realizar estas últimas acciones.
- Las acciones “lectura cuidadosa” se han realizado sobre el artículo, pero sólo en aquellas páginas que cumplen la condición de que pertenecen a partes interesantes.

- Finalmente, se ha llevado a cabo un agrupamiento de los resultados de las acciones realizadas (idea general del artículo e información detallada de los puntos más importantes) para obtener una conclusión final (conocimiento acerca del artículo).

Por otro lado, existen sistemas inteligentes que están compuestos por varios subsistemas, de tal manera que su forma de actuar se asemeja al comportamiento humano. En estos sistemas podemos apreciar la existencia de una estrategia de control que dirige la ejecución de los distintos subsistemas. Un ejemplo de este tipo de sistema inteligente es el modelo NARA (redes neuronales diseñadas sobre una arquitectura de razonamiento aproximado) [Takagi91,Takagi92], el cual es un sistema compuesto por varias redes neuronales que actúan ordenadamente: primero se ejecuta un red cuya salida determina la influencia del resto de las redes en la salida final del sistema.

Algunos ejemplos de razonamiento humano estructurado que tienen cierta semejanza con algunos modelos de Sistemas Inteligentes aparecidos en la literatura son:

- ¿Cómo actúa un médico cuando atiende a un paciente?. Primero, tras escuchar las dolencias del paciente, obtiene una primera idea de lo que padece. A continuación, le realiza una serie de análisis y con sus resultados obtiene un segundo diagnóstico. Si el médico no queda convencido con este segundo diagnóstico, le realizará más análisis y, con los resultados de estos análisis junto con el segundo diagnóstico, obtendrá una nueva idea de lo que padece. Este proceso se repite hasta que el médico consigue un diagnóstico que le convence.

  En la Figura 2.1 se recoge una representación estructurada del comportamiento del médico. Podemos observar las tareas sucesivas realizadas por el médico para obtener un diagnóstico a partir de información anterior y de los resultados de los nuevos análisis. Esta forma de razonar es bastante similar al procesamiento realizado por los controladores difusos jerárquicos presentados en [Raju93]. Este tipo de controlador será especificado más adelante usando el nuevo modelo de representación de conocimiento estructurado.

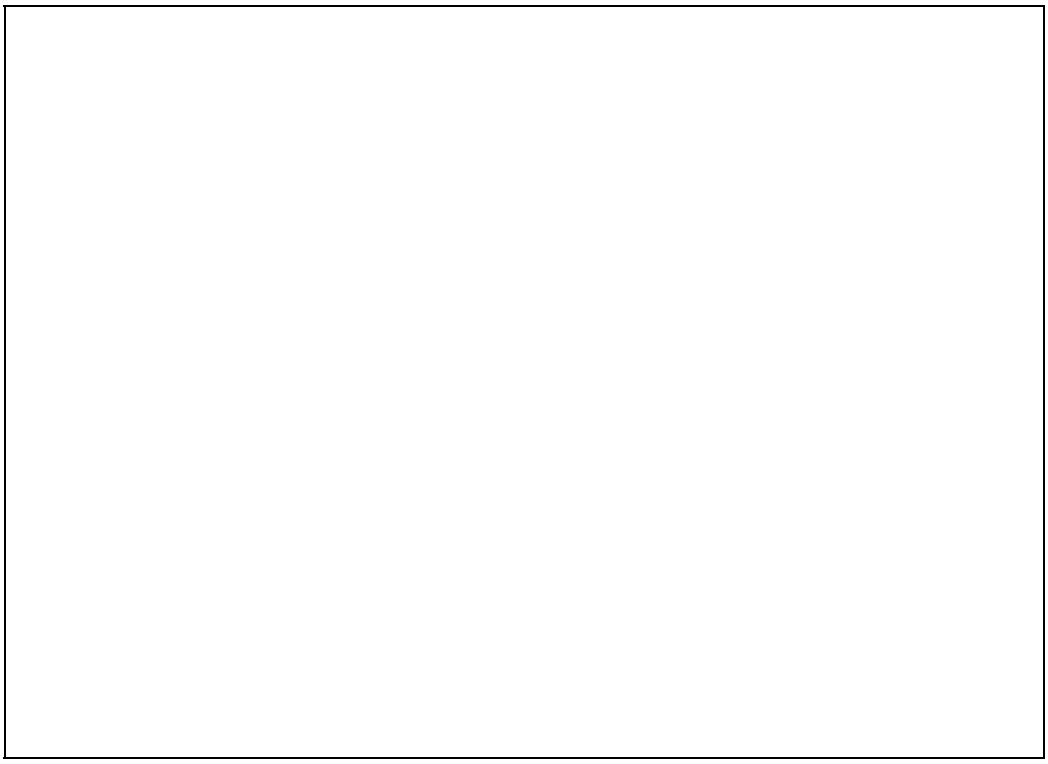


Figura 2.1: Representación estructurada del comportamiento de un médico cuando atiende a un paciente.

- ¿Cómo actúa un mecánico cuando arregla un coche?. Lo primero que hace es detectar un problema en el coche y, a continuación, lo soluciona. Si el coche sigue sin funcionar bien vuelve a detectar otro problema y a solucionarlo. Las acciones de detección de un problema y su arreglo se repiten hasta que el coche funciona correctamente.

  La Figura 2.2 presenta una representación estructurada del comportamiento de un mecánico cuando arregla un coche. Podemos apreciar una secuencia de acciones (detección y arreglo sucesivos) que se van ejecutando en un cierto orden y que, al final, se obtiene un resultado global (coche arreglado) a partir de los resultados de las acciones locales. Esta forma de actuar es bastante parecida al proceso de construcción de la arquitectura de redes neuronales denominada *Correlación en Cascada* [Fahlman90]. Esta arquitectura también será especificada más adelante usando el modelo de representación de conocimiento estructurado.

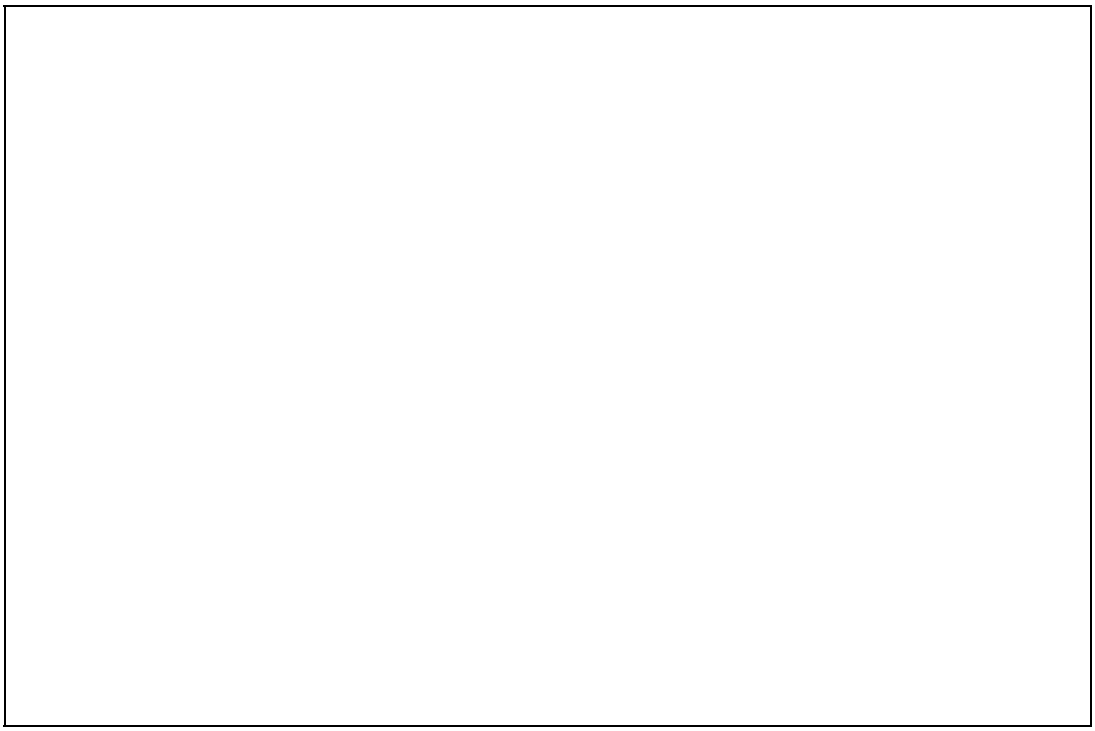


Figura 2.2: Representación estructurada del comportamiento de un mecánico cuando arregla un coche.

- Un conductor de un coche conduce de manera distinta en función de que conduzca en una carretera estrecha con curvas o en una autopista; o en función de que sea de día o de noche, etc. En definitiva, el tipo de conducción depende del contexto que le rodee.

  En la Figura 2.3 se observa un ejemplo de la estructura del comportamiento de un conductor que circula por una carretera con curvas y de noche, detecta estas características en el ambiente que le rodea y circula en consecuencia, o sea, circula lentamente y prestando atención al cambio de luces. Esta forma de razonar es similar a la actuación del modelo de fusión dependiente del contexto [Saffiotti93, Saffiotti95, Saffiotti97], a la de los sistemas basados en reglas difusas tradicionales [Sugeno93] y a la del modelo de mezcla adaptativa de redes neuronales [Jacobs91]. En estos modelos percibimos una secuencia de dos acciones que se ejecutan de manera ordenada: detección del contexto actual seguida de una actuación específica en función del contexto detectado.

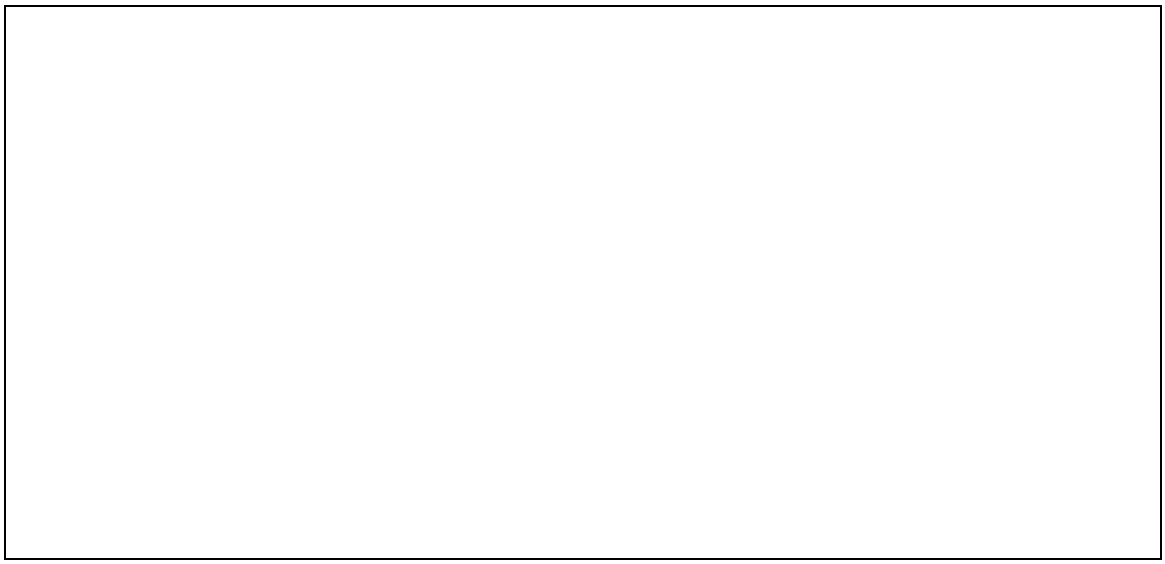


Figura 2.3: Ejemplo de la estructura del comportamiento de un conductor.

Parece razonable representar los sistemas inteligentes intentando abstraer su estructura, o sea, interpretándolos como sistemas constituidos por varios subsistemas que actúan de acuerdo a una estrategia global de ejecución. Para conseguir esto, vamos a definir un modelo de representación del conocimiento que permita especificar las tareas que realiza un sistema, a diferentes niveles de abstracción, y el conocimiento de control que dirige a estas tareas. El propósito de este nuevo modelo (MORSE) consiste en posibilitar la fácil representación de los sistemas inteligentes, siendo suficientemente general como para modelar gran cantidad de ejemplos que usan razonamiento estructurado. Además, debe permitir de forma cómoda la deducción de las propiedades generales de un sistema a partir del análisis de las características de su representación mediante el nuevo modelo. Esto, nos ha llevado a definir un nuevo modelo de representación de conocimiento estructurado, orientado a la consecución de los anteriores objetivos (fácil representación, generalidad, análisis cómodo de características), en vez de utilizar uno de los modelos para representar conocimiento estructurado ya definidos [Chandrasekaran87, Wielinga92, McDermott88, Steels90, Chandrasekaran92]. De esta manera, en este capítulo pretendemos:

a) Definir un modelo de representación del conocimiento que permita recoger las estrategias seguidas en las distintas formas de actuar estructuradas mencionadas anteriormente, o sea, que sea capaz de representar:

   - un conjunto de acciones que se ejecutan en un cierto orden;

   - la necesidad, que se tiene en algunos casos, de conocer la salida de acciones anteriores para ejecutar una;

- una serie de condiciones que se deben cumplir para que se realice una acción y que dependen de las variables de entrada;

- la combinación final de resultados de diversas acciones para obtener un resultado global;

- la independencia en el tipo de las acciones.

b) Modelar a los sistemas inteligentes presentados en diferentes trabajos, tratando de construir modelos generales que representen a varios sistemas particulares.

c) Analizar los modelos con el propósito de conseguir describir las propiedades de los sistemas inteligentes derivadas de su estructura y, de este modo, ser capaz de elegir uno para resolver un problema.

2.2.- MORSE: un modelo de representación modular

En esta sección, definimos un modelo de representación del conocimiento que permite especificar la estructura de un sistema, o sea, los módulos que lo componen y la estrategia de control a la que obedecen estos módulos. A continuación, definiremos un “módulo de representación” que será el componente básico de un “modelo MORSE”. Este modelo será la representación estructurada de un sistema y permitirá describir algunos ejemplos de razonamiento humano estructurado.

Un **módulo de representación** (MOR) es una abstracción de la idea de sistema donde se especifican explícitamente sus variables de entrada, las condiciones de activación definidas sobre estas variables y la salida proporcionada cuando se ejecuta (Figura 2.4). El objetivo perseguido con la definición de un MOR consiste en obtener la capacidad para representar a un sistema de la manera más general posible, olvidándonos de la descripción de gran cantidad de detalles de su funcionamiento interno, destacando la información de entrada, de salida y las condiciones de activación del sistema.

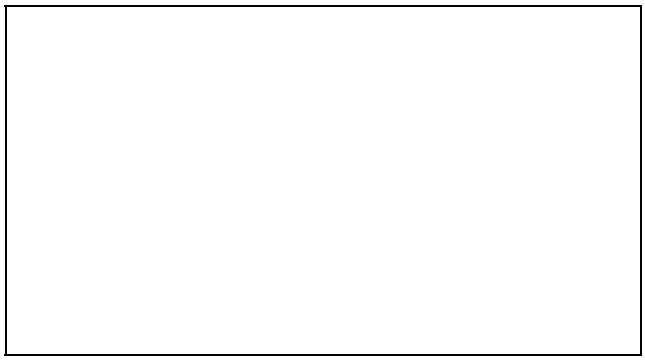


Figura 2.4: Módulo de Representación.

Como ejemplo de MOR podemos imaginar a una red neuronal artificial multicapa hacia adelante [Lippmann87] con dos entradas x_1, x_2 pertenecientes a [0,1] y una salida Y. Su representación gráfica se ilustra en la Figura 2.5, pudiendo apreciar que la condición para que se dispare el sistema consiste en que la entrada pertenezca al cuadrante [0,1]x[0,1].

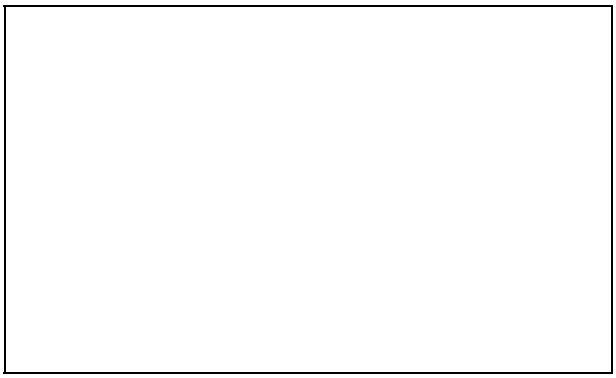


Figura 2.5: Red neuronal multicapa hacia adelante representada como un módulo de representación.

A partir del módulo básico MOR y como abstracción del concepto de sistema estructurado definimos un **modelo MORSE** (Módulos de Representación para un Sistema Estructurado) [Castro99b] (Figura 2.6) como una estructura compuesta por:

* una sucesión de niveles con uno o más módulos de representación en cada nivel y,

* un módulo de combinación que proporciona la salida global del sistema a partir de las salidas de los módulos de representación ejecutados previamente y de las entradas al sistema, definiendo implícitamente la estrategia de ejecución de los módulos del modelo MORSE;

donde :

- Se ejecutan antes los niveles superiores que los inferiores y, finalmente, se ejecuta el módulo de combinación.

- No existe orden en la ejecución de los módulos de representación de un mismo nivel.

- Las entradas a una estructura MORSE y las salidas de los módulos de representación ya ejecutados en niveles anteriores están disponibles para ser usadas por cualquier módulo que vaya a dispararse en el nivel actual de ejecución o, por el módulo de combinación.

- Los módulos de representación pueden ser de distinto tipo.

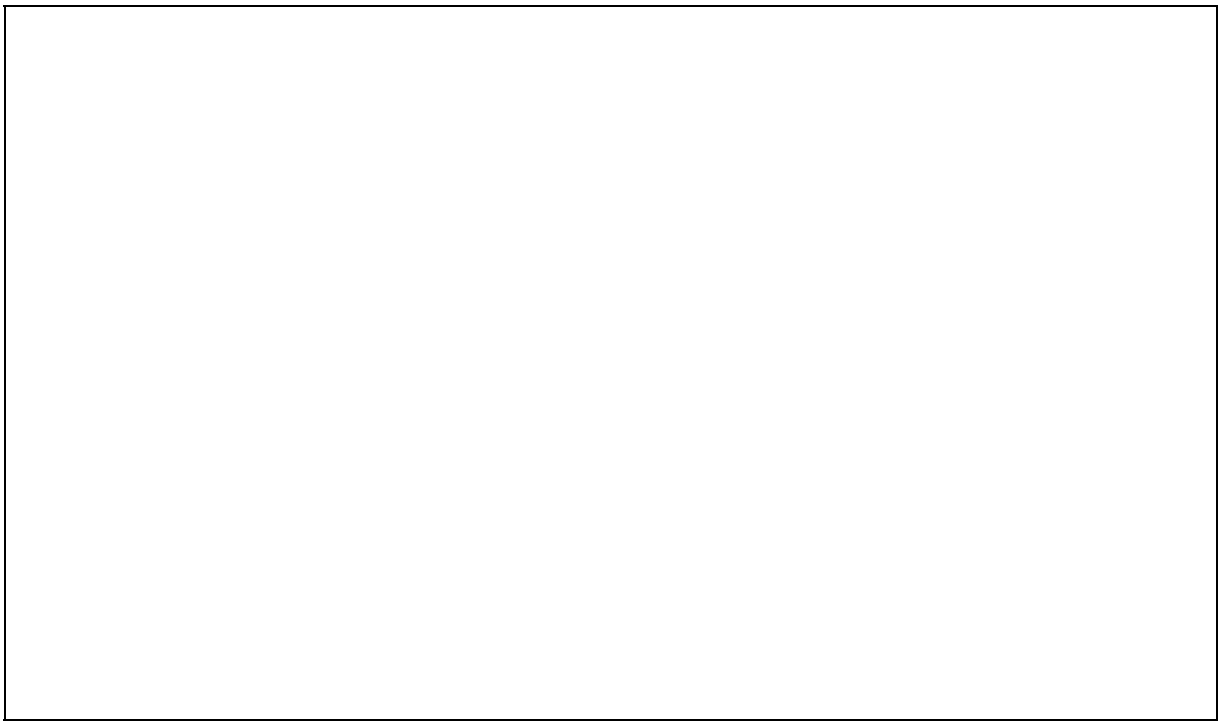


Figura 2.6: Modelo MORSE (Módulos de Representación para un Sistema Estructurado).

En la Figura 2.7 se modela la forma de actuar de un científico por medio de una estructura MORSE. En este ejemplo podemos apreciar todos los elementos de una estructura MORSE:

- Aparecen dos niveles de módulos de representación, el primero compuesto por un solo MOR (“Vistazo general”) y el segundo compuesto por dos (MOR’s de “Lectura cuidadosa”).

- Contiene a un módulo de combinación que proporciona la salida global del sistema (*Conocimiento acerca del artículo*), haciendo uso, únicamente, de las salidas proporcionadas por los MOR’s del sistema, realizando un agrupamiento de ellas para obtener la salida deseada. Podemos observar que el módulo de combinación pide primero la idea general del artículo y, a continuación, requiere las ideas específicas de cada sección, definiendo, de esta manera, una estrategia de ejecución para los módulos de este modelo MORSE. En la siguiente sección, presentamos otros ejemplos de estructuras MORSE, pudiendo apreciar diferentes operaciones en el módulo de combinación.

- El MOR “Vistazo general” del nivel superior se debe ejecutar antes que los MOR’s “Lectura cuidadosa” del segundo nivel y, tras todos ellos, se podrá ejecutar el módulo de combinación.

- Los MOR’s “Lectura cuidadosa” del segundo nivel pueden utilizar la salida del MOR del primer nivel (*Idea general*) y, todas las salidas de los MOR’s de la estructura MORSE (idea general e ideas específicas de secciones) las puede usar el módulo de combinación.

- Hay MOR’s de dos tipos diferentes: MOR “Vistazo general” y MOR’s “Lectura cuidadosa”.

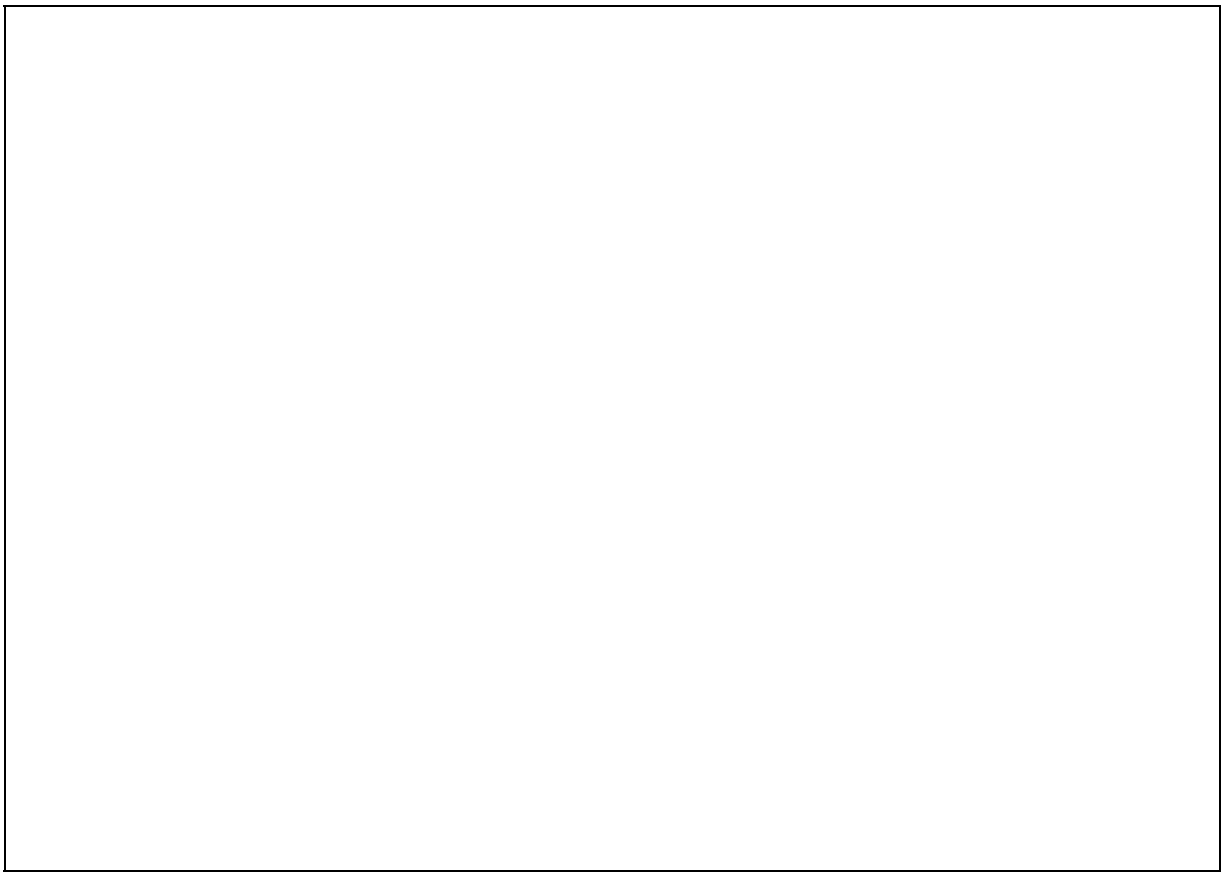


Figura 2.7: Proceso seguido por un científico cuando lee un artículo, modelada mediante una estructura MORSE.

2.3.- Generalidad del modelo

En esta sección, vamos a discutir como pueden modelarse por medio de MORSE, diferentes sistemas inteligentes presentados en varios trabajos y que se han usado para resolver diversos problemas tales como clasificación, control, aproximación, robótica o razonamiento. El objetivo es apreciar la capacidad de descripción general de este esquema de representación del conocimiento.

2.3.1.-Razonamiento no monótono

En el razonamiento no monótono [Lukaszewicz90] existen reglas generales (reglas por defecto) que son las que habitualmente se disparan para obtener una salida a partir de unas entradas concretas, y un conjunto de reglas más particulares que reemplazan a las por defecto cuando las entradas corresponden a ciertas situaciones especiales (excepciones). Esta forma de razonar se ilustra mediante un modelo MORSE en la Figura 2.8. Este sistema tiene dos niveles, el primero está compuesto

por un solo módulo de representación que abstrae la actuación de todas las reglas por defecto y el segundo está compuesto por varios módulos de representación, conteniendo cada uno las reglas particulares que se activan cuando se presenta una excepción.

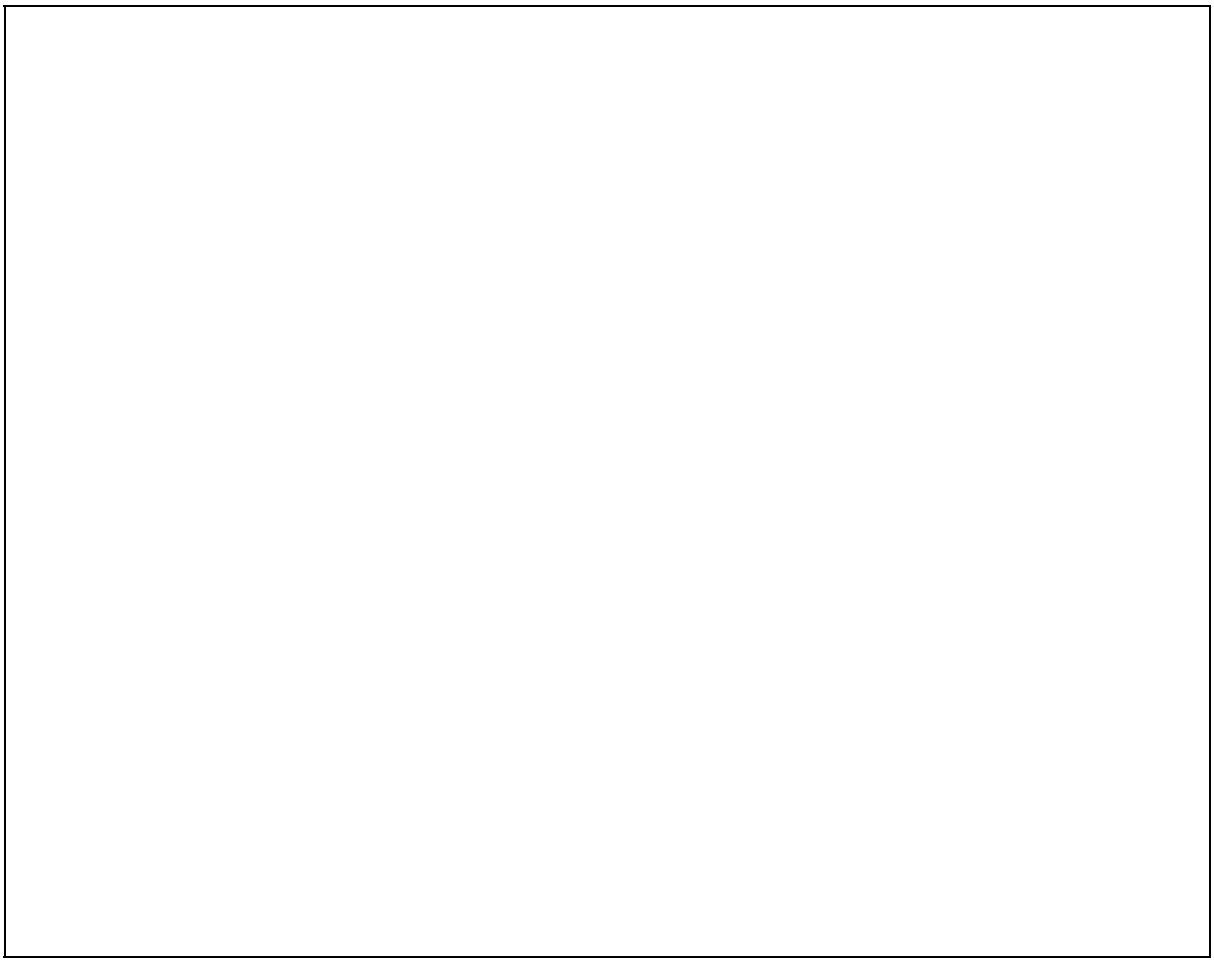


Figura 2.8: Razonamiento no monótono representado con una estructura MORSE.

En el razonamiento no monótono difuso [Castro98a], cuando se presenta una entrada puede ocurrir que dos o más reglas originen un conflicto. En este caso, el

conflicto se resuelve realizando un promedio de los resultados problemáticos. Por ejemplo, dos reglas razonables para conducir un coche podrían ser :

R1 : *Si* obstáculo cerca *entonces* frenar mucho.
R2 : *Si* curva cerrada *entonces* no frenar o frenar muy poco.

Estas reglas entran en conflicto cuando el coche encuentra un obstáculo en una curva cerrada. En este caso, un conductor actuaría de acuerdo a un compromiso (promedio) entre los resultados de R1 y de R2. El procesamiento de este tipo de razonamiento se presenta mediante un modelo MORSE en la Figura 2.9, el cual también tiene dos niveles, el primero de los cuales está constituido por un módulo de representación con las reglas del sistema, mientras que el segundo nivel contiene varios módulos con los promedios que se utilizan para resolver cada posible conflicto. Finalmente, el módulo de combinación realiza un consenso entre las salidas producidas por las reglas del sistema y las salidas proporcionadas por los módulos del segundo nivel (compromiso entre el resultado de reglas conflictivas).

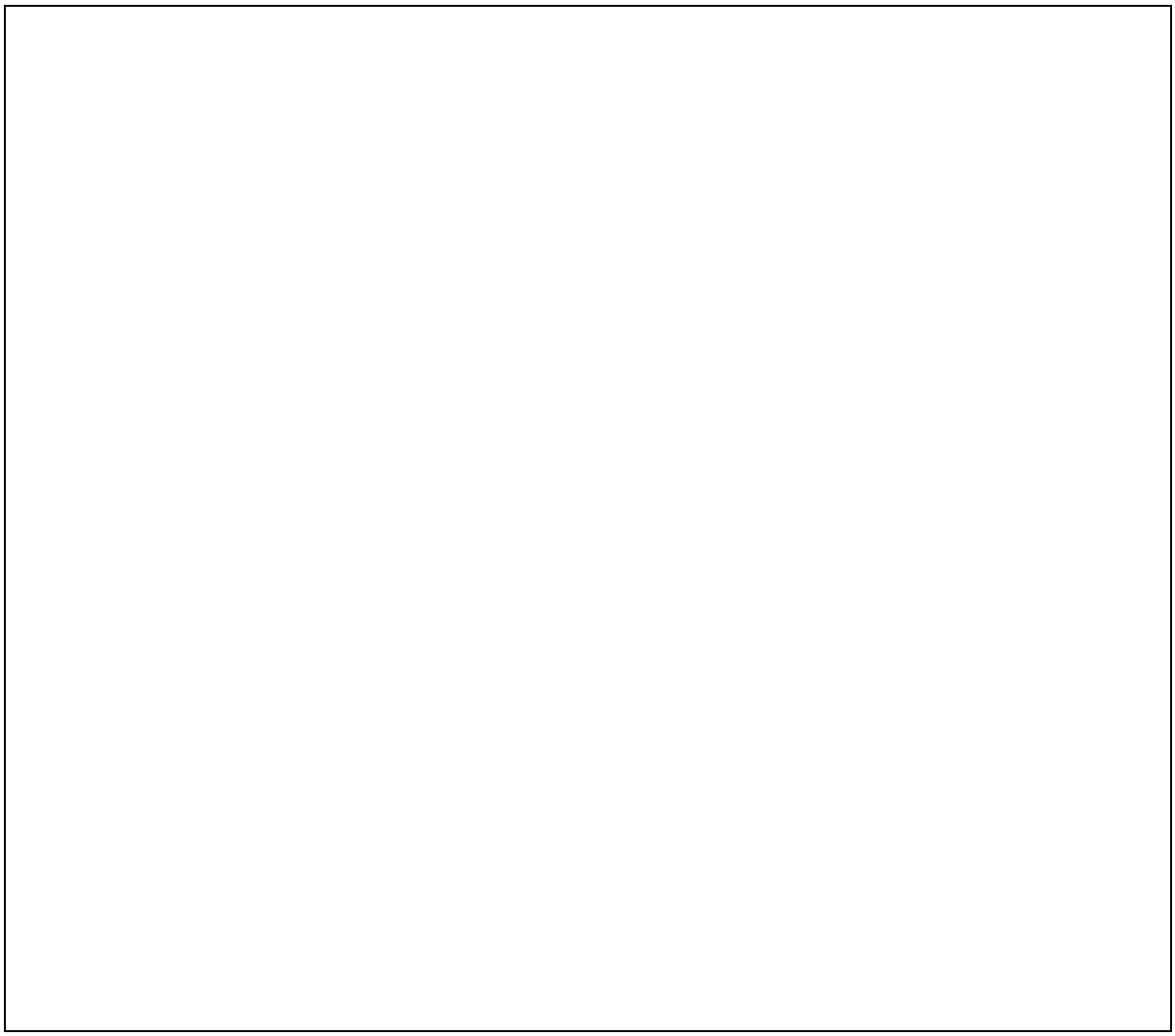


Figura 2.9: Razonamiento no monótono difuso representado con un modelo MORSE.

2.3.2.-Arquitectura Correlación en Cascada

Correlación en cascada [Fahlman90] es una arquitectura para redes neuronales artificiales. Su representación mediante un modelo MORSE se ilustra en la Figura 2.10, que tiene varios niveles con un solo módulo por nivel, donde cada módulo de representación y el módulo de combinación son perceptrones [Minsky69, Wasserman89]. Cada MOR de un nivel procesa la unión de todas las salidas de los MOR's de niveles superiores junto con las entradas globales del modelo MORSE. Finalmente, el perceptrón correspondiente al módulo de combinación procesa el

conjunto de todas las salidas de todos los módulos de representación del modelo MORSE y a las entradas globales, produciendo así la salida global del sistema.

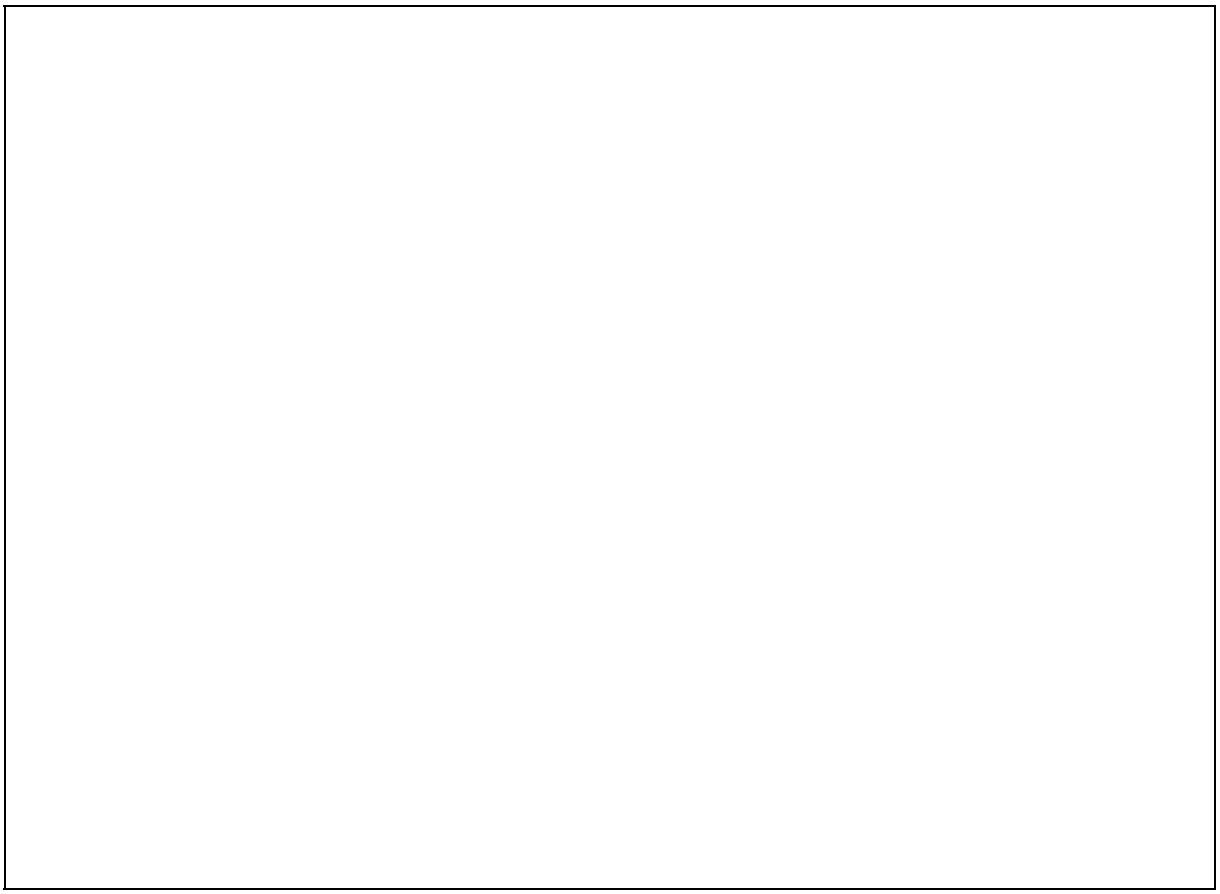


Figura 2.10: Arquitectura Correlación en Cascada representada con un modelo MORSE.

2.3.3.-Controladores difusos jerárquicos

La estructura del control difuso jerárquico [Raju93] está formada por varios niveles de controladores, donde las variables más significativas para ayudar en la consecución de una solución se eligen como entradas al controlador difuso del primer nivel, las siguientes variables en importancia se eligen como entradas para el controlador del segundo nivel y así sucesivamente. Además, todos los controladores, menos el del primer nivel, también procesan la salida producida por el controlador difuso del nivel anterior. La modelización de este controlador difuso jerárquico mediante una estructura MORSE se ilustra en la Figura 2.11. En cada nivel de este sistema hay sólo un módulo de representación (controlador difuso) que afina la salida producida por el módulo del nivel anterior. Finalmente, la salida producida por el sistema es igual a la salida del módulo de nivel más bajo.

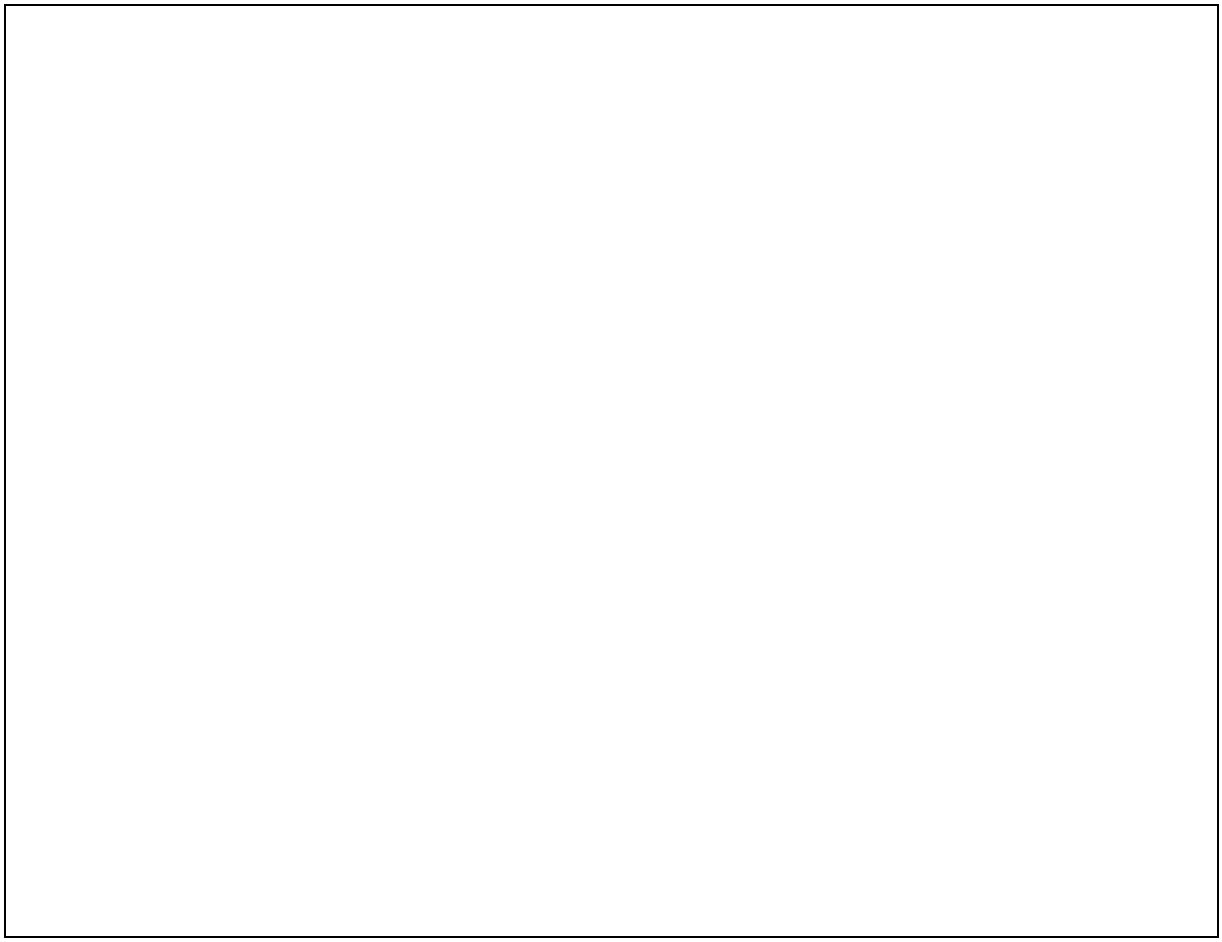


Figura 2.11: Controlador difuso jerárquico representado con un modelo MORSE.

2.3.4.-Redes Neuronales

Una red neuronal [Lippmann87] está compuesta por un conjunto de elementos de procesamiento o perceptrones, operando en paralelo y ordenados por capas. Un ejemplo de red neuronal multicapa se ilustra en la Figura 2.12 y su representación mediante un modelo MORSE en la figura 2.13. Los módulos de representación de un mismo nivel corresponden con los perceptrones de una misma capa en una red neuronal. El módulo de combinación devuelve las salidas de los módulos del último nivel que corresponden con los perceptrones de la última capa de la red neuronal multicapa.

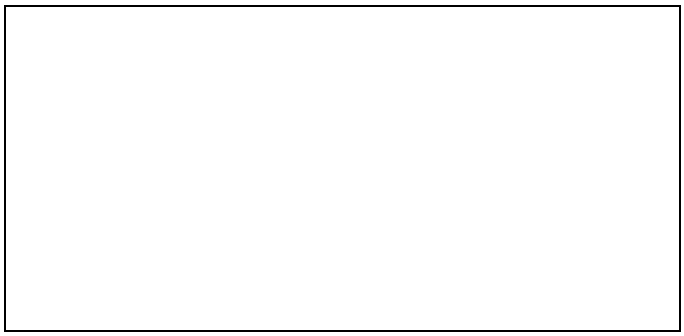


Figura 2.12: Ejemplo de Red Neuronal multicapa.

