Universidade da Coruña

FACULTADE DE INFORMÁTICA

Departamento de Tecnoloxías da Información e as Comunicacións

TESIS DOCTORAL

METODOLOGÍA PARA EL DESARROLLO DE SISTEMAS DE EXTRACCIÓN DE CONOCIMIENTO EN RNA

DOCTORANDO: JUAN RAMÓN RABUÑAL DOPICO

DIRECTORES: ALEJANDRO PAZOS SIERRA

JULIÁN DORADO DE LA CALLE

A Coruña, Junio 2002

Universidade da Coruña

FACULTADE DE INFORMÁTICA

Departamento de Tecnoloxías da Información e as Comunicacións

TESIS DOCTORAL

METODOLOGÍA PARA EL DESARROLLO DE SISTEMAS DE EXTRACCIÓN DE CONOCIMIENTO EN RNA

DOCTORANDO: JUAN RAMÓN RABUÑAL DOPICO

DIRECTORES: ALEJANDRO PAZOS SIERRA

JULIÁN DORADO DE LA CALLE

A Coruña, Junio 2002

UNIVERSIDADE DA CORUÑA

DEPARTAMENTO DE TECNOLOXÍAS
DA INFORMACIÓN E AS COMUNICACIÓNS

Facultade de Informática
Campus de Elviña, s/n.
15071 A Coruña
Telf. 981 167 000
Fax 981 167 160
E-mail: tic@udc.es

D. Alejandro Pazos Sierra y **D. Julián Dorado de la Calle**, Profesores Catedrático y Titular Interino de Universidad, del Dpto. de Tecnoloxías da Información e as Comunicacións en la Universidade de A Coruña, en cumplimiento del punto 1 del art. 18 del reglamento de estudios de doctorado,

AUTORIZAN la presentación de la tesis doctoral titulada "*Metodología para el Desarrollo de Sistemas de Extracción de Conocimiento en RNA*", que ha sido realizada bajo nuestra dirección por **D. Juan Ramón Rabuñal Dopico**, en el Departamento de Tecnoloxías da Información e as Comunicacións de la Universidade de A Coruña, y que presenta para optar al grado de Doctor en Informática.

A Coruña, 14 de Junio de 2002

Fdo.: Dr. Alejandro Pazos Sierra
Director de la tesis doctoral

Fdo.: Dr. Julián Dorado de la Calle
Director de la tesis doctoral

Agradecimientos

Debo agradecer a todos los miembros del laboratorio de Redes de Neuronas Artificiales y Sistemas Adaptativos (RNASA) de la Facultad de Informática de la Universidade da Coruña su paciencia, inestimable compañía y ayuda. Especialmente a su director, Alejandro Pazos, que me ha apoyado, dirigido y conducido para llegar a buen puerto con este trabajo.

También tengo que dar las gracias al profesor del departamento de Tecnoloxías da Información e as Comunicacións Julián Dorado por haber sufrido con interminable paciencia todas mis preguntas y dudas que me han surgido a lo largo de todo el período de desarrollo de la presente tesis.

A Daniel Rivero por su inestimable colaboración en el desarrollo y las pruebas del sistema.

A mi esposa por todo el ánimo y dedicación que me ha brindado durante estos años.

A mis padres, abuelos y tía-abuela por haberme formado y enseñado tantas cosas importantes que no se aprenden en la escuela.

Finalmente le agradezco a todos mis compañeros de trabajo del Centro de Innovación Tecnolóxica en Edificación e Enxeñería Civil (CITEEC), y especialmente a su director Jerónimo Puertas, por su generoso apoyo y comprensión.

A mis padres,

esposa,

familia y

amigos

ÍNDICE

"Si un viejo pero distinguido científico dice que algo es posible, casi seguro que está en lo cierto, pero si dice que es imposible es muy probable que se equivoque"

Arthur C. Clarke

CAPÍTULO

1 INTRODUCCIÓN

1.1 Motivación

La Inteligencia Artificial (en adelante IA) es la ciencia que permite el diseño de sistemas inteligentes, es decir, que exhiben características que se asocian con la inteligencia humana, como son el entender lenguaje natural, el aprendizaje, el razonamiento, etc. Su objetivo es por una lado, hacer computadoras más útiles y, por otro, entender los principios que hacen posible la inteligencia. Para ello se construyen o se programan computadoras para que hagan tareas que si fuesen realizadas por un ser humano se diría que son inteligentes.

Una de las más rápidas y sólidas conclusiones que surgieron en las tres primeras décadas de las investigaciones de la IA fue que la inteligencia necesita conocimiento, y pronto se descubrió que el conocimiento posee algunas propiedades poco deseables, como que es voluminoso, es difícil de caracterizar con exactitud, cambia constantemente y se organiza de tal forma que se corresponde con la forma más eficiente en que pueda ser usado.

De esta manera, se puede deducir que las técnicas de IA utilizan conocimiento representado de tal forma que:

1. El conocimiento identifique las generalizaciones. En otras palabras, no es necesario representar de forma separada cada situación individual. En lugar de esto se agrupan las situaciones que comparten propiedades importantes.
2. Debe ser comprendido por las personas que lo proporcionan. En IA, el conocimiento que se suministra lo proporcionan personas haciéndolo siempre en términos que ellos comprenden. Esto es más notorio en los Sistemas Expertos (en adelante SSEE) que son unos programas informáticos que recogen, para su adecuado funcionamiento, el conocimiento de uno o varios expertos.
3. Puede modificarse fácilmente para corregir errores y reflejar los cambios en la visión del mundo.
4. Puede usarse en gran cantidad de situaciones aún cuando no sea totalmente preciso o completo.
5. Puede usarse para ayudar a superar los problemas creados por su propio volumen, ayudando a acotar el rango de posibilidades que normalmente deben ser consideradas.

Así, se puede concluir que el objetivo de las técnicas de IA es apoyar el uso eficaz del conocimiento. En las soluciones en las que se usan técnicas de IA se ponen de manifiesto tres técnicas muy importantes para el tema que aquí nos ocupa:

1. **La búsqueda**. – Proporciona una forma de resolver problemas en los que no se dispone de un método directo para la obtención de una solución.
2. **La estructuración**. – Proporciona una forma de resolver problemas complejos explotando las estructuras de los objetos involucrados.

3. **La abstracción**. – Proporciona una forma de separar aspectos y variaciones importantes de aquellas características menos relevantes y que podrían colapsar el proceso de búsqueda.

Clásicamente las técnicas de IA se dividían en dos grandes grupos: la **IA Conexionista**, donde se incluyen las Redes de Neuronas Artificiales (en adelante RR.NN.AA), los **Sistemas Conexionistas** (en adelante SC) y la **IA Simbólica**, donde se incluyen los Sistemas Expertos (en adelante SS.EE). Hoy en día se contempla un nuevo gran grupo que incluye la **Computación Biológica**, donde se incluyen los Algoritmos Genéticos (en adelante AA.GG) y la Programación Genética (en adelante PG).

1.1.1 Sistemas Conexionistas

Desde la primera mitad del siglo XX se han empezado a desarrollar modelos computacionales que han intentado emular el comportamiento del cerebro humano [MCCU 43]. Aunque se han propuesto una gran cantidad de ellos, todos usan una estructura en red en la cual los nodos o neuronas son procesos numéricos que involucran estados de otros nodos según sus uniones. Una clase de estos modelos computacionales son las RR.NN.AA.

Las RR.NN.AA se han hecho muy populares debido a la facilidad en su uso (ver figura 1) e implementación y la habilidad para aproximar cualquier función matemática.

Las RR.NN.AA, con su marcada habilidad para obtener resultados de datos complicados e imprecisos, pueden utilizarse para extraer patrones y detectar tramas que son muy difíciles de apreciar por humanos u otras técnicas computacionales. Una red neuronal entrenada puede usarse como un "experto" para categorizar la información que se ha dado para su análisis. Este experto puede usarse para proveer proyecciones de nuevas situaciones.

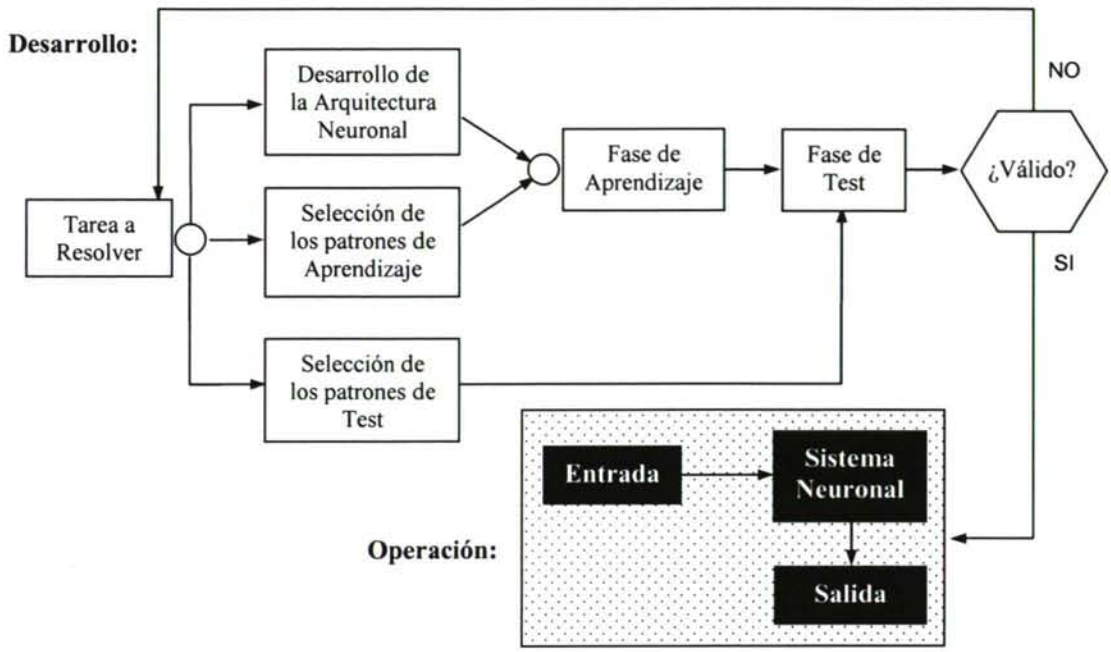


***Fig. 1** - Modo de trabajo con Redes de Neuronas Artificiales*

Las ventajas más reseñables de las RR.NN.AA son las siguientes:

1. **Aprendizaje adaptativo**. Capacidad de aprender a realizar tareas basadas en un entrenamiento o una experiencia inicial.
2. **Autoorganización**. Una red neuronal puede crear su propia organización o representación de la información que recibe durante la etapa de aprendizaje.
3. **Tolerancia a fallos gracias a poseer la información distribuída o vía información redundante**. La destrucción parcial de una red puede conducir a una degradación de su estructura; sin embargo, algunas capacidades de la red se pueden retener, incluso sufriendo daños considerables.
4. **Capacidad de generalización**. Ante la entrada de datos nuevos es capaz de producir resultados coherentes de acuerdo con la naturaleza del problema para el cual han sido entrenadas.

5. **Operación en tiempo real**. El cómputo neuronal puede realizarse en paralelo, bien vía software o mediante máquinas especiales para obtener esta ventaja (hardware conexionista o masivamente paralelo).

Pero los sistemas neuronales presentan ciertos inconvenientes. Uno importante es que habitualmente realizan un complejo procesamiento que supone millones de operaciones, por lo que no es posible seguir paso a paso el razonamiento que les ha llevado a extraer sus conclusiones. Sin embargo, en redes pequeñas, mediante simulación o por el estudio de los pesos sinápticos sí es posible saber, al menos, qué variables de las introducidas han sido relevantes para tomar la decisión.

1.1.2 Sistemas Expertos

Los SS.EE son el producto más conocido de la rama simbólica de la IA. La forma en que representan el conocimiento, habitualmente mediante símbolos, es apropiada cuando es posible extraer un conjunto de reglas y normas. Su origen se sitúa a mediados de los años setenta; sin embargo, es a partir de la década de los ochenta cuando se desarrollan aplicaciones en toda su plenitud. Un sistema experto recopila en un programa informático el conocimiento de especialistas en una materia. Sus dos componentes principales son la *Base de Conocimiento* y el *Motor de Inferencia*. El saber de un experto se representa mediante el uso de símbolos, creando una base de conocimiento, posteriormente se diseña un programa de inferencia que manipula la información simbólica almacenada en dicha base de conocimiento mediante procesos de búsqueda (ver figura 2) y resolución de conflictos.

La tarea de adquisición del conocimiento es una tarea compleja que precisa de varios actores:

1. El ingeniero del conocimiento que es el especialista informático que extrae el conocimiento del especialista humano y lo plasma en el programa informático.

2. El especialista humano, que es quien posee el conocimiento, la experiencia.
3. El usuario del sistema, encargado de utilizar el sistema experto.

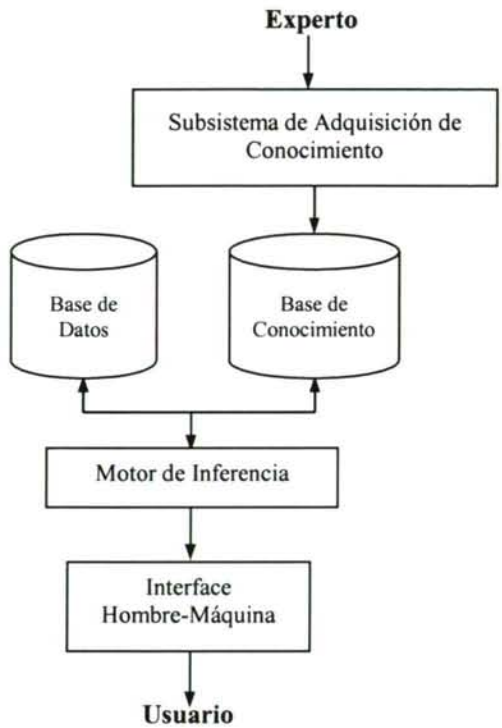


***Fig. 2** - Modo de trabajo con Sistemas Expertos*

Los SS.EE tratan de representar el conocimiento de forma simbólica, partiendo de la premisa de que los expertos humanos utilizan gran número de reglas heurísticas específicas en un determinado campo, las cuales son incorporadas al sistema.

Las motivaciones para desarrollar y utilizar un SE son las siguientes (Edward Feigenbaum):

1. Los expertos son escasos y costosos.
2. Permite mayor accesibilidad a la experiencia.
3. Tiene conocimientos utilizables de manera inmediata, descentralizada y duradera.

4. El conocimiento tiene poder organizador.
5. El razonamiento es accesible a los usuarios.
6. Su funcionamiento es consistente.

Pese a su innegable potencia y utilidad, los SS.EE presentan una serie de inconvenientes como son su programación y mantenimiento, la dificultad de elicitar y representar adecuadamente el conocimiento, el elevado coste en tiempo y dinero para educir el conocimiento de los especialistas humanos, la poca flexibilidad a cambios y la dificultad que presenta la manipulación de información incompleta, inconsistente o errónea. Otro hecho importante a tener en cuenta es que la justificación de sus conclusiones generalmente se reduce a la enumeración o "traza" de las reglas usadas.

1.1.3 Planteamiento

La idea que se presenta en esta tesis es diseñar y construir un sistema que tenga la potencia y la utilidad de los dos grupos: las RR.NN.AA y los SE. Las RR.NN.AA son de fácil implementación y utilización, así como otras características que las hacen idóneas para resolver problemas en muchos campos; sin embargo muchos desarrolladores e investigadores evitan su uso porque se consideran "cajas negras", es decir, son sistemas donde a partir de una serie de entradas produce unas salidas de respuesta y el proceso o explicación de por qué produce esas salidas es desconocido. Por ello, en muchos ámbitos, como puede ser la toma de decisiones en medicina, donde suelen dar buenos resultados, su utilización es bastante difícil, a pesar de su elevada fiabilidad, ya que no explican el por qué de su correcto funcionamiento y de las soluciones que aporta. Como ejemplo, se puede citar el ámbito del diagnóstico médico. Un sistema computacional diseñado para dar el diagnóstico, además de ser correcto, debe poder dar una justificación de cómo y porqué ha llegado a dicho diagnóstico, para que pueda ser utilizado en la clínica.

Sin embargo, los SS.EE poseen como núcleo principal y garantía de éxito el hecho de poder dar una explicación de la solución o respuesta que producen. Por ello,

en esta tesis se pretende desarrollar un sistema que realice la extracción automática de reglas a partir de RR.NN.AA ya entrenadas y esta forma obtener el conocimiento que una RNA extrae del problema que resuelve.

1.2 Objetivos

Hasta ahora se han utilizado diferentes técnicas de extracción de reglas a partir de RR.NN.AA, pero mayormente aplicadas sobre RR.NN.AA multicapa debido a que son más fáciles de tratar y en muchos de ellos centrados en ciertas arquitecturas y modos de entrenamiento de las mismas. En cambio, son prácticamente nulos los métodos que traten la extracción de reglas de las RR.NN.AA como sistemas independientes de su arquitectura, entrenamiento y distribución interna de pesos, conexiones y funciones de activación. **Ésta es la principal causa por la que se ha planteado la realización de ésta tesis doctoral: buscar un mecanismo de extracción automática y eficiente de las reglas de transferencia de información existentes en una RNA entrenada. Con esto se pretende hacer explícito el conocimiento que una RNA extrae de un problema. En esta tesis se propone como tarea fundamental proporcionar una metodología para la extracción de reglas de RR.NN.AA independientemente de su arquitectura: multicapa o monocapa, alimentada hacia delante o recurrente, y de los modos de entrenamiento que han tenido**.

Como se puede suponer, la extracción de reglas de una RNA de arquitectura Recurrente (en adelante RNAR) se complica de forma exponencial debido a que existen ciclos de conexiones realimentadas y por tanto intervienen estados pasados de las activaciones de las neuronas. Además, su capacidad para almacenar conocimiento es considerablemente muy superior a las RR.NN.AA alimentadas hacia delante debido a que pueden existir muchas más conexiones entre los elementos de proceso al no existir restricciones en su conectividad. Las RR.NN.AA.RR son utilizadas en problemas dinámicos donde intervienen características temporales como la predicción de series temporales. La tarea de extracción de conocimiento implementados por los métodos desarrollados hasta ahora se vuelve ardua y en gran medida imposible para la mayoría

de ellos. Se presenta, por tanto, el diseño de una metodología y de algoritmos que sean aplicables a todo tipo de RR.NN.AA y, para ello, es imprescindible que cumplan una serie de características [TICK 98]:

1. **Independientes de las arquitecturas de la RNA.** Un mecanismo de extracción de reglas que pueda trabajar con todos los tipos de redes neuronales, incluidas las profusamente interconectadas y las recurrentes.
2. **Independientes de los algoritmos de entrenamiento de la RNA.** Muchos de los algoritmos propuestos se centran en un proceso de entrenamiento de RNA determinado para el cual se favorece la extracción de reglas. Por lo tanto no valen para otros tipos de entrenamiento.
3. **Debe ser correcto y coherente.** Muchos mecanismos de extracción de reglas generan sólo aproximaciones de funcionamiento. Es deseable que las reglas describan la RNA tan correctamente como sea posible.
4. **Debe tener un alto nivel expresivo.** El lenguaje de reglas (sintaxis) caracteriza la compactación del conocimiento simbólico extraído. Generalmente, los lenguajes potentes son los deseables debido a que se puede producir un conjunto de reglas muy compacto y fácil de entender.

Como el objetivo es crear un sistema automático de extracción de reglas que se aproxime al máximo a las características anteriores, se va a tomar como base lo realizado en el proyecto de fin de carrera [RABU 99], de quién realiza el presente trabajo de investigación, donde se ha desarrollado un sistema de entrenamiento común de RR.NN.AA tanto alimentadas hacia delante como recurrentes, mediante AA.GG. De esta forma, partiendo de un sistema de entrenamiento de RR.NN.AA independiente de su arquitectura, se diseñará y se realizará un algoritmo de extracción de reglas de las redes entrenadas.

Se propone como método principal para la extracción de reglas de RR.NN.AA la utilización de la computación evolutiva debido a que ésta ha demostrado su enorme potencial en las tareas de búsqueda donde el espacio de soluciones crece de forma

exponencial según el número de variables que tenga el problema a resolver. En concreto, se propone la utilización de la PG debido a que conlleva la ventaja de que la forma de representación y estructuración de la información se realiza mediante un árbol semántico. Esta representación en árbol es la forma natural de representación de una regla. Regla que será, en cierta medida, fácilmente entendible por el ser humano.