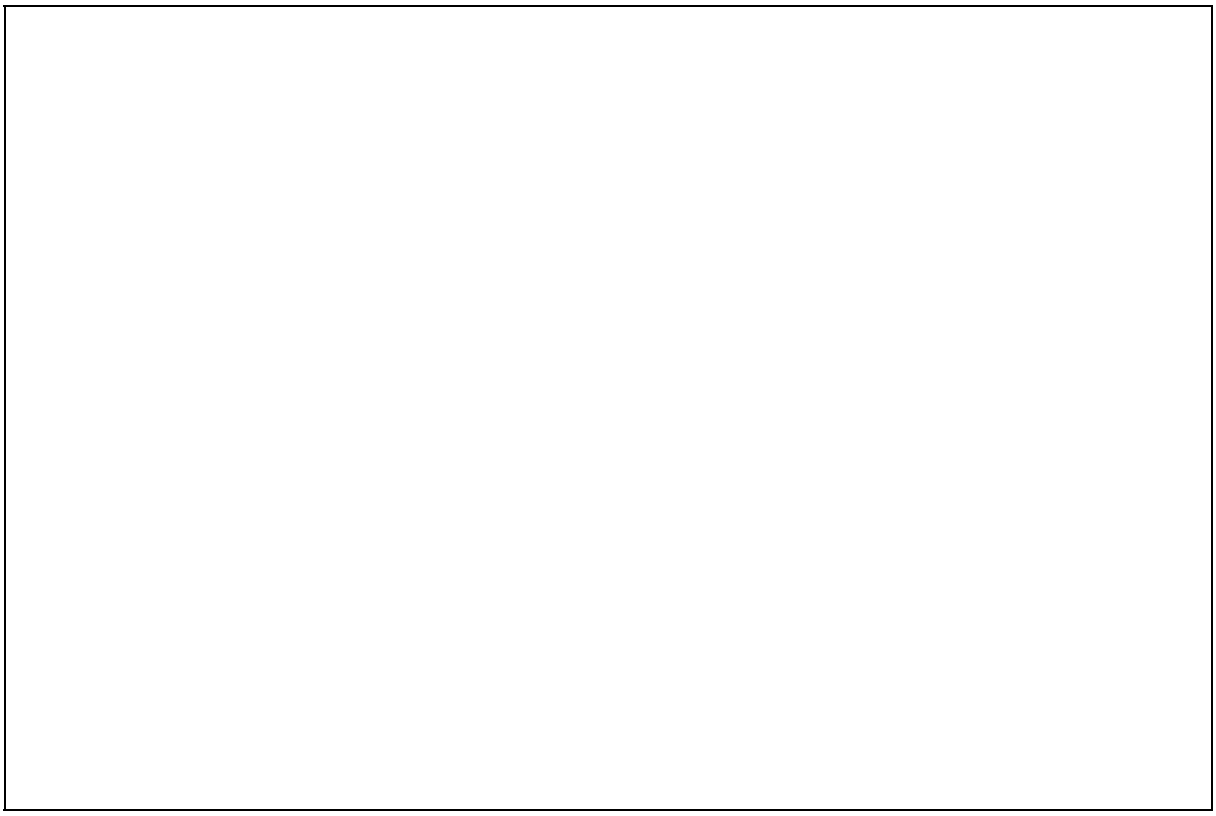


Figura 2.13: Red Neuronal representada con un modelo MORSE.

2.3.5.-Árboles de Decisión

Un árbol de decisión modelado mediante una estructura MORSE se ilustra en la Figura 2.14. Este sistema puede estar compuesto por varios niveles con uno o más módulos de representación por nivel. Los módulos de representación pueden ser de dos tipos:

a) módulos *decisión* que aportan una salida para saber qué módulos de representación se activan en el siguiente nivel o,

b) módulos *dar resultado* cuya salida se puede usar como salida global del sistema.

Finalmente, el módulo de combinación proporciona la salida de un módulo *dar resultado* como salida global en función de las salidas de los módulos *decisión*. Particularizando los módulos de representación de este sistema a herramientas concretas obtenemos modelos presentados en distintos trabajos, por ejemplo :

- si los módulos *dar resultado* de la Figura 2.14 devuelven un valor de clasificación y los módulos *decisión* consisten en una consulta sobre una variable, obtenemos el modelo presentado en [Quinlan86]. El algoritmo de construcción de este modelo se denomina ID3 y utiliza una medida de entropía [Abramson63] para saber como de informativa es una variable a la hora de diseñar los módulos *decisión*. Este modelo se usa para resolver problemas de clasificación;

- si los módulos *dar resultado* devuelven un valor constante y los módulos *decisión* son una consulta sobre una variable, obtenemos el modelo presentado en [Breiman84] . Este modelo se construye con la misma filosofía que el modelo presentado en el párrafo anterior, la diferencia radica en que se aplica a problemas de aproximación de funciones;

- si los módulos *dar resultado* son redes neuronales multicapa y los módulos *decisión* de la Figura 2.14 son perceptrones binarios, obtenemos el modelo presentado en [Chiang94] . En el proceso de construcción de este modelo, los perceptrones de los módulos “decisión” se diseñan usando una medida de *Correlación* sobre el error actual del sistema;

- si los módulos *dar resultado* y *decisión* son perceptrones binarios, obtenemos el modelo presentado en [Sirat90] . Este modelo se usa para resolver problemas de clasificación. Su proceso de construcción consiste en ir entrenando perceptrones binarios con el algoritmo *Pocket* [Gallant86] (Apéndice IV) hasta conseguir una partición del espacio de entrada en regiones que contengan puntos de la misma clase, teniendo en cuenta que un perceptrón define un hiperplano que corta el espacio de entrada en dos regiones;

- si los módulos *dar resultado* devuelven un valor de clasificación y los módulos *decisión* son una combinación lineal de las entradas, obtenemos el modelo presentado en [Park90] . Este modelo se usa para resolver problemas de clasificación y en su proceso de construcción los módulos *decisión* se optimizan para cortar el mayor número posible de enlaces Tomek [Tomek76], teniendo en cuenta que un enlace Tomek nos indica la frontera entre dos puntos de distinta clase;

- si los módulos *dar resultado* están compuestos por un clasificador simple tal como un perceptrón binario, una función gaussiana o incluso una pequeña red neuronal, y los módulos *decisión* son perceptrones, entonces obtenemos el modelo presentado en [Alché94]. El paso básico de construcción de este modelo se denomina *Trio Learning* y consiste en ajustar simultáneamente, mediante un esquema de aprendizaje supervisado, los parámetros de un perceptrón que divide un espacio en dos regiones y los de los clasificadores que actúan en cada una de estas regiones.

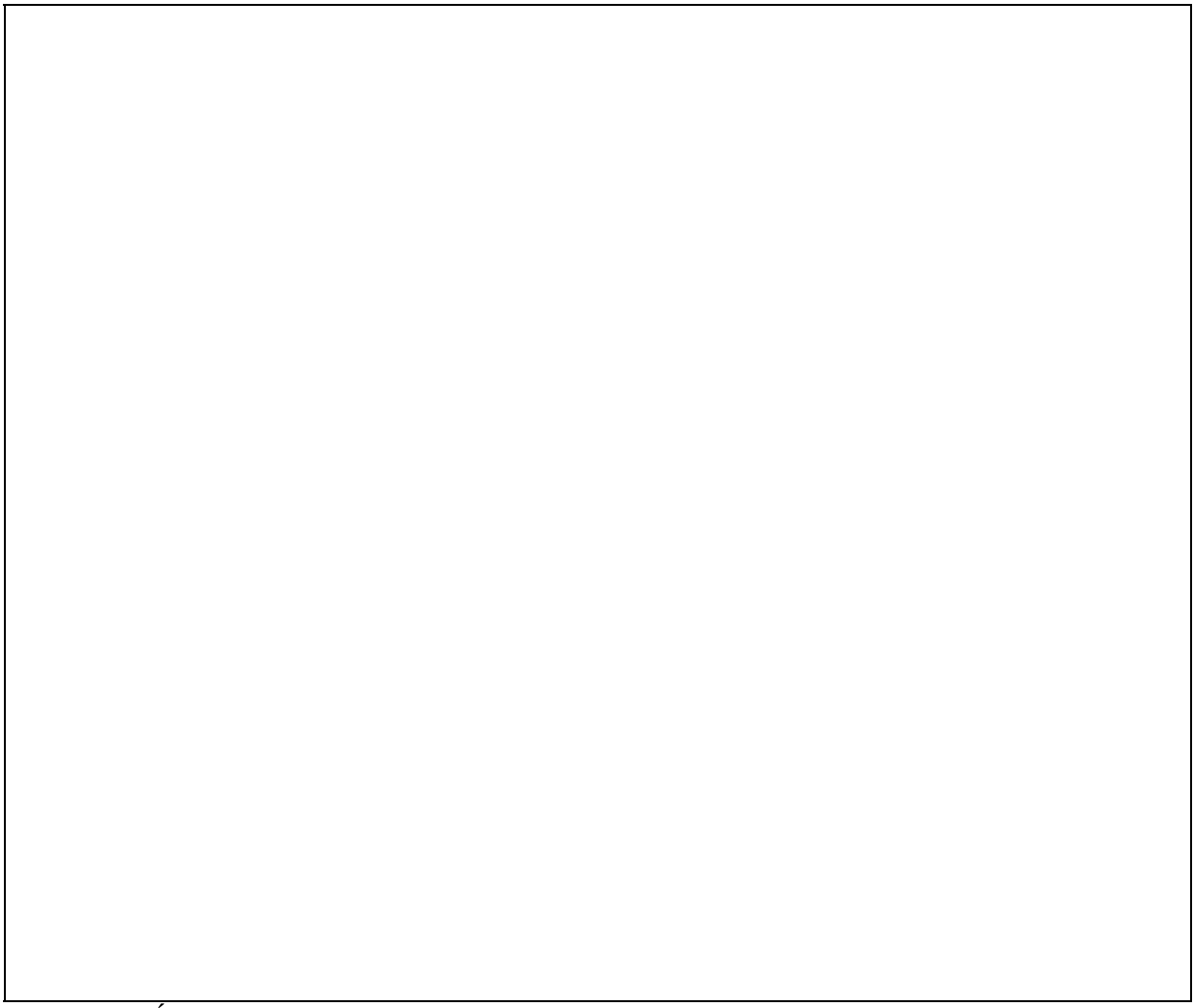


Figura 2.14: Árbol de Decisión representado con un modelo MORSE.

2.3.6.- Sistema MORSE Contextual : Sistemas basados en reglas difusas, fusión dependiente del contexto, mezcla adaptativa de redes neuronales y NARA

En esta sección presentamos un modelo MORSE denominado *Sistema MORSE Contextual*, que tiene la peculiaridad de que en función de la composición de sus módulos de representación y de su forma de construcción da lugar a diferentes tipos particulares de Sistemas Inteligentes. El Sistema MORSE Contextual (Figura 2.15) tiene dos niveles: en el primer nivel, un módulo se encarga de determinar el contexto al que pertenece la entrada actual al sistema y, los módulos del segundo nivel se encargan de producir una salida para cada contexto posible de las entradas. Finalmente, el módulo de combinación producirá la salida global del sistema en función de las salidas de los módulos del segundo nivel y del contexto al que pertenece cada entrada.

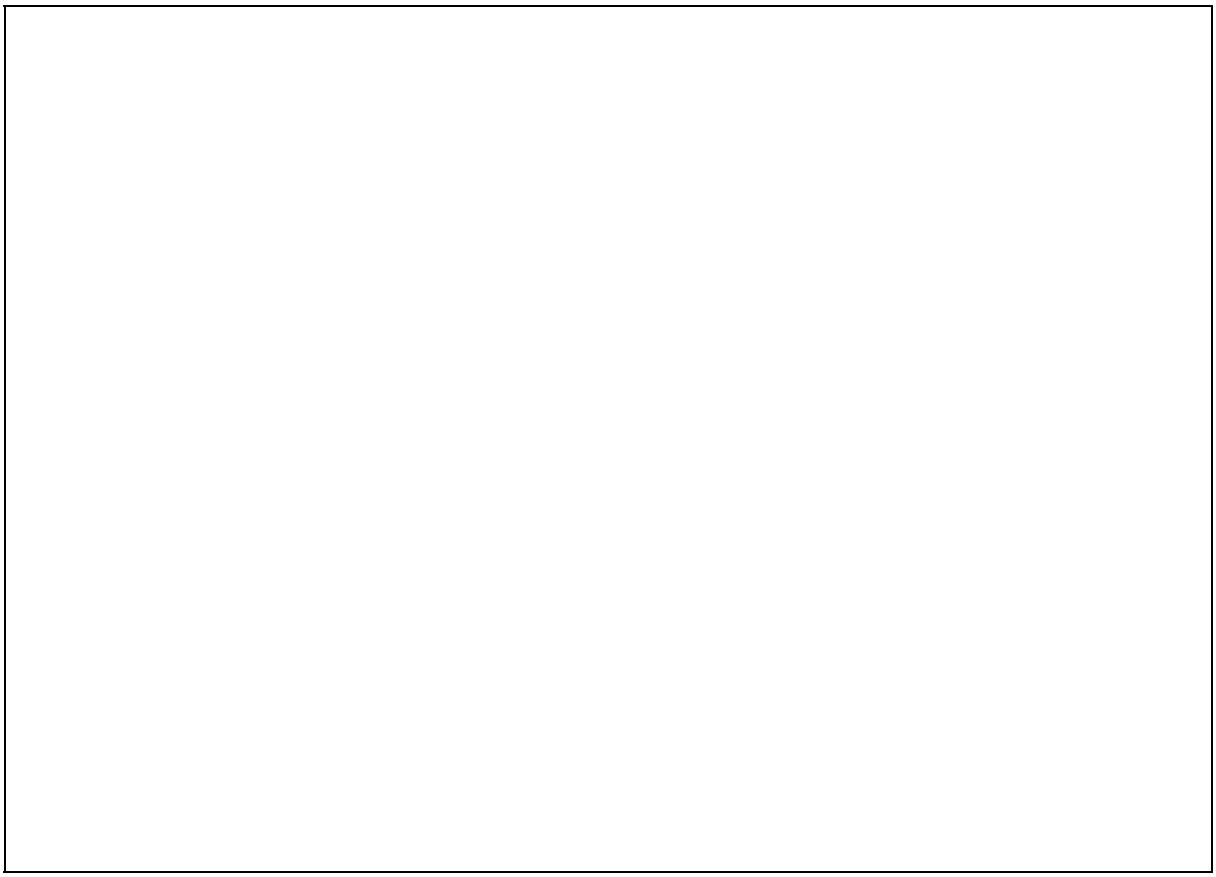


Figura 2.15: Sistema MORSE Contextual.

En función de la composición de los módulos de este sistema obtenemos distintos tipos de Sistemas Inteligentes:

- si todos los módulos están compuestos por controladores difusos obtenemos el modelo de fusión dependiente del contexto [Saffiotti93, Saffiotti95, Saffiotti97]. Este modelo consiste en la implementación de un sistema con

varios comportamientos por medio de controladores difusos, cuya activación e influencia en el resultado final del sistema dependerá de a) el contexto actual que se determina usando reglas difusas, b) de un operador de combinación difusa y c) de un operador de defuzzificación. En [Saffiotti97], este modelo se usa para controlar el movimiento de un robot autónomo que debe actuar de manera diferente dependiendo del contexto en el que se encuentre;

- si el módulo del primer nivel está compuesto por un técnica de agrupamiento [Jain88] y los módulos del segundo nivel por reglas difusas [Klir95], obtenemos los modelos conocidos como sistemas basados en reglas difusas [Chiu94, Sugeno93, Sun94, Takagi85, Yager94]. Estos modelos tratan de describir el comportamiento de un sistema mediante reglas si-entonces difusas, obtenidas a partir de una partición difusa del espacio de entrada y de la asociación de una relación entrada-salida a cada subespacio difuso generado. La activación de estas reglas lingüísticas se realizará mediante inferencia difusa [Klir95] dando lugar al proceso que se denomina *razonamiento aproximado*;

- si los módulos del sistema están compuestos por redes neuronales obtenemos el modelo de mezcla adaptativa de redes neuronales [Jacobs91] y el modelo NARA (redes neuronales diseñadas sobre una arquitectura de razonamiento aproximado) [Takagi91, Takagi92]. La diferencia entre ellos radica en la forma de entrenar a las redes neuronales que componen el sistema. En [Jacobs91] todas las redes del modelo se entrenan simultáneamente usando el algoritmo Backpropagation [Rumelhart86] sobre una función de error basada en probabilidades gaussianas; mientras que en el modelo NARA [Takagi91, Takagi92], primero se entrena la red que determina el contexto con el fin de aproximar el comportamiento de una partición difusa previamente establecida y, a continuación, se entrenan el resto de las redes del sistema para aproximar el comportamiento global del sistema.

2.4.-Análisis de los esquemas MORSE

En esta sección utilizamos tres características de los modelos MORSE para analizar los sistemas inteligentes en base a su estructura. Éstas son:

1) Número de niveles de un modelo MORSE.
2) Número de módulos en cada nivel.
3) Salidas de módulos de representación (ejecutados en niveles anteriores) que son usadas por cada módulo del sistema.

Observando los valores particulares para estas características ofrecidos por una estructura MORSE que modela a un sistema inteligente concreto, podemos apreciar sus propiedades generales. Este hecho facilita la elección de este sistema inteligente para resolver un problema, esto es, si tenemos dudas acerca de que sistema inteligente usar para solucionar un problema, podemos seguir la siguiente metodología:

1.- Modelar los sistemas disponibles mediante estructuras MORSE.

2.- Apreciar los valores particulares para las características anteriores que ofrece cada estructura MORSE.

3.- Deducir las propiedades generales de cada sistema inteligente derivadas de su estructura.

4.- Finalmente, elegir aquel sistema cuyas propiedades se amolden mejor al tipo de solución que estamos buscando.

2.4.1.-Características de una estructura MORSE

A continuación, presentamos las características de una estructura MORSE junto con una explicación acerca de la propiedad general que cada valor particular de una característica lleva asociado:

1) **Número de niveles de un modelo MORSE**: existen Sistemas Inteligentes cuya representación mediante un modelo MORSE sólo *necesita dos niveles de módulos de representación*, por ejemplo, el modelo MORSE que representa al razonamiento no monótono o el sistema MORSE Contextual. Por otro lado, hay otras representaciones que *necesitan varios niveles*, por ejemplo, los modelos MORSE que representan a la correlación en cascada, a los controladores difusos jerárquicos, a los árboles de decisión o a las redes neuronales. La principal diferencia entre un sistema

que sólo *necesita dos niveles* y otro que *necesita varios niveles* radica en la facilidad para conocer el procesamiento realizado por el sistema, lo que conlleva mayor o menor dificultad a la hora de extraer conocimiento de éste. Por ejemplo, es más fácil comprender el funcionamiento de un sistema MORSE Contextual donde se tiene identificado un módulo que determina el contexto y los módulos que corresponden a comportamientos particulares, que comprender la semántica de cada módulo en una arquitectura correlación en cascada, en un controlador difuso jerárquico, en un árbol de decisión o en una red neuronal. Cuanto más niveles tenga el modelo MORSE que representa a un sistema, más difícil será la comprensión de la operación de este sistema. En el caso de las redes neuronales multicapa, es posible comprender el procesamiento realizado por una red neuronal con una sola capa oculta (modelo MORSE con dos niveles) [Benítez97] frente a la dificultad de extraer conocimiento de una red con más capas ocultas. Por otro lado, cuanto mayor sea el número de niveles de un modelo MORSE mayor será la posibilidad de ajuste y, por lo tanto, de conseguir una buena solución.

En definitiva, cuando queramos diseñar un Sistema Inteligente para resolver un problema, en función de la necesidad de comprender su funcionamiento frente a la exactitud de su solución, un sistema inteligente u otro será elegido teniendo en cuenta el número de niveles de su representación mediante una estructura MORSE.

2) **Número de módulos en cada nivel**: hay modelos MORSE que *tienen un solo módulo de representación en cada nivel*, por ejemplo, los modelos MORSE correspondientes a la correlación en cascada y a los controladores difusos jerárquicos; frente a otros que usan *varios módulos en sus niveles*, por ejemplo, los modelos MORSE de los árboles de decisión o de las redes neuronales. La utilización de varios módulos en un mismo nivel implica la posibilidad de realizar varios procesamientos independientes en un mismo paso. Este hecho parece ventajoso frente a la realización de un único procesamiento (uso de un solo módulo por nivel). Sin embargo, para usar varios módulos en un mismo nivel se tienen que resolver dos problemas:

- **¿Cuál es el número de procesamientos necesarios en un nivel?**. Este problema, en el ámbito de las redes neuronales multicapa, se traduce en conocer cuántos perceptrones son necesarios en una capa. A raíz de este problema surgió la arquitectura de redes neuronales denominada *Correlación en Cascada*, que sólo utiliza un perceptrón en cada capa.

- **¿Cuáles son los procesamientos a realizar en un mismo nivel?**. En el campo de las redes neuronales, los algoritmos de entrenamiento [Rumelhart86, Delgado97,

Delgado99] solucionan este problema, siendo ésta una de las mayores ventajas de este tipo de sistema inteligente. En el ámbito de los árboles de decisión, el espacio de entrada del problema se va dividiendo en regiones y, de este modo, los procesamientos a realizar en un nivel consisten en resolver el problema global en cada región del espacio de entrada independientemente (técnica de diseño de algoritmos denominada *Divide-y-Vencerás* [Brassard88]). Este mecanismo es posible gracias a que se dispone de herramientas que consiguen una división bastante óptima del espacio de entrada, o sea, dividen el problema de la mejor manera posible para maximizar cierto criterio de bondad, como puede ser reducir la entropía de cada región o lograr distinguir una región que apenas tiene error. Ejemplos de herramientas de división utilizadas son la medida de entropía para seleccionar la variable más informativa [Abramson63], el algoritmo de entrenamiento de perceptrones [Gallant86, Minsky69], la Correlación de error, etc.

Por lo tanto, cuando busquemos un Sistema Inteligente para resolver un problema, si se conocen métodos para establecer varios procesamientos en un mismo paso, podremos elegir un Sistema Inteligente que tenga *varios módulos por nivel en su representación MORSE*. Por otro lado, si no se conocen estos métodos o no es posible su aplicación, elegiremos un Sistema Inteligente que tenga *un solo módulo por nivel en su modelización con MORSE*.

3) **Salidas de módulos de representación (ejecutados en niveles anteriores) que son usadas por cada módulo del sistema**: hay modelos MORSE cuyos módulos *emplean las salidas de todos los módulos disparados en niveles anteriores*, por ejemplo, el modelo MORSE que representa a la arquitectura Correlación en Cascada. Los módulos de otras estructuras MORSE sólo *emplean las salidas proporcionadas por los módulos disparados en su nivel anterior*, por ejemplo, los modelos MORSE que representan a los controladores difusos jerárquicos y a las redes neuronales. Otros módulos sólo emplean un *subconjunto de las salidas proporcionadas por los módulos de su nivel anterior*, por ejemplo, los módulos del modelo MORSE que representa a los árboles de decisión.

El hecho de que un módulo de una estructura MORSE pueda *aprovechar todas las salidas proporcionadas en niveles anteriores*, significa que este módulo puede ajustar los resultados de cualquier procesamiento realizado anteriormente para obtener el máximo beneficio en su propio procesamiento. Esto es bastante ventajoso a la hora de diseñar un Sistema Inteligente, pero presenta el problema de tener un gran coste computacional. Por ejemplo, es bastante costoso entrenar un perceptrón en la arquitectura Correlación en Cascada que esté en un nivel muy bajo debido a que tiene

que procesar gran cantidad de variables de entrada (entradas globales del problema y salidas de perceptrones de niveles superiores). Por otra parte, los módulos de un modelo MORSE que *sólo emplean las salidas proporcionadas en su nivel anterior*, sólo pueden ajustar las salidas de los procesamientos realizados en el paso previo para obtener el máximo rendimiento a su funcionamiento. Con esto se pierde la ventaja de poder ajustar cualquier resultado anterior pero reduce el coste computacional. Parece deseable llegar a un compromiso entre el aprovechamiento de las salidas de niveles anteriores en un modelo MORSE y el coste de su diseño y ejecución.

Un modelo MORSE con módulos que *sólo emplean un subconjunto de las salidas proporcionadas en su nivel anterior*, sólo ajusta una parte del procesamiento realizado en un paso por el sistema, olvidándose del resto. Esto implica un ajuste más exacto debido a la localidad del área de actuación, pero puede traer dos problemas:

- Este ajuste puede implicar que el sistema necesite más recursos que otro que realice un ajuste teniendo en cuenta todo el procesamiento del sistema en un paso. Por ejemplo, puede ocurrir que la actuación realizada por un nodo de un árbol de decisión en una región también sea útil en otra región, sin embargo, con la filosofía de diseño de los árboles de decisión (modelos MORSE con módulos que aprovechan un subconjunto de las salidas proporcionadas en el nivel anterior) se necesitarían dos nodos en el árbol, uno por cada región.

- Al realizar el ajuste sobre una parte del procesamiento, el funcionamiento del sistema se puede especializar en el tratamiento de un ámbito local del problema. Por ejemplo, los árboles de decisión suelen fallar en los ejemplos de prueba debido a que se especializan en la clasificación de los ejemplos de entrenamiento en regiones locales. Por otro lado, a las redes neuronales (modelos MORSE con módulos que aprovechan todas las salidas proporcionadas en un nivel anterior) les cuesta más aprender los ejemplos de entrenamiento que a los árboles de decisión pero, sin embargo, clasifican mejor los ejemplos de prueba (buena generalización).

Por lo tanto, a la hora de resolver un problema, elegiremos un Sistema Inteligente cuya representación MORSE tenga módulos que *aprovechan todas las salidas producidas en el procesamiento de un paso del sistema* o *un subconjunto de ellas*, en función de la necesidad de generalización y de la importancia del ahorro de recursos frente a la exactitud de los resultados.

2.4.2-Análisis de algunos sistemas inteligentes representados con MORSE

En la Figura 2.16, se ilustran los sistemas inteligentes representados con MORSE en las secciones anteriores. La organización de la figura se basa en los valores particulares para las características de un modelo MORSE que ofrecen las representaciones de estos sistemas inteligentes. Estos valores son:

- La estructura MORSE que modela el razonamiento no monótono y el Sistema MORSE Contextual tienen sólo dos niveles, con uno o más módulos por nivel y los módulos del segundo nivel usan la salida proporcionada por el módulo del nivel anterior. La diferencia entre ellos radica en que los módulos del primer nivel del razonamiento no monótono producen una salida útil como salida global además de información acerca de la activación de los módulos del siguiente nivel, mientras que los módulos del primer nivel del Sistema MORSE Contextual sólo producen información para activar los módulos del siguiente nivel.

- La estructura MORSE que modela a la Correlación en Cascada tiene uno o más niveles, con un módulo por nivel y sus módulos usan las salidas producidas por todos los módulos de niveles anteriores.

- La estructura MORSE que modela a los controladores difusos jerárquicos tiene uno o más niveles, con un módulo por nivel y sus módulos sólo usan la salida proporcionada por el módulo del nivel anterior.

- La estructura MORSE que modela a los árboles de decisión tiene uno o más niveles, con varios módulos por nivel y sus módulos sólo usan la salida proporcionada por uno de los módulos del nivel anterior.

- La estructura MORSE que modela a las redes neuronales tiene uno o más niveles, con varios módulos por nivel y sus módulos usan las salidas proporcionadas por todos los módulos del nivel anterior.

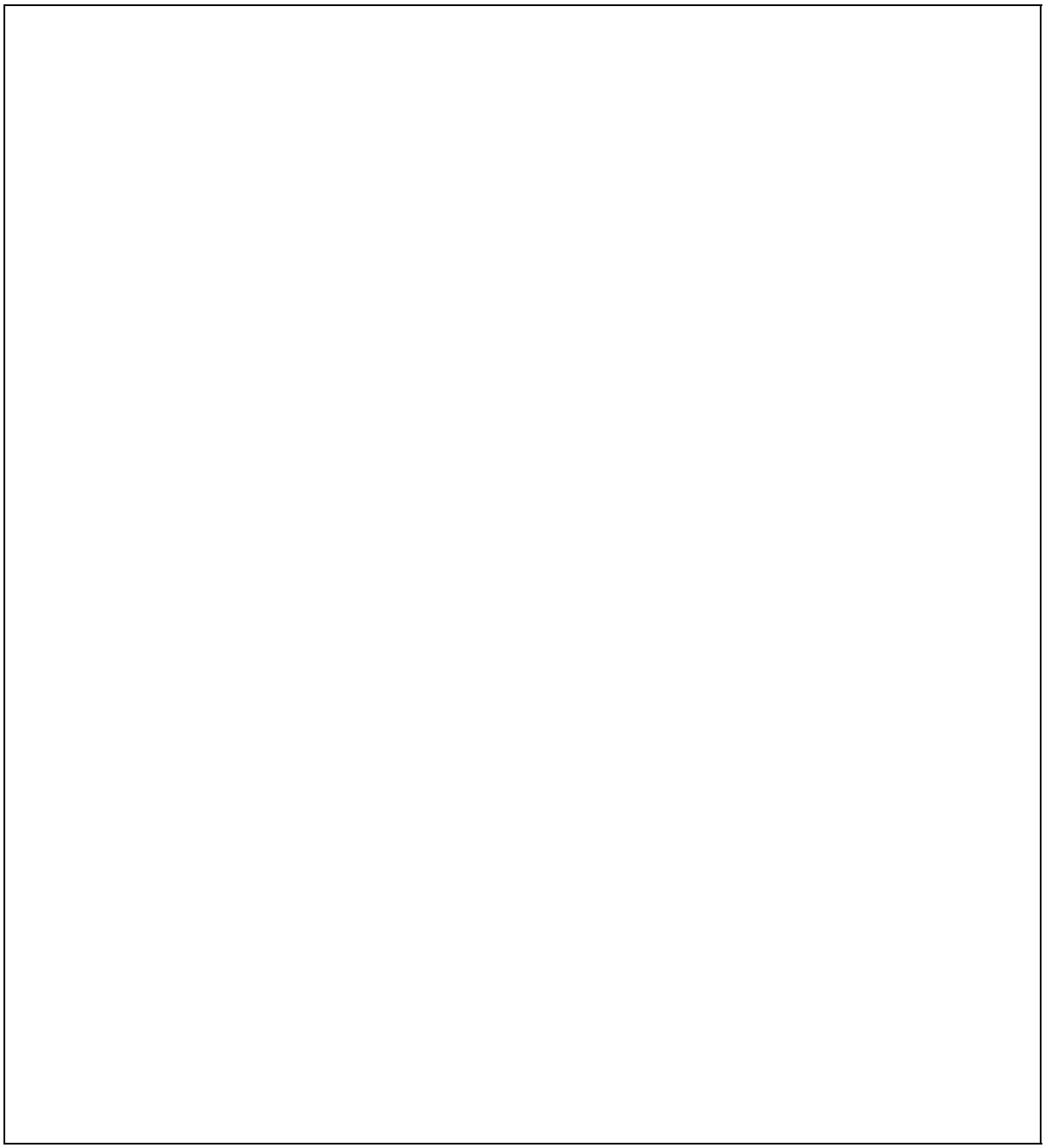


Figura 2.16: Organización de modelos MORSE que representan a sistemas inteligentes particulares.

2.4.3.-¿Qué Sistema Inteligente usar?

El análisis de las secciones anteriores ofrece información sobre las propiedades generales de varios sistemas inteligentes derivadas del estudio de su estructura. Esta información puede facilitar la elección de uno de estos sistemas inteligentes para resolver un problema particular. Algunas ideas extraídas del análisis anterior que ayudan a elegir un sistema inteligente son:

- Si deseamos identificar un sistema que resuelva un problema pero con la condición de que su dinámica sea “simple” y tenga una interpretación comprensible, deberíamos usar un sistema basado en el modelo de razonamiento no monótono o alguno que pudiera ser representado por el sistema MORSE contextual, ya que la representación MORSE de estos sistemas sólo necesita dos niveles. Otra opción podría ser utilizar cualquier otro sistema inteligente pero limitando la profundidad de su representación MORSE, por ejemplo, podríamos usar una red neuronal multicapa con una sola capa oculta [Benítez97] o un árbol de decisión con profundidad limitada.

- Supongamos que disponemos de un método para poder realizar varios procesamientos en un mismo paso (algoritmo de entrenamiento Backpropagation [Rumelhart86] o herramientas para dividir el espacio de entrada [Abramson63, Gallant86, Minsky69, Chiang94]). Entonces, si el factor de generalización es importante (buenos resultados sobre los ejemplos de prueba), podríamos usar redes neuronales para resolver un problema. Por otro lado, si se dispone de un conjunto de ejemplos de entrenamiento completo y consistente y, por tanto, el factor de generalización no es muy importante, podríamos utilizar un sistema inteligente basado en árboles de decisión.

- Si no disponemos de un método para poder realizar varios procesamientos en un mismo paso y no nos preocupa el coste computacional, entonces podríamos usar la arquitectura Correlación en Cascada. Si el factor de coste computacional es importante, podríamos pensar en utilizar un controlador difuso jerárquico.

En esta sección hemos presentado las conclusiones referentes a una serie de sistemas inteligentes particulares, que fueron elegidos en este capítulo con el fin de ilustrar el beneficio del estudio de sistemas en el nivel de su estructura. El mismo procedimiento se podría seguir con otros tipos de sistemas inteligentes (representarlos con MORSE, ver los valores particulares para cada característica y deducir propiedades generales), logrando de esta manera poseer información acerca de ellos que facilitaría su uso correcto a la hora de resolver un problema.

2.5.- Resumen y notas finales

En este capítulo hemos definido un nuevo modelo para representar conocimiento estructurado, hemos conseguido modelar varios sistemas inteligentes,

logrando incluir varios de ellos, que fueron presentados de manera independiente, en una misma representación y, finalmente, hemos presentado una serie de características de una estructura MORSE que, en función de sus valores concretos, deciden una serie de propiedades generales para un sistema inteligente.

Todos estos logros han permitido identificar un procedimiento para ayudar en la elección de un tipo de sistema inteligente u otro cuando se quiera resolver un problema. Este procedimiento consiste, básicamente, en los siguientes pasos:

1) Representar mediante estructuras MORSE a todos los sistemas inteligentes disponibles.

2) Observar los valores concretos para las características de las estructuras MORSE, que ofrecen las representaciones obtenidas en el paso anterior.

3) Deducir, a partir de los valores apreciados en el paso anterior, las propiedades generales de cada sistema inteligente.

4) Elegir aquel sistema cuyas propiedades generales se amolden mejor al tipo de solución que deseamos alcanzar.

Con todo esto, podemos apreciar que con MORSE, a diferencia de otros esquemas de modelización de conocimiento estructurado, hemos logrado analizar las propiedades de un sistema inteligente desde el punto de vista de su estructura, dando lugar a una metodología para poder usar los resultados de este análisis en casos prácticos.

Capítulo 3

APRENDIZAJE ESTRUCTURADO

3.1.-Introducción

En este capítulo, vamos a definir un método de aprendizaje automático que va a simular la estructura de algunos tipos de aprendizaje humano. Para conseguir esto, tras observar la estructura de varios ejemplos de aprendizaje humano, comprobaremos si la estructura de algunos de los modelos de sistemas inteligentes ya conocidos se asemejan a la estructura del comportamiento humano. En este paso, utilizaremos el modelo MORSE y su uso en la representación de varios sistemas inteligentes. Encontraremos dos sistemas inteligentes (razonamiento no monótono y correlación en cascada), cuya estructura es parecida a la del aprendizaje humano y, finalmente, definiremos el método de aprendizaje automático MEGAS como una combinación de la estructura de estos dos sistemas. Apreciaremos que MEGAS es fácilmente modelable mediante MORSE, ya que se ha definido en el nivel de la estructura de sistemas. En definitiva, en este capítulo veremos un ejemplo de como aprovechar MORSE para definir un nuevo método de aprendizaje automático.

Uno de los objetivos de la Inteligencia Artificial es simular el comportamiento humano y, dentro de esta idea general, se ha prestado gran atención a los procesos de aprendizaje. En este capítulo pretendemos definir un método de aprendizaje automático que trata de imitar la estructura del proceso de aprendizaje humano. Para ello, partimos de la apreciación de que nuestra forma de pensar es estructurada: todos tenemos unas reglas generales que actúan de manera consensuada con otras reglas más particulares que dependen del contexto en el que se encuentre cada persona. Estas reglas concretas surgen, durante el proceso de aprendizaje de cada persona, para mejorar los resultados obtenidos tras aplicar solamente las reglas generales en cada contexto específico. Veamos un ejemplo de este hecho:

> ¿*Cómo aprende una persona a vender en una tienda*?. Al principio, el vendedor actúa de acuerdo a las reglas generales de conducta de su sociedad, o sea, cuando llega un cliente le saluda, responde a sus preguntas y le enseña los productos que le pide. Con estas reglas, el vendedor es capaz de vender a ciertos clientes pero se da cuenta que a otros, que no tienen las ideas muy claras acerca de lo que quieren comprar, no consigue venderles nada. Entonces aprende que, ante este tipo de clientes (clientes indecisos), es aconsejable

comentarles lo bueno que es cierto producto para convencerles de que es lo que buscan (regla concreta), frente al resto de clientes (clientes no indecisos) con los que no es necesario realizar un esfuerzo adicional para venderle algo. Sin embargo, esta regla concreta debe utilizarse de forma consensuada con la actuación de las reglas generales con el fin de evitar que ciertos clientes indecisos se sientan incómodos ante la actitud asfixiante del vendedor para convencerles acerca de la bondad de un producto.

En este ejemplo podemos observar un primer paso del vendedor en el que afronta la tarea de forma general, obteniendo una solución aproximada (uso de reglas generales de conducta) y, a continuación, delimita los casos donde la primera solución falla (clientes indecisos), que estudia con más detenimiento para resolverlos correctamente, sin olvidarse de la solución obtenida en el primer paso. Este es el tipo de comportamiento humano (aprendizaje) que vamos a intentar modelar por medio de una nueva definición de sistema inteligente, cuyo diseño se centrará en la forma de estructurar lo que se aprende. Para conseguir esto, vamos a ver si alguno de los sistemas inteligentes, cuya estructura fue estudiada en el capítulo anterior mediante MORSE, coincide con la estructura de la forma de aprender del vendedor.

En el capítulo anterior estudiamos dos tipos de sistemas inteligentes en función de su estructura, que se aproximaban en alguna de sus características a la forma de razonar humana del ejemplo anterior. Estos sistemas son:

- El modelo de razonamiento no monótono [Lukaszewicz90] que consiste en un primer módulo que contiene las reglas por defecto y una serie de módulos conteniendo reglas que se activan en situaciones particulares. Podemos apreciar cierta similaridad con las reglas generales del vendedor y la regla concreta de convencer a un cliente que se activa cuando se detecta un cliente indeciso.

- La arquitectura de redes neuronales Correlación en Cascada [Fahlman90] cuyos módulos son perceptrones donde, durante el proceso de aprendizaje, un nuevo módulo se añade al sistema con el fin de reducir su error global y, este nuevo módulo actúa sobre la zona del espacio de entrada del problema con mayor error. Este proceso se asemeja al seguido por el vendedor cuando se percató de que no conseguía vender nada a los clientes indecisos y, entonces creó una regla concreta que actuaba ante este tipo de clientes y que mejoraba su número de ventas.

Podemos observar que la estructura de estos dos tipos de sistemas inteligentes, se aproxima a la del aprendizaje humano del ejemplo anterior pero no coincide totalmente. Vamos a combinar la estructura de ambos sistemas para conseguir definir una metodología de aprendizaje

automático, cuya estructura coincide con la del ejemplo de aprendizaje humano. Denominaremos a este método MEGAS (MEcanismo General de Aprendizaje eStructurado) y consiste basicamente en:

1) Diseñar un sistema que proporcione una solución aproximada a un problema.

2) Delimitar las zonas donde esta primera solución falla.

3) Diseñar sistemas locales en estas zonas, cuya solución actúe de forma consensuada con la salida proporcionada por el sistema diseñado en el primer paso, con el objeto de mejorar el funcionamiento de todo el sistema global como solución del problema.

La idea del uso de un primer sistema que ofrece una salida aproximada y de sistemas que actúan localmente en varias regiones procede de la estructura del modelo de razonamiento no monótono y, la idea del ajuste por zonas del error producido por el primer sistema, procede de la estructura de la metodología de diseño de la arquitectura de redes neuronales Correlación en Cascada.

Podemos apreciar como, a partir del estudio de la estructura de varios sistemas inteligentes mediante el modelo MORSE, hemos logrado definir fácilmente un nuevo método de diseño de sistemas inteligentes a nivel de estructura, que logra simular cierto tipo de razonamiento humano y que, como veremos más adelante, se puede modelar con MORSE.

A continuación, presentaremos una serie de ejemplos que muestran como aprende el ser humano de acuerdo a la estructura definida por MEGAS. Seguidamente, definiremos esta metodología y mostraremos algunas de sus ventajas mediante un par de ejemplos. Finalmente, presentamos una particularización de MEGAS donde sus componentes son redes neuronales artificiales (redes neuronales estructuradas), que implementamos y realizamos una serie de pruebas para comprobar los beneficios que aporta el uso estructurado de una herramienta estándar como son las redes neuronales artificiales, frente a su utilización no estructurada.

3.2.-Estructura del aprendizaje humano

Con el propósito de conseguir otra muestra de que los seres humanos solemos aprender de acuerdo a la filosofía de MEGAS, pedimos a diez personas, de manera separada, que dedujeran reglas para clasificar puntos (x,y) con $x \in \{1,...,100\}$ e $y \in \{1,...,50\}$ entre dos posibles

clases (clase 0 y clase 1), y que explicaran el razonamiento seguido en su deducción. Para poder realizar esta tarea les mostramos un conjunto de puntos de cada clase. Ilustramos estos puntos en la Figura 3.1 y los presentamos a continuación:

Puntos de la clase 0

(79, 6) (55, 10) (49, 4) (86, 6) (69, 2) (27, 8) (1, 9) (29, 4)
(22, 5) (78, 5) (66, 8) (27, 10) (30, 4) (89, 9) (25, 12) (76, 2)
(36, 11) (21, 11) (45, 4) (17, 11) (48, 4) (58, 1) (31, 11) (26, 8)
(57, 9) (44, 9) (24, 1) (51, 2) (23, 13) (60, 2) (21, 12) (12, 10)
(88, 10) (97, 6) (36, 5) (22, 12) (98, 2) (59, 1) (59, 9) (70, 1)
(77, 6) (85, 8) (6, 5) (57, 10) (24, 13) (92, 7) (63, 4) (36, 1)
(88, 5) (13, 3) (46, 7) (48, 6) (84, 5) (55, 2) (47, 3) (46, 1)
(80, 8) (28, 8) (98, 9) (9, 9) (62, 10) (75, 11) (36, 7) (51, 1)
(24, 12) (13, 10) (46, 9) (66, 11) (37, 3) (94, 2) (46, 2) (23, 14)
(22, 13) (9, 7) (79, 4) (71, 11) (25, 6) (75, 2) (19, 5) (74, 1)
(73, 2) (65, 8) (44, 7) (100, 3) (94, 9) (58, 10) (14, 2) (92, 11)
(11, 10) (45, 6) (8, 2) (78, 9) (23, 12) (81, 6) (81, 9) (17, 9)
(68, 1) (89, 6) (17, 4) (51, 6)

Puntos de la clase 1

(52, 29) (64, 50) (87, 48) (40, 47) (63, 48) (62, 44) (77, 21) (45, 39)
(39, 26) (81, 12) (79, 22) (74, 13) (90, 15) (75, 30) (6, 39) (60, 37)
(31, 30) (59, 21) (97, 23) (13, 41) (25, 14) (82, 36) (70, 33) (81, 19)
(98, 25) (21, 32) (87, 30) (43, 37) (73, 36) (1, 14) (93, 46) (12, 32)
(45, 23) (22, 44) (79, 13) (47, 19) (43, 28) (60, 19) (91, 15) (23, 47)
(60, 35) (51, 13) (31, 19) (100, 26) (61, 38) (33, 17) (31, 33) (55, 45)
(2, 15) (61, 39) (25, 50) (47, 26) (79, 31) (9, 37) (55, 17) (79, 44)
(25, 33) (47, 17) (41, 24) (4, 12) (50, 41) (81, 15) (62, 27) (2, 49)
(100, 24) (25, 43) (90, 16) (40, 38) (83, 39) (41, 12) (8, 43) (32, 22)
(15, 25) (100, 12) (87, 42) (9, 26) (66, 27) (47, 20) (15, 18) (49, 33)
(14, 43) (36, 36) (19, 42) (74, 44) (4, 23) (66, 14) (15, 44) (12, 25)
(88, 24) (76, 48) (32, 31) (64, 15) (5, 30) (18, 26) (21, 16) (26, 12)
(79, 42) (57, 19) (8, 27) (17, 13)

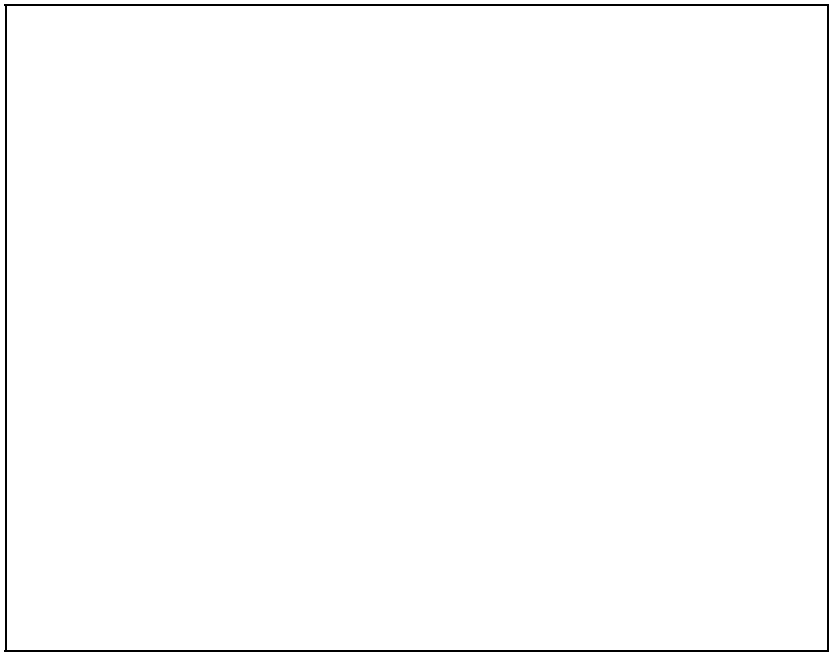


Figura 3.1: Puntos de las clase 0 y 1. Las cruces corresponden a los puntos de la clase 1 y los rombos corresponden a los puntos de la clase 0.

Básicamente, observamos dos formas de resolver el problema que finalmente obtenían la misma solución. Siete personas dedujeron una primera regla de clasificación en la primera mirada al problema que clasificaba incorrectamente una serie de puntos y, las otras tres personas, dedujeron inicialmente dos reglas de clasificación que dejaban una zona del espacio de entrada sin tratar. Sin embargo, esta zona y el área del espacio de entrada donde se concentraban los puntos incorrectos de la primera forma de resolución, coincidían. Esto provocaba que los siguientes pasos de ajuste en las dos formas de resolución fueran los mismos, dando lugar finalmente al mismo conjunto de reglas para resolver el problema. Veamos más detenidamente las acciones realizadas por las personas que llevaron a cabo la prueba:

1) Tras un primer vistazo, siete personas dedujeron el siguiente "conjunto a" de reglas de clasificación y las otras tres el "conjunto b":

Conjunto a

Si $(y \geq 13)$ entonces $(x,y) \in$ clase 1, si no $(x,y) \in$ clase 0.