En la solución del sistema se propone añadir la funcionalidad de auto-explicación a las RR.NN.AA. Para ello, se va a desarrollar un sistema que evolucione el conjunto de reglas que modelizan el funcionamiento de una red. Este conjunto de reglas extraídas de la RNA constituyen lo que se conoce como un sistema basado en reglas. En otras palabras, se intenta obtener una representación entendible del conocimiento que atesora una red neuronal.

Se buscará, además, aportar una metodología clara y concisa sobre las diferentes tareas en el proceso de extracción de reglas de las RR.NN.AA, atendiendo a clasificar las diferentes formas según su arquitectura y los problemas que pretenden resolver.

Se comprobará el correcto funcionamiento del sistema propuesto, así como su metodología, con casos experimentales basados en problemas genéricos. Una vez verificado y contrastado su correcto funcionamiento con otras técnicas de extracción existentes se probarán casos reales basados en tareas de clasificación. En todos los casos se probará la extracción directa de reglas de dichos datos y mediante el entrenamiento de RR.NN.AA, para en cada caso extraer las reglas pertinentes a cada red. Se contrastarán los resultados obtenidos con otras técnicas existentes. Finalmente, se probará el algoritmo con RR.NN.AA recurrentes en tareas de predicción de series temporales, para ello se utilizará en un primer momento una RNA entrenada para predecir una serie temporal caótica de laboratorio y, posteriormente, se trabajará con series temporales reales.

Una vez realizados estos casos de prueba, se introducirá un nuevo ámbito de trabajo: la extracción de conocimiento de las RR.NN.AA en forma de rangos de operatividad. Una de las grandes ventajas de la CE, y por ende de la PG es su

flexibilidad en la codificación de los diferentes tipos de problemas que pueden abarcar. Así, de esta forma se introducirán nuevos operadores aplicables al proceso de búsqueda de reglas: los rangos. Estas pruebas están orientadas a las RR.NN.AA que realizan tareas de clasificación y se busca obtener los rangos donde las diferentes clasificaciones son efectuadas. De esta manera se puede dar, en función de los rangos obtenidos, una exhaustiva lista de requerimientos de funcionalidad de estas redes.

Finalmente, se harán una serie de investigaciones sobre una de las que se considera como fundamentales tareas de identificación de funcionalidad de las RR.NN.AA: su capacidad de generalización. Esta capacidad consiste en que, ante la entrada de datos nuevos, la RNA sea capaz de producir resultados coherentes de acuerdo con la naturaleza del problema para el cual ha sido entrenada. Esta característica de las RR.NN.AA es la que le ha dado su principal éxito y por su enorme potencia en la aplicación de múltiples tareas complejas del mundo real, desde tareas de clasificación de patrones y de reconocimiento de secuencias hasta de predicción de series temporales.

Para ello, se probarán diferentes configuraciones y formas de extracción de reglas para obtener el conocimiento sobre generalización que tienen las redes. Se buscarán, principalmente, dotando al sistema de un método automático de creación de nuevos casos de prueba que una vez aplicados a la RNA podrán refinar e incrementar las reglas obtenidas hasta el punto de dar el comportamiento general de la RNA.

1.3 Estructuración de la tesis

La presente tesis está estructurada en varios capítulos de forma que partiendo de una serie de datos sobre los métodos y técnicas existentes actualmente, analizar las diversas formas de conseguir uno por uno los diferentes objetivos que se acaban de presentar.

En el capítulo 2, se verá qué son las RR.NN.AA, tanto desde la perspectiva biológica como desde la perspectiva artificial de sus componentes y la interrelación entre ellos. Se verá cual es el proceso de entrenamiento de estas RR.NN.AA y los tipos,

así como las características específicas de las redes recurrentes. Con ello, se pretende identificar los datos que son necesarios sobre la complejidad de las diferentes estructuras de las RR.NN.AA para poder diseñar un algoritmo de extracción de conocimiento de ellas. Así mismo, se mostrará el concepto de Identificación de Sistemas (en adelante IS), el cuál dará las reglas y la metodología necesaria para poder afrontar el reto de construir un sistema de extracción robusto. Se verán los diferentes modelos existentes en la teoría de IS clasificados por su nivel de complejidad y según las diferentes estructuras de sistemas a modelar. Posteriormente, se darán una serie de técnicas existentes en la extracción de conocimiento de base de datos (KDD). Estas técnicas se basan en analizar los datos para, a partir de ellos inducir reglas, por lo tanto realizan un proceso similar al que se hace cuando se trata de extraer reglas de una RNA en función de sus patrones de entrada-salida. Así, es imprescindible analizar los métodos existentes en esta disciplina para aprovechar su experiencia en el análisis sobre las relaciones que se pueden abstraer de los datos. Finalmente, se aportarán más nociones de lo que se entiente por PG, su funcionamiento y la forma en la que se podrá realizar el algoritmo de extracción de reglas que obtenga todas las ventajas de la CE y concretamente de la PG.

En el capítulo 3, se mostrarán los métodos de extracción de reglas de RR.NN.AA que se han desarrollado hasta el momento. Se dará una clasificación de los mismos donde se indicarán las características más relevantes de cada uno y los principales problemas que puedan plantear. De esta forma, se puede evitar cometer los mismos errores que muchos de ellos tienen y así disponer de las ventajas que puedan tener y la forma de poder incorporarlas al sistema que se propondrá posteriormente.

El capítulo 4 es el encargado de sentar las bases de desarrollo del sistema de extracción de reglas. En el capítulo 5, se sigue la metodología de IS aportada en el capítulo 2 para el desarrollo de las diferentes fases de todo sistema de extracción de conocimiento. Además, se puede considerar como una metodología adaptada a la extracción de reglas donde se explica, paso a paso, cómo se puede desarrollar un

sistema de extracción que sea robusto, fiable y con reglas extraídas altamente expresivas.

En el capítulo 6, se describe el sistema que, siguiendo la metodología expuesta en el capítulo 5, se ha desarrollado para el proceso de extracción de conocimiento de una RNA. En él se explicarán las diferentes partes y módulos del sistema y se describirán, una por una, las diferentes fases que tiene el sistema mostrado.

El capítulo 7 es el encargado de mostrar los resultados de las pruebas que se han llevado a cabo una vez desarrollado el sistema. Se expondrán los diferentes resultados ordenados por nivel de complejidad. Así, se comenzará por casos de laboratorio y de reducidos espacios de búsqueda mediante los cuales se podrá comprobar el correcto funcionamiento del sistema en comparación con otros métodos existentes. Posteriormente, se expondrán los resultados de casos reales comparados con los resultados obtenidos por otros métodos.

En el capítulo 8, se indicarán las conclusiones que se han obtenido del desarrollo de la presente tesis, tanto de la metodología expuesta como de la creación del sistema de extracción y de los resultados obtenidos.

En el capítulo 9 se mostrarán las diferentes líneas de investigación futura que se abren a partir de los nuevos horizontes de experimentación abiertos a raíz del desarrollo de la presente tesis y del sistema expuesto.

El capítulo 10 es la sección donde se encuentran las referencias bibliográficas que se han utilizado en las diferentes partes de la documentación.

Por último, está la sección de anexos donde se encuentra el manual de usuario de la aplicación desarrollada. En él se muestra con todo detalle las diferentes opciones incluidas en la aplicación con pantallas autoexplicativas de ejemplos de ejecuciones de extracción de reglas de varias RR.NN.AA.

"Lo que un hombre imagina
otro lo hace realidad"

Julio Verne

CAPÍTULO

2 FUNDAMENTOS

Para poder diseñar y realizar un sistema que manipule las RR.NN.AA es necesario conocer la técnica con un cierto grado de profundidad. Si se afronta el reto de suplir alguno de los defectos o problemas que presentan es preciso estudiar a fondo su estructura, su forma de funcionar, los casos en los que son aplicables, etc. Para ello, se van a exponer los conceptos fundamentales de las neuronas artificiales y su interconexión formando la RNA. El estudio se centrará en explicar aquellas partes relativas al tema que se tratará en esta tesis. Para ello, es necesario conocer las diferentes arquitecturas de RR.NN.AA, los diferentes tipos de neuronas o elementos de proceso existentes, el proceso de entrenamiento y la puesta en funcionamiento de las mismas. Se explicará en detalle la construcción de los ficheros de entrada-salida para el proceso de entrenamiento y ejecución, incluyendo la normalización de los mismos.

Antes de continuar con el proceso de manipulación de RR.NN.AA, para ver la forma de extracción de conocimiento de las mismas, se deben revisar las diferentes formas existentes de afrontar el modelado de procesos que existen. Para ello, existe una

disciplina denominada IS que proporciona metodología y algoritmos de modelado de procesos.

Una vez encontrada la metodología y algunas técnicas de modelado que pueden resultar útiles para el proceso de extracción de reglas de RR.NN.AA, es necesario conocer qué técnicas existentes tratan a los datos y no a las estructuras de los sistemas como núcleo del proceso de extracción de conocimiento. Para ello existe una disciplina denominada "minería de datos" o "Data Mining" que a partir de ingentes cantidades de datos simples intenta abstraer conocimiento en forma de reglas. Así, es necesario adentrarse en esta materia para ver qué técnicas y algoritmos utiliza en dicho proceso.

Como se podrá observar en el capítulo 4, se ha decidido la implementación de la Programación Genética (PG) como algoritmo inductivo del proceso de extracción de reglas de RR.NN.AA utilizando el descubrimiento de conocimiento de los patrones entrada-salida de la RNA. Pero antes de llevar a cabo dicha tarea de implementación es preciso conocer cómo funcionan los métodos evolutivos, los operadores genéticos y cómo se pueden codificar los problemas en forma de individuos y material genético, y cómo actúa el proceso de evolución natural en la búsqueda de soluciones.

2.1 Redes de Neuronas Artificiales

La construcción de máquinas inteligentes constituye un mito desde la antigüedad. Un sistema siempre puede ser simulado en una computadora y, por supuesto, un modelo de RNA también. Estas simulaciones han dado una idea muy precisa de los rendimientos potenciales de las máquinas basadas en RR.NN.AA [GRAU 97][GERS 97].