Conjunto b

Si $(y \leq 11)$ entonces $(x,y) \in$ clase 0.
Si $(y \geq 15)$ entonces $(x,y) \in$ clase 1.

Usando las reglas del "conjunto a", existen puntos mal clasificados y, si se usan las reglas del "conjunto b", dejamos una zona del espacio de entrada del problema sin tratar.

2) El siguiente paso consistió en estudiar cuidadosamente la zona del espacio de entrada donde aparecían los puntos mal clasificados por el "conjunto a", que coincidía con la zona sin tratar por el "conjunto b". Para esto, todas las personas seleccionaron los puntos pertenecientes a esta zona:

Puntos conflictivos de la clase 0

(25, 12) (23, 13) (21, 12) (22, 12)
(24, 13) (24, 12) (23, 14) (22, 13)
(23, 12)

Puntos conflictivos de la clase 1

(81, 12) (74, 13) (25, 14) (1, 14)
(79, 13) (51, 13) (4, 12) (41, 12)
(100, 12) (66, 14) (26, 12) (17, 13)

3) De la observación de estos puntos, las diez personas dedujeron las mismas dos reglas para la clasificación:

1. Si $(y \leq 11)$ entonces $(x,y) \in$ clase 0.
2. Si $(y \geq 12)$ entonces $(x,y) \in$ clase 1,
   salvo que $(x \in \{21,22,23,24,25\} \wedge y \in \{12,13,14\})$ en cuyo caso $(x,y) \in$ clase 0.

En ese paso podemos apreciar la aparición de una regla (excepción de la 2ª regla) con una zona de actuación concreta que se utiliza para afinar la actuación de una regla general.

Estas últimas reglas siguen fallando al clasificar el punto (25,14). En este momento la prueba finalizaba ya que no existían puntos que dieran información más detallada. En la Figura 3.2 ilustramos la clasificación deducida por las diez personas que realizaron la prueba y el espacio del cual se extrajeron los puntos para clasificar.

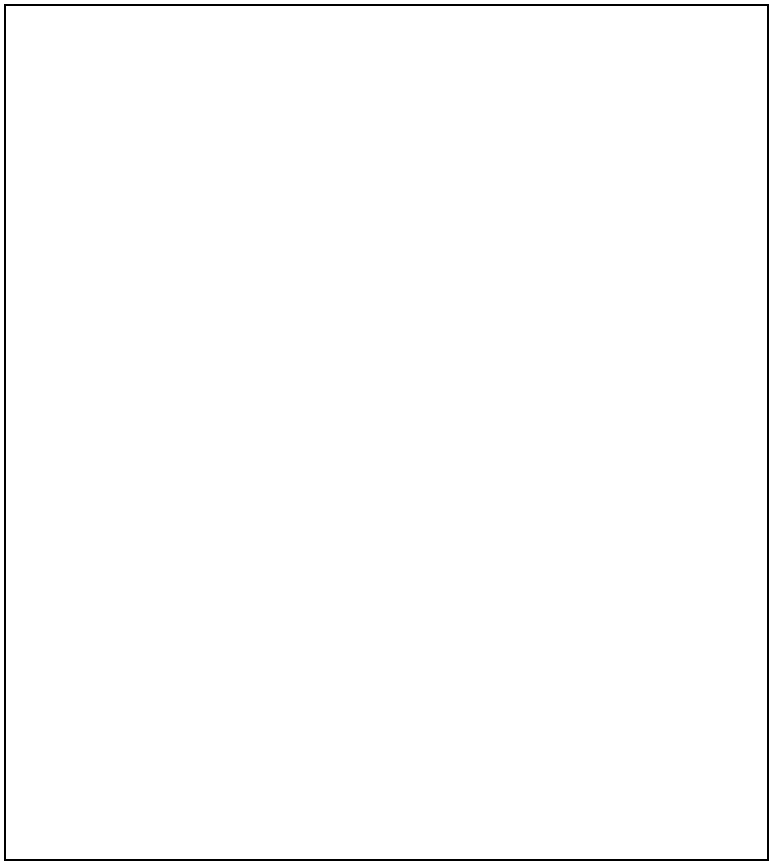


Figura 3.2: a) Espacio deducido en la prueba. b) Espacio original del cual se extrajeron los puntos.

Podemos apreciar que el razonamiento seguido por todas las personas coincide con la filosofía de MEGAS: primero, encontraron reglas que resolvían aproximadamente el problema, a continuación, detectaron las zonas donde había fallos y, finalmente, dedujeron reglas locales a estas zonas que afinaban el comportamiento de las reglas generales.

El procedimiento para encontrar una solución exacta a este problema de clasificación usando la filosofía de MEGAS consiste en:

a) Resolver el problema de forma general, obteniendo una primera regla de clasificación cuya acción, para un punto (x,y), se resume en que si ($y \leq 11$) el punto pertenece a la clase 0 y, en caso contrario, el punto pertenece a la clase 1 (Figura 3.3.a). Podemos modelar esta regla así:

Si $((y-11) \leq 0)$ **¡Error! Argumento de modificador no especificado.**entonces $(x,y) \in$ **¡Error! Argumento de modificador no especificado.**clase 0, si no $(x,y) \in$ **¡Error! Argumento de modificador no especificado.**clase 1.

b) Cuando se aplica la regla general anterior, observamos que aparecen puntos mal clasificados en el rectángulo con vértices (20,11), (26,11), (20,14) y (26,14) (Figura 3.3.b).

c) El siguiente paso consiste en buscar reglas de clasificación que actúen localmente en el anterior rectángulo, tales que, al actuar de forma consensuada con la regla general encontrada en el primer paso, clasifiquen correctamente todos los puntos del rectángulo. La regla local sería (Figura 3.3.c):

Si $(y \leq 14)$ entonces $(x,y) \in$ clase 0, si no $(x,y) \in$ clase 1,

y el consenso de actuación entre esta regla local y la general que resuelve la clasificación en el rectángulo, es (Figura 3.3.d):

Si (((1-λ(x) * (y-11) + λ(x) * (y-14))≤ 0) **¡Error! Argumento de modificador no especificado.**entonces $(x,y) \in$ clase 0, si no $(x,y) \in$ clase 1,

donde

$$\lambda(x) = \begin{cases} \dfrac{x-20}{3} & si\ x \in [20,23] \\ \dfrac{26-x}{3} & si\ x \in [23,26] \end{cases}.$$

Actuando con este consenso en el rectángulo y con la regla general en el resto del espacio de entrada, resolvemos satisfactoriamente el problema de clasificación.

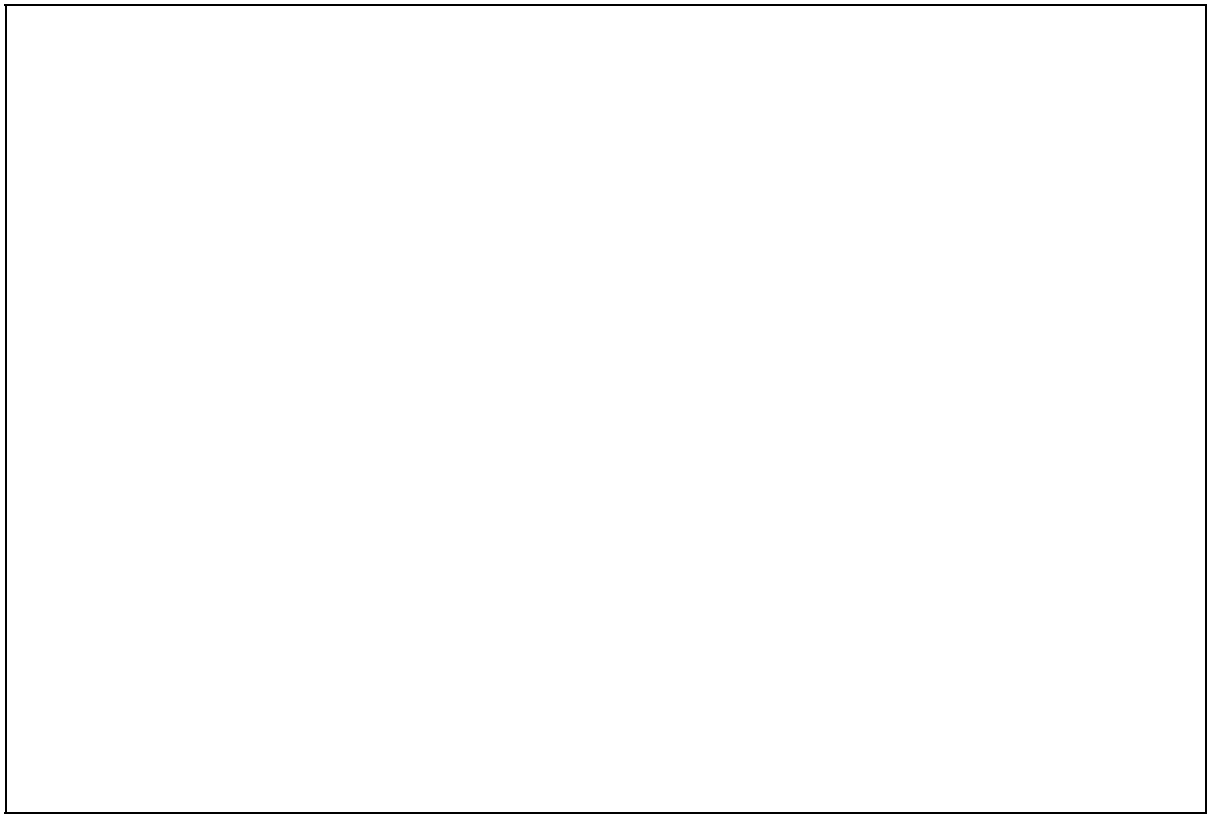


Figura 3.3: Procedimiento para encontrar una solución exacta al problema de clasificación usando la filosofía de MEGAS. a) Regla general, b) zona errónea, c) regla local, d) resultado del consenso entre la regla general y la regla local.

En la Figura 3.4 ilustramos este sistema solución al problema de clasificación representado con un modelo MORSE. Observamos que un sistema diseñado con la filosofía de MEGAS es fácilmente modelable mediante una estructura MORSE. Como veremos en la siguiente sección, no sólo el sistema diseñado con MEGAS es representable mediante el modelo MORSE, sino que también el propio MEGAS es modelable con MORSE.

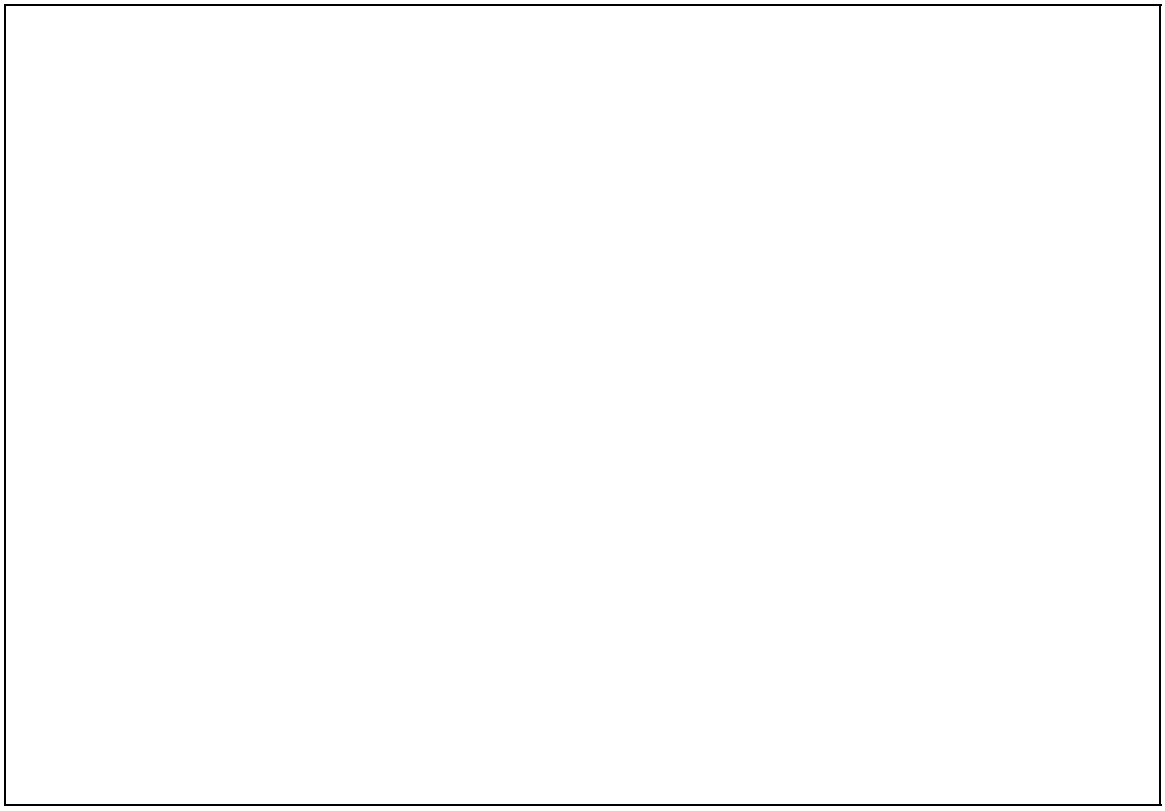


Figura 3.4: Solución al problema de clasificación de la prueba representado con un modelo MORSE.

3.3.-MEGAS: mecanismo general de aprendizaje estructurado

En esta sección vamos a definir un método de aprendizaje automático, denominado MEGAS (MEcanismo General de Aprendizaje eStructurado) [Castro99c], inspirado en la estructura aparecida en los ejemplos de comportamiento humano expuestos en las secciones anteriores. Vamos a definir MEGAS a partir de componentes abstractos, dando lugar a un metaalgoritmo, que puede dar lugar a la definición de diferentes algoritmos en función de la herramienta concreta que se utilice en cada componente y del propósito perseguido con su uso. Este método de aprendizaje automático puede concebirse como el conjunto de la ejecución de los siguientes pasos para resolver un problema:

1. Usar un módulo de aprendizaje para diseñar un sistema que resuelva el problema de forma global, proporcionando una solución aproximada.

2. Detectar las zonas del espacio de entrada del problema donde la solución del paso anterior no es totalmente satisfactoria.

3. En cada una de las zonas detectadas, usar un módulo de aprendizaje para diseñar un sistema que resuelva el problema localmente en dicha zona. Estos sistemas locales se diseñarán teniendo en cuenta la solución obtenida en el primer paso, este hecho implicará que sus salidas tendrán que actuar de acuerdo a un consenso con la salida ofrecida por el sistema global.

Observando los pasos que definen MEGAS se plantean dos cuestiones:

1) ¿Cómo se podrían realizar varias ejecuciones seguidas de MEGAS?, o sea, ¿cómo se podría definir la recursión de MEGAS?.

2) ¿Hasta qué nivel de granularidad debe llegar la recursión de MEGAS?, o sea, ¿cuándo debe parar un sistema recursivo de MEGAS?.

Podemos pensar en varias respuestas a la primera pregunta:

- podríamos volver a aplicar MEGAS en cada zona local definida en el segundo paso, o
- podríamos volver a aplicar MEGAS, tras diseñar un primer sistema estructurado con él, donde el sistema del primer paso fuera este sistema estructurado ya diseñado, o
- podríamos volver a aplicar MEGAS sobre la unión de varias zonas locales definidas en el segundo paso.

Cada una de las anteriores opciones tiene sus ventajas y sus inconvenientes. Unas alternativas pueden obtener resultados exactos pero mala generalización, otras alternativas podrían obtener una buena generalización pero les costaría aprender a manipular correctamente los ejemplos de entrenamiento. En definitiva, consideramos que el estudio de la posible recursión de MEGAS requiere un tratamiento aparte y, es por esto, que se lo dedicamos en el siguiente capítulo de esta tesis.

Respecto a la segunda pregunta, la respuesta más obvia parece ser que la recursión continúe hasta lograr una buena cota de exactitud en los resultados. Sin embargo, fijar esta cota puede no ser trivial en ciertos problemas. En el siguiente capítulo, veremos con unos cuantos ejemplos, varias formas para ajustar de la manera más óptima posible a esta cota, intentando alcanzar los mejores resultados en exactitud sobre los ejemplos de entrenamiento o en generalización.

El módulo de aprendizaje del primer paso de MEGAS debe diseñar un sistema que proporcione una visión general del problema a resolver, en vez de diseñar un sistema cuya salida se acerque a la mejor solución del problema. Por ejemplo, podríamos pensar en usar, en este primer paso, una red neuronal artificial no muy grande con el fin de conseguir, en un periodo corto de tiempo, una visión general del problema, ya que el entrenamiento de una red con tamaño moderado tiene una convergencia rápida a pesar de no obtener una solución muy exacta. Otra opción para este sistema del primer paso de MEGAS podría consistir en usar un sistema basado en reglas difusas (Apéndice II) con pocas reglas y con un mecanismo de diseño simple y rápido.

El segundo paso de MEGAS, que consiste en delimitar las zonas conflictivas del problema, podríamos realizarlo con cualquier técnica de agrupamiento conocida, intentando agrupar los ejemplos erróneos comunes, o con cualquier otra herramienta de clasificación (árboles de decisión, redes neuronales, ...) que permita distinguir los ejemplos problemáticos del resto. En el caso de utilizar una técnica de agrupamiento en este paso de MEGAS, se podría distinguir entre un agrupamiento nítido [Jain88] o un agrupamiento difuso [Bezdek93]. La principal diferencia entre ellos radica en que el agrupamiento nítido detectaría zonas del espacio de entrada del problema con fronteras claramente definidas, por contra el agrupamiento difuso delimitaría zonas cuyas fronteras serían borrosas, o sea, existiría un área alrededor de una frontera donde no se sabría con exactitud a que zona pertenecen los puntos que caen dentro. En este caso, un punto que se situara dentro de una frontera difusa, activaría los sistemas de las zonas locales cercanas a la frontera, que tendrían una influencia en el resultado final del sistema diseñado con MEGAS proporcional al grado de pertenencia del punto a cada zona.

Diseñaremos los sistemas locales del paso 3 en función del consenso existente entre su salida y la ofrecida por el sistema global. Por ejemplo, si el consenso consiste en elegir la salida producida por el sistema local, entonces los módulos de aprendizaje del paso 3 intentarían resolver exactamente el problema en su zona local. Por otro lado, si el consenso consiste en realizar un promedio entre la salida del sistema local y la del sistema global, entonces los módulos de aprendizaje del paso 3 intentarían resolver el problema en su zona sin olvidar las salidas producidas por el sistema global. Por ejemplo, al final de la siguiente sección y en la sección 3.6 de este capítulo, el consenso consiste en sumar la salida del sistema global con la del sistema local, por lo tanto, diseñamos los sistemas locales intentando reducir el error producido por el sistema global en su zona.

En la Figura 3.5 mostramos MEGAS representado con un modelo MORSE y en Alg. 3.1 aparece el metaalgoritmo correspondiente.

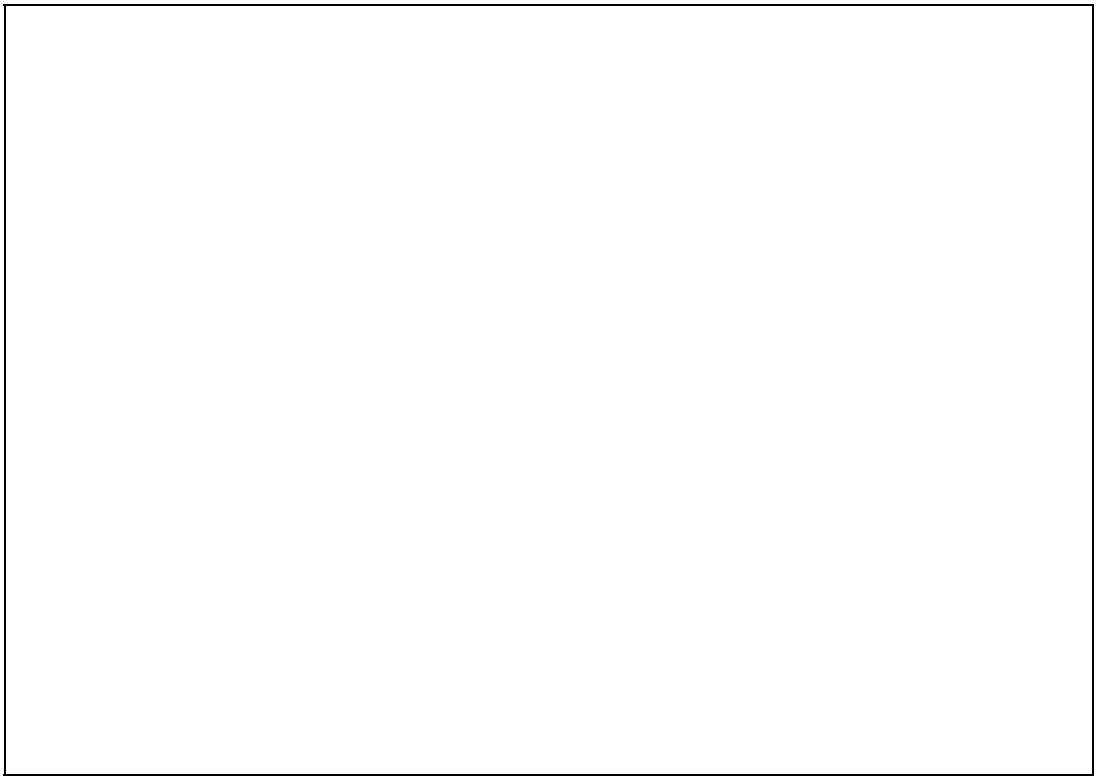


Figura 3.5: MEGAS representado con un modelo MORSE.

Entrada: Un problema.
Salida: Un sistema estructurado que resuelve el problema de entrada.

1.- Diseñar un sistema para recoger información general del problema.

2.- Delimitar las zonas del espacio de entrada del problema donde se concentran los puntos conflictivos.

3.- En cada zona extraída en el paso 2, hacer:

3.1.-Diseñar un sistema en esta zona para resolver el problema, teniendo en cuenta la información proporcionada por el sistema del paso 1.

Algoritmo 3.1: Metaalgoritmo para MEGAS.

En una ejecución de MEGAS los módulos de aprendizaje de cualquier nivel pueden ser de distinto tipo. Por ejemplo, podríamos combinar redes neuronales con sistemas difusos o con cualquier otro método de aprendizaje. La elección en cada caso dependerá de la información que

dispongamos acerca de la zona donde va a actuar y de los objetivos perseguidos con el módulo en cuestión.

3.3.1.-Convergencia de MEGAS

Definición

Diremos que un sistema de aprendizaje A converge a una función f(x) si la función aprendida F(x) se aproxima uniformemente a f(x) en todos los ejemplos de entrenamiento, o sea:

$$\left|F(x) - f(x)\right| \leq \varepsilon \quad \forall x \in U\,,$$

donde *U* es el conjunto de ejemplos de entrenamiento usados para identificar A y ε es un número real positivo.

A continuación, vamos a estudiar la convergencia de los sistemas basados en MEGAS. Para ello usaremos la siguiente notación:

- $F_1(x)$ será la función que calcula el sistema identificado por el módulo de aprendizaje del primer nivel de un sistema basado en MEGAS.

- $F_{zona\_errónea}(x)$ será la función que calcula un sistema identificado por un módulo de aprendizaje local a una zona errónea.

- $F_{total}(x)$ será la función que calcula la salida final proporcionada por el sistema basado en MEGAS.

Suponiendo que el consenso entre la salida de $F_1(x)$ y la de $F_{zona\_errónea}(x)$ consiste en elegir la salida de $F_{zona\_errónea}(x)$, entonces $F_{total}(x)$ es igual a:

$$F_{total}(x) = \begin{cases} F_{zona\_errónea\_1}(x) & si\ x \in zona\_errónea\_1 \\ \ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots\ldots\ldots \\ F_{zona\_errónea\_n}(x) & si\ x \in zona\_errónea\_n \\ F_1(x) & En\ otro\ caso \end{cases}$$

Para estudiar la convergencia del sistema basado en MEGAS analizaremos que ocurre cuando se le presenta una entrada particular $x_0 \in U$. Pueden ocurrir dos cosas:

1) que x_0 no pertenezca a ninguna zona errónea, entonces

$$\left|F_{total}(x_0) - f(x_0)\right| = \left|F_1(x_0) - f(x_0)\right| \leq \varepsilon ,$$

ya que si fuera mayor que ε, x_0 hubiera caído en alguna zona errónea.

2) que x_0 pertenezca a alguna zona errónea. En este caso, es necesario que los módulos locales de un sistema basado en MEGAS converjan para conseguir que:

$$\left|F_{total}(x_0) - f(x_0)\right| = \left|F_{zona\_errónea}(x_0) - f(x_0)\right| \leq \varepsilon .$$

Por tanto, podemos afirmar el siguiente resultado:

Teorema

Un sistema basado en MEGAS converge si sus módulos locales convergen.

Más concretamente, si cada módulo local converge solamente sobre los ejemplos de su zona de actuación, también se puede asegurar la convergencia del sistema basado en MEGAS.

Este teorema de convergencia está de acuerdo con la filosofía de construcción de un sistema basado en MEGAS. Todo el esfuerzo de aproximación a la solución recae sobre los módulos locales, mientras que el sistema del primer nivel se utiliza principalmente para obtener una visión general del problema.

3.4.- Ejemplos

En esta sección presentamos dos ejemplos que muestran las ventajas de usar la filosofía de MEGAS frente al uso de un único módulo de aprendizaje para resolver un problema. El problema planteado en los dos ejemplos consiste en distinguir los puntos que caen dentro de un círculo (Figura 3.6) de los que caen fuera, en el espacio bidimensional $(x,y) \in [-1,1]x[-1,1]$. En el primer ejemplo, resolvemos el problema a mano, usando solamente rectas sobre el espacio de entrada del problema. El objetivo de este ejemplo es mostrar las ideas básicas de MEGAS. En el segundo ejemplo, planteamos la resolución del problema como una aproximación de puntos. Para ello, generamos una serie de ejemplos que consisten en puntos aleatorios del espacio de

entrada del problema y un valor asociado a ellos de (-1) o (+1), en función de que caiga dentro o fuera del círculo de la Figura 3.6. Intentamos resolver este problema por medio de un sistema inteligente que se diseña con el propósito de aproximar los ejemplos de entrenamiento generados. El objetivo de este segundo ejemplo consiste en comparar empíricamente la bondad de los sistemas diseñados con MEGAS. Para realizar esto, diseñaremos un sistema inteligente con la filosofía de MEGAS, otro sin el uso de esta filosofía pero con la misma herramienta y compararemos sus resultados.

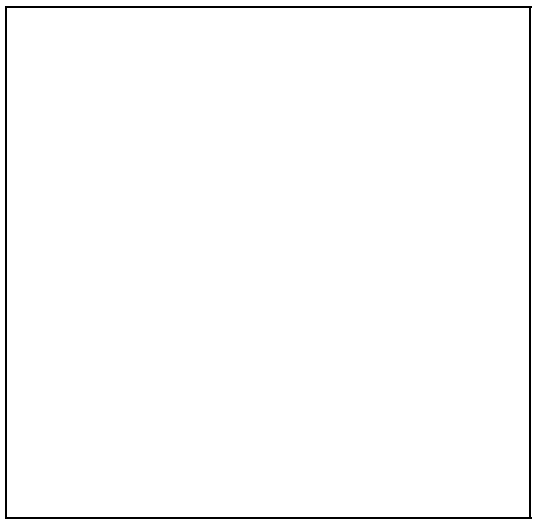


Figura 3.6: Círculo usado en los ejemplos.

3.4.1.-Ejemplo 1: Delimitación manual de un círculo con MEGAS

Supongamos que deseamos diseñar un sistema que devuelva una salida negativa si recibe como entrada a un punto dentro del círculo de la Figura 3.6 y que devuelva una salida positiva en caso contrario. El diseño de este sistema está restringido a que sus reglas sólo puedan utilizar rectas sobre el espacio bidimensional de entrada. Veamos como actuar con la filosofía de MEGAS:

a) Una primera aproximación a la solución del problema podría consistir en la delimitación del cuadrado más pequeño que contiene al círculo. Conseguimos esta primera solución con un sistema compuesto por la siguiente regla:

Regla general: entrada = (x,y) $\rightarrow$ salida = máximo{(-1-x),(x-1),(-1-y),(y-1)},

observemos que esta regla es equivalente a esta otra regla en forma más estándar:

Si (x∈ **¡Error! Argumento de modificador no especificado.**[-1,1] $\wedge$ y∈ **¡Error! Argumento de modificador no especificado.**[-1,1]) *entonces* la salida es negativa, *en otro caso* la salida es positiva.

b) El siguiente paso consistiría en detectar las zonas del espacio de entrada del problema donde existen puntos erróneos. Estas zonas aparecen en la Figura 3.7 denotadas por A, B, C y D. A continuación, realizamos una partición del espacio de entrada intentando delimitar lo más posible las zonas erróneas anteriormente detectadas. Como resultado de esta partición obtenemos los cuadrantes A, B, C y D de la Figura 3.8.

Figura 3.7: Zonas donde hay puntos erróneos.

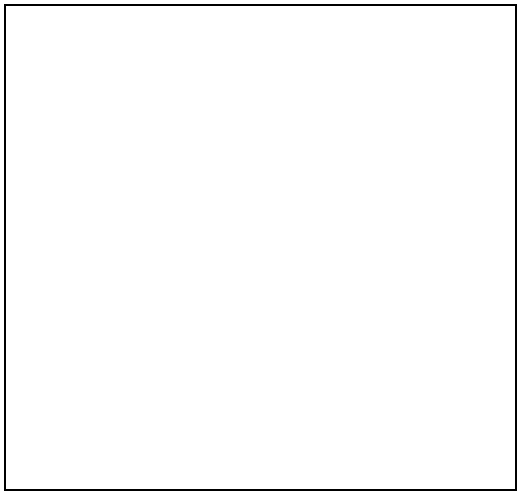


Figura 3.8: Cuadrantes para delimitar las zonas con puntos erróneos.

c) En cada uno de los anteriores cuadrantes intentaremos encontrar una regla local tal que, al actuar de forma combinada con la regla general, consigamos una buena aproximación a la solución del problema. Estas reglas locales podrían ser:

- ***Regla A***: entrada = (x,y) ∈ **¡Error! Argumento de modificador no especificado.**cuadrante A $\rightarrow$ salida = (x+y-1) (Figura 3.9),
- ***Regla B***: entrada = (x,y) ∈ **¡Error! Argumento de modificador no especificado.**cuadrante B $\rightarrow$ salida = (y-x-1),
- ***Regla C***: entrada = (x,y) ∈ **¡Error! Argumento de modificador no especificado.**cuadrante C $\rightarrow$ salida = (-x-y-1),
- ***Regla D***: entrada = (x,y) ∈ **¡Error! Argumento de modificador no especificado.**cuadrante D $\rightarrow$ salida = (x-y-1);

ya que al actuar de acuerdo al siguiente consenso con la acción de la regla general del sistema:

Salida(Regla general) $+\sqrt{2}$ **¡Error! Argumento de modificador no especificado.**Salida(regla i), con i = A,B,C,D;

obtenemos la aproximación a la solución reflejada en la Figura 3.10. Ilustramos el sistema que proporciona esta solución en la Figura 3.11 representado con un modelo MORSE.

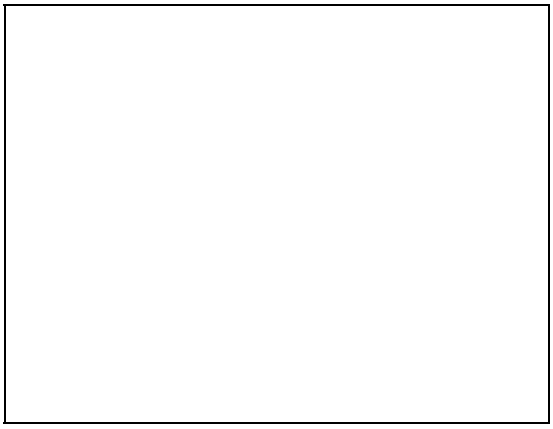


Figura 3.9: Clasificación realizada por la regla local del cuadrante A.

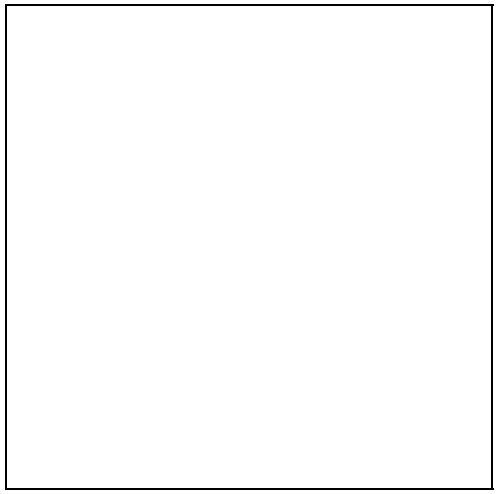


Figura 3.10: Aproximación al círculo lograda con cinco reglas, usando MEGAS.

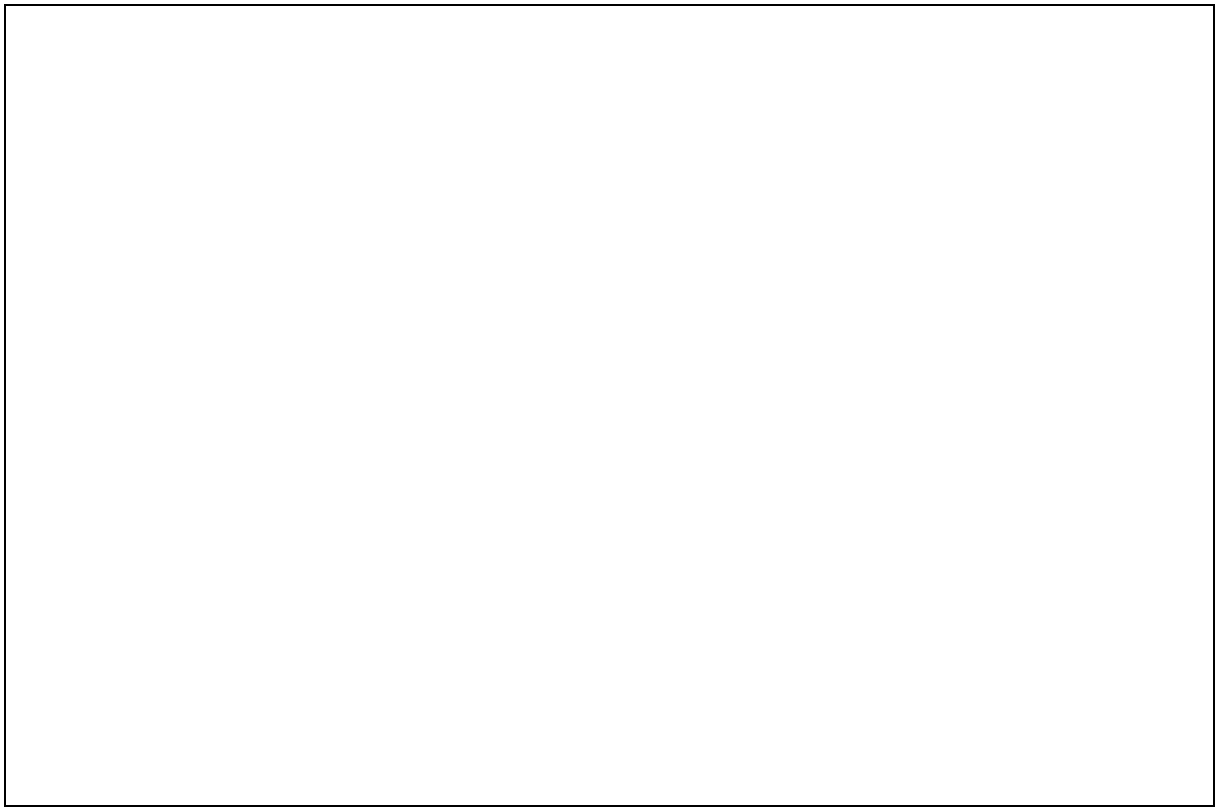


Figura 3.11: Solución a la aproximación del círculo representado con un modelo MORSE.

Podemos apreciar la buena solución (Figura 3.10) alcanzada al problema de la delimitación de un círculo con tan solo cinco reglas que usan funciones lineales al seguir la filosofía de MEGAS, frente a lo complicado que sería encontrar una solución equivalente sin el uso de esta filosofía. Para comprobar experimentalmente esta afirmación, realizamos el siguiente ejemplo de esta sección.

3.4.2.-Ejemplo 2: Aproximación de puntos dentro y fuera de un círculo

Hemos generado aleatoriamente 200 puntos fuera del círculo de la Figura 3.6, asociándoles una salida de (+1) y, 200 puntos dentro del círculo, asociándoles una salida de (-1). Mostramos estos puntos en la Figura 3.12. Con ellos, vamos a plantear el problema de identificar un sistema difuso con reglas tipo TSK [Takagi85] (Apéndice II), que aproxima la salida asociada a cada punto cuando se introduce como entrada. Para poder identificar el sistema difuso con reglas TSK a partir de ejemplos, implementamos el método de identificación presentado en [Lee93,Lee96] (Apéndice V).

Figura 3.12: Puntos usados para identificar el sistema que resuelve el problema de aproximación. Los rombos son los puntos fuera del círculo de la Figura 3.6 y las cruces los puntos dentro del círculo.

Realizamos dos experimentos, uno ha consistido en construir un solo módulo del método de [Lee93,Lee96] (Apéndice V) y el otro ha consistido en construir varios módulos de este mismo método con la filosofía de MEGAS. Hemos comprobado el beneficio de usar MEGAS tanto en el número de reglas del modelo difuso obtenido como en su nivel de aproximación a la solución del problema.

El método de identificación de sistemas difusos con reglas de tipo TSK presentado en [Lee93,Lee96] consiste, básicamente, en usar un algoritmo genético (AG) [Goldberg89, Holland75, Michalewicz96] con codificación binaria que busca, al mismo tiempo, el número de reglas TSK, las funciones de pertenencia de los conjuntos difusos de los antecedentes de estas reglas y los parámetros de los consecuentes, intentando minimizar el número de reglas del modelo y su error cuadrático medio.

Al comienzo de la ejecución de este método, hay que establecer como parámetros del AG, el número de subconjuntos difusos primarios que componen cada variable de entrada y su rango de actuación. Con esto, conseguimos una cota superior al número de reglas del sistema que es igual al producto del número de subconjuntos difusos primarios de cada variable de entrada.

Los valores de parámetros de ejecución, comunes a todos los experimentos, fueron:

- Número de generaciones = 5000.
- Tamaño de la población = 10.
- Probabilidad de cruce = 0.6.

- Probabilidad de mutación = 0.0333.

En el primer experimento, intentamos encontrar la mejor solución al problema propuesto usando un sólo módulo de aprendizaje (una sola ejecución del método de identificación de sistemas difusos). Para ello, ejecutamos el AG con 10 subconjuntos difusos primarios en cada variable de entrada, moviéndose cada variable en el intervalo [-1.5,1.5]. Esta ejecución dio como resultado un sistema difuso con 32 reglas TSK con un error cuadrático medio igual a 0.093. Ilustramos esta solución en la Figura 3.13 representada con un modelo MORSE.

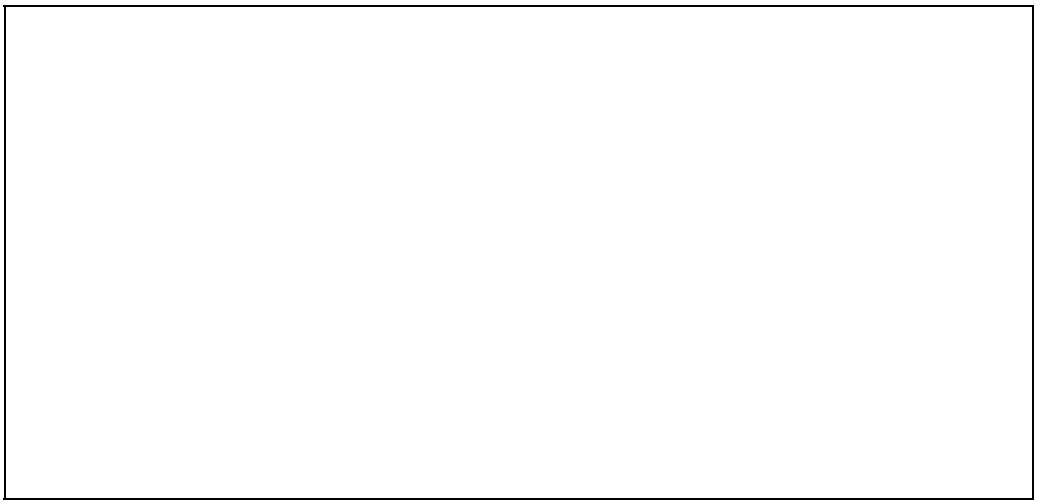


Figura 3.13: Sistema obtenido como solución en el primer experimento, representado con un modelo MORSE.

En el segundo experimento, intentamos encontrar una solución al problema usando la filosofía de MEGAS:

a) Lo primero que hicimos fue ejecutar el AG para obtener, de manera rápida, una primera aproximación a la solución del problema. Lo ejecutamos con sólo dos subconjuntos difusos primarios en cada regla, buscando, por lo tanto, un sistema difuso que no tuviese más de 4 reglas TSK. El resultado fue un sistema difuso (*sistema_difuso_1*) con sólo 2 reglas y con un error cuadrático medio bastante alto, 0.299.

b) El siguiente paso consistió en estudiar los puntos con mayor error en el *sistema_difuso_1* para intentar delimitar las zonas erróneas del espacio de entrada. Para ello, dibujamos los puntos que superaban el error cuadrático medio (Figura 3.14). En esta figura podemos apreciar tres zonas donde, aproximadamente, se concentran puntos erróneos: la zona derecha, la zona

izquierda superior y la zona izquierda inferior. Por lo tanto, las zonas que inferimos para realizar un estudio local en ellas fueron:

- Zona 1: [0.0, 1.5] x [-1.5,1.5] (zona derecha).
- Zona 2: [-1.5,0.0] x [0.0, 1.5] (zona izquierda superior).
- Zona 3: [-1.5,0.0] x [-1.5,0.0] (zona izquierda inferior).

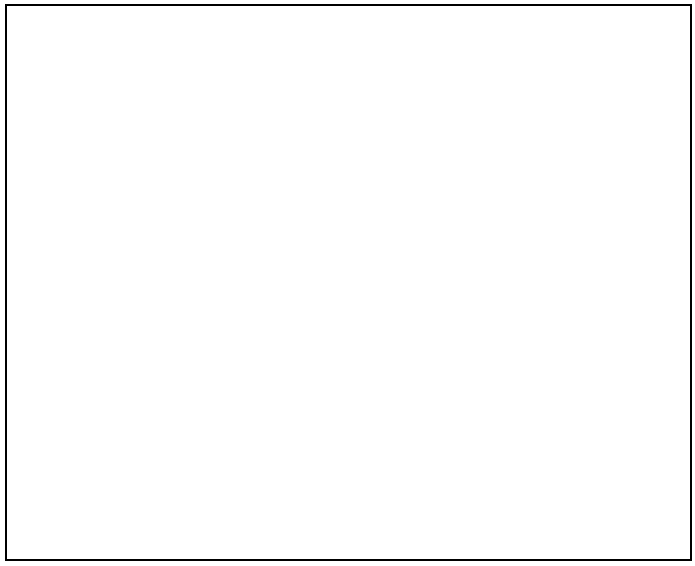


Figura 3.14: Puntos que superan el error cuadrático medio en la primera parte del segundo experimento.

c) En cada una de estas zonas ejecutamos el AG, no para aproximar los valores de salida (+1,-1) asociados a cada punto en el planteamiento original del problema, si no para aproximar el error producido por el *sistema_difuso_1*. Con esto conseguimos encontrar sistemas difusos en cada zona que, al actuar de forma combinada con el *sistema_difuso_1* (suma de las salidas de ambos sistemas), logran una buena aproximación a la solución del problema original. Para la zona 1 ejecutamos el AG con 2 subconjuntos difusos primarios para la variable X y con 4 subconjuntos difusos primarios para la variable Y, el resultado fue un sistema difuso con 8 reglas TSK. En las zonas 2 y 3 ejecutamos el AG con dos subconjuntos difusos primarios para cada variable de entrada, en cada zona obtuvimos un sistema difuso con 4 reglas TSK.

Finalmente, obtuvimos como solución el sistema que se ilustra mediante un modelo MORSE en la Figura 3.15. Este sistema final tiene un error cuadrático medio igual a 0.069, usando 18 reglas TSK de las cuales sólo 2 actúan en todo el espacio de entrada.