El primer modelo de neurona artificial fue propuesto por McCulloch y Pitts [MCCU 43] donde modelizaban una estructura y un funcionamiento simplificado de las neuronas del cerebro, considerándolas como dispositivos con *"m"* entradas, una única salida y solo dos estados posibles: activa o inactiva.

Una RNA era, en ese planteamiento inicial, una colección de neuronas de McCulloch y Pitts, todas sincronizadas, donde las salidas de unas neuronas estaban conectadas a las entradas de otras. Algunos de los planteamientos de McCulloch y Pitts se han mantenido desde 1943 sin modificaciones, otros por el contrario han ido evolucionando, pero todas las formalizaciones matemáticas que se han realizado desde entonces, sobre las RR.NN.AA, aún sin pretender ser una modelización exacta de las redes de neuronas biológicas, sí han resultado un punto de partida útil para el estudio de las mismas.

Una de las definiciones que se estima más certera de RNA es la siguiente: "Las redes neuronales son conjuntos de elementos de cálculo simples, usualmente adaptativos, interconectados masivamente en paralelo y con una organización jerárquica que le permite interactuar con algún sistema del mismo modo que lo hace el sistema nervioso biológico" [KOHO 88]. Su aprendizaje adaptativo, auto-organización, tolerancia a fallos, operación en tiempo real y fácil inserción dentro de la tecnología existente, han hecho que su utilización se haya extendido en áreas como la biológica, financiera, industrial, medio ambiental, militar, salud, etc. [HILE 95]. Están funcionando en aplicaciones que incluyen identificación de procesos [GONZ 98], detección de fallos en sistemas de control [ALDR 95], modelación de dinámicas no lineales [MEER 98] [WANG 98], control de sistemas no lineales [BLOC 97][LEVI 93] y optimización de procesos [OLLE 98][ALTI 98][AGUI 98].

En general, se puede encontrar que una RNA se suele caracterizar por tres partes fundamentales: la topología de la red, la regla de aprendizaje y el tipo de entrenamiento.

En este afán de emular el cerebro, esto es simular tanto su estructura como su funcionamiento, se han desarrollado numerosos modelos de RR.NN.AA [FREE 93], entre los que se pueden mencionar: Perceptron (1957), Adeline y Madeline (1960), Avalancha (1967), Retropropagación (1974), Hopfield y SOM (1980), ART (1986), etc. De los modelos anteriores se puede apreciar que esta idea tiene más de 40 años, sin embargo, sólo en las últimas décadas se ha desarrollado la tecnología que permita su aplicación de manera eficiente.

Cabe destacar, para concluir esta breve introducción, que las RR.NN.AA, gracias al masivo paralelismo de su estructura, gozan de una serie de ventajas:

- Tolerancia a fallos.
- Degradación lenta del funcionamiento global del sistema ante sucesivos fallos de neuronas individuales.
- Capacidad de aprendizaje "socrático"; esto es, a partir de ejemplos.
- Memoria distribuida de tipo asociativo.

Estas propiedades hacen atractivas a las RR.NN.AA para determinadas aplicaciones comerciales, militares e industriales [HERN 93]. Sin embargo, estas mismas características hacen que la utilidad de las redes no se pueda buscar en las mismas tareas que los ordenadores convencionales de procesamiento en serie realizan tan satisfactoriamente [FOGE 90].

2.1.1 Fundamentos y Conceptos Básicos de las RR.NN.AA

Warren McCulloch y Walter Pitts pueden ser considerados como los padres de las RR.NN.AA, ya que fueron los primeros en diseñar una neurona artificial. En los años 40 suponían que las neuronas biológicas eran de carácter binario, lo cual resulta ser bastante inexacto, pero sirvió de base para posteriores estudios sobre el sistema nervioso.

Así, McCulloch y Pitts proponen el modelo de neurona que lleva su nombre, el cual es un dispositivo binario con un umbral fijo que hay que superar para que cambie de estado. Recibe sinapsis excitadoras de otros elementos, los cuales tienen la característica de ser del mismo valor. Puede recibir sinapsis inhibidoras que son de acción total, lo cual supone que la recepción de una impide el cambio de estado del elemento, sin importar la cantidad de sinapsis excitadoras que hubiese.

La neurona artificial o elemento formal está conceptualmente inspirada en la neurona biológica. Esto es, los investigadores están en su inmensa mayoría pensando en la organización cerebral cuando consideran configuraciones y algoritmos de RNA.

Se va a considerar una neurona como un elemento formal o módulo o unidad básica de la red que recibe información de otros módulos o del entorno; la integra, la computa y emite una única salida que se va a transmitir idéntica a múltiples neuronas posteriores [WASS 89].

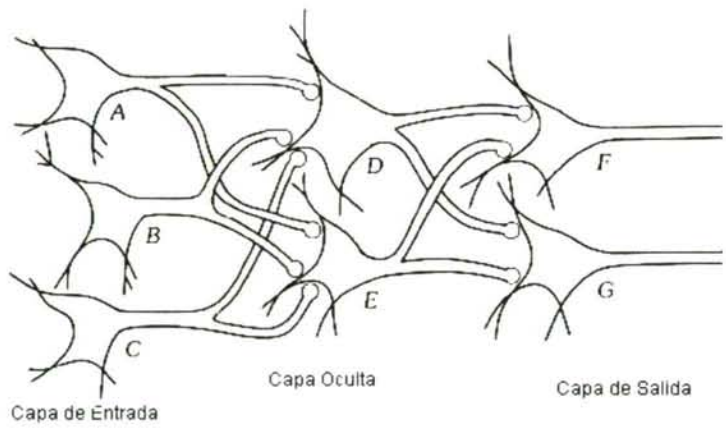


***Fig. 3** - Red de Neuronas Biológicas*

Considérese a los pesos sinápticos como referencia a la noción biológica de la fuerza de unión entre los elementos; es decir, a la fuerza de la sinapsis. Se considera que una sinapsis es fuerte; es decir, tiene un alto grado de conexión, cuando la información que transmite contribuye, en gran medida, a un nuevo estado o a la alteración que se produzca en la neurona receptora y, por tanto, en la respuesta que ésta elabora.

En las RR.NN.AA este peso o fuerza sináptica va a ser un valor numérico que va a ponderar las señales que se reciben por dicha sinapsis. El peso de las conexiones se denotan con una W con dos subíndices:

$\mathbf{W_{ij}}$ = *Peso de la conexión entre la neurona j (que emite) y la neurona i (que recibe).*

La combinación de las señales que recibe una neurona se puede computar como sigue:

$$NET_i(t) = \sum_{j=0}^{N-1} \left[W_{ij} * O_j * (t-1) \right]$$

La función de activación y la función de transferencia son las encargadas de definir el nuevo estado de activación A_i y la respuesta O_i de la neurona. Tanto el estado de activación como la salida de la neurona van a estar en función de las entradas que recibe en un determinado momento y del estado de activación previo que tenga esa neurona.

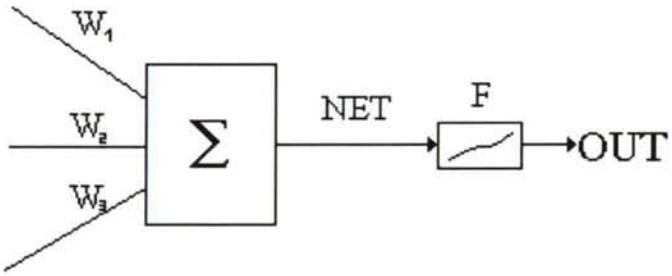


***Fig. 4** - Neurona Artificial*

2.1.1.1 Función de activación de la neurona

Va a relacionar la información de entrada de la neurona con el siguiente estado de activación que tenga esa neurona. Existen dos modelos de función de activación:

- Modelos acotados: El valor de la activación de la neurona puede ser cualquiera dentro de un rango continuo de valores.
- Modelos No acotados: No existe ningún límite para los valores de activación.

Cuando se diseña una red debe establecerse como van a ser los valores de activación de cada neurona y se debe decidir la función de activación (FA) con la que cada neurona procesará las entradas. Por lo tanto, la FA va a actuar sobre las señales de

entrada, sobre los pesos sinápticos asociados con cada entrada y sobre el valor de activación que tenía la neurona en el momento de recibir las señales.

$$A_i(t) = FA\left(A_i(t\text{-}1), NET_i(t\text{-}1)\right)$$

FA = Función de activación.

2.1.1.2 Tipos de neuronas

La linealidad de las funciones que definen a los elementos de la red es quizás lo que va a proporcionar la característica más definitoria. Así, se pueden clasificar las neuronas en lineales y no lineales.

2.1.1.2.1 Neuronas lineales

Una neurona es lineal cuando su salida es linealmente dependiente de sus entradas, es proporcional a las funciones de transferencia y de activación.

Esto conlleva a ciertos problemas como la falta de persistencia en las respuestas, de modo que cambios muy pequeños en las entradas pueden producir fluctuaciones bastante grandes en las respuestas, o la falta de adecuación simultánea, pues es imposible que con neuronas lineales la respuesta de una neurona se adapte tanto a señales grandes como a pequeñas.

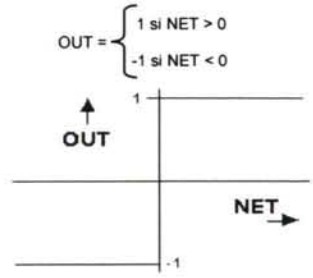
2.1.1.2.2 Neuronas no lineales

En estas neuronas o bien la función de activación o bien la función de transferencia (o ambas) es una función no lineal, dando lugar a que la respuesta de la neurona no sea función lineal de sus entradas.

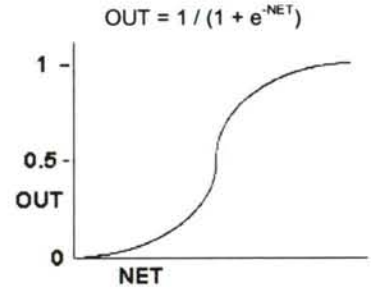
Este tipo de neuronas van a producir respuestas acotadas, desapareciendo los problemas de fluctuación y la falta de adecuación a señales pequeñas y grandes.