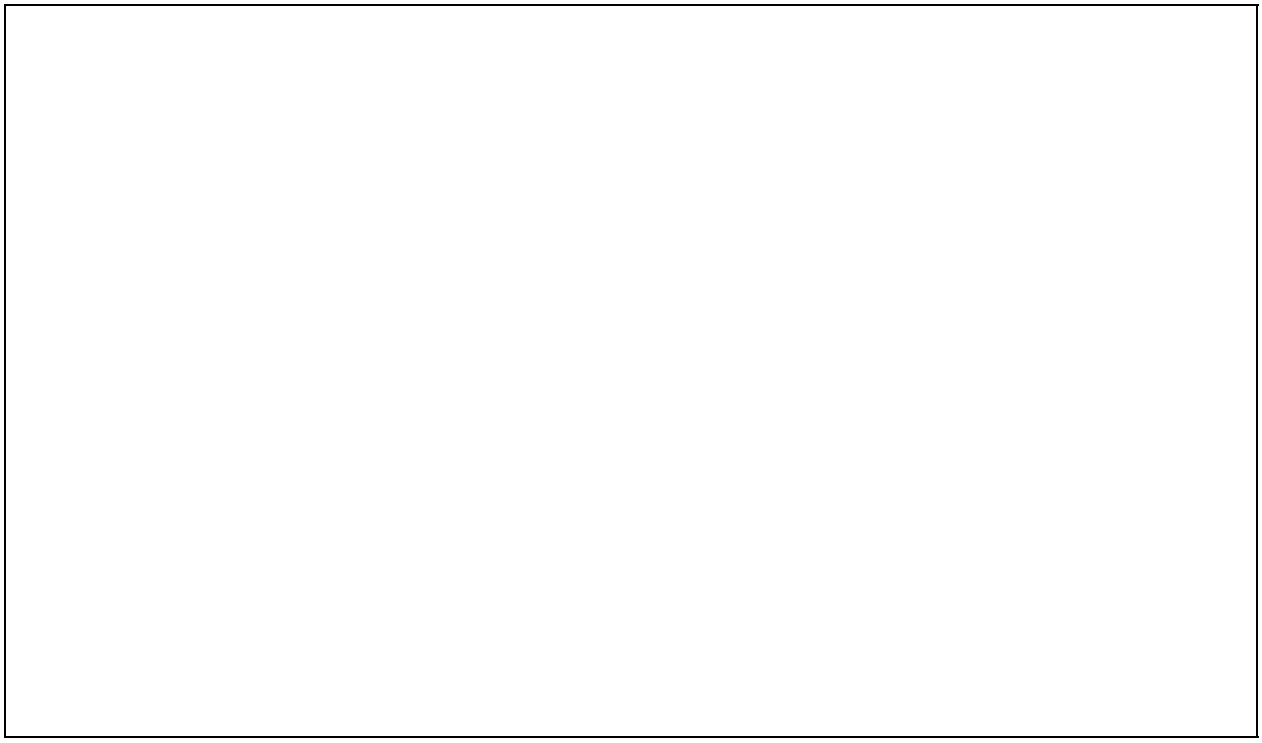


Figura 3.15: Sistema obtenido como solución en el segundo experimento, representado con un modelo MORSE.

Podemos apreciar que con el uso de la filosofía de MEGAS hemos obtenido una solución mejor, tanto en número de reglas (18 frente a 32) como en error de aproximación (0.069 frente a 0.093).

Para finalizar con este ejemplo, hacemos un comentario acerca del coste de ejecución de los dos experimentos, resumido en la Tabla 3.I:

◊ En el primer experimento realizamos:

- **5000** generaciones de un AG con una población de **10** cromosomas y **7200** bits por cromosoma.

◊ En el segundo experimento realizamos:

- **5000** generaciones de un AG con **10** cromosomas y **282** bits por cromosoma (solución *sistema_difuso_1*).
- **5000** generaciones de un AG con **10** cromosomas y **576** bits por cromosoma (solución *sistema_zona_1*).

- **5000** generaciones de un AG con **10** cromosomas y **282** bits por cromosoma (solución *sistema_zona_2*).
- **5000** generaciones de un AG con **10** cromosomas y **282** bits por cromosoma (solución *sistema_zona_3*).

Con estos datos se deduce que encontrar una solución al problema sin MEGAS costó manipular 72000 bits durante 5000 generaciones, mientras que con el uso de MEGAS, encontrar una solución costó aproximadamente la manipulación de sólo 14220 bits (2820 + 5760 + 2820 + 2820) durante 5000 generaciones.

					COSTE
		Generaciones	Cromosomas	bits/cromosoma	bits/ 5000 generaciones
1º experimento	Único sistema	5000	10	7200	**72000**
2ºexperimento	Sistema difuso 1	5000	10	282	2820
	Sistema zona 1	5000	10	576	5760
	Sistema zona 2	5000	10	282	2820
	Sistema zona 3	5000	10	282	2820
		TOTAL			**14220**

Tabla 3.I: Tabla resumen del coste de ejecución de los dos experimentos.

3.5.-Ventajas de usar sistemas difusos o redes neuronales con MEGAS frente a su uso no estructurado

En esta sección, comentamos las ventajas de usar MEGAS cuando sus módulos de aprendizaje están constituidos por algún método de identificación de sistemas difusos publicado o por redes neuronales, frente al uso directo de uno de estos métodos para resolver un problema de clasificación o de aproximación de funciones.

- En el método de identificación de sistemas difusos basado en un AG utilizado en el ejemplo 2 de la sección anterior, se empleaba un término difuso para valorar una variable de entrada, término que era compartido por todas las reglas que tenían la variable. Esto implica que cuando un AG intenta encontrar un único sistema para resolver el problema del círculo, se enfrenta a la dificultad de que el ajuste de un parámetro de una regla para mejorar su acción en cierta zona influye en la aproximación que se realiza en otra zona.

  Por ejemplo, supongamos que ajustamos reglas para manipular correctamente los ejemplos de la zona 1 de la Figura 3.16, con esto estamos fijando las funciones de pertenencia que afectan a la zona 1, o sea, L_2, E_1 y E_2 en la Figura 3.16. Si, a continuación, deseamos ajustar reglas

para actuar correctamente en la zona 2 de la Figura 3.16, sería aconsejable no modificar la función de pertenencia E_2 (beneficiosa para la zona 1). El ajuste de las reglas para la zona 2 fijaría la función de pertenencia L_1. Finalmente, aparece el problema de que si deseamos ajustar reglas para la zona 3, sólo podemos modificar los parámetros de los consecuentes de estas reglas, ya que los parámetros de sus antecedentes (E_1 y L_1) han sido fijados anteriormente para actuar beneficiosamente en otras zonas (zona 1 y zona 2).

Sin embargo, este problema no aparece si usamos MEGAS, ya que, en este caso, además de existir parámetros que afectan a varias zonas, puede haber parámetros que sólo influyen en zonas locales.

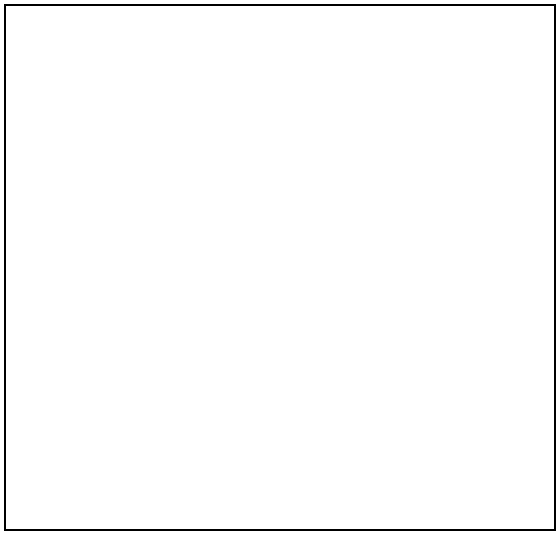


Figura 3.16: Zonas locales para resolver el problema del círculo y funciones de pertenencia asociadas a ellas (E_1, E_2, L_1 y L_2).

- El anterior problema también aparece en los métodos de identificación de sistemas difusos presentados en [Jang93, Takagi85, Sun94]. En estos métodos, se particiona el espacio de entrada de un problema y se asocia una regla difusa a cada partición. Sin embargo, las reglas que actúan sobre el mismo intervalo de una variable de entrada comparten la misma función de pertenencia asociada con este intervalo. Este hecho conlleva que cuando modificamos una función de pertenencia compartida debido al ajuste de una regla difusa, alteramos la acción del resto de reglas difusas que usan esa función de pertenencia.

  Por ejemplo, supongamos que deseamos diseñar un sistema de reglas difusas para distinguir las clases A, B, C y D de la Figura 3.17. Cuando se aplican directamente los métodos

anteriores, primero se realiza un agrupamiento de los puntos del espacio de entrada, obteniéndose cuatro grupos que implican cuatro reglas difusas, una por clase (Figura 3.18) y, a continuación, se ajustan los parámetros de estas reglas para resolver correctamente el problema de clasificación. Sin embargo, cuando ajustamos la regla que actúa sobre los puntos de la clase D, modificamos la función de pertenencia U_2 de la Figura 3.18 y esta función también influye sobre la regla que actúa sobre los puntos de la clase A. Un hecho similar ocurre con la función de pertenencia V_2 y la regla de la clase B y la regla de la clase A. El problema consiste en que no es fácil encontrar los parámetros óptimos para U_2 y V_2 debido a la diferente naturaleza de las clases A, B y D.

Este problema no se presenta si estos métodos de identificación de sistemas difusos se usan con la filosofía de MEGAS, ya que pueden existir conjuntos difusos que actúen en el mismo intervalo de una variable de entrada pero que pertenezcan a módulos diferentes, consiguiendo así tener reglas con distintos conjuntos difusos asociadas al mismo intervalo de una variable de entrada.

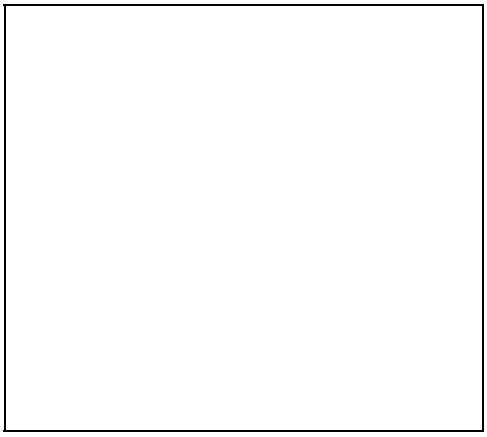


Figura 3.17: Clases A, B, C y D.

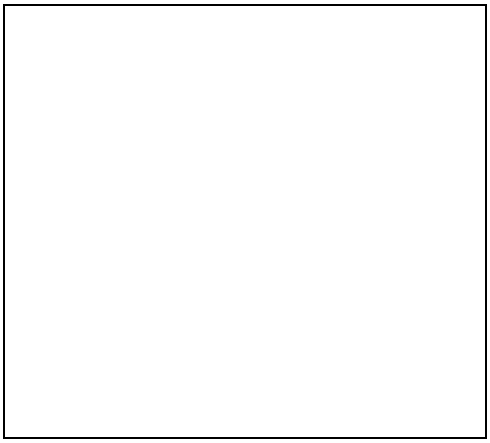


Figura 3.18: Agrupamiento del espacio ilustrado en la Figura 3.17 y nombres de las funciones de pertenencia de los antecedentes de las reglas asociadas con cada grupo (U_1, U_2, V_1 y V_2).

- El problema presentado en los dos casos anteriores no aparece cuando se usan directamente los métodos de identificación de sistemas difusos presentados en [Chiu94, Sugeno93, Yager94], ya que en estos métodos los conjuntos difusos de cada regla pertenecen a ella, pudiendo ajustar los parámetros de sus funciones de pertenencia de forma independiente en cada regla. Sin embargo, en estos métodos el número de reglas sigue siendo igual al número de grupos detectados inicialmente. Este hecho ocasiona que el uso de estos métodos mediante la filosofía de MEGAS presente una ventaja frente a su uso directo para resolver un problema, en lo referente al número de reglas del sistema diseñado

  Por ejemplo, supongamos el problema de distinguir entre las dos clases que aparecen en la Figura 3.19. El primer paso, cuando estos métodos se usan directamente, consistiría en realizar un agrupamiento del espacio, dando lugar a tres grupos y, por lo tanto, diseñaríamos un sistema compuesto por tres reglas difusas. Sin embargo, aplicando la filosofía de MEGAS, primero obtendríamos una regla general, que se aplicaría en todo el espacio y que fallaría en la zona donde se concentran los ejemplos de la clase B y, a continuación, construiríamos una nueva regla que actuaría en esta zona para clasificar correctamente los ejemplos de la clase B. Con esto, obtendríamos un sistema clasificador que resolvería correctamente el problema, compuesto por sólo dos reglas difusas frente al anterior sistema compuesto por tres reglas.

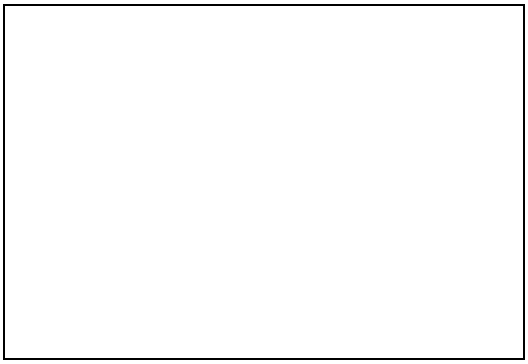


Figura 3.19: Clases A y B.

- El uso de los métodos de identificación de sistemas difusos presentados en [Takagi91, Takagi92] con la filosofía de MEGAS tiene la ventaja comentada en el párrafo anterior respecto al número de reglas, ya que en estos métodos se realiza un agrupamiento previo para conocer el número de reglas del sistema. Por otra parte, estos métodos utilizan una red neuronal para particionar el espacio de entrada y asociar conjuntos difusos a los antecedentes de cada regla, red que se entrena sin indicarle que los conjuntos difusos pertenecen a cada regla de manera independiente, por lo tanto, es posible que estos métodos también presenten el problema referente a la existencia de conjuntos difusos compartidos en el antecedente de varias reglas.

- La resolución del problema del círculo presentado en la sección anterior, usando directamente redes neuronales, consistiría en ajustar parámetros de rectas hasta encontrar una solución. Sin embargo, una recta influye en todo el espacio de entrada y no sólo en el área donde su acción es beneficiosa. Por ejemplo, la recta $x + y - \sqrt{2} = 0$ 12es beneficiosa para resolver el problema de la delimitación del círculo en el cuadrante A (Figura 3.20), sin embargo, esta recta influye en el resto de los cuadrantes. La red neuronal está obligada a buscar un consenso global entre la actuación de varias rectas para llegar a una solución aceptable al problema del círculo. Este problema se soluciona con MEGAS, haciendo que la recta $x + y - \sqrt{2} = 0$ 13sea local al cuadrante A. Por lo tanto, parece adecuado usar MEGAS con redes neuronales como módulos de aprendizaje para resolver problemas de aproximación. En la siguiente sección, presentamos una serie de pruebas empíricas en las que usamos redes neuronales con la filosofía de MEGAS para aproximar funciones.

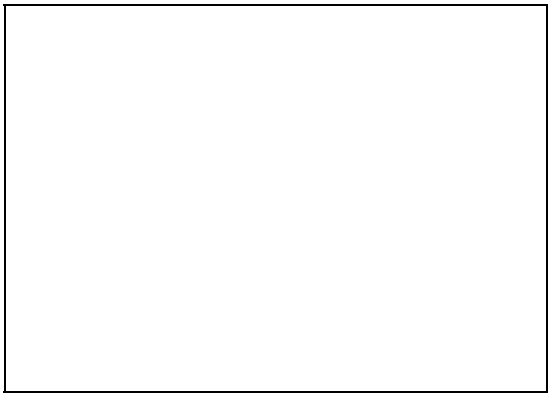


Figura 3.20: Influencia de una recta en el problema del círculo.

3.6.-Redes neuronales estructuradas

En esta sección presentamos una particularización de MEGAS donde sus componentes son redes neuronales artificiales hacia adelante con una capa oculta [Lippmann87, Wasserman93] (redes neuronales estructuradas). Hemos implementado este método y lo hemos usado para resolver problemas de aproximación, comparando sus resultados con los obtenidos por una red neuronal artificial sin estructurar que resolvía los mismos problemas, pudiendo apreciar de esta manera las ventajas experimentales de usar una herramienta de forma estructurada frente a su uso sin estructurar.

Hemos utilizado el algoritmo de agrupamiento presentado en [Chiu94] (Apéndice III) para detectar las zonas del espacio de entrada de un problema donde existen puntos con error mayor que un umbral fijo. Esta técnica de agrupamiento se basa en la definición de una función potencial sobre los puntos que manipula, la cual devuelve una medida acerca de la distancia de un punto con el resto. A continuación, se van extrayendo los máximos de esta función que serán usados como centros prototipos de los grupos, o sea, los grupos que devuelve este algoritmo consistirán en una vecindad de radio r_a (por defecto, 0.3) alrededor de estos centros. Esta técnica de agrupamiento posee la ventaja de determinar automáticamente el número de grupos.

El método para resolver problemas de aproximación basado en MEGAS, cuyos módulos son redes neuronales artificiales, consta de los siguientes pasos:

1. Entrenar una red neuronal (NN_1) con un número fijo N de neuronas en su capa oculta durante un número máximo de ciclos con el fin de resolver el problema de aproximación.

2. Seleccionar los puntos de entrenamiento con error en NN_1 mayor que cierto umbral β.

3. Aplicar una técnica de agrupamiento sobre los puntos seleccionados en el paso anterior.

4. Entrenar en cada grupo una red neuronal (NN_{local}) con N neuronas en su capa oculta para aproximar el error producido por NN_1 sobre los puntos de entrenamiento.

5. Finalmente, la salida del sistema para un punto de entrada particular P, será igual a la suma de la salida producida por NN_1 y la producida por la red local del grupo al que pertenece el punto P (si esta última salida existe).

En el Apendice I describimos las cuatro funciones (F1, F2, F3 y F4) que hemos usado para extraer puntos con el fin de probar los métodos aproximadores de esta sección.

En las redes neuronales estructuradas implementadas, el número máximo de ciclos que entrenamos a una red neuronal artificial es igual a 10000 ciclos, el umbral β es igual a 0.0002 para la función F1 y 0.02 para el resto de las funciones y, finalmente, como ya hemos indicado anteriormente, usamos el algoritmo de agrupamiento presentado en [Chiu94] (Apéndice III) con un radio de vecindad r_a igual a 0.3, para realizar el paso 3 del diseño de las redes neuronales estructuradas. Entrenamos todas las redes neuronales con el algoritmo Backpropagation [Rumelhart86] (Apéndice IV). En Alg. 3.2 aparece el algoritmo de diseño de la red neuronal estructurada usada en esta sección y en la Figura 3.21 ilustramos el sistema obtenido con este método representado con un modelo MORSE.

Entrada: Un problema de aproximación.
Salida: Una red neuronal estructurada que resuelve el problema de aproximación de entrada.

1.- Entrenar con el algoritmo Backpropagation una red neuronal artificial hacia adelante con una sola capa oculta de N neuronas, durante 10000 ciclos, con el objetivo de resolver el problema de aproximación.

2.- Seleccionar los puntos de entrenamiento cuyo error producido por la red neuronal diseñada en el paso anterior es mayor que β.

3.- Aplicar la técnica de agrupamiento presentada en [Chiu94] para agrupar los puntos seleccionados en el paso 2.

4.- Dentro de cada grupo devuelto por el paso anterior, hacer:

4.1.- *Entrenar, con el algoritmo Backpropagation, una red neuronal artificial hacia adelante con una sola capa oculta con N neuronas, con el objetivo de aproximar el error producido por la red neuronal diseñada en el paso 1 al aplicarle los puntos pertenecientes al grupo.*

Algoritmo 3.2: Algoritmo particular de diseño de una red neuronal estructurada.

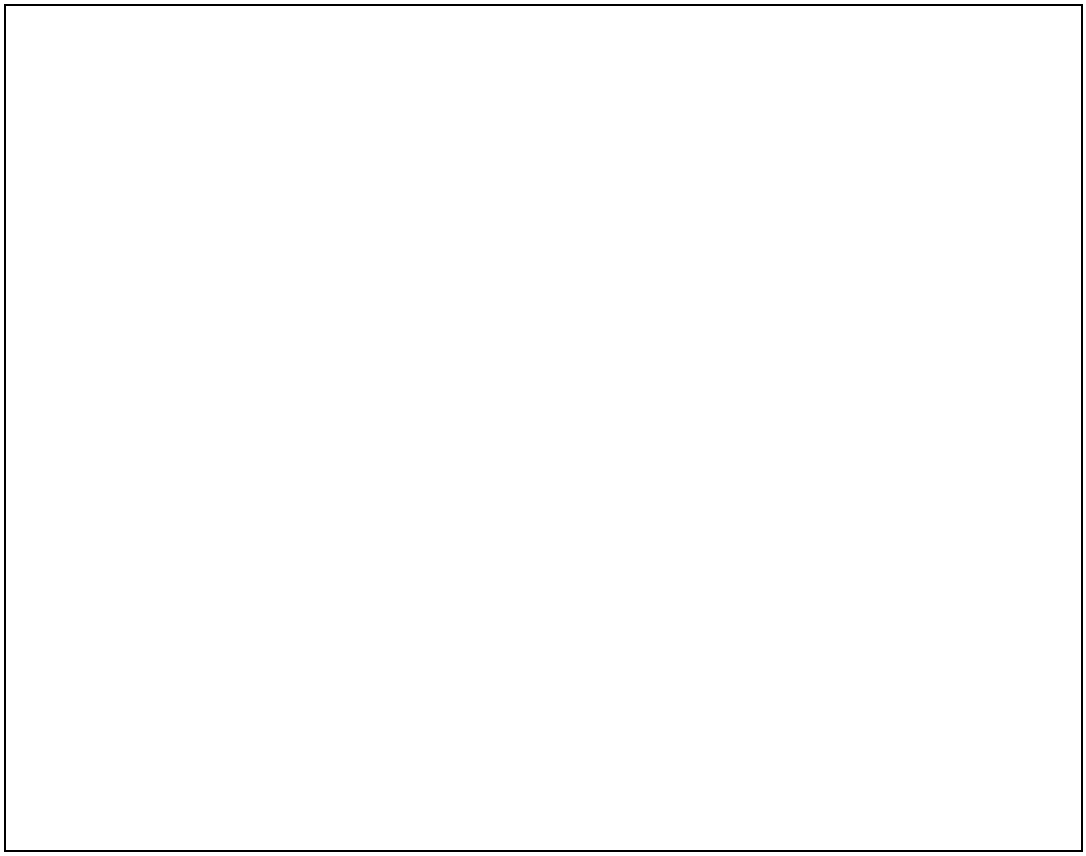


Figura 3.21: Sistema obtenido tras aplicar un método para resolver problemas de aproximación basado en MEGAS, representado con un modelo MORSE.

Para comprobar la bondad de nuestra construcción, hemos comparado los resultados obtenidos al emplear redes neuronales estructuradas en aproximación con los obtenidos al emplear una única red neuronal hacia adelante. La idea es comparar los usos estructurados y no estructurados de una misma herramienta. El número máximo de ciclos que hemos entrenado a esta red neuronal global es igual a 10000.

En las tablas que presentamos a continuación aparecen los resultados de tres sistemas que usan redes neuronales para aproximar: dos sistemas usan la filosofía de MEGAS y el tercero corresponde con el uso de una única red neuronal con una capa oculta para resolver directamente el problema. La diferencia entre los dos sistemas aproximadores basados en MEGAS reside en la forma de entrenar las redes neuronales locales de cada grupo: el primer sistema aproximador basado en MEGAS entrena la red neuronal de cada grupo usando solamente los puntos de ese grupo, mientras que el segundo sistema aproximador basado en MEGAS, entrena la red neuronal de cada grupo usando los puntos de ese grupo y los puntos fuera de él, de forma proporcional a la distancia que los separa del centro del grupo, esto es,

- Si un punto P$\notin$ grupo, entonces la modificación de los pesos de la red neuronal de este grupo se multiplica por

$$\gamma = -\frac{1}{2r_a}d + 1,$$

donde r_a es el radio del grupo (definido en el método de agrupamiento) y d es la distancia del punto P al centro del grupo;

- Si el punto P$\in$ grupo entonces la modificación de los pesos de la red neuronal del grupo no se altera.

Este cambio en el entrenamiento de las redes neuronales locales lo realizamos con el propósito de conseguir mejores resultados del sistema en generalización. Por esto, a este segundo tipo de sistema aproximador basado en MEGAS lo denominaremos *MEGAS con generalidad*.

Los resultados proporcionados por los sistemas aproximadores, reflejados en las tablas de esta sección, corresponden con la media de los valores de salida ofrecidos por cinco ejecuciones independientes. La información que ofrecen las tablas está dividida en cuatro columnas:

1) Exactitud: Refleja el error cuadrático medio de los puntos de entrenamiento. Mide la exactitud de cada uno de los métodos de aproximación al manipular los puntos de entrenamiento.

2) Generalización: Recoge el error cuadrático medio de los puntos de prueba. Mide la capacidad de generalización de los métodos de aproximación.

3) Espacio necesario: Presenta el número de elementos que un sistema aproximador necesita almacenar. En los sistemas basados en MEGAS, el valor de esta columna es igual a la suma del número de neuronas en la capa oculta de las redes que utiliza (red NN_1 y redes locales en cada grupo), mientras que en las redes neuronales sin estructuración, el valor de esta columna corresponde con el número de neuronas de su capa oculta.

4) Tiempo de respuesta: Indica el número de elementos que deben activarse en el sistema aproximador para obtener una salida a partir de una entrada concreta, ofreciendo una medida aproximada del tiempo de respuesta del sistema. En las redes neuronales sin estructuración, este valor también coincide con el número de neuronas en su capa oculta. En los sistemas basados en MEGAS, este valor es igual a la suma del número de neuronas

en la capa oculta de la primera red neuronal NN_1 y la media del número de neuronas en la capa oculta de las redes neuronales locales de cada grupo.

En la tabla 3.II podemos apreciar los resultados ofrecidos por los tres sistemas aproximadores al ser aplicados sobre las cuatro funciones de test (F1, F2, F3 y F4). Dos de ellos basados en MEGAS, con y sin generalidad, que usan una red neuronal con dos neuronas en su capa oculta (N=2) y el otro sistema aproximador que consiste en una red neuronal con cuatro neuronas en su capa oculta (N=4), que intenta resolver el problema globalmente.

El objetivo de esta tabla es comparar el funcionamiento de sistemas basados en MEGAS frente a otros sistemas, sin estructuración, que usan la misma herramienta y que tienen el mismo tiempo de respuesta.

Podemos observar como los sistemas con MEGAS consiguen mejores resultados, tanto en exactitud como en generalización, que los sistemas no estructurados. Por contra, los sistemas estructurados necesitan almacenar mayor número de elementos de procesamiento, sin embargo, el tiempo de respuesta es idéntico. Por otro lado, podemos apreciar como el sistema basado en MEGAS con generalidad frente al sistema basado en MEGAS sin generalidad, consigue mejores resultados en generalización (hecho lógico debido al diferente entrenamiento de las redes locales de estos sistemas) y en exactitud, ya que las redes locales del sistema con generalidad utilizan más puntos en su entrenamiento y esto le permite alcanzar resultados más exactos sobre los puntos de su zona local.

		Exactitud	Generalización	Espacio necesario	Tiempo de respuesta
F1	MEGAS	0.0001322	0.0002182	26.8	4
	MEGAS con generalidad	0.00008	0.0001664	26.8	4
	Red neuronal global	0.0003714	0.0004002	4	4
F2	MEGAS	0.009749	0.0272574	22	4
	MEGAS con generalidad	0.0059092	0.0173732	22.4	4
	Red neuronal global	0.1383246	0.135558	4	4
F3	MEGAS	0.0133962	0.0270488	23.2	4
	MEGAS con generalidad	0.0091786	0.0196028	22.4	4

	Red neuronal global	0.3266776	0.3732034	4	4
F4	MEGAS	0.3257794	0.526196	27.2	4
	MEGAS con generalidad	0.285552	0.467812	26.4	4
	Red neuronal global	0.4954714	0.5686486	4	4

Tabla 3.II: Resultados de tres sistemas aproximadores al ser aplicados sobre los puntos extraídos de las funciones F1, F2, F3 y F4.

En la tabla 3.III podemos apreciar los resultados de tres sistemas aproximadores al ser aplicados sobre las cuatro funciones de test (F1, F2, F3 y F4). Estos sistemas son: 1) red neuronal con cuatro neuronas en su capa oculta que resuelve el problema globalmente (que aparece también en la tabla 3.II) y, 2) dos sistemas basados en MEGAS, con y sin generalidad, que usan la anterior red global como NN_1 y que tienen cuatro neuronas en la capa oculta de sus redes neuronales locales.

El objetivo de esta tabla consiste en mostrar la posibilidad de reutilizar un sistema, por medio de MEGAS, con el propósito de mejorar sus resultados. La metodología de diseño de un sistema por medio de MEGAS consiste en 1) realizar una primera aproximación a la solución del problema, 2) si esta solución es buena se acaba el proceso, si no 3) se diseñan grupos y redes locales, produciendo la salida final de un sistema estructurado. En la tabla 3.III se supone que el sistema, cuyos resultados se han de mejorar, es la red neuronal global con cuatro neuronas en su capa oculta. Entonces, esta red se coloca como el sistema diseñado en el primer paso de MEGAS y, a continuación, se diseñan grupos y redes locales, obteniendo los sistemas basados en MEGAS que aparecen en la tabla 3.III, que mejoran los resultados de la red neuronal global, aprovechándola. Por lo tanto, en esta tabla podemos apreciar el beneficio de aplicar el proceso de agrupamiento y de entrenamiento de redes neuronales locales para mejorar los resultados en exactitud y en generalización de un sistema diseñado previamente.

		Exactitud	Generalización	Espacio necesario	Tiempo de respuesta
F1	Red neuronal global	0.0003714	0.0004002	4	4
	MEGAS	0.0000122	0.0000192	45.6	8
	MEGAS con generalidad	0.0000054	0.0000108	45.6	8
	Red neuronal	0.1383246	0.135558	4	4

F2	global				
	MEGAS	0.0067666	0.0096892	46.4	8
	MEGAS con generalidad	0.0026478	0.004882	46.4	8
F3	Red neuronal global	0.3266776	0.3732034	4	4
	MEGAS	0.0064734	0.0174026	45.6	8
	MEGAS con generalidad	0.002075	0.008344	45.6	8
F4	Red neuronal global	0.4954714	0.5686486	4	4
	MEGAS	0.167848	0.3877718	56	8
	MEGAS con generalidad	0.1189564	0.3143556	56	8

Tabla 3.III: Resultados de tres sistemas aproximadores al ser aplicados sobre los puntos extraídos de las funciones F1, F2, F3 y F4.

Por lo tanto, observando las tablas de esta sección podemos concluir que:

1) el uso de redes neuronales con la filosofía de MEGAS consigue mejores resultados en aproximación que la utilización de redes neuronales sin estructurar con el mismo tiempo de respuesta, como se puede apreciar en la tabla 3.II, y

2) podemos usar la metodología de diseño definida por MEGAS para ser capaz de reutilizar un sistema ya diseñado cuando se necesite afinar su salida como solución de un problema, esto es, si diseñamos un sistema inteligente para resolver un problema y no obtenemos resultados aceptables, en vez de volver a diseñar un nuevo sistema desde el principio, podemos aplicar la filosofía de MEGAS para mejorar los resultados, consiguiendo de esta manera aprovechar el trabajo realizado al diseñar el primer sistema. Esta conclusión se puede obtener de los resultados presentados en la tabla 3.III.

3.6.1.-Propiedades de las redes neuronales estructuradas

A continuación, presentamos las propiedades de la utilización de redes neuronales artificiales con la filosofía de MEGAS deducidas de la experimentación mostrada en esta sección. Podemos resumir estas propiedades en los siguientes puntos:

1. Una red neuronal estructurada parece obtener, experimentalmente, mejores resultados de aproximación sobre los puntos de entrenamiento de un problema que una red neuronal sin estructurar. Por lo tanto, si lo que necesitamos es una buena aproximación sobre los puntos de entrenamiento, usaremos redes neuronales estructuradas frente al uso directo de una sola red.

2. Una red neuronal estructurada parece obtener, empíricamente, mejores resultados de aproximación sobre los puntos de prueba de un problema que una red neuronal sin estructurar. Por lo tanto, si el principal uso de un sistema consiste en tratar ejemplos no vistos durante el entrenamiento, parece conveniente usar redes neuronales estructuradas frente al uso de una única red neuronal.

3. Si hemos diseñado una red neuronal artificial para resolver un problema y los resultados obtenidos no son satisfactorios, en vez de rechazar esta red entrenada y volver a diseñar una nueva con una topología diferente, podríamos usar la metodología de diseño de MEGAS para conseguir mejorar los resultados aprovechando la red entrenada, o sea, realizaríamos un agrupamiento de los puntos de entrenamiento con mayor error producido por la red entrenada y, a continuación, diseñaríamos redes locales en cada grupo para aproximar el error producido por la primera red, obteniéndose de esta manera un sistema que logra mejores resultados que el uso único de la primera red diseñada, aprovechando el trabajo realizado en su construcción.

3.7.-Resumen y notas finales

En este capítulo hemos definido, en base a componentes generales, un método de aprendizaje automático cercano al proceso de razonamiento del aprendizaje humano, denominado MEGAS. Hemos mostrado, por medio de ejemplos, la ventaja de este método cuando sus componentes son sistemas basados en reglas difusas o redes neuronales, frente al uso directo de estos sistemas inteligentes, en cuanto al número de recursos necesarios para resolver un problema y a la consecución de mejores resultados.

Hemos deducido con la experimentación realizada que, cuando disponemos de redes neuronales artificiales para resolver un problema de aproximación de funciones y necesitamos obtener buenos resultados en exactitud y en generalización, es conveniente usar estas redes con la filosofía de MEGAS (redes neuronales estructuradas), frente a su uso sin estructurar para resolver el problema. Además, hemos mostrado como se puede utilizar la

metodología MEGAS para mejorar los resultados proporcionados por un sistema ya diseñado, reutilizándolo y, de esta manera, aprovechando el trabajo realizado al construirlo.

Capítulo 4

SEPARATE: UN MÉTODO DE APRENDIZAJE AUTOMÁTICO BASADO EN PARTICIONES SEMIGLOBALES

4.1.-Introducción

En el capítulo anterior presentamos el método de aprendizaje automático MEGAS, que consiste en realizar sobre un problema y un sistema que lo resuelve aproximadamente, un paso de procesamiento (detección de zonas erróneas e identificación de sistemas locales en estas zonas) para encontrar una solución mejor al problema. Es posible que un solo paso de ajuste sea insuficiente para alcanzar el grado de exactitud requerido para la solución. Por otra parte, llevar a cabo varios pasos seguidos de MEGAS no es trivial: si cada paso de ajuste se ejecuta sobre cada región local obtenida con MEGAS, al realizar varios pasos seguidos se perdería la visión general del problema, resultando un sistema especializado en manipular los puntos de entrenamiento de un modo local y, por otro lado, si se mantiene una visión de conjunto de las regiones locales obtenidas con MEGAS, podemos estar considerando regiones donde no es necesario el ajuste ya que han alcanzado la exactitud deseada en su interior.

En este capítulo vamos a estudiar las características que presentan las distintas formas de llamar recursivamente a MEGAS, obteniendo finalmente un método de aprendizaje automático denominado SEPARATE, que permite realizar varios pasos sucesivos de MEGAS. Una ventaja adicional de la definición iterativa de MEGAS es que, como este método de aprendizaje automático está definido en base a componentes generales, existe la libertad de instanciarlos de manera independiente en cada paso, en función de la mejor herramienta para lograr una solución. Para poder definir un mecanismo de iteración para MEGAS vamos a emplear la división en regiones que hace MEGAS y las propiedades de dos tipos de sistemas inteligentes derivadas de su estructura, que fueron mencionadas en el capítulo 2. Estos dos tipos de sistemas que se suelen usar para resolver problemas de clasificación o de aproximación, son:

a) **Sistemas inteligentes basados en el modelo de árbol** [Breiman84, Quinlan86, Quinlan93, Alché94, Chiang94, Frean90, Park90, Sirat90]. El diseño de estos

sistemas consiste en que si no se puede resolver directamente el problema, su espacio de entrada se divide en varias regiones y se intenta resolver, independientemente, el problema global en cada región. El mecanismo de división local en cada región se puede repetir hasta llegar a regiones del espacio de entrada en las cuales sea sencillo resolver el problema (técnica de diseño de algoritmos denominada Divide-y-Vencerás [Brassard88]). Finalmente, se obtiene una estructura de árbol donde, para una entrada particular, se decide en cada nodo que nodo hijo activar hasta alcanzar un nodo hoja que devuelve una salida para el valor de entrada.

Debido a la filosofía de diseño de este tipo de sistemas inteligentes, son *fáciles de construir* ya que sólo hay que darles los ejemplos de entrada y ellos se encargan de crear su propia estructura y, consiguen unos *resultados bastante exactos sobre los ejemplos de entrenamiento*. Por contra, cuando estos sistemas se están diseñando, el conocimiento acerca de la estructura general del problema se pierde (actuación local en cada región) y este hecho ocasiona una *mala generalización* (malos resultados sobre los ejemplos de prueba) y una *alta sensibilidad ante el ruido* de los ejemplos de entrenamiento.

a) **Sistemas inteligentes basados en el modelo de red neuronal** [Lippmann87, Mezard89, Wasserman89]. El funcionamiento de estos sistemas consiste, básicamente, en la actuación conjunta de una serie de elementos computacionales o neuronas que influyen sobre todo el espacio del problema. Por lo tanto, el diseño de estos sistemas consiste en el ajuste del funcionamiento simultáneo de todas sus neuronas. Este hecho implica una *alta dificultad de construcción*, tanto por el proceso de ajuste como por la necesidad de conocer a priori el número de neuronas necesarias para resolver un problema y, unos *resultados sobre los ejemplos de entrenamiento menos exactos* que los obtenidos por los sistemas inteligentes del apartado anterior. Por contra, la influencia global de las neuronas permite apreciar la estructura general del problema en la fase de diseño del sistema consiguiendo, de esta manera, una *buena generalización* (buenos resultados sobre los ejemplos de prueba) y un *buen comportamiento ante la presencia de ruido* en los ejemplos de entrenamiento.

En este capítulo, pretendemos definir un mecanismo de iteración para MEGAS que reúna las ventajas de los dos tipos de sistemas inteligentes comentados anteriormente. Para esto, resolveremos los inconvenientes de uno usando las ventajas del otro y viceversa, esto es, los inconvenientes de dificultad en el diseño y poca exactitud sobre los ejemplos de entrenamiento que presentan los sistemas inteligentes basados en el modelo de red neuronal, serán resueltos usando la *técnica de diseño de*

algoritmos Divide-y-Vencerás y, los inconvenientes de mala generalización y de alta sensibilidad ante la presencia de ruido en los ejemplos de entrenamiento que presentan los sistemas inteligentes basados en el modelo de árbol, los intentaremos resolver por medio de la *actuación semiglobal de los elementos del sistema inteligente en su fase de diseño*.

El método de aprendizaje automático denominado SEPARATE (SEmiglobal PARtitions to design Automatic sysTEms) consiste en dividir, recursivamente, el espacio de entrada de un problema (técnica *Divide-y-Vencerás*), mediante el proceso de afinamiento definido por MEGAS, hasta alcanzar un sistema solución aceptable para el problema. Sin embargo, a diferencia de los sistemas inteligentes basados en el modelo de árbol, en un momento concreto de la fase de diseño, la tarea de dividir no se realiza localmente en una región particular sino que se realiza teniendo en cuenta el mayor número posible de regiones, esto es, una partición en SEPARATE intenta delimitar los ejemplos de entrenamiento erróneamente resueltos en el tiempo actual de diseño teniendo en cuenta todas las regiones del espacio de entrada menos las que no contienen a ningún ejemplo erróneo (regiones correctas). De esta manera, conseguimos que algunos elementos del sistema diseñado por SEPARATE (elementos que definen las particiones del espacio) tengan una actuación semiglobal en su fase de diseño.

Algunos términos que emplearemos más adelante son concretados a continuación:

- Un *ejemplo erróneo* será un ejemplo mal clasificado en un problema de clasificación o, un ejemplo con un error mayor, simultáneamente, que cierto umbral y que el error medio de la región a la que pertenece en un problema de aproximación.

- Una *región correcta* será una región del espacio de entrada de un problema que no contenga ningún ejemplo erróneo.

- Un *sistema resolutor* (clasificador o aproximador) será un sistema que, a partir de una serie de ejemplos pertenecientes a una región particular, es capaz de aprender y, a continuación, devolver una salida para cada entrada que se le presente.

- Un *elemento de procesamiento* será un sistema que se entrenará para distinguir los ejemplos erróneos de un problema, obteniendo un sistema que clasifica los ejemplos de entrada como correctos o erróneos.

- Una *zona errónea* será la zona del espacio de entrada de un problema determinada globalmente como errónea por un elemento de procesamiento.

En la siguiente sección presentamos el método de diseño SEPARATE suponiendo, por motivos de claridad, que en un paso de MEGAS sólo se deduce una región con ejemplos erróneos. Al final del capítulo, presentamos una implementación particular de SEPARATE junto con los resultados de su aplicación donde, el consenso entre las salidas proporcionadas por distintos módulos para una misma entrada consiste en la elección de la salida del último módulo identificado en el proceso de diseño de SEPARATE.

4.2.-SEPARATE: una metodología para diseñar sistemas inteligentes

En esta sección explicamos detalladamente el método SEPARATE (Semiglobal Partitions to design Automatic Systems) [Castro99d] para diseñar un sistema inteligente que resuelve un problema de clasificación o de aproximación. Para ello, se supone que disponemos de un conjunto de ejemplos de entrenamiento (pares entrada-clasificación o entrada-valor de aproximación), de un conjunto de ejemplos de prueba y de un *sistema resolutor* (clasificador o aproximador).

Inicialmente, en la resolución de un problema con SEPARATE, el espacio de entrada del problema está compuesto por una sola región que abarca todos los ejemplos de entrenamiento y, se supone diseñado un sistema resolutor que ofrece una salida para cada entrada posible. El algoritmo SEPARATE consiste, básicamente, en repetir sucesivamente el siguiente proceso hasta conseguir que todos los ejemplos de entrenamiento sean no erróneos:

1. Delimitar los ejemplos erróneos de los correctos, sin tener en cuenta en este paso a los ejemplos no erróneos contenidos en regiones correctas.

2. Operar de manera independiente sobre los ejemplos de la zona delimitada para intentar solucionar el problema en esta zona.

Más concretamente, el método SEPARATE consiste en ir aplicando iterativamente los siguientes pasos, que dividen el espacio del problema en regiones, hasta conseguir que todas las regiones sean correctas:

A) Construir un elemento de procesamiento que distinga los ejemplos erróneos y que actúe sobre el mayor número posible de regiones del espacio de entrada, sin tener en cuenta las regiones correctas. En este capítulo, como ya se comentó anteriormente, suponemos, por motivos de claridad, que el elemento de procesamiento sólo delimita una región con ejemplos erróneos. Sin embargo, este paso coincide con el proceso de detección de zonas conflictivas de MEGAS y, en este proceso no existía la restricción de tener que delimitar una sola región. En definitiva, en este paso de distinción de ejemplos erróneos, de forma similar al paso 2 de MEGAS, podríamos utilizar cualquier técnica de agrupamiento [Jain88] o cualquier procedimiento de clasificación (redes neuronales, árboles de decisión, etc), que pueda devolver una o varias regiones con ejemplos erróneos, sin embargo, en este capítulo estudiaremos este paso de delimitación con la restricción de que sólo pueda devolver una región no correcta.

B) Realizar sobre la zona delimitada con ejemplos erróneos un **Procesamiento Especial**, que consiste en diseñar un sistema resolutor para resolver el problema en esta zona, cuya salida actuará de acuerdo a un consenso con las salidas disponibles previamente para los puntos de esta zona. Este Procesamiento Especial puede ser de dos tipos:

B.1. Procesamiento Especial que olvida las divisiones anteriores. Por ejemplo, supongamos que tenemos un sistema que resuelve un problema de aproximación, que ha dividido el espacio de entrada del problema en varias regiones (Figura 4.1.a) y, que una recta ha delimitado una zona con ejemplos erróneos que contiene varias divisiones anteriores (Figura 4.1.b), entonces en esta zona olvidaríamos las aproximaciones previas de cada región particular y diseñaríamos una nueva aproximación para toda la zona (aprox5 en Figura 4.1.c).

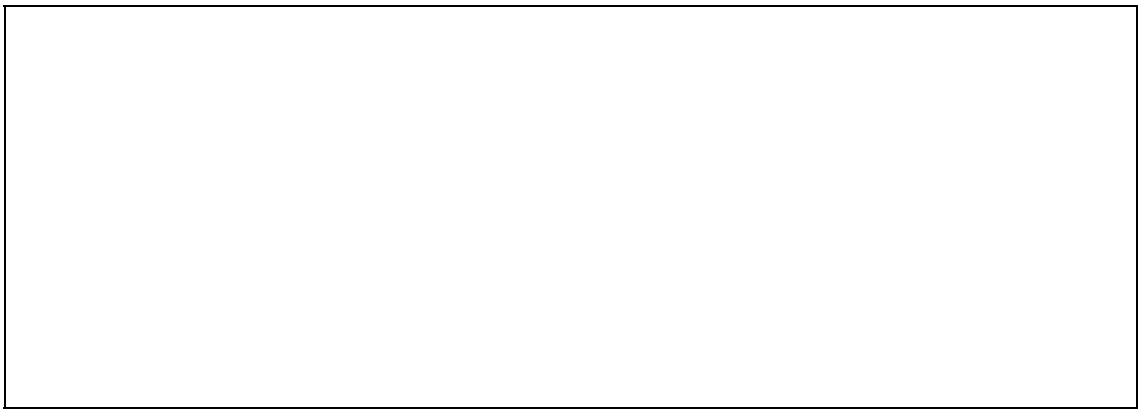


Figura 4.1: Ejemplo de Procesamiento Especial sin tener en cuenta las divisiones anteriores. a) Sistema aproximador que ha dividido el espacio de entrada del problema en varias regiones. b) Recta que delimita una zona con ejemplos erróneos. c) Procesamiento Especial (aprox5) sobre la zona errónea que no tiene en cuenta las divisiones anteriores.

B.2. Procesamiento Especial que tiene en cuenta las divisiones anteriores. Por ejemplo, supongamos que tenemos un sistema clasificador que ha dividido el espacio de entrada de un problema en varias regiones (Figura 4.2.a) y una recta que ha delimitado una zona con ejemplos erróneos (Figura 4.2.b), entonces en esta zona diseñaríamos un clasificador para cada una de las regiones en su interior (clasif E.C, clasif E.D y clasif E.B en Figura 4.2.c).