Como ejemplo de funciones no lineales se pueden destacar las siguientes:

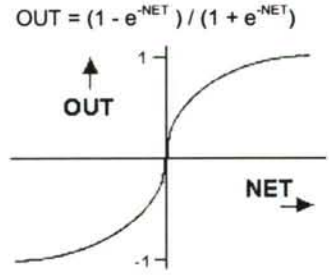
- Umbral



- Sigmoide



- Hiperbólica tangente



2.1.2 Flujo de datos en las RR.NN.AA

2.1.2.1 Redes alimentadas hacia delante.

Son aquellas en las que, como su nombre indica, la información se mueve en un único sentido, desde la entrada hacia la salida. Estas redes están clásicamente organizadas en "capas". Cada capa agrupa a un conjunto de neuronas que reciben sinapsis de las neuronas de la capa anterior y emiten salidas hacia las neuronas de la capa siguiente. Entre las neuronas de una misma capa no hay sinapsis.

En este tipo de redes existe al menos una capa de entrada, formada por las neuronas que reciben las señales de entrada a la red y una capa de salida, formada por

una o más neuronas que emiten la respuesta de la red al exterior. Entre la capa de entrada y la de salida existen una o más capas intermedias.

En redes así construidas es evidente que la información sólo puede moverse en un sentido: desde la capa de entrada hasta la capa de salida, atravesando todas y cada una de las capas intermedias una sola vez.

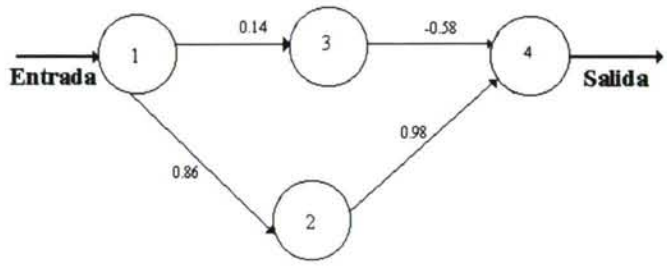


***Fig. 5** - RNA alimentada hacia delante*

El procesamiento de la información en estas redes se realiza en un tiempo predeterminado e igual para todas las posibles configuraciones de señales de entrada.

El hecho de que no haya conexión entre las neuronas de una misma capa hace que no haya tiempos de espera en los que las neuronas estén interactuando unas sobre otras hasta que toda la capa adquiera un estado estable. Se trata por tanto de redes rápidas en sus cálculos.

2.1.2.2 Redes con Retroalimentación total o parcial.

En este tipo de redes los elementos pueden enviar estímulos a neuronas de capas anteriores, de su propia capa o a ellos mismos, por lo que desaparece el concepto de agrupamiento de las neuronas en capas. Cada neurona puede estar conectada a todas las demás; de este modo, cuando se recibe información de entrada a la red, cada neurona tendrá que calcular y recalcular su estado varias veces, hasta que todas las neuronas de la red alcancen un estado estable. Un estado estable es aquel en el que no ocurren cambios en la salida de ninguna neurona. No habiendo cambios en las salidas, las entradas de todas las neuronas serán también constantes, por lo que no tendrán que

modificar su estado de activación ni su respuesta, manteniéndose así un estado global estable [PEAR 90].

En este tipo de redes no hay forma de saber cuánto se tardará en alcanzar un estado estable. Además, unos estímulos de entrada provocarán que se alcance el estado estable en menos tiempo que otros.

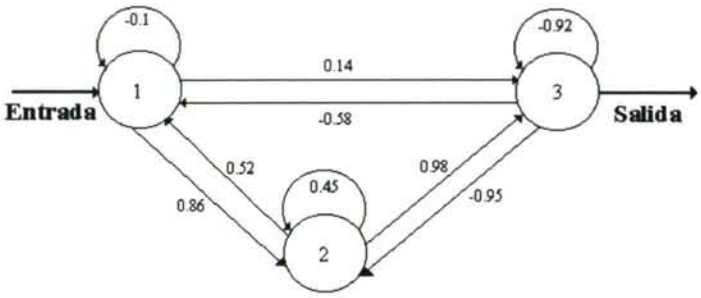


***Fig. 6** - RNA con retroalimentación*

Las redes retroalimentadas emulan más fielmente la estructura del cerebro humano, en donde los fenómenos de retroalimentación son fundamentales.

2.1.3 Manejo de la información en las RR.NN.AA

En este apartado, se presentan los tipos de información que maneja una RNA, así como los métodos para alimentarla con dicha información; en otras palabras, se verá cómo se le presenta a la red el problema que debe resolver. También se verán los problemas típicos con los que las redes se enfrentan.

En el cerebro todas las neuronas actúan del mismo modo. Una neurona concreta no puede distinguir características como "rojo" o "frío". Lo único que una neurona biológica distingue es el potencial de membrana, su estado de activación y sus secuencias de disparo. Estos tres últimos elementos pueden ser expresados numéricamente.

La información de entrada en números tiene que ser adecuada para las funciones de activación y transferencia de la RNA y se deben distribuir esas informaciones O_j entre las neuronas de la capa de entrada de forma adecuada.

Se observa, por tanto, que a la hora de diseñar una red es necesario realizar un análisis tanto de la información de entrada que hay que suministrarle para presentarle el problema, como de la información de salida que la RNA proporcionará como solución a dicho problema.

En el análisis de la información es necesario centrarse en las siguientes características:

2.1.3.1.1 Información global o divisible

Según el tipo de información que van a recibir, se clasifican las redes en **no distribuidas y distribuidas.**

Las redes no distribuidas son aquellas en donde la información de entrada se compone de elementos ortogonales o perpendiculares, independientes entre sí. Un ejemplo típico de este tipo de red lo constituyen las redes **reconocedoras de caracteres.** En este tipo de redes lo que se desea es que la red reconozca caracteres gráficos. La información de entrada a la red será un carácter representado sobre una matriz de forma que cada elemento de la matriz tome dos valores: cero o uno.

Por el contrario, las redes distribuidas, son aquellas en donde la información de entrada es global y no divisible en elementos ortogonales separables como en el caso anterior. En las redes distribuidas todas las neuronas de entrada reciben una parte de la información global. Es decir, la información global "se distribuye" entre ellas, de este modo cada neurona recibe un porcentaje de la información total y no el 100% de la información relativa a un elemento informativo como en el caso anterior.

2.1.3.1.2 Dominio de los datos

Si los datos de entrada, es decir, la información del problema que se desea que la RNA resuelva, no se pueden representar mediante valores dicotómicos, se tendrán que modificar las funciones de la red para hacerla así capaz de trabajar con valores O_j con los que se puedan representar los datos.

Si, por ejemplo, se trabaja con información incierta, en donde sólo se puede dar un valor de la probabilidad de que un determinado elemento pertenezca a un conjunto, se necesitará que la función sea continua para poder expresar valores de probabilidad y el rango de los O_j irá de 0 a 1.

Si los valores de O_j son continuos y los datos también, se tendrá que volver a normalizar los datos de entrada para hacerlos corresponder con el rango de valores que los O_j pueden tomar. Es decir, es necesario reajustar la escala de los datos para que se adapte a la escala de los O_j. Así, si por ejemplo un dato de entrada es una cantidad en euros (desde 0 a 1.000 €) y los O_j se mueven desde -1 a 1 se tendrá que establecer una correspondencia en la que a 0 € le corresponda -1, a 1.000 € le corresponda +1, a 500 € le corresponda 0, etc.

2.1.3.2 Problemas de clasificación y de evaluación.

Una vez decidido como representar la información, el siguiente paso es considerar qué tipo de problema va a resolver la red. La mayoría de los problemas se dividen en dos grandes grupos y, consecuentemente, existen dos grandes tipos de redes de neuronas. Las redes del primer tipo, asocian una determinada configuración de entrada, o **patrón de entrada,** con una salida. Se denomina a estas redes, **redes asociadoras, clasificadoras o reconocedoras de patrones.** El segundo tipo de redes, maneja problemas en los que lo que se pide es un "juicio", por lo que se denominan **redes evaluadoras**.

Otro grupo de RR.NN.AA lo constituyen modelos específicos para tareas especiales como: restauración de patrones, predicción, etc.

2.1.4 Entrenamiento de las RR.NN.AA

Norbert Wiener en su libro "Dios y el Golem" [WIEN 64] da tal vez la definición más neutral de aprendizaje de todas las conocidas hasta ahora. Allí definió un sistema que aprende como: "Un sistema organizado puede definirse como aquel que transforma un cierto mensaje de entrada en uno de salida, de acuerdo con algún principio de

transformación. Si tal principio está sujeto a cierto criterio de validez de funcionamiento, y si el método de transformación se ajusta a fin de que tienda a mejorar el funcionamiento del sistema de acuerdo con ese criterio, se dice que el sistema aprende".

Esta definición es válida para definir el aprendizaje bajo los dos puntos de vista en que éste puede entenderse: el ontogenético, que es el aprendizaje de un individuo humano o no, y el filogenético, aprendizaje que afecta a la especie.

Uno de los principales objetivos de los sistemas autónomos es emular la habilidad que posee el hombre para interactuar con el ambiente y aprender de dichas interacciones. Es así como necesitan de una estructura flexible, capaz de desempeñarse en ambientes de operación dinámicos sujetos a diversas incertezas y perturbaciones. Dentro de las aproximaciones existentes en la teoría de IA, surgen las RR.NN.AA como elementos capaces de proveer de dicha estructura flexible, mediante la integración con diversos sistemas de aprendizaje [ASCE 98]. Tales sistemas están orientados hacia diferentes operaciones y pueden ser clasificados en dos tipos [BROW 94][LIN 96]: aprendizaje supervisado y no supervisado, y dentro de este, aprendizaje auto-organizativo y aprendizaje por reforzamiento [HOSK 92].

Una vez diseñada la arquitectura de la red (capas y número de neuronas por capa) y las funciones que la regirán, se tiene que proceder a entrenar a la red para que "aprenda" el comportamiento que debe tener; es decir, para que aprenda a dar la respuesta adecuada a la configuración de estímulos o patrón de entrada que se le presente [PAZO 96].

Una excepción a esta regla general la constituyen las redes de Hopfield, que no son entrenadas sino construidas, de modo que tengan ya inicialmente el comportamiento deseado. Por este motivo, se ha dicho que las redes de Hopfield simulan el comportamiento "instintivo" mientras que las demás redes simulan el comportamiento "aprendido".