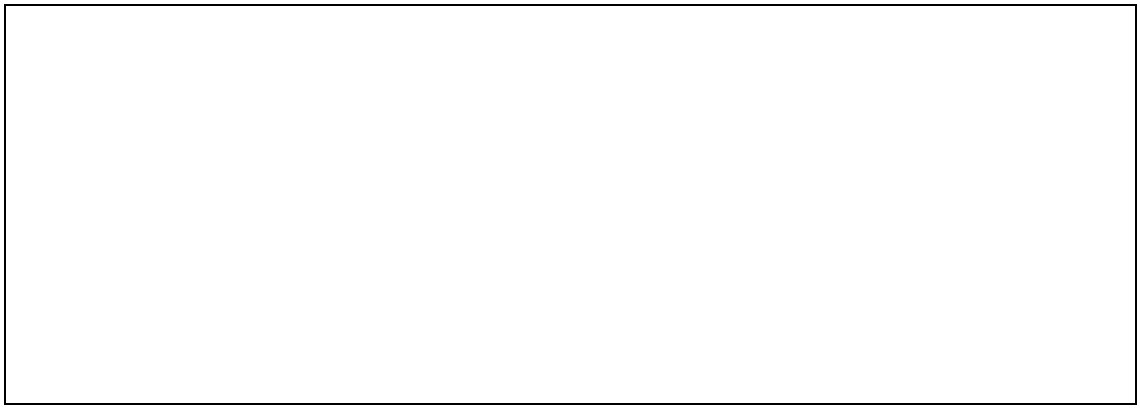


Figura 4.2: Ejemplo de Procesamiento Especial que tiene en cuenta las divisiones anteriores. a) Sistema clasificador que ha dividido el espacio de entrada del problema en varias regiones. b) Recta que delimita una zona con ejemplos erróneos. c) Procesamiento Especial (clasif E.B, clasif E.C y clasif E.D) sobre la zona errónea que tiene en cuenta las divisiones anteriores.

C) Ajustar los sistemas resolutores que actúan sobre regiones que contienen algún ejemplo erróneo y que no están dentro de la zona errónea. Con este paso

pretendemos seguir ajustando los ejemplos que pertenecen a regiones no correctas pero que no han caído dentro de la zona errónea y que, por tanto, no se han visto afectados por el Procesamiento Especial. El objetivo principal de este paso es conseguir convertir en correctas a aquellas regiones que proceden de una división de una región no correcta realizada por el elemento de procesamiento y que han caído fuera de la zona errónea. La motivación de este paso es que puede existir una región del espacio de entrada del problema que no se ha podido convertir en correcta debido a la existencia de unos cuantos ejemplos conflictivos, los cuales pueden caer en la zona errónea al ser dividida esta región por el elemento de procesamiento, permitiendo, de esta manera, que la parte de la región que ha quedado fuera de la zona errónea se pueda convertir en correcta.

En Alg. 4.1 mostramos la metodología de diseño SEPARATE en forma de algoritmo. Podemos apreciar que se dejan una serie de cuestiones que hay que concretar para poder utilizar SEPARATE. Estas cuestiones son:

* ¿Qué herramienta se usa como elemento de procesamiento?
* ¿Con qué se instancia un sistema resolutor?
* ¿Qué tipo de Procesamiento Especial se realiza?
* ¿Cuál es el consenso entre la salida del sistema diseñado en la zona errónea y el resto de salidas disponibles previamente para cada ejemplo?

Entrada: Problema de clasificación o de aproximación.
Salida: Un sistema inteligente que resuelve el problema de entrada.
Condiciones iniciales: Sistema resolutor diseñado que proporciona una salida para cada ejemplo de entrada.

1.- Determinar regiones correctas.

2.- Mientras exista alguna región no correcta hacer:

2.1.- Construir un elemento de procesamiento para distinguir los ejemplos erróneos del resto de los ejemplos que pertenezcan a regiones no correctas.

2.2.- Realizar en la zona delimitada por el elemento de procesamiento un tipo de Procesamiento Especial, esto es, diseñar un sistema resolutor en esta zona cuya salida actúe de acuerdo a un consenso con las salidas ofrecidas

anteriormente para los ejemplos de esta zona.

2.3.-*Ajustar los sistemas resolutores que actúan sobre los ejemplos que están fuera de la zona delimitada por el elemento de procesamiento y que no pertenecen a regiones correctas.*

Algoritmo 4.1: Metodología de diseño SEPARATE en forma de algoritmo.

En la Figura 4.3 ilustramos el método de diseño SEPARATE representado con MORSE. En la implementación particular de SEPARATE presentada en este capítulo, el consenso entre salidas (consenso(Y_1,Y_2) en Figura 4.3) consistirá en tomar la salida proporcionada por el último sistema resolutor diseñado (Y_2 en Figura 4.3). Por lo tanto, los módulos *Nuevo sistema resolutor* intentarán resolver exactamente el problema original en su zona de influencia.

Figura 4.3: Método de diseño SEPARATE representado con MORSE.

Podemos apreciar que el método de diseño SEPARATE va dividiendo el espacio de entrada de un problema en regiones, resolviendo el problema en cada una de ellas (técnica Divide-y-Vencerás), por lo que los sistemas inteligentes que construye tienen la

ventaja de *facilidad de diseño* (ellos mismos definen su propia estructura) y de *exactitud sobre los ejemplos de entrenamiento*. Además, cuando se diseña un elemento de procesamiento para dividir, se tienen en cuenta todas las regiones con ejemplos erróneos, consiguiendo así una visión bastante global de la estructura del problema y, de esta manera, los sistemas inteligentes diseñados con SEPARATE tienen una capacidad de *generalización* y un *comportamiento ante el ruido* presente en los ejemplos de entrenamiento comparable al de los sistemas inteligentes basados en el modelo de red neuronal.

4.2.1.-Convergencia de SEPARATE

Definición

Como ya comentamos en la sección 3.3.1, diremos que un sistema de aprendizaje A converge a una función f(x) si la función aprendida F(x) se aproxima uniformemente a f(x) en todos los ejemplos de entrenamiento, o sea:

$$\left|F(x) - f(x)\right| \leq \varepsilon \quad \forall x \in U\,,$$

donde *U* es el conjunto de ejemplos de entrenamiento usados para identificar A y ε es un número real positivo.

En esta sección estudiamos la convergencia de un sistema diseñado con SEPARATE. En la sección 3.3.1 se comentó que si los módulos locales de un sistema basado en MEGAS convergen, el sistema converge. Como el método de diseño SEPARATE consiste en una iteración de llamadas a MEGAS, el método SEPARATE restringido a una sola iteración identifica a un sistema basado en MEGAS, por lo que se puede afirmar el siguiente teorema:

Teorema

Si el módulo “Nuevo Sistema Resolutor” de SEPARATE converge, entonces el sistema identificado con SEPARATE converge.

El anterior teorema procede del uso de MEGAS por parte de SEPARATE. Sin embargo, debido a la naturaleza iterativa de SEPARATE, se puede hacer un estudio adicional de su convergencia. Para ello usaremos la siguiente notación:

- $F^0(x)$ será la función que calcula el sistema resolutor del primer nivel en la primera iteración de MEGAS.

- $F^j(x)$ (j=1, ...,n) será la función que calcula el módulo local "Nuevo Sistema Resolutor" en la j-ésima iteración de MEGAS.

Con esta notación, suponiendo que el consenso entre una salida local de una zona errónea y una global consiste en elegir la local y, que los módulos de SEPARATE que ajustan las salidas globales sólo se limitan a devolverlas sin realizarle ninguna modificación, la función que calcula el sistema SEPARATE tras realizar n iteraciones es igual a:

$$F_{total}(x) = \begin{cases} F^n(x) & si\ x \in zona\_errónea\ de\ n\_ésima\ iteración \\ F^j(x) & si\ x \in zona\_errónea\ de\ j\_ésima\ iteración \wedge \\ & \wedge\ x \notin zona\_errónea\ de\ i\_ésima\ iteración\ con\ n \geq i > j \\ F^0(x) & en\ \ otro\ \ caso \end{cases}.$$

Veamos que ocurre con la convergencia del sistema SEPARATE si se asegura que cada sistema resolutor diseñado (sistema local a una zona errónea o sistema resolutor del primer nivel en la primera iteración) aproxima un ejemplo de entrenamiento. Supongamos que existen (n+1) ejemplos de entrenamiento y que se introduce una entrada particular $x_0 \in U$ al sistema SEPARATE. Puede ocurrir que:

- x_0 no pertenezca a ninguna zona errónea, entonces

$$\left|F_{total}(x_0) - f(x_0)\right| = \left|F^0(x_0) - f(x_0)\right| \leq \varepsilon$$

  ya que si fuera mayor que ε, x_0 pertenecería a la zona errónea de la primera iteración.

- $x_0 \in$ zona errónea de j_ésima iteración y $x_0 \notin$ zona errónea de i-ésima iteración con n≥i>j, entonces

$$\left|F_{total}(x_0) - f(x_0)\right| = \left|F^j(x_0) - f(x_0)\right| \leq \varepsilon$$

  ya que si fuera mayor que ε, x_0 pertenecería a la zona errónea de la (j+1)_ésima iteración.

- $x_0 \in$ zona errónea de n_ésima iteración, entonces:

  1.- x_0 es el único ejemplo perteneciente a la zona errónea de la n_ésima iteración, ya que con los sistemas resolutores que calculan $F^k(x)$ (k=0,...,(n-1)) se aproximan correctamente los n ejemplos de entrenamiento restantes de U.

  2.- $$\left|F_{total}(x_0) - f(x_0)\right| = \left|F^n(x_0) - f(x_0)\right| \leq \varepsilon$$

  ya que si fuera mayor que ε, el sistema resolutor que calcula $F^n(x)$ no aproximaría ningún ejemplo de entrenamiento.

  Por lo tanto, podemos afirmar el siguiente teorema:

Teorema

Un sistema diseñado con SEPARATE converge al realizar no más de n iteraciones si cada sistema resolutor aproxima al menos un ejemplo de entrenamiento, siendo (n+1) el número de ejemplos de entrenamiento

Una consecuencia de este resultado es que para garantizar el buen funcionamiento del método SEPARATE sólo debemos asegurar que los sistemas resolutores aproximen correctamente al menos un ejemplo de entrenamiento.

4.2.2.-Ejemplo de aplicación de SEPARATE

Consideremos el problema de distinguir nueve puntos positivos de ocho puntos negativos en la figura 4.4.a [Cios92]. Para diseñar un sistema clasificador con SEPARATE, primero debemos concretar una serie de cuestiones del algoritmo:

* Los elementos de procesamiento para dividir consisten en sistemas cuya acción se puede representar mediante una recta sobre el espacio de entrada del problema.
* El sistema resolutor consiste en un clasificador que etiqueta a todos los puntos de una región con la clase mayoritaria de la región, o sea, la clase con la que están etiquetados el mayor número de puntos de entrenamiento de la región.
* El Procesamiento Especial en las zonas erróneas tiene en cuenta las divisiones anteriores.
* El consenso entre salidas consiste en escoger la salida proporcionada por el último sistema resolutor diseñado.

Ya que hay mayor número de puntos positivos en todo el espacio, el primer sistema resolutor clasificaría todos los puntos como positivos. De esta manera, la primera recta intenta separar los puntos negativos (ejemplos erróneos) del resto (Figura 4.4.b). Esta primera recta divide el espacio de entrada en dos regiones y los sistemas resolutores de cada región clasificarían los puntos de una región como negativos y los de la otra como positivos.

El siguiente paso consistirá en determinar qué puntos son erróneos (Figura 4.4.c) y diseñar una recta para intentar delimitarlos (Figura 4.4.d). Esta nueva recta junto con la primera origina cuatro regiones en el espacio de entrada. Cuando estas regiones son clasificadas por los sistemas resolutores, observamos que dos de ellas no contienen puntos erróneos (regiones correctas), por lo tanto, al diseñar la siguiente recta para delimitar puntos erróneos no se tendrán en cuenta estas regiones correctas (Figura 4.4.e). Con esta tercera recta (Figura 4.4.f), que sólo tiene influencia en un subconjunto del espacio de entrada, se divide éste en seis regiones que no contienen puntos erróneos cuando son clasificadas por los sistemas resolutores, por lo tanto, el algoritmo finalizaría en este paso consiguiendo clasificar correctamente todos los puntos.

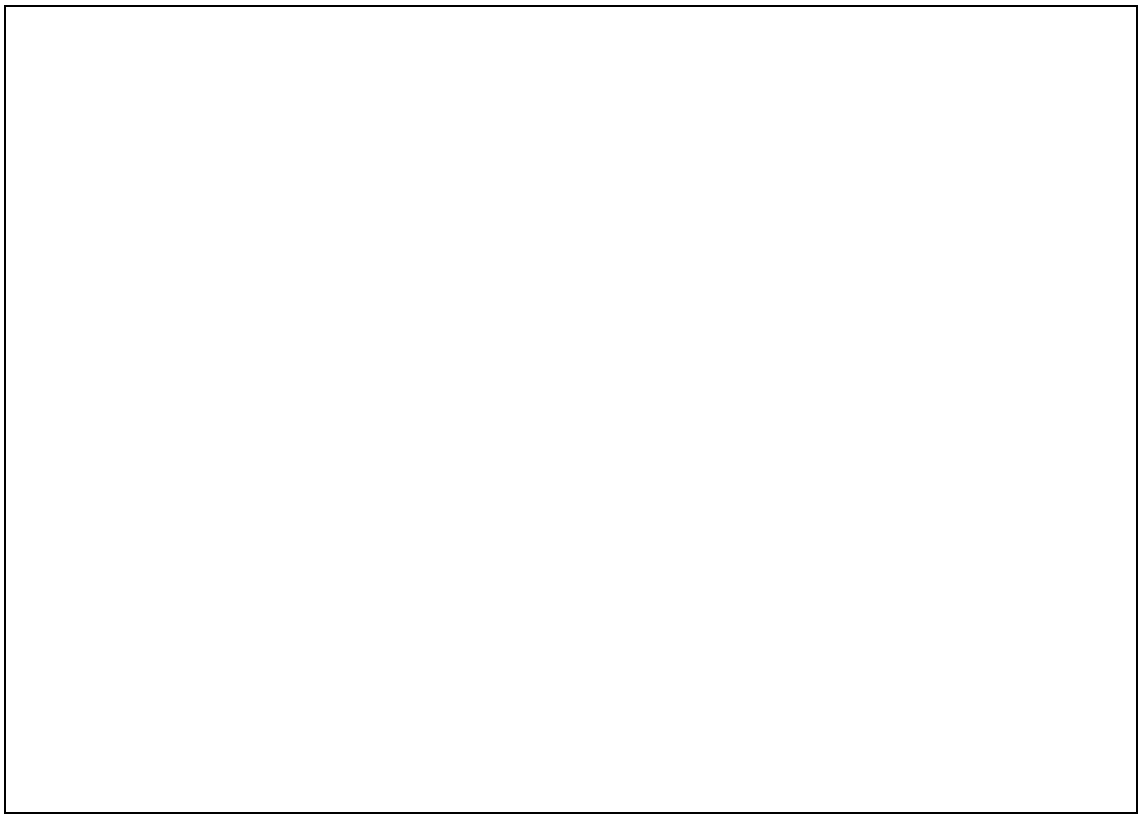


Figura 4.4: Ejemplo de aplicación de SEPARATE. a) Espacio de entrada de un problema de clasificación. b) Actuación de la primera recta delimitadora de ejemplos erróneos y clasificación de regiones. c) Determinación de puntos erróneos. d) Actuación conjunta de las dos primeras

rectas delimitadoras y clasificación de regiones. e) Área de influencia para diseñar la siguiente recta y puntos erróneos. f) Actuación conjunta de las tres rectas diseñadas y clasificación de regiones.

Podemos apreciar que sólo hemos necesitado tres rectas para resolver el problema de clasificación anterior. Un sistema inteligente basado en el modelo de red neuronal podría también resolver este problema con sólo tres rectas [Mezard89], pero sería bastante difícil ajustarlas para conseguirlo. Por otro lado, con un sistema inteligente basado en el modelo de árbol se necesitarían al menos cuatro rectas para conseguirlo (Figura 4.5). Estas ventajas se obtienen gracias a que SEPARATE ha ido insertando rectas una a una, intentando encontrar las zonas donde hacia falta su acción, en vez de disponer a priori de un número fijo de rectas e intentar organizarlas adecuadamente, que es lo que hubiera hecho un sistema inteligente basado en el modelo de red neuronal. Por otro lado, SEPARATE ha diseñado sus rectas para que tuvieran influencia beneficiosa en el mayor número posible de regiones, a diferencia de lo que haría un sistema inteligente basado en el modelo de árbol, que diseñaría cada una de sus rectas para actuar sobre una sola región particular.

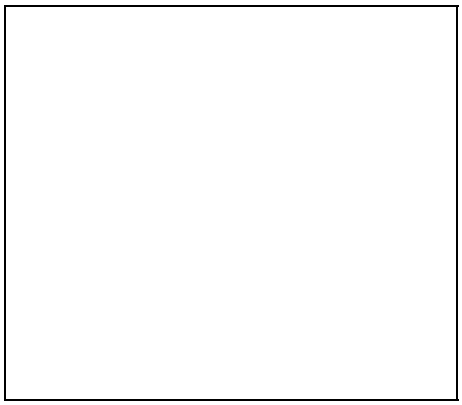


Figura 4.5: Resolución del problema de clasificación de la Figura 4.4 con un sistema inteligente basado en el modelo de árbol que usa cuatro rectas.

4.3.-Implementación particular de SEPARATE

En esta sección presentamos una implementación de SEPARATE donde el consenso entre salidas consiste en elegir la proporcionada por el último sistema resolutor diseñado y, los elementos de procesamiento que dividen el espacio de entrada

de un problema son perceptrones binarios [Minsky69, Wasserman89] entrenados con el algoritmo *Pocket* [Gallant86] (Apéndice IV). Esta implementación posee características especiales debido a las propiedades del sistema elegido como elemento de procesamiento (perceptrón binario). En la siguiente sección presentamos estas características y, en la sección 4.3.2 mostramos una forma de representar un sistema diseñado con SEPARATE que facilita el manejo de su información en un ordenador. En estas dos secciones, no se especifica la herramienta utilizada para instanciar al sistema resolutor, ya que ésta depende del tipo de problema (clasificación o aproximación de funciones) que se vaya a resolver. En la sección 4.3.3 presentamos la implementación particular de SEPARATE para resolver problemas de aproximación y en la sección 4.3.4, para resolver problemas de clasificación, instanciando el sistema resolutor con una herramienta para aproximación y clasificación, respectivamente.

4.3.1.-Características de la implementación particular de SEPARATE

Para asegurar un correcto funcionamiento de SEPARATE, cuando sus elementos de procesamiento son perceptrones binarios, es necesario tener en cuenta las siguientes eventualidades:

- Supongamos que una zona errónea delimitada por un perceptrón binario contiene totalmente a una o más regiones del espacio de entrada (por ejemplo, Figura 4.6.b). No sería razonable realizar el Procesamiento Especial de la zona errónea sobre estas regiones, ya que o perderíamos las actuaciones ejecutadas anteriormente sobre estas regiones en el caso de un Procesamiento Especial que no tuviera en cuenta las divisiones anteriores o, volveríamos a repetir la misma actuación en esa región en el caso de un Procesamiento Especial que tuviera en cuenta las divisiones anteriores. Por lo tanto, la actuación especial que realizamos sobre la zona errónea sólo afecta a las partes de las regiones divididas por el perceptrón que caen dentro de la zona errónea (por ejemplo, Figura 4.6.c).

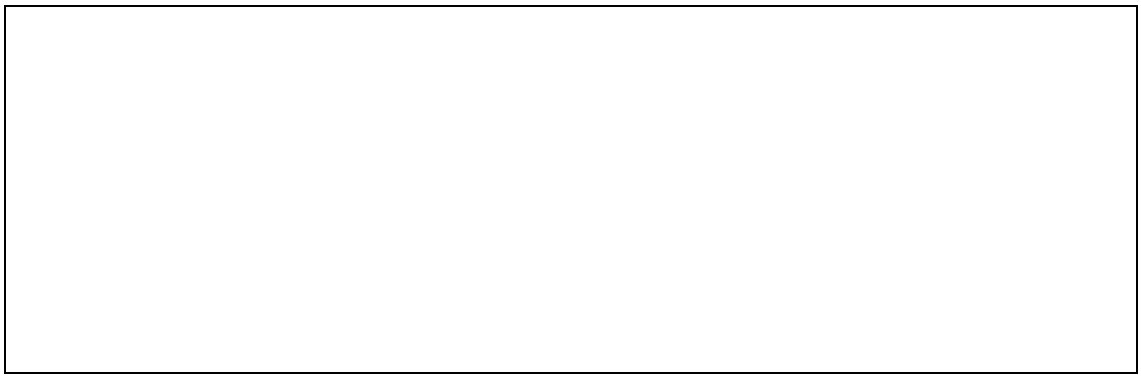


Figura 4.6: Ejemplo de región totalmente incluida en la zona errónea (b) que no se ve afectada por el Procesamiento Especial de esta zona (c).

- Por otro lado, puede ocurrir que la parte de una región dividida que cae dentro de la zona errónea no contenga ningún ejemplo erróneo (Figura 4.7.b), debido a que el perceptrón se entrena teniendo en cuenta a todas las regiones con ejemplos erróneos. En este caso, esta región tampoco se vería afectada por el Procesamiento Especial de la zona errónea (Figura 4.7.c).

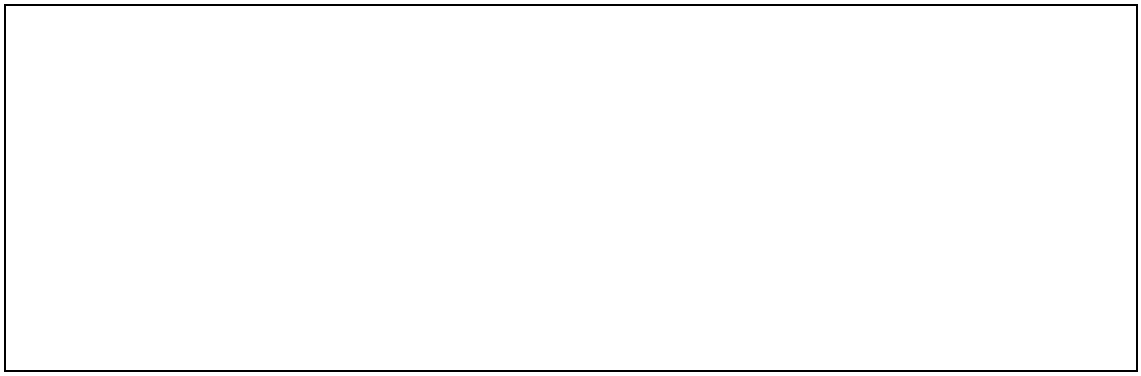


Figura 4.7: Ejemplo de parte de región dividida que cae dentro de la zona errónea (b) y que no se ve afectada por el Procesamiento Especial (c) al no tener ningún ejemplo erróneo en su interior.

- Si al entrenar un perceptrón binario para delimitar los ejemplos erróneos no se consigue que el Procesamiento Especial de la zona errónea influya en alguna parte del espacio de entrada del problema, bien porque no se ha logrado dividir ninguna region o porque, si se ha dividido alguna, la parte que ha caído en la zona errónea no tiene ejemplos erróneos, entonces volveríamos a entrenar al perceptrón sin tener en cuenta a la región con menor proporción de ejemplos erróneos. Este paso se repite hasta conseguir que el Procesamiento Especial influya en alguna parte del espacio o hasta que el perceptrón se entrene sobre los puntos de una sola región. En este último caso forzaríamos a que el perceptrón dividiera a los ejemplos de esta única región.

Para poder utilizar un perceptrón binario como elemento de procesamiento en un paso de SEPARATE, marcaremos los ejemplos erróneos con un *uno* y los no erróneos con un *cero*, mientras que los ejemplos pertenecientes a una región correcta *no se marcarán* para que no influyan en el entrenamiento del perceptrón. Si entrenamos un perceptrón sobre una sola región y no consigue dividirla, se repite el siguiente proceso hasta conseguir que el perceptrón divida a la región original:

1) Se selecciona, de la variable de entrada con mayor dominio, el valor intermedio de su intervalo de definición.

2) Con el valor seleccionado, se dividen los ejemplos de la región actual de entrenamiento del perceptrón en dos grupos: aquellos que tienen un valor en la variable con mayor dominio menor que el valor seleccionado y, aquellos cuyo valor es mayor o igual.

3) Se elige el grupo con mayor número de ejemplos erróneos y se vuelve a entrenar al perceptrón con los ejemplos de este grupo.

En Alg. 4.2 presentamos los *pasos* de esta implementación particular de SEPARATE con todas las características necesarias para su correcto funcionamiento. Podemos apreciar que en este algoritmo no hemos fijado las cuestiones referentes al sistema resolutor y al tipo de Procesamiento Especial. Estas cuestiones se resolverán en las secciones donde se conozca el tipo de problema a resolver.

Entrada: Problema de aproximación o de clasificación.
Salida: Sistema que resuelve el problema de entrada.
Condiciones iniciales: Sistema resolutor aplicado sobre todo el espacio de entrada del problema que ofrece una solución para cada ejemplo de entrenamiento.

1.- Entrenar un perceptrón binario para delimitar ejemplos erróneos.

2.- Si el perceptrón divide al menos una región del espacio de entrada, teniendo por lo menos un ejemplo erróneo en la parte que cae en la zona errónea, ir a 4, en caso contrario ir a 3.

3.- Olvidar la región con menor proporción de ejemplos erróneos e ir a 1, excepto

que el perceptrón se haya entrenado sobre una sola región en cuyo caso, volver a entrenar al perceptrón forzando la división de la región.

4.- *Realizar el Procesamiento Especial sobre las partes de las regiones divididas correctamente que caen en la zona errónea, o sea, diseñar un nuevo sistema resolutor para intentar resolver el problema en la zona errónea.*

5.- *Ajustar los sistemas resolutores del resto del espacio de entrada que está fuera de la zona errónea, sin incluir los sistemas resolutores que actúan sobre regiones correctas.*

6.- *Si alguna región no contiene ejemplos erróneos entonces marcarla como correcta.*

7.- *Si todas las regiones son correctas "Terminar", si no ir a 1.*

Algoritmo 4.2: Pasos de la implementación particular de SEPARATE con todas las características necesarias para su correcto funcionamiento.

4.3.2.-Representación por bloques de la implementación particular de SEPARATE

En esta sección presentamos otra forma de representar un sistema identificado con SEPARATE, que facilita el manejo de su información en un ordenador. En esta representación se distinguen tres tipos de bloques (Figuras 4.8 y 4.9):

- **Módulos resolutores**: intentan clasificar o aproximar correctamente los ejemplos de entrenamiento de una región (sistemas resolutores).

- **Nodos decisores**: son los perceptrones binarios que dividen con hiperplanos el espacio de entrada del problema.

- **Nodos índice**: indican a los ejemplos de entrada que nodo decisor consultar para poder moverse a través de la representación por bloques hasta alcanzar un módulo resolutor que le asigne un valor de salida.

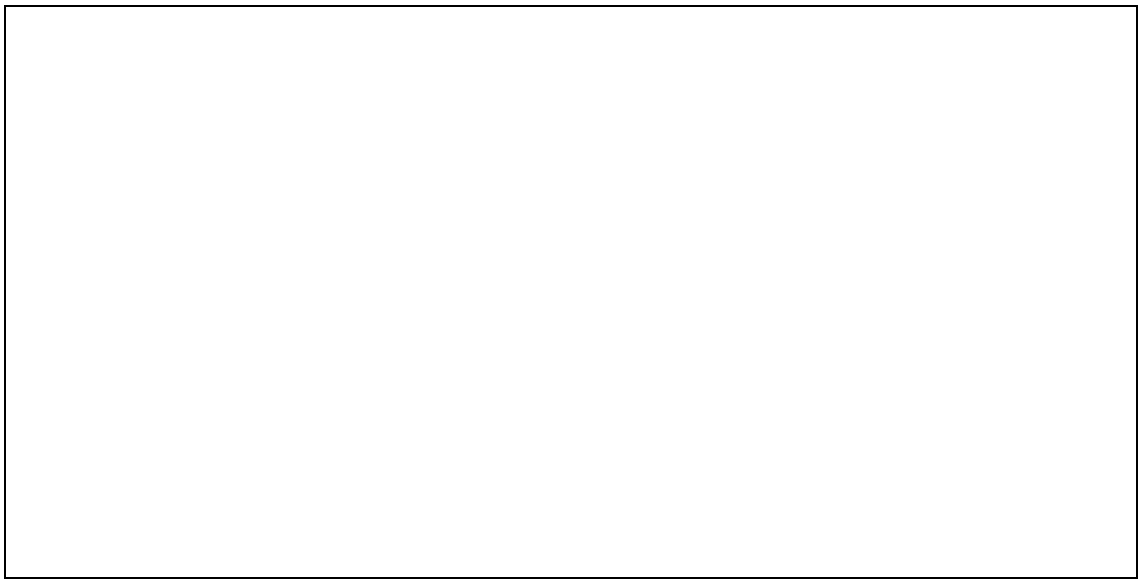


Figura 4.8: Ejemplo de la representación por bloques usada en la implementación de SEPARATE que tiene en cuenta las divisiones anteriores al realizar el Procesamiento Especial y el espacio asociado con esta representación.

Hay dos tipos de representaciones por bloques en función del tipo de Procesamiento Especial que realicemos en las zonas erróneas: si este Procesamiento Especial no tiene en cuenta las divisiones anteriormente realizadas, entonces los nodos índice que apuntan al mismo nodo decisor tienen un hijo común en la representación por bloques (Figura 4.9) y, si el Procesamiento Especial tiene en cuenta las divisiones anteriores, entonces cada nodo índice tiene sus propios descendientes en la representación (Figura 4.8). En la Figura 4.9 podemos apreciar como se representa una región no correcta que no se ve afectada por la división realizada por un perceptrón binario. En la figura, el perceptrón correspondiente al nodo decisor 3 se entrena sobre dos regiones con ejemplos erróneos, sin embargo, sólo consigue dividir a una de ellas. El nodo índice que representa a esta región no dividida pasa a apuntar al siguiente nodo decisor, en este caso, el nodo decisor 4.

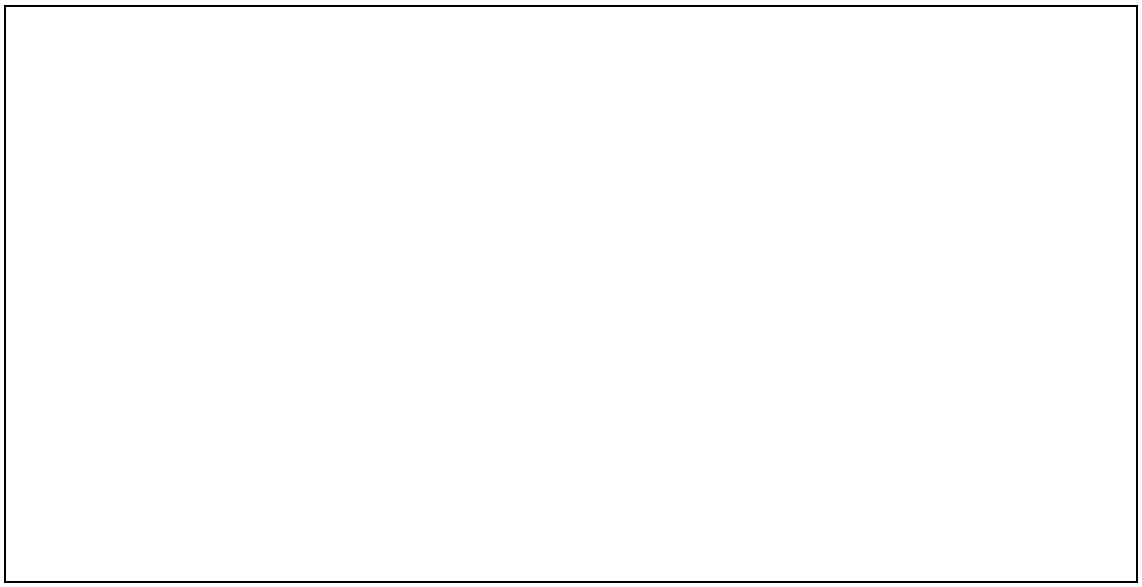
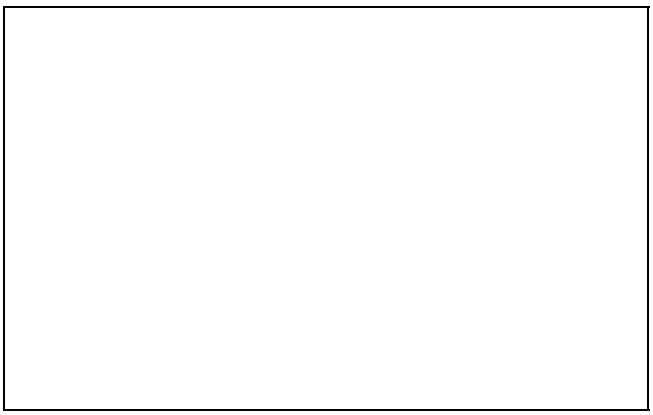


Figura 4.9: Ejemplo de la representación por bloques usada en la implementación de SEPARATE que no tiene en cuenta las divisiones anteriores al realizar el Procesamiento Especial y el espacio asociado con esta representación.

En la Figura 4.10 presentamos la resolución del problema de clasificación de la Figura 4.4.a usando SEPARATE junto con el proceso de construcción de la representación por bloques asociada con el sistema inteligente diseñado. En la Figura 4.10.a podemos apreciar la representación con el primer módulo resolutor que afecta a todo el espacio del problema. En la figura 4.10.b observamos la representación por bloques del sistema con la primera recta y los módulos resolutores que actúan sobre cada una de las regiones del espacio. En la Figura 4.10.c vemos la representación por bloques del sistema con las dos primeras rectas y los módulos resolutores asociados a las regiones creadas. Finalmente, en la figura 4.10.d observamos la representación por bloques del sistema con las tres rectas y los módulos resolutores que resuelven el problema de clasificación global.



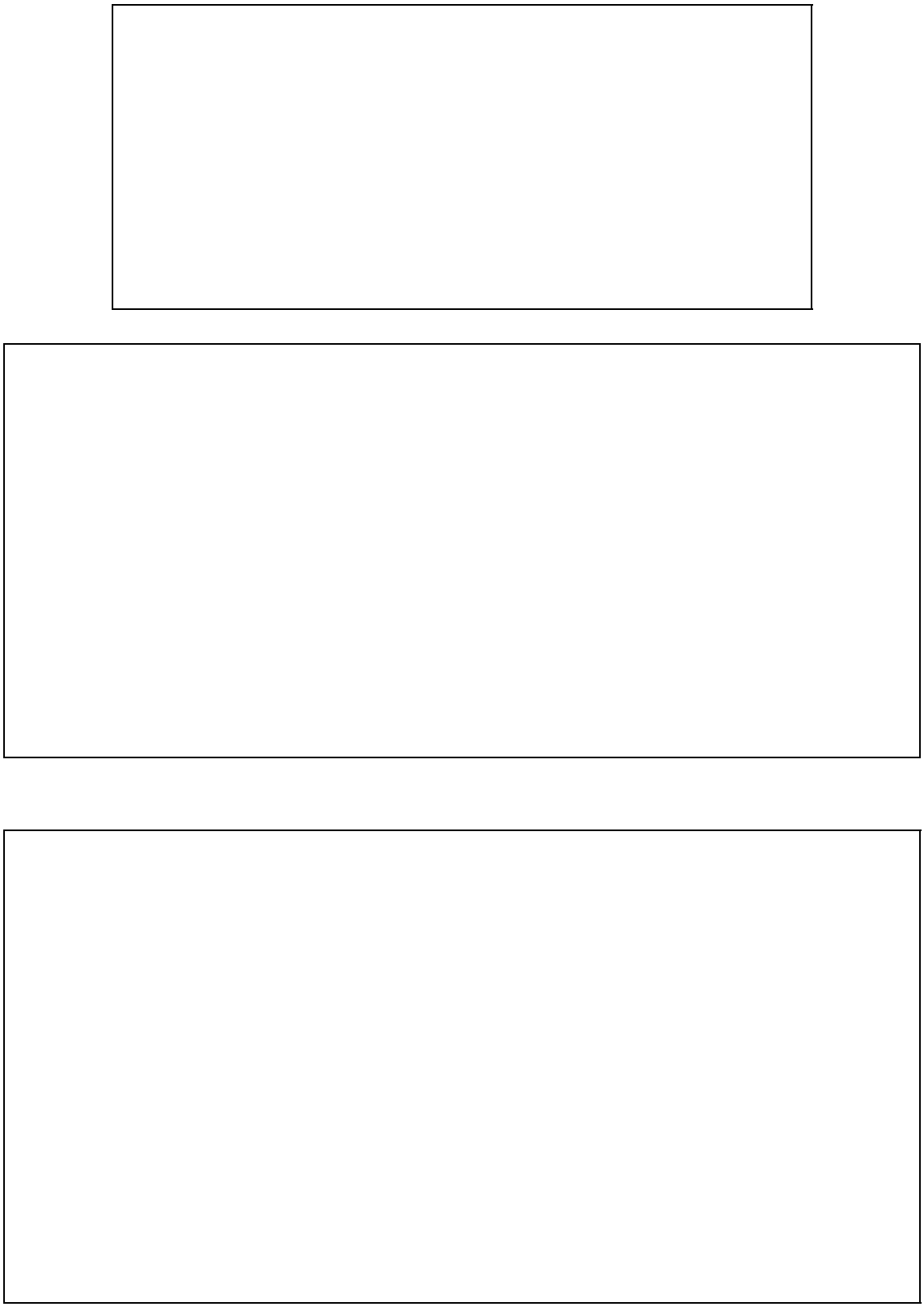


Figura 4.10: Resolución del problema de clasificación de la Figura 4.4.a junto con el proceso de construcción de la representación por bloques. a) Primer clasificador. b) Actuación de la primera recta y de los módulos resolutores en cada región. c) Actuación de las dos primeras rectas y de los módulos en las regiones creadas. d) Actuación de las tres rectas y de los módulos resolutores que resuelven el problema de clasificación global.

4.3.3.-Implementación para resolver problemas de aproximación

En esta sección vamos a presentar la implementación particular del método SEPARATE para diseñar sistemas inteligentes que resuelvan problemas de aproximación. En esta implementación, el *Procesamiento Especial* realizado en las zonas erróneas no tiene en cuenta las divisiones anteriores y los *módulos resolutores* consisten en una red neuronal hacia adelante con una sola capa oculta compuesta por dos neuronas y entrenada con el algoritmo de aprendizaje supervisado Backpropagation [Rumelhart86] (Apéndice IV). Como ya indicamos anteriormente los *elementos de procesamiento* de esta implementación de SEPARATE consisten en perceptrones binarios [Minsky69, Wasserman89] y el *consenso* entre salidas consiste en elegir la salida proporcionada por el último módulo resolutor diseñado.

Para poder realizar un estudio comparativo, también implementamos una variante del método de aprendizaje automático [Chiang94] que construye un *árbol de redes neuronales*. En la versión original de este método las regiones del espacio de entrada de un problema se van dividiendo localmente de tal forma que se delimiten, en una zona particular, los puntos con error mayor que el error cuadrático medio de la región, utilizando para ello una medida de correlación de error. En cada región se usa una red neuronal multicapa para aproximar sus puntos. Sin embargo, en la implementación de este método usada en esta sección, dividimos los puntos mediante un perceptrón binario y las redes de los nodos hoja tienen una sola capa oculta con dos neuronas, con el propósito de poder comparar este método con la implementación particular de SEPARATE. También usamos *redes neuronales con una capa oculta* para resolver globalmente un problema de aproximación, con el fin de comparar sus resultados con los de SEPARATE.

En los experimentos se han empleado simulaciones de las funciones F1, F2, F3 y F4 presentadas en el Apéndice I para probar los métodos de aproximación comentados anteriormente.

En todos los métodos de aproximación implementados, un punto obliga a ajustar los parámetros de un sistema durante su fase de entrenamiento, si su error es mayor que cierto umbral ε (por defecto, $\varepsilon=0.001$) y, un método finaliza cuando el error cuadrático medio de todos los puntos es menor que una constante β (por defecto, $\beta=0.02$).

En SEPARATE y en el árbol de redes neuronales, un punto es erróneo si su error es simultáneamente mayor que ε y que el error cuadrático medio de la región a la que pertenece, una región es correcta si su error cuadrático medio es menor que β, el número de ciclos en el entrenamiento de una red neuronal en una región es igual a 2000 y el número de ciclos para un perceptrón binario es igual a 1000. En las redes neuronales que actúan globalmente para resolver un problema, el número de ciclos de entrenamiento es igual a 10000.

En Alg. 4.3 mostramos el algoritmo de esta implementación particular de SEPARATE para resolver problemas de aproximación.

Entrada: Problema de aproximación.
Salida: Sistema que resuelve el problema de aproximación.
Condiciones iniciales: Red neuronal artificial aplicada sobre todo el espacio de entrada del problema que ofrece un valor de aproximación para cada punto de entrenamiento.

1.- *Entrenar un perceptrón binario para delimitar puntos erróneos.*

2.- *Si el perceptrón divide al menos una región del espacio de entrada, teniendo por lo menos un punto erróneo en la parte que cae en la zona errónea, ir a 4, si no ir a 3.*

3.- *Olvidar la región con menor proporción de puntos erróneos e ir a 1, excepto que el perceptrón se haya entrenado sobre una sola región en cuyo caso, volver a entrenar al perceptrón forzando la división de la región.*

4.- *Diseñar una red neuronal para aproximar los puntos de la zona errónea delimitada por el perceptrón.*

5.- *Volver a entrenar las redes neuronales de los puntos que caen fuera de la zona errónea y que no pertenecen a una región correcta.*

6.- *Si el error cuadrático medio de alguna región es menor que β, marcarla como correcta.*

7.- *Si todas las regiones son correctas "Terminar", si no ir a 1.*

Algoritmo 4.3: Algoritmo de la implementación particular de SEPARATE para resolver problemas de aproximación.

4.3.3.1.-Resultados

Los resultados que aparecen en las tablas de esta sección son la media de los valores de salida ofrecidos por cinco ejecuciones independientes de los métodos de aproximación implementados. La información que ofrecen las tablas está dividida en cuatro columnas:

1) Exactitud: Refleja el error cuadrático medio de los puntos de entrenamiento. Mide la exactitud de cada uno de los métodos de aproximación al manipular los puntos de entrenamiento.

2) Generalización: Recoge el error cuadrático medio de los puntos de prueba. Mide la capacidad de generalización de los métodos de aproximación.

3) Espacio necesario: Presenta el número de elementos que un sistema aproximador necesita almacenar. En SEPARATE y en los árboles de redes, el valor de esta columna es igual a la suma de dos términos: el número de nodos decisores (perceptrones) y el número de módulos resolutores (redes neuronales con una capa oculta compuesta por dos neuronas). En las redes neuronales globales, este valor corresponde con el número de neuronas en su capa oculta.

4) Tiempo de respuesta: Indica el número de elementos que deben activarse en el sistema aproximador para obtener una salida a partir de una entrada concreta y ofrece una medida aproximada del tiempo de respuesta del sistema. En las redes neuronales globales, este valor también coincide con el número de neuronas en su capa oculta. En SEPARATE y en los árboles de redes este valor es igual a la media de los elementos activados por cada posible camino en el árbol o en la representación por bloques de SEPARATE, o sea, este valor es igual a:

$$\frac{\Sigma_{nodos\_decisores}}{Numcaminos} + Numneuronas$$

donde $\Sigma_{nodos\_decisores}$ corresponde a la sumatoria de los nodos decisores activados al recorrer cada posible camino desde el nodo raíz a un módulo resolutor (nodo hoja), *Numcaminos* es el número de caminos posibles y *Numneuronas* es el número de neuronas en la capa oculta de la red neuronal que compone un módulo resolutor (en esta sección, este número de neuronas es igual a dos).

Para un problema de aproximación particular, usamos el valor "tiempo de respuesta" obtenido por el sistema diseñado con SEPARATE para fijar (de manera orientativa) el número de neuronas en la capa oculta de la red neuronal que intenta resolver globalmente este problema. Este paso es necesario ya que cuando se quiere usar una red neuronal para resolver un problema debemos indicar su topología de antemano.

En la Tabla 4.I podemos apreciar los resultados ofrecidos por los tres métodos aproximadores (SEPARATE, árboles de redes y redes neuronales) cuando aprenden los puntos de entrenamiento de la función F1, fijando los parámetros que controlan la precisión del sistema final a ε=0.001 y β=0.02. Observamos que todos los métodos resolvían fácilmente este problema. Por lo tanto, volvimos a repetir estos experimentos pero, en esta ocasión, exigiendo mayor precisión en los resultados (ε=0.00001 y β=0.0002). Las salidas ofrecidas se presentan en la Tabla 4.II. Podemos apreciar como es posible controlar la precisión sobre los ejemplos de entrenamiento en SEPARATE para conseguir una mayor exactitud.

		Exactitud	Generalización	Espacio necesario	Tiempo de respuesta
F1	SEPARATE	0.0122848	0.0128420	0.0+1.0=1.0	2.00
	Árbol de redes	0.0122900	0.0131708	0.0+1.0=1.0	2.00
	Red neuronal	0.0184000	0.0188450	2.0	2.00

Tabla 4.I: Resultados de los tres métodos aproximadores al aprender los puntos de entrenamiento de la función F1 con ε=0.001 y β=0.02.

		Exactitud	Generalización	Espacio necesario	Tiempo de respuesta
F1	SEPARATE	0.0001038	0.0002412	7.6+8.6=16.2	6.00
	Árbol de redes	0.0000562	0.0785804	244.2+245.2=489.4	20.36
	Red neuronal	0.0001760	0.0001900	6.0	6.00

Tabla 4.II: Resultados de los tres métodos aproximadores al aprender los puntos de entrenamiento de la función F1 con ε=0.00001 y β=0.0002.

En la Tabla 4.III presentamos los resultados ofrecidos por los tres métodos aproximadores al aprender los puntos de entrenamiento de las funciones F2 y F3, fijando ε=0.001 y β=0.02. Más adelante comentaremos estos resultados.

		Exactitud	Generalización	Espacio necesario	Tiempo de respuesta
F2	SEPARATE	0.0116446	0.0219600	9.0+10.0=19.0	6.97
	Árbol de redes	0.0011284	14.265536	353.4+354.4=707.8	14.76
	Red neuronal	0.0432710	0.0419862	7.0	7.0
F3	SEPARATE	0.0101590	0.0303850	17.4+18.4=35.8	9.17
	Árbol de redes	0.0034352	1.0258614	316.4+317.4=633.8	28.96
	Red neuronal	0.3307180	0.3746302	10.0	10.00

Tabla 4.III: Resultados de los tres métodos aproximadores al aprender los puntos de entrenamiento de las funciones F2 y F3 con ε=0.001 y β=0.02.