En la vida de las redes con comportamiento aprendido se distinguen dos periodos o fases claramente diferenciados. Durante **la fase de aprendizaje** se entrena a la red para que vaya modificando sus pesos sinápticos, adaptándolos paulatinamente para que la respuesta de la red sea la correcta. Después, viene **la fase de funcionamiento real o fase de ejecución,** durante la cual la red ya es operativa y sus pesos sinápticos no volverán a ser modificados. Durante esta fase se usa la red como si se tratara de cualquier otro programa informático convencional.

2.1.4.1 Aprendizaje supervisado

Con esta técnica de aprendizaje el entrenamiento consiste en presentarle a la red repetitivamente patrones de estímulos de entrada pertenecientes a un **juego de ensayo.**

El juego de ensayo está formado por parejas "patrón de estímulos - respuesta correcta" y debe de ser elegido cuidadosamente. Cada pareja se denomina **hecho.** En el juego de ensayo debe estar representada equilibradamente toda la información que la red necesite aprender.

Al realizar el entrenamiento la respuesta que da la red a cada patrón se compara con la respuesta correcta ante dicho patrón y, en virtud de esa comparación, se reajustan los pesos sinápticos. El reajuste de los pesos sinápticos está orientado a que, ante el patrón de entrada, la red se acerque cada vez más a la respuesta correcta.

Cuando ante un patrón de entrada la red ya responde bien, se pasa al siguiente patrón del juego de ensayo y se procede de la misma manera.

Cuando se termina con el último patrón del juego de ensayo, se tiene que volver a empezar con el primero, ya que los pesos se han seguido modificando.

En casos sencillos, al cabo de unos pocos pasos de entrenamiento completos, con todos los elementos del juego de ensayo, los pesos sinápticos de todas las neuronas se estabilizan en torno a unos valores óptimos. Se dice entonces que el algoritmo de aprendizaje **converge.** Es decir, después de sucesivas presentaciones de todos los

patrones estimulares del juego de ensayo, la red, responderá correctamente a todos ellos y se puede considerar entrenada y dar por terminada la fase de aprendizaje.

2.1.4.2 Aprendizaje no supervisado

En este tipo de aprendizaje, no se le especifica a la red cuál debe ser la respuesta correcta; es decir, no hay una comparación entre la respuesta de la red y la respuesta deseada. Además, en este modelo de aprendizaje no existe ninguna influencia externa a la red, puesto que no se le informa de si un resultado fue correcto o no; tan sólo se le suministran grandes cantidades de datos con los que la red pueda construir sus propias asociaciones. Se necesita, por tanto, una cantidad mucho mayor de patrones de entrada durante el entrenamiento para que la red pueda ajustar correctamente sus pesos sinápticos. Por supuesto, los procedimientos de aprendizaje son diferentes a los utilizados con el modelo de entrenamiento supervisado.

En este tipo de aprendizaje, lo que de hecho se está haciendo es exigirle a la red que capte por sí misma alguna de las características de los datos de entrada.

Evidentemente, muchos de los aprendizajes básicos que realizan los sistemas biológicos son de este tipo. Los recién nacidos (al igual que los ciegos de nacimiento que recuperan la visión en edad adulta) aprenden a organizar los datos visuales sin ayuda de "profesor" que les indique para cada patrón de estímulos de entrada, cual es la organización - interpretación correcta de dichos estímulos; es decir, la respuesta del subsistema neurológico de visión que servirá a su vez de entrada a otros subsistemas.

De hecho, en el aprendizaje no supervisado se pretende que las neuronas se auto-organicen aprendiendo a captar las "regularidades" de los datos de entrada sin suministrarles ningún tipo de criterio o ayuda externa que dirija dicha autoorganización.

2.1.4.3 Ejecución

Tras la fase de entrenamiento viene la fase de ejecución, durante la cual se le pedirá a la red que responda a estímulos diferentes a los presentados durante la fase de

entrenamiento. Gracias a los ejemplos aprendidos del juego de ensayo, la red deberá ser capaz de **generalizar** y dar respuestas correctas ante patrones de estímulos nuevos.

En otras palabras, una vez terminado el aprendizaje, una red puede generalizar; es decir, ante entradas similares a las de su juego de ensayo, producirá salidas correctas. Hay que tener en cuenta que es muy difícil conseguir la capacidad de generalización de una red sin utilizar grandes cantidades de datos y que estos sean muy variados.

Para operar con una red entrenada, el proceso es el mismo que cuando se realizaba el entrenamiento: se le sigue suministrando información de entrada a la red, sólo que ahora no se realizará ningún ajuste en los pesos sinápticos. La red reconocerá o evaluará y dará una respuesta. Pero ahora no se conoce la respuesta correcta y no se puede por tanto comparar con nada la respuesta de la red.

Para conseguir el mejor rendimiento de generalización, los datos usados para el entrenamiento deben cubrir un rango de hechos suficientemente amplio. En general, cuando aumenta el tamaño y variedad del juego de ensayo disminuye la necesidad de que los datos de entrada durante la fase de trabajo normal se parezcan mucho a los patrones del juego de ensayo; es decir, la red generalizará mejor. Pero si los datos de un problema se diferencian demasiado de todos los patrones del juego de ensayo, la red tendrá dificultades para encontrar la respuesta correcta.

2.1.4.4 Selección de los elementos del juego de ensayo

En vista de la gran cantidad de información que se le puede proporcionar a una red, se ha de buscar un criterio de selección para crear el juego de ensayo.

En el juego de ensayo debe de haber suficientes hechos: parejas "patrones de estímulos - respuesta correcta". Además, los hechos del juego de ensayo deberán cubrir ampliamente la totalidad de las características a los que la red debe de enfrentarse. El problema de decidir cuantos hechos se han de incluir en el juego de ensayo es un problema de "Teoría de Muestras" al que se está buscando solución. Un problema muy similar se plantea al establecer, en base a casos de prueba, el grado de experiencia de un experto.

No se debe desechar alegremente información por pensar que es irrelevante. A menudo, las redes de neuronas pueden encontrar asociaciones que nunca se hubiese supuesto que existiesen. Se debe, por el contrario, entrenar a la red con cualquier pieza de información disponible y, simultáneamente, entrenar otra red con los hechos mínimos que se consideran imprescindibles, para luego cotejar cual de las dos redes se comporta mejor. Puede resultar una curiosa experiencia el descubrir cuales son los hechos realmente importantes.

Por otra parte, a una red evaluadora es importante mostrarle tanto los patrones de entrada que llevan a evaluaciones positivas, como los patrones de entrada que llevan a evaluaciones negativas. Es decir, el entrenamiento de la red debe incluir situaciones que se evalúen negativamente, pues de lo contrario la red simplemente aprenderá que todo está correcto siempre. También se debe incluir en el juego de ensayo cada uno de los casos en los cuales el valor de una entrada es causa de un "mal" resultado. Por ejemplo, una red que evalúe la capacidad de un avión para volar debe saber que un indicador de combustible que marque vacío es definitivamente malo y que el avión no estará en disposición de volar.

Por otra parte, no se puede incluir en el juego de ensayo una colección exageradamente grande de hechos. Es necesario seleccionar aquellos hechos que reflejen claramente cada uno de los patrones a reconocer y las situaciones extremas de evaluación en una red evaluadora.

Lo ideal es preparar una colección amplia de hechos de entrenamiento que cubran todos los problemas a los que se pueda tener que enfrentar la red. Después de esa amplia colección de hechos, se seleccionarán algunos de ellos para el juego de ensayo, teniendo cuidado de que todos los problemas queden bien representados.

2.1.5 Redes Recurrentes

Cuando se trabaja con patrones dinámicos; es decir, con patrones de secuencias, los tipos de RR.NN.AA de más aplicación como el Perceptron multicapa, se encuentran bastante limitados ya que no permiten conexiones que unan neuronas creando bucles.

Las Redes Recurrentes son redes multicapa en las que no se impone ninguna restricción en su conectividad, con lo que se gana un número mayor de pesos por neurona y dado que las redes de neuronas representan la información de forma distribuida en sus pesos, una mayor representatividad.

Así, la principal característica de este tipo de redes es la de realimentar su salida a su entrada, evolucionando hasta un estado de equilibrio donde proporciona la salida final de la red [DEMU 94]. Esta característica las hace útiles cuando se quiere simular sistemas dinámicos; sin embargo, su entrenamiento es más lento que el de una red alimentada sólo hacia delante.

El primer modelo surge con las redes de Hopfield, mejorándose después con las máquinas de Boltzman que admiten neuronas ocultas. Estos modelos, aunque en cierto modo son los responsables del auge que está teniendo el campo de las RR.NN.AA, son poco interesantes desde el punto de vista informático ya que se comportan únicamente como memorias asociativas. En 1987 se adapta el algoritmo de retropropagación a las redes recurrentes aplicadas a patrones estáticos ("Recurrent Backpropagation") y se pudieron aplicar estas redes a los mismos problemas a los que se aplicaban las multicapa alimentadas hacia delante.

Además, otros investigadores se centran en desarrollar aproximaciones del algoritmo de aprendizaje que lo hagan más práctico surgiendo el algoritmo llamado "Real-Time Recurrent Learning" o RTRL indicado para tareas de tiempo real [MCBR 65][WILL 89].

A partir de entonces, las redes recurrentes se han venido aplicando en un buen número de tareas, desde reconocimiento del habla hasta la simulación de autómatas finitos. Sin embargo, la aplicación de redes recurrentes presenta un mayor número de problemas. En el caso de patrones estáticos, una red recurrente funciona presentándole un patrón, haciendo después evolucionar la red hasta que sus salidas se estabilizan. Sin embargo, esto no está asegurado, pudiéndose dar comportamientos oscilatorios o

caóticos y aunque existen estudios para establecer las condiciones para que esto no ocurra, se limitan a ciertas arquitecturas muy concretas como las Hopfield.

El caso de los patrones dinámicos es todavía más complicado, ya que si se sabe poco del comportamiento de una red recurrente (por ejemplo la dificultad de estabilizarse), se sabe aún menos de su comportamiento dinámico. El poco conocimiento es empírico y no existen estudios formales ni de la red recurrente más simple: una neurona aislada con una conexión a sí misma. Tampoco existen estudios teóricos que avalen utilizar un algoritmo basado en el descenso del gradiente para tareas de tratamiento de patrones dinámicos. Un problema sencillo, como es enseñar a oscilar a una neurona aislada con realimentación, da muchos problemas del tipo de mínimos locales y hasta ahora no se conoce su justificación teórica.

Además, en redes multicapa se conoce más o menos bien qué arquitectura hay que utilizar en la mayoría de los problemas, gracias a conocimientos basados fundamentalmente en la experiencia. Sin embargo, por una parte, la variedad arquitectónica en redes recurrentes es infinitamente superior, por lo que su diseño es más complicado y, por otra, la gran variedad de este tipo de patrones hace difícil su categorización.

2.1.5.1 Arquitectura y funcionamiento de las RR.NN.AA recurrentes

Las redes de neuronas recurrentes se pueden dividir en tres grupos según su función y su estructura:

- Redes recurrentes aplicadas a patrones estáticos, cuyo comportamiento dinámico consiste en converger a un punto estable y su tarea es similar a la de una red multicapa no recurrente. La manera de trabajar con estas redes consta de tres fases:
  - Introducir el patrón de entrada.
  - Hacer evolucionar la red hasta que se estabilice dejando la entrada fija.

- Obtener el patrón de salida de las neuronas de salida, que deben tener una activación estable y constante.

Estas redes se utilizan en tareas que impliquen la satisfacción de restricciones o la construcción de memorias asociativas.

- Redes parcialmente recurrentes, se aplican a reconocimiento y, a veces, a producción de secuencias. En estas redes la mayoría de las conexiones son no recurrentes, pero existe un pequeño grupo de recurrencias cuidadosamente escogidas, que permiten a la red recordar su estado en un pasado reciente sin complicar demasiado la labor del entrenamiento. En la mayoría de los casos, las recurrencias son fijas, por lo que el algoritmo de aprendizaje no debe modificarlas y se puede utilizar un algoritmo de entrenamiento del tipo "backpropagation". En estas redes existirán una serie de neuronas especiales llamadas "neuronas de contexto" que son las receptoras de las recurrencias. Estas neuronas son simplemente una copia de la capa de la cual recibe la conexión en la iteración anterior. La conexión de la capa de contexto y la capa de neuronas es una conexión neurona a neurona con un peso fijo de 1.

- Redes recurrentes, donde no existe restricción en su conectividad y se pueden aplicar a reconocimiento, producción o asociación de secuencias. Estas redes se pueden dividir en dos, discretas y continuas, según se comporten sus neuronas y son estas últimas las que mejor recogen el comportamiento dinámico que se exige para las tareas temporales.

2.2 Identificación de Sistemas

La Identificación de Sistemas (IS) determina una relación funcional entre las variables de entrada y de salida de un proceso [NARE 90]. Los usos de esta relación funcional son la simulación y el control del proceso identificado [HUNT 92]. Tradicionalmente se emplean técnicas de regresión para desarrollar la identificación de un proceso, las cuales requieren de una estructura previa de la naturaleza de la relación entre las entradas y las salidas. Sin embargo, una RNA es capaz de identificar un

proceso sin conocimiento previo de él, para lo cual requiere de abundante información de entradas-salidas del proceso en el rango de interés que se desea identificar. Hay que tener claro que la red no identifica la naturaleza de relación entrada-salida, si no que desarrolla un mapa entrada-salida, basado en las muestras con que fue entrenada [POLL 92].

El término *sistema* es utilizado en una gran diversidad de maneras, es difícil dar una definición que abarque todos los usos que se le da a este término y que a la vez sea suficientemente concisa para resultar útil. La siguiente pretende ser una definición de sistema que reúne estos requisitos: "un sistema es un conjunto de objetos que interactúan entre sí o que son interdependientes entre sí" [LJUN 87].

El concepto de sistema permite plantear la comprensión de la realidad en dos grandes etapas [LJUN 87]:

1. Identificando sistemas físicos.
2. Estableciendo las reglas o leyes que los describen.

Esta descripción de los sistemas no es necesariamente completa ni precisa, sino adecuada. A un sistema que se ha identificado para interpretar parte de la realidad se le llama modelo. De hecho en ocasiones los términos modelo y sistema se usan indistintamente.

Un modelo nunca es una representación completa de la realidad y de ahí vienen sus ventajas y sus desventajas:

- ✓ Es ventajoso porque implica sencillez.
- ✓ Es desventajoso porque implica inexactitud.

Sencillez y exactitud son dos características deseables en un modelo; sin embargo, conforme se gana en una se pierde en la otra y esto ha dado lugar a toda una gama de modelos diferentes para representar un mismo sistema.

En este contexto, los modelos lineales son para muchos los modelos que sin ser demasiado complejos proporcionan una buena dosis de exactitud en la representación de

una gran variedad de sistemas físicos. Una de las consideraciones impuestas tradicionalmente a los modelos en ingeniería ha sido la *linealidad* (como pueden ser la determinística y la invariabilidad en el tiempo); sin embargo, no es la única que permite un tratamiento sencillo sin perder demasiada exactitud en un gran número de casos prácticos.

Tradicionalmente se han desarrollado numerosos modelos basados en la teoría matemática debido a que son especialmente útiles debido a su formalidad, su gran flexibilidad y su conversión directa a modelos numéricos que pueden ser simulados en una computadora. Sin embargo, al igual que otros modelos, heredan el conflicto entre sencillez y exactitud, buscándose diferentes puntos medios entre estos extremos.

***Fig.** 7 - Grados de complejidad y exactitud de los modelos*

2.2.1 Motivación para el diseño de modelos

Una motivación fundamental para plantear modelos es resolver algún problema, o simplemente dar una explicación adecuada a una situación real o ficticia.

El problema fundamental que plantea un sistema es la descripción concisa de su comportamiento entrada-salida y su comportamiento interno. De aquí se desprenden diversos problemas que dependen de lo que se desea hacer con el sistema:

- ✓ El problema de **análisis** consiste en determinar el comportamiento interno y entrada-salida de un sistema.
- ✓ El problema de **realización** o diseño consiste en construir un sistema que reproduzca el comportamiento de otro sistema dado.

- ✓ El problema de **control** consiste en elegir adecuadamente la entrada para obtener una salida deseada.
- ✓ El problema de **observación** (estimación de estados) consiste en la determinación del estado interno del sistema a partir de la medición de entradas y salidas.
- ✓ El problema de **identificación** (estimación de parámetros) consiste en determinar valores de los parámetros del sistema a partir de mediciones de entrada y salida.

2.2.2 Definición de sistemas

En un sistema las señales son funciones de una o más variables independientes y contienen información acerca de la naturaleza o comportamiento de algún fenómeno. Los sistemas reciben señales como entrada y responden a ellas produciendo otras señales a la salida. Esta relación entre señales y sistemas se puede ver representada en la figura 8.

***Fig. 8** - Diagrama de bloques de un sistema general*

De esta forma un sistema es un proceso que produce una transformación de señales. Todo sistema debe tener al menos una entrada **x** y una salida **y**, la señal de salida está relacionada con la entrada mediante una relación de transformación **y = f(x)**. Los sistemas pueden ser de tiempo continuo o de tiempo discreto en función de si la transformación de las señales de entrada en señales de salida es de tiempo continuo o de tiempo discreto.

2.2.2.1 Propiedades de los sistemas

A continuación, se describen algunas de las propiedades más importantes de los sistemas teniendo en cuenta el tipo de transformación que realiza el sistema sobre las señales de entrada:

1. **Sistemas con y sin memoria**: Un sistema se dice sin memoria si su salida en un instante dado depende solamente de su entrada en ese instante. Estos sistemas también son denominados sistemas estáticos. Un sistema cuyas salidas puedan depender de entradas en instantes anteriores al actual es denominado sistema con memoria o sistema dinámico.

2. **Causalidad**: Un sistema es causal si su salida en cualquier instante depende sólo de los valores de la entrada en el instante actual o en instantes anteriores. Estos sistemas también son denominados sistemas no anticipativos, ya que la salida del sistema no anticipa valores futuros de la entrada.

   Una consecuencia fundamental de que un sistema sea causal es el hecho de que si se prueban dos entradas a un sistema causal idénticas desde las condiciones iniciales hasta un tiempo $\mathbf{t_n}$ las salidas correspondientes también serán iguales hasta ese mismo instante.

3. **Estabilidad**: Un sistema estable es aquel en el que entradas pequeñas producen salidas que no divergen, es decir, salidas acotadas.

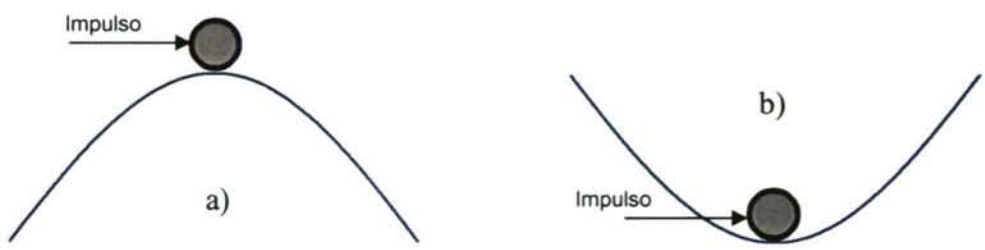


***Fig. 9** - a) Sistema Inestable. b) Sistema Estable.*

4. **Estabilidad de entrada acotada - salida acotada**: También llamado BIBO (Bounded Input-Bounded Output). Define un sistema estable donde la salida está acotada siempre y cuando su entrada esté acotada. La manera de definir el acotamiento de una señal depende del tipo de análisis que se quiera realizar. De esta manera, se puede usar el concepto de norma para introducir diferentes tipos de cotas.

5. **Invariancia en el tiempo**: Un sistema se dice invariante en el tiempo si un retardo en la señal de entrada produce una señal de salida retardada en la misma cantidad de tiempo; es decir, si y(k) es la salida correspondiente a la entrada x(k) en un sistema invariante en el tiempo, la entrada x(k-d) producirá la salida y(k-d). Cuando un sistema invariante en el tiempo se describe por ecuaciones diferenciales, o de diferencias, se obtienen coeficientes constantes. En dichos coeficientes normalmente aparecen los parámetros físicos del sistema, tales como resistencias, capacitancias, masas, coeficientes caloríficos, etc.