En la Tabla 4.IV aparecen los resultados sobre la función F4 con ε=0.001 y β=0.02. Podemos observar unos malos resultados de generalización tanto en SEPARATE como en el árbol de redes. Para evitar este problema, volvimos a repetir los experimentos pero esta vez exigiendo menos precisión sobre los ejemplos de entrenamiento (ε=0.1 y β=0.2). Los resultados obtenidos aparecen en la Tabla 4.V. Podemos apreciar que podemos utilizar el control sobre la precisión en SEPARATE para regular los resultados de generalización.

		Exactitud	Generalización	Espacio necesario	Tiempo de respuesta
F4	SEPARATE	0.0077708	0.6000542	146.0+147.0=293.0	21.18
	Árbol de redes	0.0027468	1.2062460	306.4+307.4=613.8	24.90
	Red neuronal	0.4302750	0.4997488	22.0	22.00

Tabla 4.IV: Resultados de los tres métodos aproximadores al aprender los puntos de entrenamiento de la función F4 con ε=0.001 y β=0.02.

		Exactitud	Generalización	Espacio necesario	Tiempo de respuesta
F4	SEPARATE	0.1342494	0.4091022	46.4+47.4=93.8	14.81
	Árbol de redes	0.1080942	1.1183344	132.4+133.4=265.8	22.76
	Red neuronal	0.4573924	0.5259092	15.0	15.00

Tabla 4.V: Resultados de los tres métodos aproximadores al aprender los puntos de entrenamiento de la función F4 con ε=0.1 y β=0.2.

		Exactitud	Generalización	Espacio necesario	Tiempo de respuesta
	SEPARATE	0.0001038	0.0002412	7.6+8.6=16.2	6.00

F1	Árbol de redes	0.0000562	0.0785804	244.2+245.2=489.4	20.36
	Red neuronal	0.0001760	0.0001900	6.0	6.00
F2	SEPARATE	0.0116446	0.0219600	9.0+10.0=19.0	6.97
	Árbol de redes	0.0011284	14.265536	353.4+354.4=707.8	14.76
	Red neuronal	0.0432710	0.0419862	7.0	7.0
F3	SEPARATE	0.0101590	0.0303850	17.4+18.4=35.8	9.17
	Árbol de redes	0.0034352	1.0258614	316.4+317.4=633.8	28.96
	Red neuronal	0.3307180	0.3746302	10.0	10.00
F4	SEPARATE	0.1342494	0.4091022	46.4+47.4=93.8	14.81
	Árbol de redes	0.1080942	1.1183344	132.4+133.4=265.8	22.76
	Red neuronal	0.4302750	0.4997488	22.0	22.00

Tabla 4.VI: Mejores resultados de los tres métodos aproximadores al aprender los puntos de entrenamiento de las funciones F1, F2, F3 y F4.

4.3.3.2.-Análisis de resultados

En la Tabla 4.VI presentamos los mejores resultados de cada método aplicado sobre cada función. A continuación, analizaremos estos resultados desde diversos puntos de vista (Diseño, Exactitud, Generalización, Espacio necesario y Tiempo de respuesta):

- Diseño: Esta es una de las mayores ventajas de SEPARATE frente al uso de redes neuronales que resuelven globalmente un problema. Mientras que SEPARATE posee la característica (extraída de los sistemas inteligentes basados en el modelo de árbol) de que él mismo define su propia estructura (cantidad y ordenación de sus elementos computacionales) durante la fase de diseño, en las redes neuronales debemos indicar, a priori, la topología de la red, esto es, debemos indicar el número de neuronas y sus interconexiones antes de entrenar a la red.

- Exactitud: En la Tabla 4.VI observamos que el árbol de redes es el método que consigue mayor exactitud sobre los ejemplos de entrenamiento. También podemos apreciar que los resultados de exactitud de los sistemas diseñados con SEPARATE son peores que los de los árboles pero, son bastante mejores que los obtenidos por las redes neuronales globales. Estos resultados de exactitud de las redes neuronales se deben a que tienen que realizar un ajuste global y simultáneo de todos sus elementos computacionales, impidiendo este hecho una buena exactitud sobre los ejemplos de entrenamiento. Por lo tanto, si se nos plantea un problema de aproximación y tenemos mucha confianza en los ejemplos de entrenamiento, tanto en consistencia (ejemplos sin ruido) como en completitud (los ejemplos abarcan todo el espacio del problema), usaremos un árbol de redes o SEPARATE para resolverlo.

- Generalización: Como podemos ver en la Tabla 4.VI, los resultados de los árboles de redes son muy malos en cuanto a generalización. Esto se debe a que se han especializado demasiado en los ejemplos de entrenamiento. Por otra parte, los sistemas diseñados por SEPARATE y las redes neuronales globales han conseguido unos buenos resultados en generalización. Los resultados de SEPARATE son un poco mejores que los de las redes neuronales debido a que hemos realizado un ajuste de la precisión de los resultados sobre los ejemplos de entrenamiento para obtener, finalmente, una buena generalización de SEPARATE. Sin embargo, podríamos haber realizado un estudio similar, ajustando la topología de la red, para mejorar los resultados en generalización de las redes. Por lo tanto, cuando queramos diseñar un sistema que generalice bien, habrá que decidir si es más sencillo ajustar la precisión en el entrenamiento de un sistema SEPARATE o ajustar la topología de una red neuronal.

- Espacio necesario: Como podemos apreciar en la Tabla 4.VI, esta característica es una gran ventaja de las redes neuronales globales, ya que almacenando pocos elementos computacionales (pocas neuronas), es capaz de conseguir unos resultados aceptables. Por otra parte, vemos que los árboles y los sistemas diseñados con SEPARATE necesitan almacenar una gran cantidad de elementos computacionales para conseguir unos buenos resultados. Por lo tanto, cuando el hecho de almacenar elementos computacionales (necesidad de memoria) sea un factor importante a la hora de elegir un método de aprendizaje automático para resolver un problema, elegiremos a las redes neuronales frente a los sistemas SEPARATE y los árboles.

- Tiempo de respuesta: Esta característica no es útil para comparar las redes neuronales globales y los sistemas diseñados con SEPARATE ya que hemos utilizado el valor de esta característica en los sistemas SEPARATE para determinar la topología de las redes. Sin embargo, podemos apreciar que los árboles de redes necesitan mucho tiempo para ofrecer una solución, debido a que han requerido mucha profundidad en el árbol para conseguir una buena precisión.

4.3.3.3.-Comentarios finales

Observando experimentalmente los buenos resultados en exactitud de los sistemas diseñados con SEPARATE, su facilidad de diseño (ellos definen su propia estructura) y su capacidad para controlar sus resultados en generalización (exigiendo mayor o menor precisión en el entrenamiento), consiguiendo de este modo resultados

comparables a los de las redes neuronales, parece aconsejable usar SEPARATE para identificar sistemas inteligentes.

Un posible inconveniente de SEPARATE es la cantidad de memoria requerida para guardar los sistemas identificados con él. Sin embargo, a pesar de tener que almacenar muchos elementos, hemos observado experimentalmente que los sistemas identificados con SEPARATE no requieren mucho tiempo para devolver una salida cuando se le presenta una entrada particular.

4.3.4.-Implementación para resolver problemas de clasificación

En esta sección presentamos la implementación particular del método SEPARATE para diseñar sistemas inteligentes que resuelven problemas de clasificación binaria, donde el *Procesamiento Especial* de las zonas erróneas no tiene en cuenta las divisiones anteriores, los *módulos resolutores* consisten en una red neuronal hacia adelante con una capa oculta compuesta por dos neuronas entrenada con *Backpropagation* (Apéndice IV), los *elementos de procesamiento* son perceptrones binarios [Minsky69, Wasserman89] y el *consenso* entre salidas consiste en elegir la salida proporcionada por el último sistema resolutor diseñado.

Hemos utilizado el problema de clasificación denominado PIMA (*www.ics.uci.edu/~mlearn/ MLRepository.html*, [Smith88]) para probar este método de identificación. El problema PIMA consiste en conocer si una persona es diabética o no a partir de ocho variables de entrada que corresponden con datos personales del paciente. Este problema consta de 768 ejemplos, que se dividen en dos conjuntos de 576 y 192 elementos que usaremos como conjunto de entrenamiento y de prueba, respectivamente.

El problema PIMA tiene la particularidad de obtener malos resultados en generalización. En este apartado, lo usaremos para probar la capacidad de SEPARATE en la obtención de buenos resultados en generalización controlando la exactitud exigida sobre los ejemplos de entrenamiento.

En SEPARATE, un ejemplo es erróneo si es mal clasificado y una región es correcta si no contiene ningún ejemplo mal clasificado. Marcaremos los ejemplos positivos (diabéticos) con un 1.0 y los negativos (no diabéticos) con un 0.0. Entrenamos las redes neuronales de los módulos resolutores si la salida producida por la red y la salida de clasificación deseada difieren en más de un umbral (por defecto, 0.2). Clasificamos un ejemplo como positivo si su salida es mayor que 0.5 y como negativo

en caso contrario. El número de ciclos de entrenamiento para una red neuronal en una región es igual a 2000 y el número de ciclos para un perceptrón binario es igual a 1000.

En Alg 4.4 mostramos el algoritmo de esta implementación particular de SEPARATE para resolver problemas de clasificación.

Entrada: Problema de clasificación.
Salida: Sistema que resuelve el problema de clasificación.
Condiciones iniciales: Red neuronal artificial aplicada sobre todo el espacio de entrada del problema que ofrece un valor de clase para cada ejemplo de entrenamiento.

1.- *Entrenar un perceptrón binario para delimitar los ejemplos mal clasificados.*

2.- *Si el perceptrón divide al menos una región del espacio de entrada, teniendo por lo menos un ejemplo mal clasificado en la parte que cae en la zona errónea, ir a 4, si no ir a 3.*

3.- *Olvidar la región con menor proporción de ejemplos mal clasificados e ir a 1, excepto que el perceptrón se haya entrenado sobre una sola región en cuyo caso, volver a entrenar al perceptrón forzando la división de la región.*

4.- *Diseñar una red neuronal para clasificar los ejemplos de la zona errónea delimitada por el perceptrón.*

5.- *Volver a entrenar las redes neuronales de los ejemplos que caen fuera de la zona errónea y que no pertenecen a una región correcta.*

6.- *Si alguna región no contiene ejemplos mal clasificados, marcarla como correcta.*

7.- *Si todas las regiones son correctas "Terminar", si no ir a 1.*

Algoritmo 4.4: Algoritmo de la implementación particular de SEPARATE para resolver problemas de clasificación.

4.3.4.1.-Resultados

Los resultados presentados en este apartado corresponden con la media de cinco ejecuciones independientes sobre particiones distintas de los ejemplos del problema PIMA. La información que ofrece las tablas de este apartado aparece repartida en dos columnas:

- Exactitud: contiene el tanto por ciento de aciertos sobre los ejemplos de entrenamiento.
- Generalización: presenta el tanto por ciento de aciertos sobre los ejemplos de prueba.

La Tabla 4.VII presenta los resultados al aplicar SEPARATE para clasificar los ejemplos del problema PIMA, con la condición de parar el proceso cuando se alcanza un tanto por ciento de aciertos τ sobre los ejemplos de entrenamiento (τ=79, τ=80, τ=81 y τ=82). En esta tabla podemos apreciar como disminuye la generalización conforme exigimos mayor exactitud al sistema diseñado con SEPARATE. Por lo tanto, si queremos diseñar un sistema con SEPARATE que generalice bien, habrá que estudiar cuál es la cota óptima de aciertos, exigida sobre los ejemplos de entrenamiento durante la fase de diseño, para conseguir los mejores resultados en generalización.

PIMA		
τ	Exactitud	Generalización
79	79.99	78.85
80	80.65	77.18
81	81.42	77.08
82	82.42	76.24

Tabla 4.VII: Resultados de los sistemas inteligentes para clasificar diseñados con SEPARATE al manipular los ejemplos de entrenamiento del problema PIMA. Interrumpimos el diseño de estos sistemas inteligentes cuando se alcanza un tanto por ciento de aciertos fijo (τ) sobre los ejemplos de entrenamiento.

La Tabla 4.VIII presenta los resultados alcanzados cuando hemos resuelto el problema PIMA con (1) el algoritmo C4.5 [Quinlan93] basado en un árbol de decisión, (2) una red neuronal hacia adelante con una sola capa oculta compuesta por seis neuronas, entrenada con *Backpropagation* (Apéndice IV), y (3) SEPARATE forzando la parada de su entrenamiento con un valor τ=79.

En esta tabla, podemos observar como los mejores resultados en generalización los obtienen los sistemas diseñados con SEPARATE, frente a la red neuronal que obtiene unos resultados aceptables tanto en exactitud como en generalización pero sin mejorar los obtenidos por SEPARATE. Por otra parte, el algoritmo C4.5 logra unos resultados muy buenos en exactitud pero, por contra, tiene unos malos resultados en generalización. Este es el comportamiento lógico, tanto de los sistemas inteligentes basados en el modelo de red neuronal (les cuesta alcanzar buenos resultados en exactitud pero obtienen unos resultados aceptables en generalización), como del algoritmo C4.5 que diseña sistemas basados en el modelo de árbol (obtienen resultados muy buenos en exactitud pero bastante malos en generalización). Por otro lado, vemos como un sistema diseñado con SEPARATE logra una situación intermedia entre ambos tipos de sistemas, alcanza unos resultados en exactitud mejores que los de la red neuronal y peores que los del algoritmo C4.5 pero, por contra, obtiene unos resultados en generalización mejores que los del algoritmo C4.5 e incluso mejores que los ofrecidos por la red neuronal.

	PIMA	
	Exactitud	Generalización
SEPARATE	79.99	78.85
Red Neuronal	78.26	76.03
C4.5	86.92	73.00

Tabla 4.VIII: Resultados ofrecidos por un sistema inteligente diseñado con SEPARATE forzando su parada en τ=79, por una red neuronal hacia adelante con una sola capa oculta compuesta por seis neuronas y por el algoritmo C4.5, cuando resuelven el problema PIMA.

4.3.4.2.-Análisis de resultados

Observando las pruebas experimentales realizadas con la implementación particular de SEPARATE para resolver problemas de clasificación, podemos deducir dos cosas:

1. La disponibilidad que ofrece la metodología SEPARATE de poder regular los resultados en generalización de los sistemas diseñados con ella, por medio del control sobre la exactitud requerida en los ejemplos de entrenamiento durante la fase de diseño. Por tanto, el diseño de un sistema inteligente con la metodología SEPARATE proporciona al diseñador un parámetro más para dirigir los resultados de su sistema y, además, este parámetro posee una regla básica de utilización: si se requiere mayor generalización debemos reducir la cota de aciertos y, si la

generalización es de menor importancia debemos aumentar la cota de aciertos sobre los ejemplos de entrenamiento.

2. La comprobación empírica de que los sistemas inteligentes diseñados con SEPARATE se encuentran en un lugar intermedio entre los sistemas basados en el modelo de red neuronal y los sistemas inteligentes basados en el modelo de árbol. Obtienen mejores resultados en exactitud que los sistemas inteligentes basados en el modelo de red neuronal, pero peores que los obtenidos por los sistemas basados en el modelo de árbol. Por contra, obtienen unos resultados en generalización mucho mejores que los alcanzados por los sistemas inteligentes basados en el modelo de árbol. Por lo tanto, si deseamos diseñar un sistema que ofrezca buenos resultados, tanto en generalización como en exactitud, lo realizaremos por medio de la metodología SEPARATE frente a su diseño con un algoritmo basado en un árbol, que no logra buenos resultados en generalización y, frente a una red neuronal que no puede alcanzar resultados en exactitud tan buenos como los obtenidos por un sistema diseñado con SEPARATE.

4.4.-Resumen y notas finales

En este capítulo hemos presentado la metodología de diseño SEPARATE con el objetivo de definir correctamente un mecanismo de iteración para MEGAS. Para llevar a cabo esto, hemos estudiado las propiedades de los sistemas inteligentes basados en el modelo de árbol (fáciles de construir, buenos resultados sobre los ejemplos de entrenamiento, mala generalización y alta sensibilidad ante el ruido), y de los sistemas inteligentes basados en el modelo de red neuronal (alta dificultad de construcción, peores resultados sobre los ejemplos de entrenamiento, buena generalización y buen comportamiento ante el ruido). De esta manera, hemos definido SEPARATE intentando arreglar los problemas de un tipo de sistema aprovechando las ventajas del otro y viceversa. SEPARATE diseña sistemas inteligentes que resuelven problemas de aproximación o de clasificación utilizando la técnica de diseño de algoritmos *Divide-y-Vencerás* y, permitiendo una actuación semiglobal de los elementos de un sistema que se encargan de dividir el espacio de entrada de un problema. Así, SEPARATE logra diseñar sistemas que son fáciles de construir, que alcanzan unos resultados exactos sobre los ejemplos de entrenamiento y que tienen un buen comportamiento respecto a la generalización.

Experimentalmente, aconsejamos el uso del método de diseño SEPARATE debido a varias razones:

- No hay que preocuparse por la organización de los elementos del sistema inteligente, ya que ellos mismo se ordenan en la fase de diseño, a diferencia de intentar resolver un problema con una red neuronal, donde hay que indicar a priori la topología de ésta.

- Podemos mejorar los resultados de generalización de los sistemas inteligentes diseñados por medio del control de la precisión requerida al manipular los ejemplos de entrenamiento de un problema. De esta manera, conseguimos una buena generalización, comparable a la de las redes neuronales y mejor que la de los sistemas inteligentes basados en el modelo de árbol.

- Sus buenos resultados en exactitud. Cuando tengamos unos ejemplos de entrenamiento consistentes y completos, podemos exigir una gran precisión al sistema diseñado teniendo que almacenar menos elementos que los sistemas inteligentes basados en el modelo de árbol.

Podemos apreciar que hemos definido el método SEPARATE a partir de componentes abstractos: elementos de procesamiento que particionan y módulos resolutores. Estos componentes pueden ser particularizados mediante diferentes herramientas en función del problema que vayamos a resolver y de su disponibilidad, pudiendo dar lugar a una gran variedad de sistemas inteligentes, cada uno con sus propias propiedades particulares, aunque conservando las características generales heredadas de SEPARATE. Así, podemos observar una de las ventajas de estudiar y diseñar algoritmos en el nivel de conocimiento definido por MORSE, o sea, en el nivel de la estructura: podemos definir metaalgoritmos, que proporcionan una serie de propiedades básicas a un sistema, los cuales se pueden concretar en varios algoritmos, que añaden propiedades particulares a un sistema en función de las necesidades del problema a resolver, además de las básicas heredadas del metaalgoritmo.

Finalmente, es destacable comentar que el método de aprendizaje automático que define la implementación particular de SEPARATE donde sus elementos de procesamiento son perceptrones binarios y los módulos resolutores son redes neuronales, lo podemos considerar como un algoritmo constructivo de redes neuronales. Este campo de estudio de la Inteligencia Artificial tiene un gran interés [Campbell95, Cios92, Fahlman90, Frean90, Littmann96, Mezard89, Moody96, Romaniuk93, Roy93, Tajine98, Zollner92], debido al gran uso de las redes neuronales y a su inconveniente referente al desconocimiento de que topología de red es la adecuada para resolver un problema.

Capítulo 5

SISTEMA HÍBRIDO PARA RAZONAMIENTO APROXIMADO

5.1.-Introducción

Una de las características más deseables para un sistema inteligente es que su funcionamiento sea comprensible para el usuario, bien porque se pueda expresar mediante reglas lingüísticas o bien porque su dinámica sea fácil de entender, ya que en este caso posee dos ventajas:

1. Su comportamiento puede ser validado por un experto del campo donde se aplica.

2. Si es necesario modificar el sistema porque las condiciones de actuación han cambiado o porque no funciona correctamente, podemos saber en que zona local es necesaria la modificación en vez de cambiar el sistema entero.

Parece pues interesante intentar la construcción de una particularización de MEGAS que sea capaz de diseñar sistemas inteligentes con la característica de que su procesamiento sea inteligible. Abordaremos este problema intentando diseñar sistemas inteligentes que sean fácilmente traducibles a un sistema basado en un tipo especial de reglas difusas. Además, estos sistemas inteligentes diseñados con un algoritmo basado en MEGAS deberán poseer la propiedad (heredada del metaalgoritmo) de lograr buenos resultados en aproximación.

En las siguientes secciones, comentaremos las propiedades de tres tipos de sistemas inteligentes de los cuales se pueden extraer reglas que explican su funcionamiento: sistemas basados en reglas difusas, árboles de decisión y redes neuronales artificiales. El objetivo es instanciar los módulos de MEGAS con estos sistemas inteligentes para conseguir definir un *sistema híbrido para razonamiento aproximado*. Presentaremos este sistema en la sección 5.3. A continuación, veremos como podemos transformar este sistema híbrido en un sistema basado en reglas (no convencionales), y como se puede utilizar su procedimiento de diseño para afinar el

conocimiento proporcionado por un experto. Finalmente, presentaremos algunos ejemplos del uso de este sistema híbrido.

5.2.-Sistemas inteligentes usados en el sistema híbrido

Para lograr definir un método de aprendizaje automático basado en MEGAS que identifique sistemas con un procesamiento inteligible y, teniendo en cuenta que MEGAS se define a partir de componentes generales que podemos instanciar de manera independiente, estudiamos en esta sección diversos sistemas inteligentes de los cuales se pueden extraer reglas, para ver como aprovechar sus ventajas y evitar sus inconvenientes en función de su utilización dentro de un sistema basado en MEGAS. Estos sistemas son:

- Sistemas basados en reglas difusas
- Árboles de decisión
- Redes neuronales artificiales.

5.2.1-Sistemas basados en reglas difusas

Los sistemas basados en reglas difusas (Apéndice II) se suelen utilizar para tareas de modelado, clasificación o control. Tienen la característica de estar constituidos por reglas difusas que usan variables lingüísticas [Zadeh75a, Zadeh75b, Zadeh75c], consiguiendo una dinámica directamente comprensible por cualquier persona. Presentan el inconveniente de requerir un número muy grande de reglas difusas para conseguir resultados precisos de aproximación.

Existen varios métodos para identificar sistemas basados en reglas difusas **[**Chiu94, Sugeno93, Sun94, Takagi85, Yager94**]**, normalmente a partir de un conjunto de ejemplos del comportamiento del sistema real a modelar. En la implementación presentada en este capítulo hemos usado el procedimiento para generar reglas difusas a partir de ejemplos presentado en [Wang92] (que denominaremos método *Wang-Mendel*), sin tener en cuenta el conocimiento de un experto. Este método es simple, su entrenamiento no requiere mucho tiempo y logra unos resultados aceptables.

Supongamos que disponemos de un conjunto de pares de datos entrada-salida, entonces los pasos para diseñar un sistema basado en reglas difusas de acuerdo al método Wang-Mendel [Wang92] son los siguientes:

1) Dividir los espacios de entrada y de salida en regiones difusas: el intervalo de influencia de cada variable de entrada y de salida se divide en 2N+1 regiones, denotadas por PN (Pequeño N), ..., P1 (Pequeño 1), CE (Centro), G1 (Grande 1), ..., GN (Grande N) y, se asocia una función de pertenencia a cada región. Por ejemplo, en la Figura 5.1 ilustramos la división del intervalo de influencia de la variable X en cinco regiones (N=2). En este caso, la forma de cada función de pertenencia es triangular.

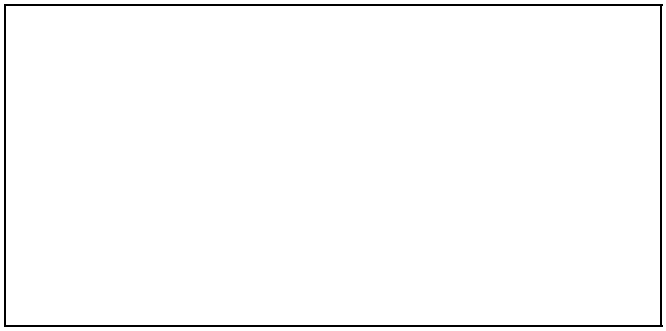


Figura 5.1: División del intervalo de influencia de la variable X en cinco regiones usando funciones de pertenencia triangulares.

2) Generar reglas difusas asociadas a cada par de entrada-salida: supongamos el par $[(x_1,x_2);y]$; primero, calculamos el grado de pertenencia de x_1, x_2 e y en las diferentes regiones. A continuación, asociamos x_1, x_2 e y con la región de máximo grado. Finalmente, obtenemos una regla de cada par de datos entrada-salida, por ejemplo:

$[(x_1,x_2);y]$ => [x_1 (0.8 en G1, max), x_2 (0.7 en P1, max); y (0.9 en CE, max)] =>

Regla: si X_1 es G1 $\wedge$ X_2 es P1 entonces Y es CE.

3) Asignar un grado a cada regla: supongamos la regla "*si X_1 es A $\wedge$ X_2 es B entonces Y es C*" y, que $A(x_1)$, $B(x_2)$ y C(y) son el grado de pertenencia de los componentes del par $[(x_1,x_2);y]$ a los conjuntos difusos A, B y C, entonces el grado de esta regla, denotado por D(regla), es igual a:

$$D(\text{regla}) = A(x_1)\, B(x_2)\, C(y).$$

Ya que usualmente hay muchos pares de datos y cada par genera una regla, es bastante probable que aparezcan reglas con el mismo antecedente pero con consecuente diferente. Resolvemos este conflicto aceptando sólo la regla con grado máximo.

4) Definir una aplicación entrada-salida a partir de la base de reglas difusas: con los tres pasos anteriores obtenemos una base de reglas difusas. A partir de unos valores de entrada particulares, determinamos el grado de activación de los antecedentes de cada regla por medio de una operación producto. A continuación, usamos el método de defuzzificación del *centroide* [Klir95] para calcular los valores de las variables de salida. Si *k* es el número de reglas difusas entonces el valor devuelto por el sistema para una variable de salida es:

$$y = \frac{\sum_{i=1}^{k} \text{gra}\,do(antecedente_i) \cdot \bar{y}_i}{\sum_{i=1}^{k} \text{gra}\,do(antecedente_i)},$$

donde *grado(antecedente$_i$)* es el grado de activación de los antecedentes de la regla *i* para una entrada particular, e $\bar{y}_i$ es el valor central del conjunto difuso asociado con la variable de salida Y en la regla *i*.

Con este procedimiento obtenemos una base de reglas difusas y un mecanismo para deducir una salida a partir de una entrada, definiendo de esta manera un sistema basado en reglas difusas. Si los conjuntos difusos de las reglas en este sistema son triangulares, con el objetivo de hacerlas más comprensibles, podemos sustituir las proposiciones *<variable> es A* por las proposiciones *<variable> es aproximadamente* $\bar{a}$, donde $\bar{a}$ es el valor central del conjunto difuso *A*. Por ejemplo, la regla:

si X_1 es A y X_2 es B entonces Y es C,

se sustituye por:

si X_1 es aproximadamente $\bar{a}$ y X_2 es aproximadamente $\bar{b}$
entonces Y es aproximadamente $\bar{c}$.

El cambio de formato de proposiciones anterior es correcto excepto para las proposiciones *<variable> es Pequeño N* y *<variable> es Grande N* de las reglas difusas obtenidas. En este caso, sustituimos la proposición difusa

<variable> es Pequeño N

por la proposición

<variable> es aproximadamente más pequeño que $\bar{s}$

donde $\bar{s}$ es el valor más grande en el conjunto difuso *Pequeño N* con un valor de pertenencia igual a uno y, sustituimos la proposición difusa

<variable> es Grande N

por la proposición

<variable> es aproximadamente más grande que $\bar{b}$

donde $\bar{b}$ es el valor más pequeño en el conjunto difuso *Grande N* con valor de pertenencia igual a uno.

En Alg. 5.1 presentamos el algoritmo que describe el método Wang-Mendel [Wang92] para generar una base de reglas difusas a partir de ejemplos usado en este capítulo.

Entrada: Pares entrada-salida.
Salida: Base de reglas difusas de un sistema que aproxima el comportamiento de los ejemplos de entrada.

1.-Dividir los espacios de entrada y de salida en regiones difusas, asociando una función de pertenencia a cada región.

2.-Generar una regla difusa por cada par entrada-salida.

3.-Asignar un grado a cada regla, que será igual al producto de los grados de pertenencia a las regiones difusas de los términos del par entrada-salida con el que se formó la regla.

4.-Si hay varias reglas que entran en conflicto entonces

4.1.-*Seleccionar la que tiene mayor grado y Descartar el resto.*

Algoritmo 5.1: algoritmo que describe el método Wang-Mendel para generar una base de reglas difusas a partir de ejemplos.

5.2.2.-Árboles de decisión

Los árboles de decisión [Breiman84, Quinlan86, Quinlan93, Sirat90] son un tipo de sistema inteligente empleado normalmente para resolver problemas de clasificación. Su procedimiento de diseño se basa en dividir el espacio de entrada de un problema en varias regiones e intentar resolver el problema en cada región, usando herramientas para dividir correctamente el espacio de un problema. Poseen la ventaja de que logran unos resultados sobre los ejemplos de entrenamiento muy exactos pero, por contra, tienen unos resultados en generalización muy malos.

Más concretamente, un árbol de decisión es un árbol donde cada nodo corresponde con una variable de entrada y cada arco con un posible valor de esa variable. Una hoja del árbol especifica el valor de salida asociado con los valores de entrada descritos en el camino desde la raíz a esta hoja. Un árbol de decisión es tanto más bueno cuanto más informativa es la variable de entrada que se asocia a un nodo.

Es fácil extraer un conjunto de reglas de un árbol de decisión, basta escribir una regla para cada camino desde la raíz a una hoja. Los antecedentes de estas reglas se construyen fácilmente a partir de las etiquetas de los nodos y las de los arcos, y los consecuentes corresponden con los valores de salida representados en las hojas del árbol. Por ejemplo, podemos extraer las siguientes tres reglas del árbol de decisión de la Figura 5.2:

Si $X_1 \leq a_0$ y $X_2 \leq a_1$ entonces $Y=C_1$.
Si $X_1 \leq a_0$ y $X_2 > a_1$ entonces $Y=C_2$.
Si $X_1 > a_0$ entonces $Y=C_3$.

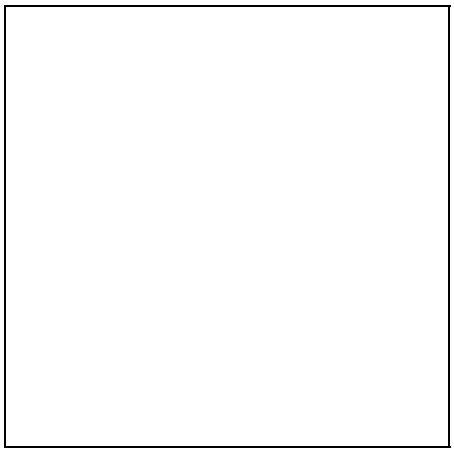


Figura 5.2: Ejemplo de un árbol de decisión.

A continuación, describimos el algoritmo ID3 [Quinlan86] y el manejo de variables continuas utilizado en C4.5 [Quinlan93], que constituyen el procedimiento usado en este capítulo para construir un árbol de decisión.

En el algoritmo ID3 [Quinlan86], la *entropía* [Abramson63] se utiliza para medir lo informativo que es un nodo. Una variable de entrada se elige en un nodo con el propósito de maximizar la diferencia de información necesaria para identificar la clase de un ejemplo, antes y después de que el valor de la variable de entrada se haya obtenido, esto es, maximizar la ganancia en información debido a esa variable de entrada.

Supongamos N clases C_1, C_2, ..., C_N y un conjunto de ejemplos, entonces la entropía de este conjunto es igual a:

$$E(conjunto) = -\sum_{i=1}^{N}(p(c_i)\cdot \log_2\ p(c_i)),$$

donde $p(C_i)$ es la probabilidad de que un ejemplo pertenezca a la clase C_i.

Por otro lado, la entropía después de conocer el valor de la variable de entrada X entre sus posibles valores x_1, ...,x_M es igual a:

$$E(conjunto / X) = \sum_{j=1}^{M} p(x_j)\cdot(-\sum_{i=1}^{N}(p(C_i / x_j)\cdot \log_2\ p(C_i / x_j))),$$

donde $p(x_j)$ es la probabilidad de que un ejemplo tenga el valor x_j en la variable de entrada X y, $p(C_i/x_j)$ es la probabilidad de que un ejemplo con el valor x_j en la variable de entrada X pertenezca a la clase C_i.

Finalmente, la ganancia en información debido a la variable de entrada X es igual a:

$$Ganancia(conjunto, X) = E(conjunto) - E(conjunto / X).$$

El algoritmo ID3 trabaja con variables de entrada discretas. La capacidad para manejar variables de entrada continuas la tomamos del algoritmo C4.5 [Quinlan93]. Supongamos que la variable de entrada X es continua y que los valores para esta variable de entrada en el conjunto de entrenamiento son, en orden creciente, A_1, A_2, ..., A_M, entonces para cada valor A_j, j=1, ..., M, los ejemplos de entrenamiento son divididos en dos conjuntos: los ejemplos con valores para X más bajos o iguales a A_j y

aquellos con valores más grandes que A_j. Para cada una de estas particiones, calculamos la ganancia y elegimos la partición que la maximiza.

Podemos apreciar que un árbol de decisión divide el espacio de entrada de un problema en varias regiones y asocia un valor de salida a cada región. En Alg. 5.2 presentamos, en forma de algoritmo, el procedimiento de diseño de un árbol de decisión usado en este capítulo.

Entrada: Pares entrada-clase, variables de entrada disponibles.
Salida: Un árbol de decisión que clasifica los ejemplos de entrada.

1.-Si todos los ejemplos de entrada son de la misma clase entonces

1.1.-Marcar este nodo con el valor de la clase de los ejemplos y Salir.

2.-Si el conjunto de variables de entrada disponibles es vacío entonces

2.1.-Marcar este nodo con la clase mayoritaria en los ejemplos de entrada y Salir.

3.-Elegir la variable de entrada con su valor asociado que maximiza la ganancia de información.

4.-Dividir los ejemplos de entrada en dos conjuntos, en función de la variable de entrada y de su valor seleccionados en el paso anterior.

5.-*Llamar a este algoritmo con cada uno de los dos conjuntos de ejemplos de entrada obtenidos en el paso 4, eliminando la variable de entrada seleccionada en el paso 2 del conjunto de variables de entrada disponibles.*

Algoritmo 5.2: Procedimiento de diseño de un árbol de decisión en forma de algoritmo.

5.2.3.-Redes neuronales artificiales

Las redes neuronales artificiales son modelos computacionales que surgieron con el objetivo de formalizar matemáticamente la estructura del cerebro [Krose93, Muller90, Simpson89]. Actualmente se consideran como modelos de cálculo (máquinas

aritméticas) capaces de desarrollar tareas cognitivas como aprendizaje, clasificación u optimización. Además, su funcionamiento, caracterizado por una operación masivamente paralela, las hace muy eficientes. Su principal ventaja es la existencia de algoritmos de entrenamiento que les permiten lograr buenos resultados en aproximación. Tienen el inconveniente de su consideración de "caja negra", o sea, un sistema que realiza eficazmente una tarea pero no se sabe cómo. Recientemente, se están estudiando métodos para extraer conocimiento de las redes neuronales, en concreto, en [Benítez97] se ha logrado comprender el funcionamiento de un tipo de red neuronal artificial (redes neuronales hacia adelante con una capa oculta) por medio del uso de reglas difusas.

En nuestro trabajo empleamos redes neuronales artificiales hacia adelante con una capa oculta [Wasserman93, Lippmann87], entrenadas con *backpropagation* [Rumelhart86] (Apéndice IV). En la Figura 5.3 ilustramos un ejemplo de este tipo de red neuronal artificial. Esta red tiene *n* entradas (x_1, ..., x_n), *h* neuronas ocultas (z_1, ..., z_h) y *m* neuronas de salida (y_1, ..., y_m). Sea τ_j la tendencia para la neurona z_j, w_{ij} el peso de la conexión entre la entrada x_i y la neurona z_j y, β_{jk} el peso de la conexión entre la neurona z_j y la neurona y_k. Los cálculos realizados por esta red son:

$$z_j = f\left(\sum_{i=1}^{n}(x_i w_{ij}) + \tau_j\right) \quad \text{y} \quad y_k = \sum_{j=1}^{h}(z_j \beta_{jk})$$

donde *f* es la función de activación sigmoide:

$$f(x) = \frac{1}{1+e^{-x}}.$$

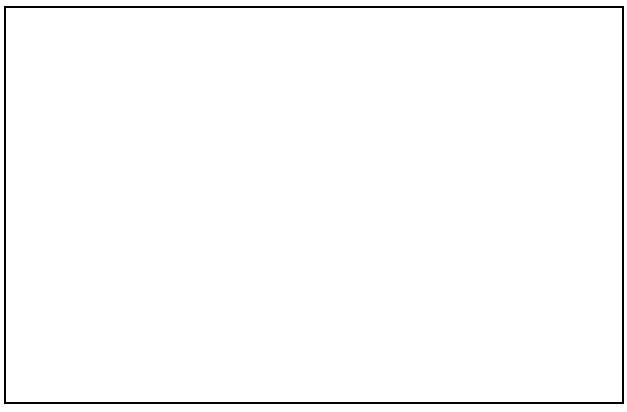


Figura 5.3: Arquitectura de una red neuronal multicapa hacia adelante con una capa oculta.

Podemos extraer directamente reglas, que caracterizan a un sistema difuso aditivo [Kosko92, Kosko94] (Apéndice II), a partir de este tipo de red neuronal artificial

una vez entrenada [Benítez97]. Por ejemplo, en la red neuronal de la Figura 5.3, para cada conexión entre una neurona oculta y una neurona de salida, deducimos una regla con el siguiente formato:

$$R_{jk}\text{: si } X_1 \text{ es } A^1_{jk} \text{ i-or } X_2 \text{ es } A^2_{jk} \text{ i-or ... i-or } X_n \text{ es } A^n_{jk} \text{ entonces } Y_k = \beta_{jk},$$

donde A^i_{jk} son los conjuntos difusos obtenidos de *f*, w_{ij} y z_j. Su función de pertenencia se calcula como:

$$\mu_{A^i_{jk}}(x) = f(x \cdot w_{ij} + \frac{\tau_j}{n})$$

e *i-or* es un operador lógico (*i-or (0,1)→(0,1)*) definido como:

$$i-or(a,b) = \frac{a \cdot b}{(1-a)\cdot(1-b) + a \cdot b}.$$

La proposición X_i *es* A^i_{jk} y el operador *i-or* tienen una interpretación lógica:

- Si el término $\frac{f^{-1}(0.9) - \frac{\tau_j}{n}}{w_{ij}} = B^i_{jk}$ es positivo, podemos leer la proposición X_i *es* A^i_{jk} como X_i *es más grande que aproximadamente* B^i_{jk}.

- Si el término $\frac{f^{-1}(0.9) - \frac{\tau_j}{n}}{w_{ij}} = B^i_{jk}$ es negativo, podemos leer la proposición X_i *es* A^i_{jk} como X_i *no es más grande que aproximadamente* $-B^i_{jk}$.

- Por otro lado, el operador i-or se interpreta como un promedio entre valores en (0,1), que tiene como elemento neutro al 0.5 y que posee la propiedad de que un valor es más influyente en el resultado cuanto más alejado está del 0.5. Por ejemplo, la decisión de un editor acerca de la publicación de un artículo científico, a partir de la evaluación del artículo realizada por dos revisores que proporcionan un número en (0,1), es similar al procesamiento ejecutado por el operador i-or con estos dos números: cuanto más extrema es la evaluación de un revisor (aceptar-1, rechazar-0), más influyente es su opinión y, cuanto más cerca está su evaluación de 0.5, menos influyente es su opinión.

5.3.-Sistema híbrido para razonamiento aproximado

En esta sección vamos a definir una particularización de MEGAS, orientada a la definición de un método de aprendizaje automático para identificar sistemas inteligentes con procesamiento inteligible y buenos resultados de aproximación. Para definir este método de aprendizaje automático vamos a utilizar los tres tipos de sistemas inteligentes, que son fácilmente traducibles a sistemas basados en reglas, presentados en las secciones anteriores: los sistemas basados en reglas difusas, los árboles de decisión y las redes neuronales artificiales hacia adelante con una sola capa oculta.

Recordemos los componentes básicos de un sistema identificado con MEGAS y, de esta manera, comentemos que tipo de sistema inteligente es el más adecuado para instanciar cada componente:

- Un primer módulo que proporciona una solución aproximada al problema de manera rápida. En este nivel parece razonable el uso de un *sistema basado en reglas difusas* con pocas reglas, aunque no ofrezca una solución exacta. Podemos comprender su funcionamiento por medio de las reglas difusas y mejorar sus resultados de aproximación con los siguientes componentes de MEGAS. En concreto, usaremos el método Wang-Mendel para identificar un sistema basado en reglas difusas por las ventajas anteriormente comentadas: simple, entrenamiento rápido y resultados aceptables.

- Un módulo que delimita los ejemplos erróneos del anterior sistema. En este paso no se ataca directamente la resolución del problema y se divide el espacio de entrada del problema. Estos dos hechos hacen aconsejable la instanciación de este componente mediante un árbol de decisión: por una parte, no se intenta resolver el problema con lo que evitamos los malos resultados en generalización de los árboles y, por otro lado, el árbol de decisión es capaz de dividir el espacio de entrada de un problema en regiones que tienen mayoritariamente ejemplos incorrectos o correctos. Además, podemos simular por medio de reglas la acción realizada por un árbol.

- Varios módulos que ajustan localmente en cada región la salida proporcionada por el sistema. Parece aconsejable que estos componentes sean *redes neuronales artificiales*, ya que constituye el último paso antes de dar la salida final del sistema y, por ello, deben alcanzar los mejores resultados posibles en aproximación. Además, actúan localmente sobre un número reducido de ejemplos por lo que su topología no debe ser muy compleja. En concreto, podemos usar redes con una capa oculta (no muy grande) de las que se pueden extraer reglas no estándar (no muchas al ser la topología simple) que expliquen su funcionamiento.

Así, el procedimiento para identificar el sistema híbrido para razonamiento aproximado [Castro99e], que mostramos en Alg 5.3 en forma de algoritmo, consiste en los siguientes pasos:

1. Diseñamos un sistema basado en reglas difusas usando el método Wang-Mendel [Wang92]. De esta manera, obtenemos un conjunto de reglas difusas que proporcionan una salida aproximada para cada entrada presentada al sistema.

2. Delimitamos los puntos con un error más grande que un umbral dado β. Llevamos a cabo esta tarea por medio de un árbol de decisión que clasifica cada punto como correcto o incorrecto. Para cada entrada que se presente al sistema, este árbol devuelve la hoja alcanzada e información acerca de si la región asociada con esta hoja tiene puntos incorrectos.

3. Entrenamos una red neuronal artificial con una capa oculta dentro de cada región con puntos incorrectos con el fin de aproximar el error producido por el sistema basado en reglas difusas.

Entrada: Un problema.
Salida: Un sistema híbrido que resuelve el problema de entrada.

1.-Diseñar un sistema basado en reglas difusas para resolver el problema, usando el método Wang-Mendel.

2.-Marcar los puntos con un error más grande que β en el sistema basado en reglas difusas como incorrectos y el resto como correctos.

3.-Construir un árbol de decisión para clasificar un punto como correcto o incorrecto, usando como puntos de entrenamiento a los puntos marcados en el paso anterior.

4.-En cada región del espacio de entrada del problema con puntos incorrectos, que ha creado el árbol de decisión, hacer:

4.1.-*Entrenar una red neuronal artificial hacia adelante con una sola capa oculta con el objetivo de aproximar el error producido por el sistema*

basado en reglas difusas al manipular los puntos de la región.

Algoritmo 5.3: Procedimiento de diseño de un sistema híbrido para razonamiento aproximado en forma de algoritmo.

El procedimiento anterior construye la arquitectura del sistema híbrido, que ilustramos en la Figura 5.4 mediante un modelo MORSE. Cuando introducimos una entrada X en esta arquitectura, se activa el sistema basado en reglas difusas (proporcionando la salida Y_1) y el árbol de decisión. A continuación, si la hoja del árbol donde la entrada X finalizó (hoja(X)) está asociada a una región con puntos incorrectos, se activa la red neuronal artificial que actúa sobre esta región, proporcionando la salida Y_{local}. Finalmente, la salida global proporcionada por el sistema híbrido es igual a:

$$SALIDA = \begin{cases} Y_1 + Y_{local\_1} & Si\ hoja(X)\ está\ asociada\ a\ región\ 1\ con\ puntos\ incorrectos. \\ & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ Y_1 + Y_{local\_p} & Si\ hoja(X)\ está\ asociada\ a\ región\ p\ con\ puntos\ incorrectos. \\ Y_1 + 0.0 & En\ otro\ caso. \end{cases}$$

Como podemos apreciar, el consenso entre salidas definido por MEGAS, consiste en la suma de las salidas locales y la salida global. Por esto, las redes neuronales locales (sistemas locales) aproximan el error del sistema basado en reglas difusas (sistema global).

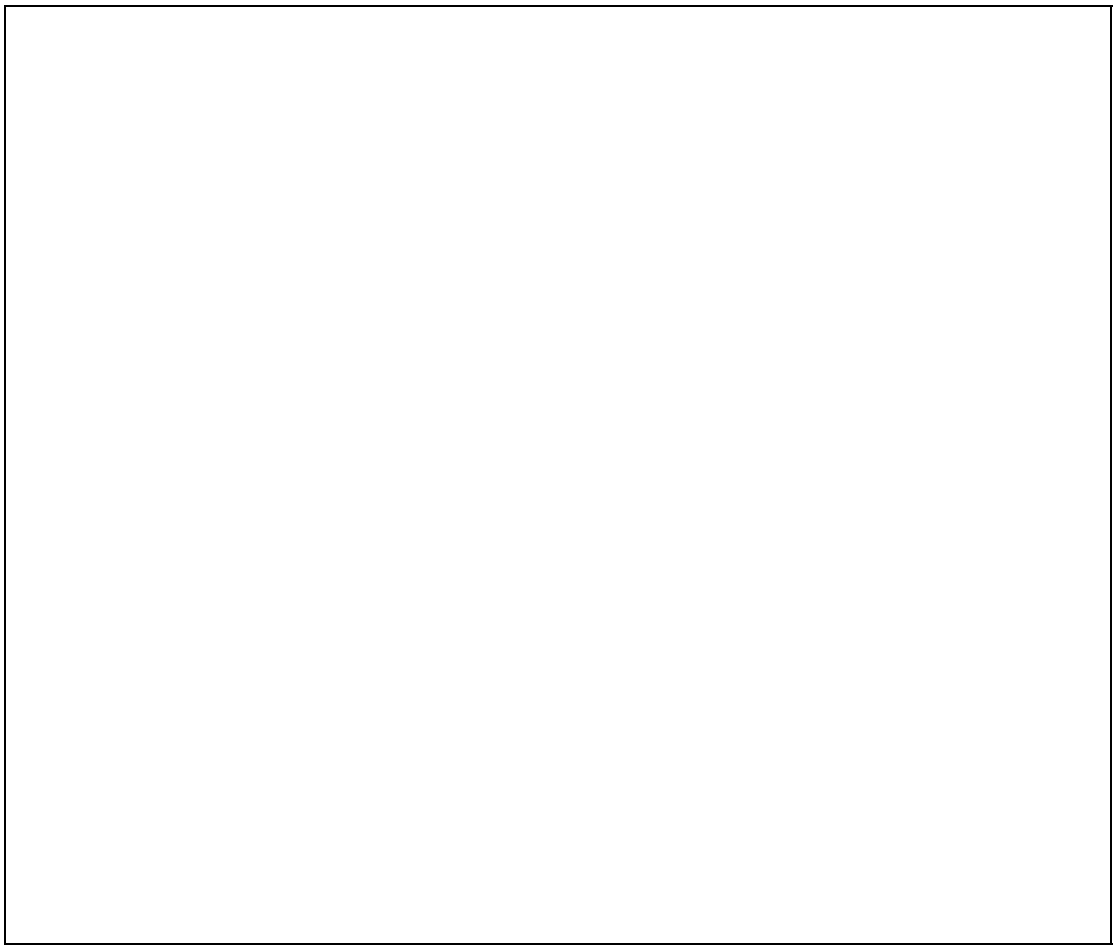


Figura 5.4: Arquitectura de un sistema híbrido compuesto por un sistema basado en reglas difusas, un árbol de decisión y redes neuronales. Representada con un modelo MORSE.

El procesamiento de este sistema, construido de acuerdo a la filosofía de MEGAS, es fácilmente inteligible, ya que podemos traducir directamente a reglas la dinámica de todos los sistemas que lo componen (sistemas basados en reglas difusas, árboles de decisión y redes neuronales multicapa hacia adelante con una capa oculta) y, la estrategia que controla el uso de estos módulos (generar salida aproximada, detectar zona local de la entrada y producir salida local “si procede” que será sumada a la salida aproximada) es comprensible. En la siguiente sección explicamos el paso de la arquitectura de la Figura 5.4 a un sistema basado en reglas.

5.4.-Interpretación del sistema híbrido

En esta sección, comentamos como podemos transformar el sistema híbrido para razonamiento aproximado en un sistema basado en un tipo especial de reglas, consiguiendo, de esta manera, comprender la acción realizada por el sistema híbrido.

Para transformar el sistema híbrido en un sistema basado en reglas que sea fácilmente comprensible, definiremos un nuevo formato de regla con las siguientes peculiaridades:

- Su consecuente consiste en una base de reglas difusas proveniente de la aplicación del método Wang-Mendel [Wang92] sobre un conjunto de pares entrada-salida o, consiste en una base de reglas difusas i-or extraída de una red neuronal artificial hacia adelante con una capa oculta ya entrenada.

- Su antecedente será una condición lógica, normalmente extraída de un árbol de decisión al recorrer un camino desde la raíz hasta una hoja del árbol.

Más concretamente, las reglas del sistema que extraemos de un sistema híbrido tienen el siguiente formato:

si <condición lógica > entonces activar <conjunto de reglas>,

donde:

- <*Condición lógica* > es "Verdad" si <conjunto de reglas> corresponde con la base de reglas difusas derivada del sistema basado en reglas difusas.
- <*Condición lógica* > corresponde con el antecedente de las reglas derivadas del árbol de decisión si <conjunto de reglas> es un conjunto de reglas difusas i-or derivadas de una red neuronal.
- *Activar <conjunto de reglas>* consiste en disparar todas las reglas del conjunto y devolver una salida.

De esta manera, el sistema basado en reglas (no convencionales) derivado de la arquitectura presentada en la Figura 5.4 está compuesto por las siguientes reglas:

◊ Si *Verdad* entonces activar *reglas difusas derivadas del sistema basado en reglas difusas.*

◊ Si *Condiciones lógicas del árbol de decisión para alcanzar la hoja asociada a la región 1 con puntos incorrectos* entonces activar *reglas difusas i-or derivadas de la red neuronal 1*.

.....

◊ Si *Condiciones lógicas del árbol de decisión para alcanzar la hoja asociada a la región p con puntos incorrectos* entonces activar *reglas difusas i-or derivadas de la red neuronal p.*

Cuando introducimos una entrada particular en este sistema basado en reglas, la salida producida es igual a la suma de las salidas proporcionadas por las reglas que han disparado su parte *Entonces*. Podemos observar que para cada entrada, como mucho, deben dispararse totalmente dos reglas: la derivada del sistema basado en reglas difusas y, algunas veces, una corrección adicional representada mediante una regla derivada de una red neuronal.

La interpretación del sistema híbrido por medio de reglas permite que comprendamos su acción y, de este modo, si es necesario modificar el sistema porque no funciona correctamente o porque sus condiciones de ejecución han cambiado, podemos deducir el área local de la modificación.

5.5.-Empleo del sistema híbrido para afinar el conocimiento proporcionado por un experto

En muchas aplicaciones en las que es necesario construir un sistema inteligente para realizar una tarea, sólo se dispone del conocimiento expresado por un experto (normalmente, en forma de reglas) acerca de como se realiza actualmente la tarea y, de un conjunto de ejemplos que corresponden con entradas para la tarea y resultados obtenidos tras llevarla a cabo. En esta sección vamos a mostrar como, a partir de esta situación, podemos usar el procedimiento de construcción del sistema híbrido para afinar el conocimiento expresado por el experto, logrando identificar un sistema inteligente cuya acción sigue siendo fácilmente comprensible.

Supongamos que tenemos el conocimiento proporcionado por el experto expresado mediante una base de reglas. La idea para afinar este conocimiento mediante el sistema híbrido consiste en colocar en el primer módulo, donde debería aparecer un sistema basado en reglas difusas, a un sistema cuya acción se base en las reglas proporcionadas por el experto. A continuación, se termina de construir el sistema híbrido usando los ejemplos entrada-salida que se dispone acerca de la tarea. En resumen, para diseñar un sistema híbrido que afine el conocimiento proporcionado por un experto, se deben realizar los siguientes pasos:

1.- Construir un sistema cuyo funcionamiento sea dirigido por las reglas que reflejan el conocimiento del experto.

2.- Marcar los ejemplos que, al introducir su entrada en el sistema diseñado en el paso anterior, no se genere la salida deseada.

3.- Construir un árbol de decisión para distinguir los ejemplos marcados del resto.

4.- En cada región, creada por el árbol de decisión, del espacio de entrada de la tarea donde existen ejemplos marcados, hacer:

4.1.- Entrenar una red neuronal artificial con el propósito de reducir el error producido por el sistema construido en el primer paso al manipular los ejemplos de la región.

Con estos pasos podemos construir un sistema híbrido que afina el conocimiento del experto. Este sistema se puede interpretar mediante reglas, que se pueden mostrar al experto para que las valide.

5.6.-Ejemplos

En esta sección, vamos a ilustrar el uso de un sistema híbrido para modelar una serie de funciones y vamos a analizar los resultados obtenidos. Para ello, utilizaremos los puntos extraídos de las funciones F1, F2, F3 y F4 presentadas en el Apéndice I. Usaremos la función F1 para mostrar como la metodología de construcción de un sistema híbrido para razonamiento aproximado puede servir para crear un sistema que afine la información aportada por un experto humano, manteniendo su interpretación lingüística. Utilizaremos las funciones F2 y F3 para comprobar los buenos resultados en aproximación del sistema híbrido, comparados con los obtenidos por una red neuronal artificial con el mismo tiempo de respuesta. Finalmente, usaremos la función F4 para comprobar como podemos mejorar el funcionamiento de un sistema híbrido, modificando localmente alguno de sus componentes, aprovechando que por construcción conocemos lo que hace cada uno de ellos. En todos los casos mostraremos la interpretación mediante reglas del sistema híbrido diseñado.

Los sistemas híbridos implementados en esta sección tienen las siguientes características:

- El sistema basado en reglas difusas divide cada variable en 3 regiones difusas.

- El árbol de decisión intenta delimitar los puntos con un error más grande que el error cuadrático medio del sistema basado en reglas difusas (β=error cuadrático medio).

- Entrenamos una red neuronal dentro de cada zona del espacio de entrada del problema durante 10000 ciclos de entrenamiento.

En las tablas de esta sección, aparte de los resultados proporcionados por el sistema híbrido, también aparecen los resultados ofrecidos por una red neuronal hacia adelante que resuelve directamente el problema de aproximación. Los datos ofrecidos en las tablas son la media aritmética de los valores de salida ofrecidos por cinco ejecuciones independientes. La información de estas tablas se reparte en cuatro columnas:

1) Exactitud: Refleja el error cuadrático medio de los puntos de entrenamiento. Mide la exactitud de cada método de aproximación al manipular los puntos de entrenamiento.

2) Generalización: Recoge el error cuadrático medio de los puntos de prueba. Mide la capacidad de generalización de los métodos de aproximación.

3) Espacio necesario: Presenta el número de reglas difusas o de reglas difusas *i-or* que un sistema necesita almacenar. En los sistemas híbridos, el valor de esta columna es igual a la suma del número de reglas difusas en el primer módulo y el número de reglas difusas i-or derivadas de cada red neuronal local.

4) Tiempo de respuesta: Indica el número de reglas difusas o de reglas difusas *i-or* que un sistema debe activar para obtener una salida a partir de una entrada particular y, ofrece una medida aproximada del tiempo de respuesta de un sistema. En los sistemas híbridos, el valor de esta columna es igual a la suma del número de reglas difusas activadas en el primer módulo (en esta sección, 4) y el número de reglas difusas i-or deducidas de la red neuronal local más grande.

5.6.1-Aproximación de F1

Supongamos que disponemos de reglas lingüísticas proporcionadas por un experto humano que aproximan los puntos extraídos de la función F1. En este caso, como ejemplo, conseguimos estas reglas lingüísticas por medio del método Wang-Mendel [Wang92] y son:

1. Si *x es aproximadamente más pequeño que -5.0* $\wedge$ *y es aproximadamente más pequeño que -5.0* entonces *F1(x,y) es aproximadamente más grande que 50.0.*

2. Si *x es aproximadamente más pequeño que -5.0* $\wedge$ *y es aproximadamente 0.0* entonces *F1(x,y) es aproximadamente 25.0.*
3. Si *x es aproximadamente más pequeño que -5.0* $\wedge$ *y es aproximadamente más grande que 5.0* entonces *F1(x,y) es aproximadamente más grande que 50.0.*
4. Si *x es aproximadamente 0.0* $\wedge$ *y es aproximadamente más pequeño que -5.0* entonces *F1(x,y) es aproximadamente 25.0.*
5. Si *x es aproximadamente 0.0* $\wedge$ *y es aproximadamente 0.0* entonces *F1(x,y) es aproximadamente más pequeño que 0.0.*
6. Si *x es aproximadamente 0.0* $\wedge$ *y es aproximadamente más grande que 5.0* entonces *F1(x,y) es aproximadamente 25.0.*
7. Si *x es aproximadamente más grande que 5.0* $\wedge$ *y es aproximadamente más pequeño que -5.0* entonces *F1(x,y) es aproximadamente más grande que 50.0.*
8. Si *x es aproximadamente más grande que 5.0* $\wedge$ *y es aproximadamente 0.0* entonces *F1(x,y) es aproximadamente 25.0.*
9. Si *x es aproximadamente más grande que 5.0* $\wedge$ *y es aproximadamente más grande que 5.0* entonces *F1(x,y) es aproximadamente más grande que 50.0.*

Estas reglas producen los siguientes resultados:

- Error cuadrático medio de los puntos de entrenamiento = 0.011591.
- Error cuadrático medio de los puntos de prueba = 0.011407.

Supongamos que necesitamos mejorar estos resultados. Podemos entonces diseñar un sistema híbrido para razonamiento aproximado que tenga a las anteriores reglas lingüísticas en su primer módulo tal y como se mencionó en la sección anterior. Este sistema híbrido utiliza redes neuronales con 2 neuronas en su capa oculta y proporciona los siguientes resultados:

- Error cuadrático medio de los puntos de entrenamiento = 0.000750.
- Error cuadrático medio de los puntos de prueba = 0.000788.

Y su interpretación por medio de reglas es:

Sistema híbrido para la función F1

1. Si *Verdad* entonces activar*:*
   * Si *x es aproximadamente más pequeño que -5.0* $\wedge$ *y es aproximadamente más pequeño que -5.0* entonces *F1(x,y) es aproximadamente más grande que 50.0.*

- Si *x es aproximadamente más pequeño que -5.0* $\wedge$ *y es aproximadamente 0.0* entonces *F1(x,y) es aproximadamente 25.0.*
- Si *x es aproximadamente más pequeño que -5.0* $\wedge$ *y es aproximadamente más grande que 5.0* entonces *F1(x,y) es aproximadamente más grande que 50.0.*
- Si *x es aproximadamente 0.0* $\wedge$ *y es aproximadamente más pequeño que -5.0* entonces *F1(x,y) es aproximadamente 25.0.*
- Si *x es aproximadamente 0.0* $\wedge$ *y es aproximadamente 0.0* entonces *F1(x,y) es aproximadamente más pequeño que 0.0.*
- Si *x es aproximadamente 0.0* $\wedge$ *y es aproximadamente más grande que 5.0* entonces *F1(x,y) es aproximadamente 25.0.*
- Si *x es aproximadamente más grande que 5.0* $\wedge$ *y es aproximadamente más pequeño que -5.0* entonces *F1(x,y) es aproximadamente más grande que 50.0.*
- Si *x es aproximadamente más grande que 5.0* $\wedge$ *y es aproximadamente 0.0* entonces *F1(x,y) es aproximadamente 25.0.*
- Si *x es aproximadamente más grande que 5.0* $\wedge$ *y es aproximadamente más grande que 5.0* entonces *F1(x,y) es aproximadamente más grande que 50.0.*

2. Si *x<=-4.1260* entonces activar*:*

- Si *x es más grande que aproximadamente 2.1257* i-or *y es más grande que aproximadamente 1.6218* entonces *F1(x,y)=3.52.*
- Si *x no es más grande que aproximadamente 1.4549* i-or *y es más grande que aproximadamente 1.3235* entonces *F1(x,y)=-4.95.*

3. Si *x>-4.1260* $\wedge$ *y<=-4.2640* entonces activar*:*

- Si *x no es más grande que aproximadamente 41.5003* i-or *y es más grande que aproximadamente 2.8877* entonces *F1(x,y)=-34.72.*
- Si *x no es más grande que aproximadamente 43.1291* i-or *y es más grande que aproximadamente 3.5756* entonces *F1(x,y)=-19.26.*

4. Si *x>-4.1260* $\wedge$ *y>-4.2640* entonces activar*:*

- Si *x no es más grande que aproximadamente 1.3908* i-or *y no es más grande que aproximadamente 476.7470* entonces *F1(x,y)=14.76.*
- Si *x no es más grande que aproximadamente 3.9940* i-or *y no es más grande que aproximadamente 65.3179* entonces *F1(x,y)=-23.45.*

Apreciamos como puede emplearse el procedimiento de diseño del sistema híbrido para razonamiento aproximado para afinar el conocimiento proporcionado por un experto humano. Además, el sistema final no pierde su interpretación lingüística.

5.6.2-Aproximación de F2 y F3

En la Tabla 5.I presentamos los resultados del método Wang-Mendel [Wang92], del sistema híbrido y de la red neuronal cuando aproximan los puntos extraídos de las funciones F2 y F3.

		Exactitud	Generalización	Espacio necesario	Tiempo de respuesta
F2	Wang-Mendel	0.8314510	0.7337390	9	4
	Sistema híbrido	0.0456634	0.0687322	17	6
	Red neuronal	0.0841702	0.0864526	6	6
F3	Wang-Mendel	0.7894130	0.8335170	9	4
	Sistema híbrido	0.1395092	0.1982892	17	6
	Red neuronal	0.3126106	0.3668976	6	6

Tabla 5.I: Resultados del método Wang-Mendel, del sistema híbrido y de la red neuronal cuando aproximan los puntos extraídos de las funciones F2 y F3.

Podemos observar como el sistema híbrido logra mejores resultados que las redes neuronales con el mismo tiempo de respuesta y, como mejoramos mucho los resultados del sistema basado en reglas difusas derivado del método Wang-Mendel con un pequeño incremento en el tiempo de respuesta. Además, la interpretación por medio de reglas de los sistemas híbridos que aproximan las funciones F2 y F3 es:

Sistema híbrido para la función F2

1. Si *Verdad* entonces activar*:*

- Si *x es aproximadamente más pequeño que 0.0* $\wedge$ *y es aproximadamente más pequeño que 0.0* entonces *F2(x,y) es aproximadamente más grande que 10.0.*
- Si *x es aproximadamente más pequeño que 0.0* $\wedge$ *y es aproximadamente 0.5* entonces *F2(x,y) es aproximadamente más pequeño que 0.0.*
- Si *x es aproximadamente más pequeño que 0.0* $\wedge$ *y es aproximadamente más grande que 1.0* entonces *F2(x,y) es aproximadamente más pequeño que 0.0.*
- Si *x es aproximadamente 0.5* $\wedge$ *y es aproximadamente más pequeño que 0.0* entonces *F2(x,y) es aproximadamente más grande que 10.0.*
- Si *x es aproximadamente 0.5* $\wedge$ *y es aproximadamente 0.5* entonces *F2(x,y) es aproximadamente 5.0.*
- Si *x es aproximadamente 0.5* $\wedge$ *y es aproximadamente más grande que 1.0* entonces *F2(x,y) es aproximadamente más pequeño que 0.0.*

- Si *x es aproximadamente más grande que 1.0* $\wedge$ *y es aproximadamente más pequeño que 0.0* entonces *F2(x,y) es aproximadamente más grande que 10.0.*
- Si *x es aproximadamente más grande que 1.0* $\wedge$ *y es aproximadamente 0.5* entonces *F2(x,y) es aproximadamente más grande que 10.0.*
- Si *x es aproximadamente más grande que 1.0* $\wedge$ *y es aproximadamente más grande que 1.0* entonces *F2(x,y) es aproximadamente 5.0.*

2. Si *x<=0.2422* $\wedge$ *y<=0.4158* entonces activar*:*
- Si *x no es más grande que aproximadamente 0.2141* i-or *y no es más grande que aproximadamente 1.2292* entonces *F2(x,y)=-5.4193.*
- Si *x no es más grande que aproximadamente 0.7155* i-or *y es más grande que aproximadamente 0.0724* entonces *F2(x,y)=-2.0625.*

3. Si *x<=0.2422* $\wedge$ *y>0.4158* entonces activar*:*
- Si *x es más grande que aproximadamente 0.2313* i-or *y es más grande que aproximadamente 0.0599* entonces *F2(x,y)=-9.6746.*
- Si *x no es más grande que aproximadamente 0.2190* i-or *y es más grande que aproximadamente 13.9512* entonces *F2(x,y)=-1.709.*

4. Si *x>0.2422* $\wedge$ *y<=0.9431* entonces activar*:*
- Si *x no es más grande que aproximadamente 0.4525* i-or *y es más grande que aproximadamente 0.4825* entonces *F2(x,y)=-2.4186.*
- Si *x no es más grande que aproximadamente 0.7545* i-or *y es más grande que aproximadamente 0.4765* entonces *F2(x,y)=2.1120.*

5. Si *x>0.2422* $\wedge$ *y>0.9431* entonces activar*:*
- Si *x no es más grande que aproximadamente 0.4784* i-or *y es más grande que aproximadamente 3.5412* entonces *F2(x,y)=-2.5743.*
- Si *x no es más grande que aproximadamente 0.3986* i-or *y es más grande que aproximadamente 0.1968* entonces *F2(x,y)=10.8843.*

Sistema híbrido para la función F3

1. Si *Verdad* entonces activar*:*
- Si *x es aproximadamente más pequeño que 0.0* $\wedge$ *y es aproximadamente más pequeño que 0.0* entonces *F3(x,y) es aproximadamente más grande que 8.0.*
- Si *x es aproximadamente más pequeño que 0.0* $\wedge$ *y es aproximadamente 0.5* entonces *F3(x,y) es aproximadamente más pequeño que 0.91.*

- Si *x es aproximadamente más pequeño que 0.0* $\wedge$ *y es aproximadamente más grande que 1.0* entonces *F3(x,y) es aproximadamente más grande que 8.0.*
- Si *x es aproximadamente 0.5* $\wedge$ *y es aproximadamente más pequeño que 0.0* entonces *F3(x,y) es aproximadamente 4.46.*
- Si *x es aproximadamente 0.5* $\wedge$ *y es aproximadamente 0.5* entonces *F3(x,y) es aproximadamente 4.46.*
- Si *x es aproximadamente 0.5* $\wedge$ *y es aproximadamente más grande que 1.0* entonces *F3(x,y) es aproximadamente 4.46.*
- Si *x es aproximadamente más grande que 1.0* $\wedge$ *y es aproximadamente más pequeño que 0.0* entonces *F3(x,y) es aproximadamente más pequeño que 0.91.*
- Si *x es aproximadamente más grande que 1.0* $\wedge$ *y es aproximadamente 0.5* entonces *F3(x,y) es aproximadamente 4.46.*
- Si *x es aproximadamente más grande que 1.0* $\wedge$ *y es aproximadamente más grande que 1.0* entonces *F3(x,y) es aproximadamente más pequeño que 0.91.*

2. Si *x<=0.1492* $\wedge$ *y<=0.0835* entonces activar*:*

- Si *x es más grande que aproximadamente 0.3577* i-or *y es más grande que aproximadamente 3.4149* entonces *F3(x,y)=-9.8658.*
- Si *x no es más grande que aproximadamente 0.8929* i-or *y es más grande que aproximadamente 0.5550* entonces *F3(x,y)=9.5552.*

3. Si *x<=0.1492* $\wedge$ *y>0.0835* entonces activar*:*

- Si *x no es más grande que aproximadamente 1.8921* i-or *y no es más grande que aproximadamente 1.9811* entonces *F3(x,y)=-5.4096.*
- Si *x no es más grande que aproximadamente 0.5515* i-or *y no es más grande que aproximadamente 1.5876* entonces *F3(x,y)=2.03.*

4. Si *x>0.1492* $\wedge$ *y<=0.7931* entonces activar*:*

- Si *x no es más grande que aproximadamente 0.2451* i-or *y no es más grande que aproximadamente 0.1790* entonces *F3(x,y)=-3.6919.*
- Si *x no es más grande que aproximadamente 0.2912* i-or *y no es más grande que aproximadamente 0.4308* entonces *F3(x,y)=5.8542.*

5. Si *x>0.1492* $\wedge$ *y>0.7931* entonces activar*:*

- Si *x no es más grande que aproximadamente 0.0489* i-or *y no es más grande que aproximadamente 0.1040* entonces *F3(x,y)=-3.4739.*
- Si *x no es más grande que aproximadamente 0.1805* i-or *y no es más grande que aproximadamente 0.6799* entonces *F3(x,y)=2.1369.*

En la interpretación de los sistemas híbridos por medio de reglas, observamos que hay proposiciones en los antecedentes de algunas reglas difusas i-or que están lejos del área de influencia de la regla, a pesar de haber obtenido esta regla después de entrenar a una red neuronal con backpropagation usando los puntos de este área. Por ejemplo, las proposiciones en el antecedente de la primera regla difusa i-or de la regla 3 del sistema híbrido para la función F3 son:

x no es más grande que aproximadamente 1.8921,
y no es más grande que aproximadamente 1.9811,

sin embargo, $x,y \in [0,1]$. Esto implica que estas proposiciones devolverán "prácticamente" un valor de verdad igual a uno para todas las posibles entradas. Por lo tanto, estas proposiciones (que corresponden a conexiones en la red neuronal) podrían sustituirse por 1. Pero como *1 i-or 1 = 1*, todo el antecedente de la regla se podría sustituir por 1, o lo que es lo mismo, todas las conexiones de entrada de la neurona de la que procede esta regla pueden sustituirse por un valor constante de entrada. Podemos observar como la acción local de las redes puede llevar a una reducción de su topología.

5.6.2-Aproximación de F4

En la Tabla 5.II presentamos los resultados del método Wang-Mendel, del sistema híbrido y de la red neuronal cuando aproximan los puntos extraídos de la función F4.

		Exactitud	Generalización	Espacio necesario	Tiempo de respuesta
F4	Wang-Mendel	0.6070620	0.6670630	9	4
	Sistema híbrido	0.4986252	0.5452146	17	6
	Red neuronal	0.4367614	0.4928016	6	6

Tabla 5.II: Resultados del método Wang-Mendel, del sistema híbrido y de la red neuronal cuando aproximan los puntos extraídos de la función F4.

Podemos apreciar como, en este caso, los resultados proporcionados por la red neuronal son mejores que los proporcionados por el sistema híbrido. Con el fin de mejorar los resultados del sistema híbrido sobre la función F4, estudiamos el error de cada red neuronal local. A continuación, presentamos la interpretación por medio de reglas de un sistema híbrido que aproxima los puntos extraídos de la función F4 junto con el error producido por cada red neuronal local:

Sistema híbrido para la función F4

1. Si *Verdad* entonces activar*:*

- Si *x es aproximadamente más pequeño que -1.0* $\wedge$ *y es aproximadamente más pequeño que -1.0* entonces *F4(x,y) es aproximadamente más grande que 3.51.*
- Si *x es aproximadamente más pequeño que -1.0* $\wedge$ *y es aproximadamente 0.0* entonces *F4(x,y) es aproximadamente 0.82.*
- Si *x es aproximadamente más pequeño que -1.0* $\wedge$ *y es aproximadamente más grande que 1.0* entonces *F4(x,y) es aproximadamente más grande que 3.51.*
- Si *x es aproximadamente 0.0* $\wedge$ *y es aproximadamente más pequeño que -1.0* entonces *F4(x,y) es aproximadamente 0.82.*
- Si *x es aproximadamente 0.0* $\wedge$ *y es aproximadamente 0.0* entonces *F4(x,y) es aproximadamente más pequeño que -1.86.*
- Si *x es aproximadamente 0.0* $\wedge$ *y es aproximadamente más grande que 1.0* entonces *F4(x,y) es aproximadamente 0.82.*
- Si *x es aproximadamente más grande que 1.0* $\wedge$ *y es aproximadamente más pequeño que -1.0* entonces *F4(x,y) es aproximadamente más grande que 0.3.51.*
- Si *x es aproximadamente más grande que 1.0* $\wedge$ *y es aproximadamente 0.0* entonces *F4(x,y) es aproximadamente 0.82.*
- Si *x es aproximadamente más grande que 1.0* $\wedge$ *y es aproximadamente más grande que 1.0* entonces *F4(x,y) es aproximadamente 0.82.*

2. Si *x<=-0.8048* $\wedge$ *y<=-0.7320* entonces activar*:*

- Si *x es más grande que aproximadamente 16.1656* i-or *y no es más grande que aproximadamente 26.0901* entonces *F4(x,y)=-0.1297.*
- Si *x no es más grande que aproximadamente 0.3936* i-or *y es más grande que aproximadamente 0.6228* entonces *F4(x,y)=-7.9672.*

Error de regla 2= 0.07123

3. Si *x<=-0.8048* $\wedge$ *y>-0.7320* entonces activar*:*

- Si *x es más grande que aproximadamente 0.5049* i-or *y no es más grande que aproximadamente 0.2564* entonces *F4(x,y)=-0.9498.*
- Si *x no es más grande que aproximadamente 0.8490* i-or *y no es más grande que aproximadamente 0.1445* entonces *F4(x,y)=-1.6077.*

Error de regla 3= 0.25544

4. Si *x>-0.8048* $\wedge$ *y<=0.3902* entonces activar*:*

- Si *x no es más grande que aproximadamente 9.3891* i-or *y es más grande que aproximadamente 16.0001* entonces *F4(x,y)=1.1873.*
- Si *x no es más grande que aproximadamente 0.5479* i-or *y es más grande que aproximadamente 2.3140* entonces *F4(x,y)=-1.1449.*

Error de regla 4= 0.59756

5. Si *x>-0.8048 ∧ y>0.3902* entonces activar*:*

- Si *x no es más grande que aproximadamente 0.2194* i-or *y es más grande que aproximadamente 0.0246* entonces *F4(x,y)=0.4846.*
- Si *x es más grande que aproximadamente 0.540* i-or *y no es más grande que aproximadamente 0.0303* entonces *F4(x,y)=-1.3991.*

Error de regla 5= 0.339517

Supongamos que el error generado por la red neuronal asociada a la regla 4 del sistema híbrido anterior es excesivo para nuestros propósitos. Con fin de mejorar los resultados globales del sistema híbrido, podemos sustituir esta red neuronal (regla 4) con dos neuronas en su capa oculta por una red neuronal con cuatro neuronas en su capa oculta. Mostramos los resultados de este nuevo sistema híbrido en la Tabla 5.III, cuando aproxima los puntos extraídos de la función F4, junto con los resultados proporcionados por una red neuronal con el mismo tiempo de respuesta. Podemos apreciar que el sistema híbrido modificado logra mejores resultados que la red neuronal. La interpretación de este nuevo sistema híbrido modificado que aproxima la función F4 es en todo igual a la interpretación presentada anteriormente salvo la regla 4 que se cambia por:

◊ Si *x>-0.8048 ∧ y<=0.3902* entonces activar*:*

- Si *x es más grande que aproximadamente 1.2643* i-or *y no es más grande que aproximadamente 0.0319* entonces *F4(x,y)=0.8025.*
- Si *x no es más grande que aproximadamente 0.0349* i-or *y es más grande que aproximadamente 1.6936* entonces *F4(x,y)=1.5397.*
- Si *x es más grande que aproximadamente 1.1732* i-or *y no es más grande que aproximadamente 0.0275* entonces *F4(x,y)=-1.6198.*
- Si *x no es más grande que aproximadamente 0.0588* i-or *y es más grande que aproximadamente 6.7003* entonces *F4(x,y)=-2.3117.*

		Exactitud	Generalización	Espacio necesario	Tiempo de respuesta
F4	Sist. hib. Modif.	0.3658262	0.4382980	19	8
	Red neuronal	0.4305438	0.4784028	8	8

Tabla 5.III: Resultados del sistema híbrido modificado y de la red neuronal cuando aproximan los puntos extraídos de la función F4.

Así pues la estructura modular de un sistema híbrido permite modificarlo localmente con el propósito de mejorar sus resultados, sin tener que cambiar el sistema entero.

5.6.4.- Conclusiones de la experimentación

Tras la serie de pruebas que acabamos de describir con el sistema híbrido para razonamiento aproximado, podemos destacar lo siguiente:

- Podemos usar la metodología de diseño de un sistema híbrido para razonamiento aproximado con el propósito de afinar las reglas lingüísticas proporcionadas por un experto para modelar el comportamiento de un sistema, manteniendo la comprensión del sistema inteligente resultante. Para ello, debemos sustituir el sistema basado en reglas difusas por el conjunto de reglas lingüísticas ofrecidas por el experto y, a continuación, sólo debemos aplicar la metodología general, es decir, utilizar el árbol de decisión para delimitar ejemplos problemáticos y las redes neuronales artificiales para conseguir que la salida del sistema sea una buena solución al problema de modelado de un sistema real.

- Podemos usar el diseño de un sistema híbrido para resolver directamente un problema, ya que logra unos buenos resultados en aproximación, con la ventaja adicional añadida de la comprensión de su funcionamiento.

- Podemos modificar localmente algunos de los componentes de un sistema híbrido para mejorar su funcionamiento global, gracias a la explicación disponible de su dinámica y a su estructura modular.

5.7.- Resumen y notas finales

En este capítulo, hemos presentado una particularización de MEGAS orientada a la creación de sistemas inteligentes con un funcionamiento fácilmente comprensible. Esta característica permite que los sistemas inteligentes sean validados con facilidad por

un experto y que, en caso de que su modificación para conseguir un mejor funcionamiento sea necesaria, no se requiera cambiar todo el sistema, ya que podemos aislar la zona a modificar puesto que conocemos el funcionamiento local del sistema. Además, estos sistemas inteligentes seguirán proporcionando buenos resultados en aproximación, propiedad que se hereda de MEGAS.

Para conseguir estos propósitos hemos considerado tres tipos de sistemas inteligentes (sistemas basados en reglas difusas, árboles de decisión y redes neuronales artificiales), cuya dinámica se podía interpretar mediante reglas, que "colocamos" dentro de la estructura de MEGAS. Instanciamos el primer componente que proporciona una salida aproximada con un sistema basado en reglas difusas, el componente que delimita ejemplos erróneos con un árbol de decisión y los componentes locales con redes neuronales artificiales.

De esta manera, definimos un sistema híbrido que es fácilmente transformable a un sistema basado en reglas (aunque estas sean no convencionales). Experimentalmente comprobamos que el procedimiento de diseño de este sistema híbrido puede ser útil para:

- Afinar el conocimiento lingüístico aportado por un experto.
- Conseguir un sistema que proporcione buenos resultados de aproximación.
- Diseñar un sistema que sea fácilmente modificable por zonas.

Capítulo 6

CONCLUSIONES Y TRABAJO FUTURO

6.1.-Conclusiones

Básicamente, en esta tesis hemos logrado interpretar cualquier sistema inteligente como un sistema estructurado, deduciendo de ello ventajas e inconvenientes de los diferentes modelos que se ofrecen en la literatura, lo que ha posibilitado la definición de nuevos sistemas estructurados que tienen buenas propiedades generales derivadas de su estructura. Más concretamente, podemos resumir estos logros en los siguientes puntos:

1) Hemos definido un nuevo modelo de representación de conocimiento estructurado (MORSE) que permite:

   - Modelar cualquier sistema inteligente de manera no uniforme, o sea, modelarlo como un sistema compuesto por varios módulos, cada uno con su función específica, que obedecen a una estrategia de control común.

   - Agrupar en una misma representación a varios sistemas inteligentes que tienen la misma estructura y, por lo tanto, las mismas propiedades derivadas de esta estructura.

2) A partir del estudio anterior, hemos conseguido realizar un análisis de sistemas inteligentes en base a las características de sus representaciones como modelos MORSE, o sea, en función de las propiedades generales derivadas de su estructura. Este análisis permite poseer información acerca de cuando debemos utilizar un tipo de sistema inteligente u otro en función de las características del problema a resolver.

3) A partir del problema de simular el "razonamiento humano estructurado" y, aprovechando los estudios realizados sobre la estructura de sistemas inteligentes, hemos definido un nuevo método de aprendizaje automático (*MEGAS*) que consigue simular algunos tipos de aprendizaje humano. Definimos MEGAS en base a componentes

generales, que podemos concretar con distintas herramientas en función del objetivo que se persiga. Por ejemplo:

- En algunas situaciones el uso de sistemas basados en reglas difusas como módulos de MEGAS frente al uso directo de estos sistemas para resolver un problema, tiene la ventaja de ahorro de recursos y de mejora de resultados.

- Hemos comprobado empíricamente, que el uso de redes neuronales artificiales con MEGAS mejora los resultados obtenidos por estas redes sin estructurar y, además permite aprovechar el trabajo realizado al identificar el modelo de un sistema que no logra unos resultados aceptables, cuando es necesario la mejora de estos.

4) A raíz de la necesidad de ejecutar, en algunos casos, varios pasos seguidos de MEGAS para resolver un problema y, aprovechando el estudio realizado anteriormente acerca de las propiedades derivadas de la estructura de los sistemas inteligentes basados en el modelo de árbol y de los sistemas inteligentes basados en el modelo de red neuronal, hemos conseguido definir, apropiadamente, un mecanismo de iteración para MEGAS, que ha dado lugar a una nueva metodología de diseño de sistemas inteligentes denominada SEPARATE. Esta metodología tiene las siguientes propiedades:

   - Debido a que utiliza la técnica de diseño de algoritmos *Divide-y-Vencerás*, los sistemas diseñados con SEPARATE logran unos buenos resultados sobre los ejemplos de entrenamiento, de manera similar que los sistemas inteligentes basados en el modelo de árbol.

   - Debido a que algunos elementos del sistema se diseñan para actuar sobre la casi totalidad del espacio de entrada de un problema, los sistemas diseñados con SEPARATE obtienen unos resultados en generalización similares a los obtenidos por los sistemas inteligentes basados en el modelo de red neuronal.

5) Hemos presentado una particularización del método MEGAS orientada a la tarea de extracción del conocimiento (sistema híbrido para razonamiento aproximado). Este método construye un sistema híbrido compuesto por diferentes sistemas inteligentes (sistemas basados en reglas difusas, árboles de decisión y redes neuronales), del cual podemos deducir fácilmente reglas (no convencionales) comprensibles que explican su funcionamiento. Los sistemas diseñados de este modo tienen dos propiedades principales que proceden de la definición del sistema en dos niveles de abstracción:

   - Ofrecen buenos resultados en aproximación, propiedad heredada de MEGAS.

- Su funcionamiento es fácil de comprender, propiedad heredada de los sistemas inteligentes particulares con los que se instanciaron los componentes de MEGAS.

En definitiva, en este trabajo apreciamos la importancia del estudio de los sistemas inteligentes en función de su estructura, ya que este hecho permite:

a) Tener una visión conjunta de distintos sistemas que están disponibles para ser usados y, poseer alguna información acerca de cual de ellos usar a la hora de resolver un problema.

b) Definir nuevos métodos de aprendizaje automático que diseñan sistemas inteligentes a un nivel alto de abstracción, que pueden ser particularizados de diferentes maneras en función de las características concretas del problema a resolver. De esta manera, construimos sistemas inteligentes que tienen dos tipos de propiedades:

    1. Propiedades generales heredadas de la definición del sistema a un alto nivel de abstracción.

    2. Propiedades particulares heredadas de la instanciación de los componentes del sistema con herramientas concretas, elegidas en función del problema a resolver y del objetivo perseguido con su resolución.

6.2.-Trabajo futuro

- Extender el modelo de representación MORSE para transformar su estrategia de ejecución estática (módulo de combinación) a una estrategia dinámica, por medio del uso del concepto de *algoritmo difuso* [Dubois80, Zadeh68, Zadeh73] y de la posibilidad de ajuste sobre este tipo de algoritmo [Castro98b, Castro99a].

- Obtener la descripción de algoritmos difusos por medio del modelo de representación MORSE o de alguna extensión suya.

- Ampliar el método MEGAS para que su primer nivel de ejecución tenga, aparte del módulo que logra una solución aproximada al problema, otro módulo que aporte información sobre la estacionalidad del sistema, con el fin de aplicar los sistemas diseñados con MEGAS en la predicción de series temporales.

- Estudiar otras particularizaciones del método MEGAS que aporten características beneficiosas para resolver ciertos tipos de problemas. En concreto, pensamos aplicar varias particularizaciones de MEGAS a problemas reales en el campo del diagnóstico médico.

- Estudiar la posibilidad de estructurar de otra forma el aprendizaje. Por ejemplo, podríamos pensar en una estructura compuesta por módulos de aprendizaje cooperando o compitiendo para lograr una solución o, compuesta por un conjunto de módulos de aprendizaje especializados en la resolución de tareas particulares y que, dinámicamente, se decide qué módulo es el más idóneo para usar en cada momento del aprendizaje.

Apéndice I

FUNCIONES CONSIDERADAS PARA SU MODELADO EN LA EXPERIMENTACIÓN REALIZADA

Las funciones empleadas para constrastar los métodos de aproximación han sido:

$F1(x,y) = x^2 + y^2$, $x, y \in [-5.0,5.0]$ (Figura I.1).

$F2(x,y) = 10\,(x-xy) / (x-2xy+y)$, $x, y \in [0.0,1.0]$ (Figura I.2).

$F3(x,y) = 42.659\,[0.1 + (x-0.5)\,(\,0.05 + (x-0.5)^4 - 10\,(x-0.5)^2\,(y-0.5)^2 + 5\,(y-0.5)^4)\,]$,
$x, y \in [0.0,1.0]$ (Figura I.3).

$F4(x,y) = x^2 + y^2 - \cos(18x) - \cos(18y)$, $x, y \in [-1.0,1.0]$ (Figura I.4).

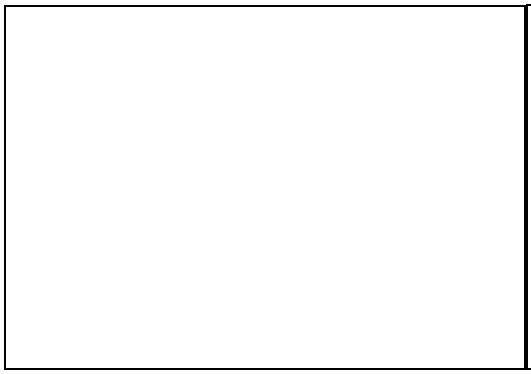


Figura I.1: Función F1(x,y)

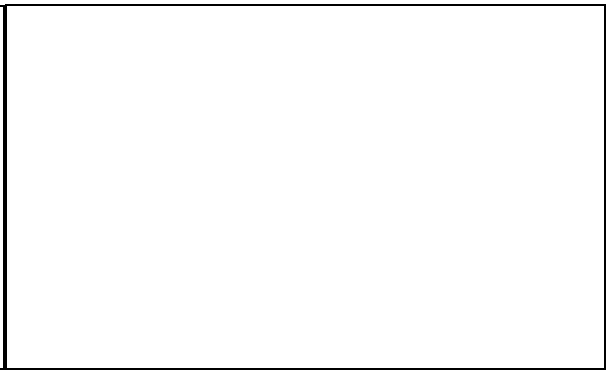


Figura I.2: Función F2(x,y)

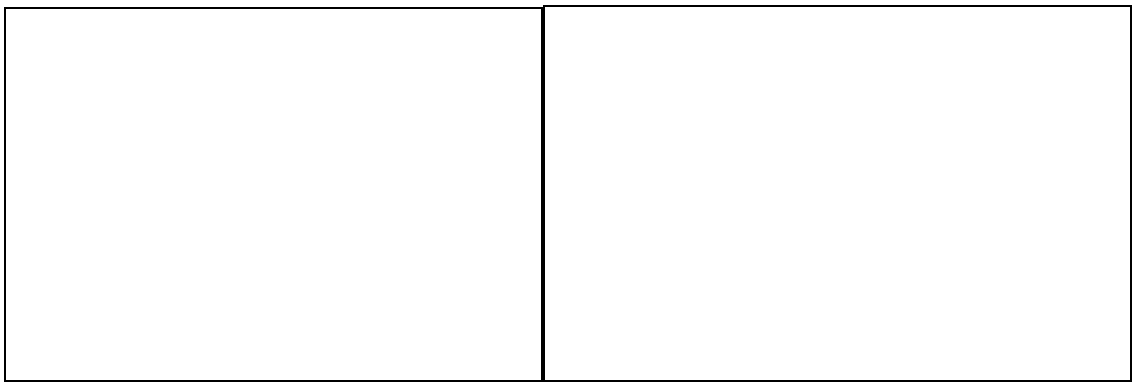


Figura I.3: Función F3(x,y)

Figura I.4: Función F4(x,y)

Hemos elegido estas funciones por su variedad en complejidad, ordenadas desde la más simple, F1, a la más compleja, F4. Las dos primeras funciones F1 (denominada *modelo esférico*) y F2 (que presenta dos discontinuidades en los puntos (0,0) y (1,1)) son funciones unimodales sencillas. La siguiente en complejidad es F3, denominada *función armónica*, que se describe en detalle en [Hwang94]. Finalmente, la *función generalizada de Rastrigin*, F4, es una función fuertemente multimodal, muy compleja.

En los experimentos realizados se han extraído, aleatoriamente, de cada función 1000 pares de entrada-salida [(x,y), F(x,y)], de los cuales hemos usado 500 como puntos de entrenamiento y los 500 restantes como puntos de prueba.

Apéndice II

SISTEMAS BASADOS EN REGLAS DIFUSAS

II.1.-Sistemas basados en reglas difusas

En la actualidad, una de las áreas de aplicación más importantes de la *Teoría de Conjuntos Difusos* y de la *Lógica Difusa*, enunciada por Zadeh en 1965 [Zadeh65], la componen los sistemas basados en reglas difusas. Este tipo de sistemas constituyen una extensión de los sistemas basados en reglas tradicionalmente empleados en Inteligencia Artificial, puesto que emplean reglas de tipo "SI-ENTONCES" en las que los antecedentes y consecuentes están compuestos por proposiciones difusas en lugar de proposiciones crisp.

En un sentido muy general, un sistema basado en reglas difusas es un sistema basado en reglas en el que la Lógica Difusa [Klir95, Zimmermann96] se puede emplear tanto como herramienta para representar distintas formas de conocimiento sobre el problema a resolver, como para modelar las interacciones y relaciones existentes entre las variables del mismo. La principal aplicación de estos sistemas inteligentes es el *modelado difuso de sistemas* [Bardossy95, Pedrycz96, Sugeno93], puesto que existe una amplia gama de sistemas reales en los que, debido a su nivel de complejidad o imprecisión, las herramientas clásicas no permiten obtener buenos resultados. El modelado difuso de sistemas se puede considerar como una vía para modelar un sistema haciendo uso de un lenguaje de descripción basado en Lógica Difusa con predicados difusos [Sugeno93].

Los sistemas basados en reglas difusas se han aplicado con éxito a una gran cantidad de problemas reales a lo largo de los últimos años [Bardossy95, Goonatilake95, Hirota93, Pedrycz96, Wang94], más concretamente, en el campo del control han producido excelentes resultados, área en la que se conocen bajo el término de *controladores difusos* [Berenji92, Driankov93, Lee90a, Lee90b]. Su aplicación a problemas de control con soluciones difíciles o imposibles para la matemática clásica, ha sido definitiva para la aceptación y rápida expansión de las técnicas basadas en la Teoría de Conjuntos Difusos.

Los sistemas basados en reglas difusas poseen algunas propiedades muy interesantes, de entre las que destaca sobremanera su capacidad de "Aproximación Universal" [Buckley93, Castro95, Castro96, Kosko94, Wang92b] . En [Castro95, Castro96, Kosko94] se demuestra que amplias clases de controladores difusos son aproximadores universales. Este resultado habilita a los sistemas basados en reglas difusas como herramientas adecuadas para aproximar a otros sistemas aun cuando los datos asociados a los mismos no tengan carácter difuso.

Podemos distinguir dos usos fundamentales de los sistemas basados en reglas difusas:

1. *Descripción lingüística* de sistemas en los que existe incertidumbre o vaguedad en las entradas y/o salidas o, en los que no importa mucho la precisión. Lo fundamental es obtener una descripción de las relaciones entre entradas y salidas subyacentes a un sistema desconocido en términos de palabras y expresiones propias del lenguaje natural.

2. *Aproximación* de sistemas. Se explota su propiedad de Aproximadores Universales. Los sistemas basados en reglas difusas aparecen como alternativas a otros modelos matemáticos clásicos en la aproximación de sistemas desconocidos. Sus ventajas se ponen de manifiesto al tratar con sistemas complejos, donde su simplicidad y facilidad de manejo los hace preferibles a otros métodos, sobre todo en lo que respecta a eficiencia de funcionamiento.

En las siguientes secciones describimos tres tipos de sistemas basados en reglas difusas: los sistemas puros, los de Takagi-Sugeno-Kang y los aditivos.

II.1.1.-Sistemas basados en reglas difusas "puros"

Las reglas lingüísticas empleadas en este tipo de sistemas presentan la siguiente forma en el caso de trabajar con sistema con múltiples entradas y una única salida:

$$R_i : \text{SI } x_1 \text{ ES } A^i_1 \text{ Y } x_2 \text{ ES } A^i_2 \text{ Y } \ldots \text{ Y } x_n \text{ ES } A^i_n \text{ ENTONCES } y \text{ ES } B_i,$$

donde x_j e y son variables lingüísticas de entrada y salida respectivamente, y los A^i_j y B_i son etiquetas lingüísticas asociadas con conjuntos difusos que determinan su semántica.

Los sistemas basados en reglas difusas "puros" pueden recibir valores crisp o conjuntos difusos como entrada y, devuelven como salida conjuntos difusos. En ellos se distinguen dos componentes principales: una *Base de Conocimiento*, que almacena el conocimiento disponible sobre el problema en forma de reglas lingüísticas de tipo "SI-ENTONCES", y un *Sistema de Inferencia*, encargado de llevar a cabo el proceso de inferencia sobre las entradas haciendo uso de la información almacenada en la Base de Conocimiento.

Los sistemas basados en reglas difusas "puros" constituyen el modelo esencial de Sistema Difuso, un marco de trabajo general en el cual se cuantifica la información lingüística procedente de los expertos humanos y se emplean los principios de la Lógica Difusa para hacer un uso sistemático de esa información. Este tipo de sistemas basados en reglas difusas constituye la derivación natural de las ideas iniciales de Zadeh [Zadeh73].

II.1.2.-Sistemas basados en reglas difusas de Takagi-Sugeno-Kang

En 1985, Takagi, Sugeno y Kang [Sugeno85, Sugeno88, Takagi85] propusieron un modelo de sistema basado en reglas difusas que era muy eficaz para tareas aproximativas. Este sistema usa reglas del tipo:

$$R_i : \text{SI } x_1 \text{ ES } A^i_1 \text{ Y } x_2 \text{ ES } A^i_2 \text{ Y } \ldots \text{ Y } x_n \text{ ES } A^i_n \text{ ENTONCES } y = p_i(x_1, x_2, \ldots, x_n),$$

con $p_i(x_1, x_2,..., x_n)$ una función lineal. Es decir, la salida adopta un carácter puramente funcional, que salvo en el caso base de que sea una constante, no tiene una interpretación lingüística simple. Este tipo de reglas suelen denominarse *reglas difusas de tipo TSK*, en alusión a sus creadores [Takagi85].

En su procedimiento de inferencia, se modela la conjunción mediante el operador producto. Así dada la entrada $\mathbf{a} = (a_1, a_2,..., a_n)$, el grado de disparo de la regla es:

$$\gamma_i = \prod (A^i_1(a_1), A^i_2(a_2), \ldots, A^i_n(a_n)).$$

La salida final de un sistema basado en reglas difusas de tipo TSK es igual a:

$$y = \frac{\sum_{i=1}^{r} \gamma_i p_i(a_1, a_2, ..., a_n)}{\sum_{i=1}^{r} \gamma_i},$$

donde *r* es el número de reglas del sistema.

La ventaja principal de estos sistemas es el hecho de presentar una ecuación del sistema compacta, que permite estimar los parámetros de p_i empleando métodos clásicos. Sin embargo, el mayor inconveniente que tienen asociado está también relacionado con la forma de los consecuentes de las reglas, que al no ser difusos, provocan que el sistema no constituya un marco de trabajo natural para representar el conocimiento experto.

II.1.3.-Sistemas difusos aditivos

Los sistemas difusos aditivos, propuestos por Kosko [Kosko92, Kosko94], son otro tipo de sistemas basados en reglas difusas caracterizados por su forma peculiar de realizar la inferencia. De hecho, la aditividad radica en la forma de realizar la agregación de la salida de las diferentes reglas más que en la forma concreta de dichas reglas.

A partir de un sistema basado en reglas difusas “puro” con reglas

$$R_i : \text{SI } x \text{ ES } A_i \text{ ENTONCES } y = B_i ,$$

la aditividad actúa agregando la salida de las distintas reglas mediante una suma ponderada:

$$B = \sum_{i=1}^{r} w_i B_i^{'},$$

donde w_i son pesos asociados a las reglas, no son el grado de disparo γ_i. La elección de estos pesos caracteriza el tipo de inferencia.

En el caso de sistemas basados en reglas de tipo TSK, la aditividad supone que la salida adopta la siguiente forma:

$$y = \sum_{i=1}^{r} w_i \gamma_i p_i(a_1, a_2, ..., a_n).$$

donde w_i tienen el mismo significado que antes y γ_i son los grados de disparo de las reglas tal como se caracterizaron en II.1.2.

Apéndice III

MÉTODO DE AGRUPAMIENTO DE CHIU

En este apéndice presentamos el método de agrupamiento presentado en [Chiu94], que tiene como principal característica que estima simultáneamente el número de grupos y sus centros. La idea básica de este método consiste en asignar a cada ejemplo una medida de su potencialidad como centro de grupo y, a continuación, seleccionar aquellos que prometan la mejor representación del conjunto de datos.

Considérese un conjunto de datos con *q* muestras de un espacio de dimensión *n*. Se supone además que todas las componentes de los datos están normalizados en los mismos rangos, de forma que todo el conjunto de datos se halla dentro de un hipercubo. Cada punto x_i del conjunto de datos es candidato a ser centro. Su potencial para serlo efectivamente se mide por el valor:

$$p_i = \sum_{j=1}^{q} e^{-\alpha\left\|x_i - x_j\right\|^2},$$

donde $\alpha=4/r_a^2$ y $r_a \in \Re^+$. Este potencial depende de la distancia del punto a todos los demás, de forma que cuantos más haya en sus cercanías, mayor es su potencial. La constante r_a define el radio de la vecindad, los puntos fuera de este radio tendrán escasa influencia sobre el potencial de un punto. Por ejemplo, supongamos el punto x_i, entonces los puntos x_j tales que $||x_i-x_j||>r_a$ están fuera de la vecindad de x_i y, por lo tanto, tendrán poca influencia sobre el potencial de x_i.

En Alg. III.1 describimos el algoritmo de este método de agrupamiento.

Entrada: Un conjunto de datos para agrupar.
Salida: Centros de los grupos que reúnen a los puntos de entrada.

1.- Calcular el potencial de todos los puntos según

$$p_i = \sum_{j=1}^{q} e^{-\alpha \left\| x_i - x_j \right\|^2},$$

donde $\alpha = 4/r_a^2$ *y* $r_a \in \Re^+$.

2.-$k \leftarrow 1$*. El centro del primer grupo es el punto con mayor valor de potencial. Sea* x_1^* *este punto y* p_1^* *su potencial. Se elimina este punto del conjunto de datos.*

3.-Actualizar el potencial de los restantes puntos de acuerdo con:

$$p_i \leftarrow p_i - p_k^* \cdot e^{-\beta \left\| x_i - x_k^* \right\|^2},$$

donde $\beta = 4/r_b^2$ *y* $r_b \in \Re^+$.

4.-$k \leftarrow k+1$*. Sea* x_k^* *el punto con máximo potencial y* p_k^* *su potencial.*

5.-Si ($p_k^* > \overline{\varepsilon}\, p_1^*$*) entonces aceptar* x_k^* *como el centro del próximo grupo e ir al paso 3. En otro caso:*

5.1.-Si ($p_k^* < \underline{\varepsilon}\, p_1^*$*) entonces rechazar* x_k^* *y finalizar el algoritmo. En otro caso:*

5.1.1.- Sean d_{min} *la mínima distancia entre cualquier par de centros de grupos ya seleccionados. Si (*$\frac{d_{min}}{r_a} + \frac{p_k^*}{p_1^*} \geq 1$*) entonces aceptar* x_k^* *como el siguiente centro e ir al paso 3. En otro caso:*

5.1.1.1.-Rechazar x_k^* *y hacer* $p_k^* \leftarrow 0$*. Seleccionar el punto con el siguiente valor más alto del potencial como el nuevo* x_k^* *y volver a hacer las comprobaciones (paso 5).*

Algoritmo III.1: Algoritmo del método de agrupamiento de Chiu.

La modificación realizada en el paso 3 de Alg. III.1 penaliza el potencial de los puntos según su proximidad al centro del grupo anterior, haciendo que los puntos cercanos a éste se desestimen como centros de los siguientes grupos. La constante r_b es el radio de vecindad que sufrirá una reducción considerable en su potencial. Para evitar la obtención de grupos muy cercanos, es conveniente que r_b sea mayor que r_a. Una buena elección es $r_b = 1'5\ r_a$.

Chiu indica que no basta con un único valor ε para establecer un criterio de parada y por ello utiliza dos, $\overline{\varepsilon}$ y $\underline{\varepsilon}$. El primero, $\overline{\varepsilon}$, establece un umbral para el potencial por encima del cual se acepta un punto como centro; el segundo, $\underline{\varepsilon}$, indica un umbral por debajo del cual se rechaza un punto como centro y, además, implica la finalización del algoritmo. Para trabajar en la zona intermedia delimitada por ambas cotas, evalúa una medida que establece un compromiso entre el potencial de un punto y su proximidad a centros ya seleccionados. Los valores orientativos que Chiu ofrece son $\overline{\varepsilon}$=0.5 y $\underline{\varepsilon}$=0.15. Si el potencial está en la zona intermedia, se comprueba si el punto ofrece un buen equilibrio entre tener un potencial razonable y estar suficientemente alejado de los centros de grupo ya fijados.

Apéndice IV

ALGORITMOS DE APRENDIZAJE

IV.1.-Algoritmo *Backpropagation* para entrenar redes neuronales artificiales

Una red neuronal artificial tiene que ser entrenada (es decir, los pesos de sus conexiones tienen que ser fijados) de modo que la aplicación de un conjunto de entradas produzca, bien de modo directo o a través de un proceso de aprendizaje, la salida esperada. El aprendizaje consiste en modificar los parámetros libres de la red, es decir, los pesos de las conexiones, para que el error cometido en la salida tienda a hacerse cero.

El algoritmo de entrenamiento *backpropagation* [Rumelhart86] es un método de aprendizaje supervisado de redes neuronales artificiales, que se utiliza, sobre todo, para entrenar redes neuronales hacia adelante con una o más capas ocultas.

Backpropagation consiste en cuantificar el error a través de la diferencia entre la salida dada por la red neuronal y la que realmente debería ser. Concretamente, en las redes neuronales artificiales multicapa, la dificultad reside en que hay un gran número de conexiones que contribuyen a dicho error, no sólo las relacionadas con las neuronas de la capa de salida. En particular, como no se tiene información sobre el papel que juegan las neuronas de las capas ocultas, no se puede calcular directamente su contribución al error total y, no se sabe como modificar correctamente los pesos de las conexiones asociadas a ellas.

El problema se puede atacar si se admite que el error que se observa en la capa de salida se debe primordialmente a la acción de las unidades ocultas ubicadas en la capa inmediatamente anterior, que a su vez se ven influenciadas por los elementos de la capa anterior y así sucesivamente. El proceso adaptativo queda definido en dos etapas que se van repitiendo hasta que se da por realizado el aprendizaje. En la primera, se evalúa el nivel de actividad de todas las neuronas de la red, manteniendo fijos los

valores de las sinapsis, lo que permite determinar la magnitud del error existente y, en una segunda etapa, dicho error se propaga hacia atrás, capa a capa, modificando convenientemente los pesos que en la etapa siguiente servirán para calcular el nuevo error. Este proceso es el que da nombre al algoritmo de aprendizaje.

Consideremos un red neuronal multicapa con *n* neurona de entrada, *h* neuronas en su capa oculta y *m* neuronas de salida (Figura IV.1). La función de activación es una función diferenciable de la entrada efectiva a la neurona:

$$a_j^p = F_j(e_j^p)$$

en la que

$$e_j^p = \sum_i w_{ij} \cdot a_i^p + \theta_j$$

es la entrada efectiva a la unidad j-ésima para el ejemplo p-ésimo.

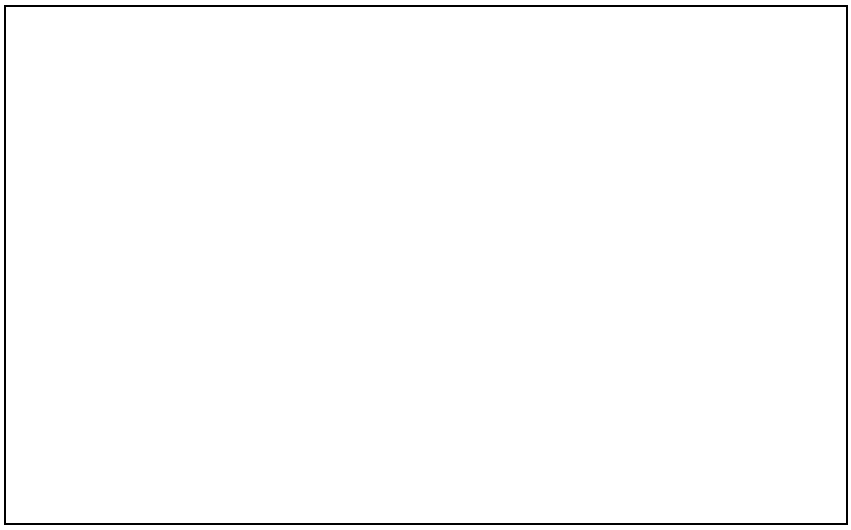


Figura IV.1: Ejemplo de red neuronal multicapa.

Tras la primera etapa en la que se evalúa el error cometido por la red, cada peso se ajusta según:

$$\Delta_p w_{ij} = \gamma \cdot \delta_j^p \cdot a_i^p,$$

donde γ es una constante de proporcionalidad que representa la *tasa de aprendizaje*. El valor de δ^p_j adopta formas distintas para el caso de neuronas de salida o internas. Se comienza haciendo el ajuste para los pesos que unen las neuronas de la capa oculta con las de salida. En este caso:

$$\delta_j^p = (d_j^p - a_j^p) \cdot F_j'(e_j^p)$$

para cualquier neurona de salida *j*. Si se trata de una neurona oculta, los valores de δ^p_j se calculan según:

$$\delta_j^p = F_j'(e_j^p) \cdot \sum_{h=1}^{N_0} (\delta_h^p \cdot w_{hj}).$$

Las dos ecuaciones anteriores ofrecen un procedimiento recursivo para calcular los deltas para todas las neuronas de una red. Este procedimiento constituye la regla delta generalizada o el algoritmo de retropropagación de errores (*backpropagation*).

A continuación, mostramos la particularización de las expresiones del algoritmo para el caso en que la función de activación sea la sigmoide, es decir, si:

$$F(e_j^p) = \frac{1}{1+\exp(-e_j^p)},$$

y su derivada es:

$$F'(e_j^p) = a_j^p \cdot (1 - a_j^p),$$

las deltas para las neuronas de salida resultan:

$$\delta_j^p = (d_j^p - a_j^p) \cdot a_j^p \cdot (1 - a_j^p),$$

y para las neuronas de la capa oculta se tiene:

$$\delta_j^p = a_j^p \cdot (1 - a_j^p) \cdot \sum_{h=1}^{N_0} (\delta_h^p \cdot w_{hj}).$$

IV.2.-Algoritmo de aprendizaje *Perceptron*

Un Perceptron (Figura IV.2) [Patterson96, Widrow60, Winston92] es una neurona artificial donde:

- Las entradas X_i, i=1, ..., n, y los pesos W_i, i=1, ..., n, son, en general, valores reales.

- Las salidas sólo pueden tomar valores binarios ($Y \in \{0,1\}$).

- La operación realizada en su interior es igual a

$$suma = \sum_{i=1}^{n} Xi \cdot Wi\,.$$

- La salida es igual al resultado ofrecido por la función umbral, *f*, al aplicarle el término *suma*, o sea,

$$Y = f(suma) = \begin{cases} 1 & si\ suma > 0 \\ 0 & en\ otro\ caso \end{cases}.$$

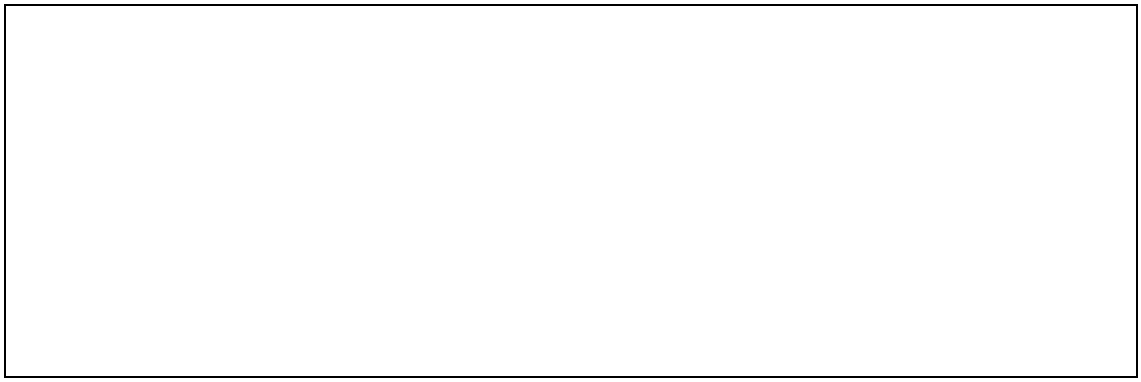


Figura IV.2: Perceptron junto a su función umbral.

El algoritmo de aprendizaje *Perceptron* consiste en modificar los pesos del perceptron, para cada ejemplo, de acuerdo a la siguiente fórmula hasta la finalización del aprendizaje:

$$W_i^{nuevo} = W_i^{viejo} + \Delta W_i, \qquad i = 1,...,n;$$

donde ΔWi es igual a:

$$\Delta W_i = \alpha \cdot (Y_{deseada} - Y) \cdot X_i = \alpha \cdot error \cdot entrada, \qquad i = 1,...,n.$$

donde $Y_{deseada}$ es la salida deseada del perceptron para una entrada particular X_i, i=1, ..., n, y, α es el coeficiente de aprendizaje que controla la velocidad del entrenamiento.

Si α es muy pequeño, el entrenamiento será lento pero estable. Por contra, si α es grande, el entrenamiento puede ser rápido pero puede ocurrir que los pesos oscilen

alrededor del valor necesario para producir salidas correctas para todos los ejemplos de entrenamiento.

IV.3.-Algoritmo de aprendizaje *Pocket*

El algoritmo de aprendizaje *Pocket* consiste en usar el algoritmo de aprendizaje *Perceptron* [Minsky69, Wasserman89] durante un número fijo de ciclos, guardando el conjunto de pesos con mayor número de éxitos al clasificar los ejemplos de entrenamiento. En [Gallant93], se prueba un teorema de convergencia que muestra que los pesos guardados por el algoritmo *Pocket* llegan a ser óptimos con una alta probabilidad después de un número finito de iteraciones.

Apéndice V

ALGORITMOS DE APRENDIZAJE *PERCEPTRON* Y *POCKET*

V.1.-Algoritmo de aprendizaje *Perceptron*

El tipo de perceptron [Patterson96, Winston92] usado en los experimentos de esta tesis (Figura V.1) es una neurona donde:

- Las entradas X_i, i=1, ..., n, y los pesos W_i, i=1, ..., n, pueden tomar valores reales.
- Las salidas toman valores binarios ($Y \in \{0,1\}$).
- La operación realizada en su interior es igual a

$$suma = \sum_{i=1}^{n} Xi \cdot Wi .$$

- La salida es igual al resultado ofrecido por la función umbral, *f*, al aplicarle el término *suma*, o sea,

$$Y = f(suma) = \begin{cases} 1 & si\ suma > 0 \\ 0 & en\ otro\ caso \end{cases}.$$

Figura V.1: Perceptron usado en los experimentos de esta tesis junto a su función umbral.

El algoritmo de aprendizaje *Perceptron* consiste en modificar los pesos del perceptron, para cada ejemplo, de acuerdo a la siguiente fórmula hasta la finalización del aprendizaje:

$$W_i^{nuevo} = W_i^{viejo} + \Delta W_i, \qquad i = 1,...,n;$$

donde ΔWi es igual a:

$$\Delta W_i = \alpha \cdot (Y_{deseada} - Y) \cdot X_i = \alpha \cdot error \cdot entrada, \qquad i = 1,...,n.$$

donde $Y_{deseada}$ es la salida deseada del perceptron para una entrada particular X_i, i=1, ..., n, y, α es el coeficiente de aprendizaje que controla la velocidad del entrenamiento.

Si α es muy pequeño, el entrenamiento será lento pero estable. Por contra, si α es grande, el entrenamiento puede ser rápido pero puede ocurrir que los pesos oscilen alrededor del valor necesario para producir salidas correctas para todos los ejemplos de entrenamiento.

V.1.-Algoritmo de aprendizaje *Pocket*

El algoritmo de aprendizaje *Pocket* consiste en usar el algoritmo de aprendizaje *Perceptron* [Minsky69, Wasserman89] durante un número fijo de ciclos, guardando el conjunto de pesos con mayor número de éxitos al clasificar los ejemplos de entrenamiento. En [Gallant93], se prueba un teorema de convergencia que muestra que los pesos guardados por el algoritmo *Pocket* llegan a ser óptimos con una alta probabilidad después de un número finito de iteraciones.

Apéndice V

MÉTODO DE IDENTIFICACIÓN DE SISTEMAS DIFUSOS DE LEE Y TAKAGI

En [Lee93], Lee y Takagi introducen un método, basado en un algoritmo genético [Goldberg89, Holland75, Michalewicz96], que permite aprender automáticamente la base de reglas difusas completa para un sistema basado en reglas difusas de tipo TSK. De este modo, la definición de los consecuentes de las reglas consiste en el aprendizaje de los parámetros de la función lineal que combina los valores de entrada para obtener las salidas.

Las funciones de pertenencia consideradas en los antecedentes de las reglas son triangulares, aunque los autores indican que el método puede trabajar con cualquier tipo de función de pertenencia parametrizada, tales como las gaussianas, acampanadas, trapezoidales o sigmoidales. Se supone una partición difusa del espacio de entrada. Cada función de pertenencia triangular asociada a cada uno de los conjuntos difusos primarios (términos lingüísticos), se representa mediante tres parámetros (Figura V.1). El primero es el punto central, es decir, el punto modal. Únicamente el centro del triángulo asociado al primer conjunto difuso primario viene especificado por una posición absoluta, mientras que los parámetros asociados a los demás, representan la distancia existente entre el punto central del triángulo actual y el del anterior. Los dos parámetros restantes representan, respectivamente, los puntos derecho e izquierdo de la base del triángulo. Ambos tienen grado de pertenencia cero.

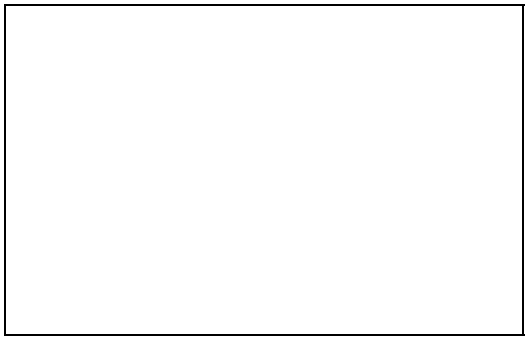


Figura V.1: Función de pertenencia triangular parametrizada.

El algoritmo genético empleado se basa en una codificación binaria. Las funciones de pertenencia se codifican agregando los valores binarios de los tres parámetros asociados en una subcadena binaria, codificando cada parámetro con ocho bits. En un cromosoma del algoritmo genético, se codifican:

1. Los antecedentes de las reglas difusas de un sistema, agregando las codificaciones parciales de las funciones de pertenencia asociadas a cada uno de los conjuntos difusos primarios de las variables de entrada, una detrás de otra.

2. Los parámetros asociados a la función lineal del consecuente de cada regla, usando también ocho bits para cada uno.

De esta manera, cada cromosoma del algoritmo genético representa una base de reglas difusas completa de un sistema de base de reglas difusas de tipo TSK. El número de reglas que forman parte de la misma dependerá del número de conjuntos difusos primarios asociados a cada una de las variables de entrada y, será igual al producto de estos.

La codificación empleada permite decidir el número óptimo de reglas difusas que formarán la base, del siguiente modo: aquellos términos lingüísticos en los que el punto central de la función de pertenencia asociada cae fuera de un límite concreto, obtenido a partir del conocimiento que se posee sobre el sistema, provocan que la regla en cuestión no forme parte de la base de reglas. Así, todas las reglas cuyo antecedente sea una combinación de los valores de entrada válidos codificados en la primera parte del cromosoma, formarán parte de la base de reglas difusas del sistema. Tomemos como ejemplo un cromosoma que codifique la base de reglas difusas de un sistema con dos variables de entrada que posean asociadas, respectivamente, m y n etiquetas lingüísticas válidas. El número total de reglas que puede poseer, como máximo, la base de reglas difusas que codifica es igual a $m \cdot n$.

El algoritmo genético identifica un sistema difuso basado en reglas de tipo TSK intentando minimizar dos términos:

* el número de reglas del sistema y

* el error cuadrático medio producido por el sistema al manipular un conjunto de ejemplos de entrenamiento.

Bibliografía

[Abramson63] N. Abramson, “Information Theory and Coding”, New York : McGraw Hill, 1963.

[Alché94] F. d’Alché-Buc, D. Zwierski & J. -P. Nadal, “Trio learning: a new strategy for building hybrid neural trees”, Int. Journal of Neural Systems, vol. 5, pp. 259-274, 1994.

[Anderson83] J.R. Anderson, “The Architecture of Cognition”, Harvard University Press, Cambridge, MA, 1983.

[Bardossy95] A. Bardossy & L. Duckstein, “Fuzzy Rule-Based Modeling With Application to Geophysical, Biological and Engineering Systems”, CRC Press, 1995.

[Benítez97] J.M. Benítez, J.L. Castro & I. Requena, “Are artificial neural networks black boxes?”, IEEE Transactions on Neural Networks, vol.8, Nº5, 1997.

[Berenji92] H. Berenji, “Fuzzy logic controllers”. En R. Yager & L. Zadeh (Eds.), “An Introduction to Fuzzy Logic Applications in Intelligent Systems”, pp. 69-96, Kluwer Academic.

[Bezdek93] J.C. Bezdek, “A review of probabilistic, fuzzy, and neural models for pattern recognition”, Journal of Intelligent and Fuzzy Systems, vol.1 (1), pp.1-25, 1993.

[Brassard88] G. Brassard & P. Bratley, “Algorithmics. Theory and Practice”, Prentice Hall International, 1988.

[Breiman84] L. Breiman, J.H. Friedman, R.A. Olshen & C.J. Stone, “Classification and Regression Trees”, Belmont, CA: Wadsworth International, 1984.

[Buchanan78] B.G. Buchanan & E.A. Feigenbaum, “DENDRAL and META-DENDRAL”, Artificial Intelligence, 11, (1), pp. 5-24, 1978.

[Buckley93] J. J. Buckley, “Sugeno type controllers are universal approximators”, Fuzzy Sets and Systems, 53, pp. 299-304, 1993.

[Campbell95] C. Campbell & C. Pérez Vicente, “The target switch algorithm: a constructive learning procedure for feed-forward neural networks”, Neural Computation, 7, pp. 1245-1264, 1995.

[Castro95] J.L. Castro, “Fuzzy logic controllers are universal approximators”, IEEE Transactions on Systems, Man and Cybernetics, 25, 4, pp. 629-635, 1995.

[Castro96] J.L. Castro & M. Delgado, “Fuzzy systems with defuzzification are universal approximators”, IEEE Transactions on Systems, Man and Cybernetics, 26, 1, pp. 149-152, 1996.

[Castro98a] J.L. Castro, E. Trillas & J.M. Zurita, "Non-monotonic fuzzy reasoning", Fuzzy Sets and Systems, 94, pp.217-225, 1998.

[Castro98b] J.L. Castro, M. Delgado & C.J. Mantas, “¿Cómo manejar un algoritmo difuso?”, Actas del VIII congreso español sobre tecnologías y lógicas fuzzy, pp. 77-84, 1998.

[Castro99a] J.L. Castro, M. Delgado & C.J. Mantas, “Fuzzy grammar for handling fuzzy algorithms”, sometido en *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*.

[Castro99b] J.L. Castro, M. Delgado & C.J. Mantas, “MORSE: a general model to represent structured knowledge”, sometido en *International Journal of Intelligent Systems.*

[Castro99c] J.L. Castro, M. Delgado & C.J. Mantas, “Structured knowledge representation and inductive learning”, sometido en *IEEE Transactions on Systems, Man and Cybernetics: Part A.*

[Castro99d] J.L. Castro, M. Delgado & C.J. Mantas, “SEPARATE: a method of machine learning based on semiglobal partitions”, sometido en *IEEE Transactions on Neural Networks.*

[Castro99e] J.L. Castro, M. Delgado & C.J. Mantas, “A hybrid system for approximate reasoning”, sometido en *IEEE Transactions on Systems, Man and Cybernetics: Part B.*

[Chandrasekaran87] B. Chandrasekaran, “Towards a functional architecture for intelligence based on generic information processing tasks”, IJCAI’87, vol.2, pp. 1183-1192, 1987.

[Chandrasekaran92] B. Chandrasekaran, T.R. Johnson & J.W. Smith, “Task-structure analysis for knowledge modeling”, Communications of the ACM, vol. 35, nº9, pp. 124-136, 1992.

[Chiang94] C. Chiang & H. Fu, “A divide-and-conquer methodology for modular supervised neural network design”, Int. Joint Conference on Neural Networks, pp. 119-124, 1994.

[Chiu94] S. Chiu, "Fuzzy model identification based on cluster estimation", Journal of Intelligent and Fuzzy Systems, vol.2, pp.267-278, 1994.

[Cios92] K.J. Cios & N. Liu, “A machine learning method for generation of a neural network architecture: a continuous ID3 algorithm”, IEEE Transactions on Neural Networks, vol. 3, Nº 2, pp. 280-290, 1992.

[Damasio90] A. Damasio et al., “Neural regionalization of knowledge access”. En: *Cold Spring Harbor Symp. On Quantitative Biology*, Vol. LV, CSHLC Press, 1990.

[Delgado97] M. Delgado, C.J. Mantas & M. Pegalajar, “A fuzzy control based algorithm to train perceptrons”, Proceedings FUZZ-IEEE’97, vol. II, pp. 1027-1031, 1997.

[Delgado99] M. Delgado, C.J. Mantas & C. Moraga, “A fuzzy rule based backpropagation method for training binary multilayer perceptrons”, Information Sciences, 113, pp. 1-17, 1999.

[Driankov93] D. Driankov, H. Hellendoorn & M. Reinfrank, “An Introduction to Fuzzy Control”, Springer-Verlag, 1993.

[Dubois80] D. Dubois & H. Prade, “Fuzzy Sets and Systems: Theory and Applications”, Academic Press, 1980.

[Fahlman90] S.E. Fahlman & C. Lebiere, "The cascade-correlation learning architecture". En Advances in Neural Information Processing Systems II, D.S.Touretzky, G.Hinton & T.Sejnowski, (Eds), San Mateo, CA: Morgan Kaufmann, 1990.

[Frean90] M. Frean, “The upstart algorithm: a method for constructing and training feedforward neural networks”, Neural Computation, 2, pp. 198-209, 1990.

[Gallant86] S.I. Gallant, “Optimal linear discriminants”, IEEE Proc. 8th Conf. Pattern Recognition, pp.849-852, 1986.

[Gallant93] S.I. Gallant, “Neural Network Learning and Expert Systems”, MIT Press, Cambridge, MA, 1993.

[Giordana97] A. Giordana, F. Neri, L. Saitta & M. Botta, “Integrating multiple learning strategies in first order logics”, Machine Learning, 27, pp. 209-240, 1997.

[Goldberg89] D.E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, New York, 1989.

[Goonatilake95] S. Goonatilake & P. Treleaven (Eds.), “Intelligent Systems for Finance and Business”, John Willey & Sons, 1995.

[Hebb49] D. Hebb, “Organization of Behaviour”, John Wiley & Sons, New York, 1949.

[Hirota93] K. Hirota (Eds.), “Industrial Applications of Fuzzy Technology”, Springer-Verlag, 1993.

[Holland75] J.H. Holland, “Adaptation in Natural and Artificial Systems”, Ann arbor: The University of Michigan Press, 1975.

[Hwang94] J.N. Hwang, S.R. Lay, M. Maechler, D. Martin & J. Schimert, “Regression modeling in backpropagation and projection pursuit learning”, IEEE Transactions on Neural Networks, vol. 5, pp. 342-353, 1994.

[Jacobs91] R.A. Jacobs, M. Jordan, S.J. Nowlan & G. E. Hinton, “Adaptive mixtures of local experts”, Neural Computation, 3, pp. 79-87, 1991.

[Jain88] A.K. Jain & R.C. Dubes, “Algorithms for Clustering Data”, Prentice Hall Advanced Reference Series, 1988.

[Jang93] J.S. Jang, "ANFIS: Adaptative-Network-Based fuzzy inference system", IEEE Transactions on Systems, Man and Cybernetics, vol. 23, nº3, 1993.

[Keil89] F. Keil, “Concepts, Kinds, and Cognitive Development”, MIT Press, Cambridge, MA, 1989.

[Klir95] G.J. Klir & B. Yuan, “Fuzzy Sets and Fuzzy Logic: Theory and Applications”, Prentice Hall, 1995.

[Kosko92] B. Kosko, “Neural Networks and Fuzzy Systems”, Prentice-Hall, 1992.

[Kosko94] B. Kosko, “Fuzzy systems are universal approximators”, IEEE Transactions on Computers, vol. 43, nº11, pp. 1324-1333, 1994.

[Krose93] B.J.A. Kröse & P.D. van der smagt, “An Introduction to Neural Networks”, University of Amsterdam, 1993.

[Lee90a] C.C. Lee, “Fuzzy logic in control systems: Fuzzy logic controller. Part I”, IEEE Transactions on Systems, Man and Cybernetics, 20, 2, pp. 404-419, 1990.

[Lee90b] C.C. Lee, “Fuzzy logic in control systems: Fuzzy logic controller. Part II”, IEEE Transactions on Systems, Man and Cybernetics, 20, 2, pp. 420-435, 1990.

[Lee93] M.A. Lee, & H. Takagi, "Integrating design stages of fuzzy systems using genetic algorithms", Proc. IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'93), San Francisco, CA, pp.612-617, 1993.

[Lee96] M.A. Lee, & H. Takagi, "Hybrid genetic-fuzzy systems for intelligent systems design". En *Genetic algorithms and Soft Computing*, F.Herrera & J.L.Verdegay (Eds.), Physica Verlag, Heidelberg, pp.226-250, 1996.

[Lippmann87] R.P. Lippmann, "An introduction to computing with neural nets", IEEE ASSP Magazine, pp.4-22, April, 1987.

[Littmann96] E. Littmann & H. Ritter, “Learning and generalization in cascade network architectures”, Neural Computation, 8, pp. 1521-1539, 1996.

[Lukaszewicz90] W. Lukaszewicz, “Non-Monotonic Reasoning: Formalization of Commonsense Reasoning”, Ellis Horwood series in Artificial Intelligence, 1990.

[McDermott88] J. McDermott, “Preliminary steps toward a taxonomy of problem-solving methods”. En S. Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pp. 225-255, Kluwer, Boston, 1988.

[Mezard89] M. Mezard & J-P. Nadal, “Learning in feedforward layerd networks: the tiling algorithm”, J. Phys. A: Math. Gen., 22, pp. 2191-2203, 1989.

[Michalewicz96] Z. Michalewicz, “Genetic algorithms + Data Structures = Evolution Programs”, Springer-Verlag, 1996.

[Michalski93] R.S. Michalski, “Inferential theory of learning as a conceptual basis for multistrategy learning”, Machine Learning, 11, pp. 111-151, 1993.

[Minsky69] M. Minsky, & S. Papert, "Perceptrons", MIT Press, 1969.

[Minsky75] M. Minsky, “A framework for representing knowledge”. En Winston, P. H. *The Psychology of Computer Vision*. New York: McGraw-Hill.

[Moody96] J.O. Moody & P.J. Antsaklis, “The dependence identification neural network construction algorithm”, IEEE Transactions on Neural Networks, vol.7, nº1, pp. 3-15, 1996.

[Muller90] B. Müller & J. Reinhardt, “Neural Networks. An Introduction”, Springer-Verlag, Berlin, 1990.

[Park90] Y. Park & J. Sklansky, “Automated design of linear tree classifiers”, Pattern Recognition, vol. 23, Nº 12, pp. 1393-1412, 1990.

[Patterson96] D.W. Patterson, “Artificial Neural Networks: Theory and Applications”, Prentice Hall, 1996.

[Pedrycz96] W. Pedrycz (Ed.), “Fuzzy Modeling: Paradigms and Practice”, Kluwer Academic, 1996.

[Quillian68] R. Quillian, “Semantic memory”. En Minsky, M. *Semantic Information Processing*. Cambridge, Massachusetts: Massachusetts Institute of Technology, Press, 1968.

[Quinlan86] J.R. Quinlan, “Induction of decision trees”, Machine Learning, vol.1, pp. 81-106, 1986.

[Quinlan93] J.R. Quinlan, “C4.5 : Programs for Machine Learning”, Morgan Kauffman, 1993

[Raju93] G.V.S. Raju & J. Zhou, "Adaptive hierarchical fuzzy controller", IEEE Transactions on Systems, Man and Cybernetics, vol.23, Nº4, pp.973-980, 1993.

[Raphael68] B. Raphael, “A computer program for semantic information retrieval”. En Minsky, M. *Semantic Information Processing*. Cambridge, Massachusetts: Massachusetts Institute of Technology, Press, 1968.

[Romaniuk93] S.G. Romaniuk & L.O. Hall, “Divide and conquer neural networks”, Neural Networks, vol.6, pp. 1105-1116, 1993.

[Roy93] A. Roy, L.S. Kim & S. Mukhopadhyay, “A polynomial time algorithm for the construction and training of a class of multilayer perceptrons”, Neural Networks, vol.6, pp. 535-545, 1993.

[Rumelhart86] D.E. Rumelhart, G.E. Hinton & R.J. Williams, “Learning internal representations by error propagation”. En *Parallel Distributed Processing : Explanations in the Microstructure of Cognition*, vol. 1 : Foundation, D.E. Rumelhart and J.L. McClelland, Eds. Cambridge, MA : MIT Press, pp.318-362, 1986.

[Saffiotti97] A. Saffiotti, "Fuzzy logic in autonomous robotics: behavior coordination", Proc. IEEE Int. Conference on Fuzzy Systems (FUZZ-IEEE'97), pp.573-578, 1997.

[Saffiotti95] A. Saffiotti, K. konolige & E.H. Ruspini, "A multivalued-logic approach to integrating planning and control", Artificial Intelligence, 76 (1-2):481-526, 1995.

[Saffiotti93] A. Saffiotti, E.H. Ruspini & K. Konolige, "Blending reactivity and goal-directedness in a fuzzy controller", Proc. of the 2nd Fuzzy-IEEE Conf., pp.134-139, 1993.

[Shortliffe75] E.H. Shortliffe & B.G. Buchanan, “A model of inexact reasoning in medicine”, Mathematical Biosciences, 23, pp. 351-379, 1975.

[Shortliffe76] E.H. Shortliffe, “Computer-Based Medical Consultations: MYCIN”, New York: Elsevier, 1976.

[Simpson89] P.K. Simpson, “Artificial Neural Systems”, Pergamon Press, 1989.

[Sirat90] J.A. Sirat & J.-P. Nadal, “Neural trees: a new tool for classification”, NETWORK, vol. 1, pp. 423-438, 1990.

[Smith88] J.W. Smith, J.E. Everhart, W.C. Dickson, W.C. Knowler & R.S. Johannes, “Using the ADAP learning algorithm to forecast the outset of diabetes mellitus”. Proceedings of the Symposium on Computer Applications and Medical Care”, IEEE Computer Society Press, pp. 261-265, 1988.

[Steels90] L. Steels, “Components of expertise”, AI Magazine, Summer issue, 1990.

[Sugeno85] M. Sugeno & M. Nishida, “Fuzzy control of model car”, Fuzzy Sets and Systems, 16, pp. 103-113, 1985.

[Sugeno88] M. Sugeno & G.,T. Kang, “Structure identification of fuzzy model”, Fuzzy Sets and Systems, 28, pp.15-33, 1988.

[Sugeno93] M. Sugeno, T. Yasukawa, "A fuzzy-Logic-based approach to qualitative modeling", IEEE Transactions on Fuzzy Systems, vol. 1, nº1, 1993.

[Sun94] C.-T. Sun, "Rule-base structure identification in an adaptive-network-based fuzzy inference system", IEEE Transactions on Fuzzy Systems, vol.2, nº1, 1994.

[Sun94b] R. Sun, “Integrating Rules and Connectionism for Robust Commonsense Reasoning”, John Wiley and Sons, New York, NW, 1994.

[Tajine98] M. Tajine & D. Elizondo, “Growing methods for constructing recursive deterministic perceptron neural networks and knowledge extraction”, Artificial Intelligence, 102, pp. 295-322, 1998.

[Takagi85] Y. Takagi, M. Sugeno, "Fuzzy identification of Systems and its Application to modeling and control", IEEE transactions on Systems, Man and Cybernetics, 15:116-132, 1985.

[Takagi91] H. Takagi & I. Hayashi, "Neural-network driven fuzzy reasoning", International Journal of Approximate Reasoning, 5:191-212, 1991.

[Takagi92] H. Takagi, N. Suzuki, T. Koda & Y. Kojima, "Neural networks designed on approximate reasoning architecture and their applications", IEEE Transactions on Neural Networks, vol.3, Nº5, pp.752-760, 1992.

[Tomek76] I. Tomek, “Two modifications of CNN”, IEEE Transactions on Systems, Man and Cybernetics. SMC-6, pp.769-772, 1976.

[Wang92] L. Wang & J.M. Mendel, “Generating fuzzy rules by learning from examples”, IEEE Transactions on Systems, Man and Cybernetics, vol.22, nº6, 1992.

[Wang92b] L.X. Wang, “Fuzzy systems are universal approximators”, En Proc. First IEEE International Conference on Fuzzy Systems (FUZZ-IEEE’92), pp.1163-1170, 1992.

[Wang94] L.X. Wang, “Adaptive Fuzzy Systems and Control”, Prentice-Hall, 1994.

[Wasserman89] P.D. Wasserman, “Neural Computing: Theory and Practice”, Van Nostrand Reinhold, New York, 1989.

[Wasserman93] P.D. Wasserman, “Advanced Methods in Neural Computing”, Van Nostrand Reinhold, 115, Fifth Avenue, New York, NY, 10003, 1993.

[Widrow60] B. Widrow & M. Hoff, “Adaptive switching circuits”. En WESCON Convention Record, vol. 4, pp. 96-104, 1960.

[Wielinga92] B.J. Wielinga, A. T. Schreiber & J.A. Breuker, “KADS: a modelling approach to knowledge engineering”, Knowledge Acquisition, 4, pp. 5-53, 1992.

[Winston92] P. H. Winston, “Artificial Intelligence”, Addison Wesley, 1992.

[Yager94] R.R. Yager, D. Filev, "Generation of fuzzy rules by mountain clustering", Journal of Intelligent and Fuzzy Systems, vol. 2, pp.209-219, 1994.

[Zadeh65] L.A. Zadeh, “Fuzzy sets”, Information and Control, 8, pp. 338-353, 1965

[Zadeh68] L.A. Zadeh, “Fuzzy algorithms”, Information and Control., 12, pp. 94-102, 1968.

[Zadeh73] L.A. Zadeh, “Outline of a new approach to the analysis of complex systems and decision processes”, IEEE Transactions on Systems, Man and Cybernetics, 3, pp. 28-44, 1973.

[Zadeh75a] L.A. Zadeh, “The concept of a linguistic variable and its applications to approximate reasoning. Part I”, Information Sciences, 8, pp. 199-249, 1975.

[Zadeh75b] L.A. Zadeh, “The concept of a linguistic variable and its applications to approximate reasoning. Part II”, Information Sciences, 8, pp. 301-357, 1975.

[Zadeh75c] L.A. Zadeh, “The concept of a linguistic variable and its applications to approximate reasoning. Part III”, Information Sciences, 9, pp. 43-80, 1975.

[Zimmermann96] H.J. Zimmermann, “Fuzzy Sets: Theory and its Applications”, Kluwer Academic, 1996.

[Zollner92] R. Zollner, H.J. Schmitz, F. Wunsch & U. Krey, “Fast generating algorithm for a general three-layer perceptron”, Neural Networks, vol. 5, pp. 771-777, 1992.