6. **Linealidad**: Un sistema lineal es aquel que posee la propiedad de superposición. Dicha propiedad se refiere a que si una entrada es la combinación lineal (suma ponderada) de varias señales, entonces la salida correspondiente es la combinación lineal de las salidas correspondientes a cada una de dichas entradas, cuando estas entradas se aplican individualmente. El sistema definido como $y(k) = a*x(k) + b$, donde a, b son constantes no es considerado un sistema lineal; sin embargo, se dice que es un sistema incrementalmente lineal o afín, ya que los incrementos de la salida responden de manera lineal a incrementos de la entrada.

La identificación de sistemas lineales ha sido ampliamente estudiada [LJUN 87] [SÖDE 89]. En estos modelos el proceso de identificación se centra en determinar los valores asociados a los términos involucrados en una función (matemática) dinámica lineal. Pero la mayoría de los sistemas del mundo real son modelos no lineales y modelos lineales a los cuales no se les puede identificar su comportamiento dinámico.

Desde los años 60 los investigadores han hecho considerables esfuerzos en desarrollar métodos para la identificación de sistemas dinámicos no lineales. Muchos de estos métodos presuponen una estructura de modelo dada a priori y entonces la identificación se reduce a un problema de estimación de parámetros. Haber y Unbehauen [HABE 90] realizan en 1990 una comparativa de métodos de identificación de sistemas dinámicos no lineales.

2.2.2.2 Metodología de la identificación

En un primer momento Norton [NORT 86] y posteriormente Söderström y Stoica [SÖDE 89] definen una serie de etapas que son consideradas dentro del área como la metodología de aplicación genérica en todo proceso de IS. Esta metodología consiste en una serie de etapas bien definidas mediante las cuales se realiza la tarea de modelado y descripción matemática de un sistema dinámico desconocido. Las etapas que han definido son las siguientes:

1. **Conocimiento previo y diseño de experimentos**: buscar información y diseñar los experimentos oportunos sobre el sistema que se desea modelar. Selección del tipo de entradas lo suficientemente "ricas" como para excitar todos los modos del sistema. Sólo pueden identificarse los modos que son observables en la salida.
2. **Toma de datos oportunos experimentales**: realizar las tomas de datos oportunas para construir un juego de ensayos representativo del sistema.
3. **Análisis y tratamiento de datos**: proceso y síntesis de los datos obtenidos en el ensayo. Destacando aquellos más relevantes e identificando el tipo de modelo a emplear. Se pueden distinguir tres tipos:
    a. Estructuras **"Caja Negra"** (black box): los parámetros no tienen una interpretación física.
    b. Estructuras a partir de modelado físico **"Caja Blanca"** (white box): los parámetros tienen una interpretación física.

   c. Estructuras a partir de modelado semi-físico **"Caja Gris"** (gray box): introducción de conocimiento a "priori" del sistema en estructuras del tipo "caja negra".

4. **Selección del tipo de modelo**: en función de los datos obtenidos del procesado del juego de ensayo, seleccionar el modelo que más se ajuste a las características obtenidas.

5. **Estimación de parámetros**: obtener los parámetros numéricos relativos al modelo que ajusten el funcionamiento del sistema. Esto implica:

   a. Elección de un criterio a minimizar.

   b. Determinación del vector de parámetros que minimiza el criterio.

   c. Analítica (sólo posible en algunos casos).

   d. Métodos numéricos iterativos. Lo que supone que pueden ser no lineales y no convexos dando problemas de convergencia y mínimos locales.

6. **Validación del modelo**: proponer y realizar nuevos juegos de ensayo para la comprobación del correcto funcionamiento del sistema modelado. Con esto se obtiene una medida de confiabilidad del modelo:

   a. Decidir si el modelo es lo suficiente bueno para la aplicación en que se usará.

   b. Decidir cuán lejos del "sistema real" está el modelo.

   c. Decidir si el modelo y los datos son consistentes con las hipótesis sobre la estructura del modelo.

   Si los resultados no son los esperados habrá que volver al paso 2.

El diagrama de las diferentes etapas de la metodología se puede observar en la figura 10.

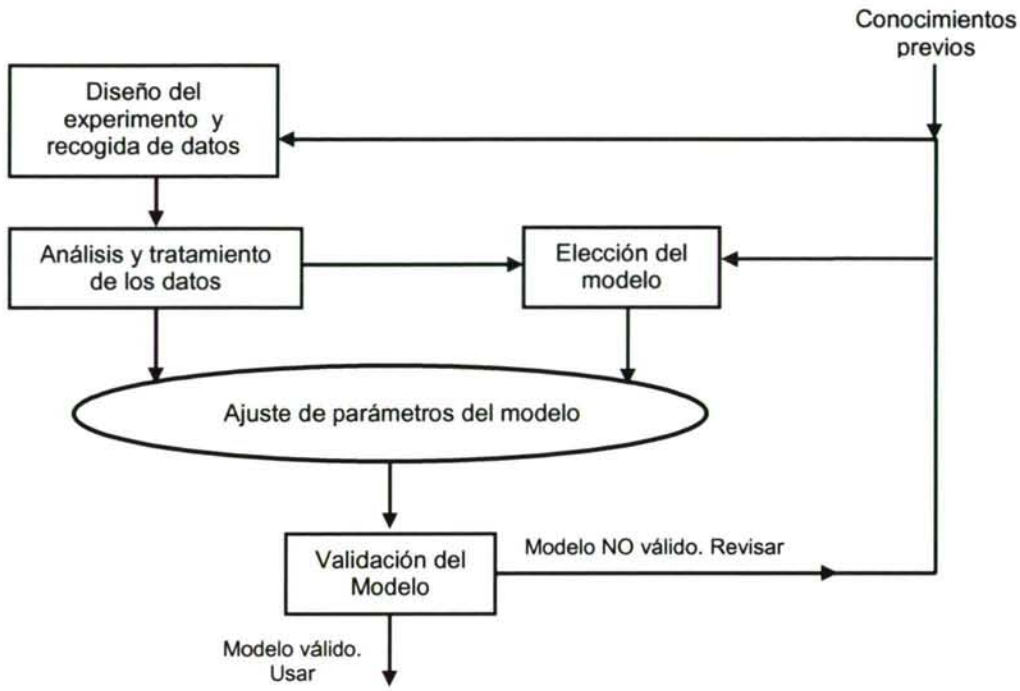


***Fig. 10** - El proceso de Identificación.*

2.2.2.3 Modelos

Existen varios tipos de modelos a aplicar en la teoría de IS. En función de la naturaleza del proceso a modelar se pueden clasificar en modelos lineales y no lineales:

- ✓ **Modelos Lineales**: No hay términos constantes. Los siguientes modelos se encuentran extensamente explicados en multitud de libros de identificación de sistemas lineales [BROC 70][WOLO 74][KAIL 80][CHEN 84][WONH 85][WILS 96].
  - ▪ ***Paramétricos***
    - ➢ Función Transferencia.
    - ➢ Ecuaciones de Estado.
    - ➢ Ecuación Diferencial Ordinaria.
    - ➢ Regresivos: tales como ARX (Autoregressive with Exogeneous Input), ARMAX (Autoregressive

Moving Average with Exogeneous Input), ARMA (Autoregressive Moving Average), BJ (Box-Jenkins), OE (Output-Error), FIR (Finite Impulse Response).

- ***No paramétricos***
  - ➢ Respuesta a impulsos.
  - ➢ Repuesta en frecuencia.

✓ **Modelos NO Lineales**: Los coeficientes dependen de x ó y. Hay términos constantes.

- Hammerstein [ESKI 91].
- Wiener [GREB 94][WIGR 93][WIGR 94][WILL 89].
- Feedback Block-Oriented Model [POTT 98][POTT 00].

En los modelos lineales la salida actual depende linealmente de las entradas y salidas en instantes de tiempo anteriores:

$$y(t) = -a_1 y(t-1) - .... - a_n y(t-n) + b_0 u(t-k) + ... b_m u(t-k-m)$$

Donde las variables "u" e "y" son perturbaciones sobre un punto de operación: $y(t) = Y - Y_0$

$u(t) = U - U_0$

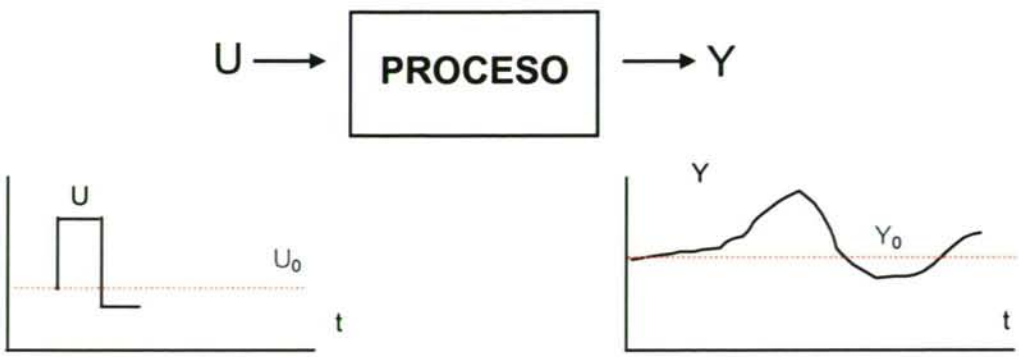


***Fig. 11** - Modelo Lineal*

Los modelos no lineales vistos anteriormente han sido aplicados con buenos resultados en diferentes casos del mundo real, desde procesos químicos [ESKI 91] [POTT 00][KALA 95][CHOU 00], procesos biológicos [KORE 73][HERN 92], procesado de señales [STAP 85] hasta control automático [FRUZ 97].

2.2.2.4 Identificación de Sistemas mediante RNA

La identificación de modelos dinámicos de procesos usa modelos como el error de salida (OE), modelos de regresión como ARMAX o NARMAX, estructuras de respuesta al impulso finito (FIR), etc. Estos modelos están basados en varias suposiciones referentes al verdadero proceso, las más importantes de ellas tienen que ver con la linealidad e invarianza en el tiempo de los parámetros [POLL 92]. Estas suposiciones permiten con un experimento local extrapolar globalmente a otras condiciones de operación. La validez de la extrapolación dependerá del curso en la validez de las suposiciones de base. La experiencia ha mostrado que para un número de sistemas de procesos estas suposiciones son razonables. Sin embargo, si el proceso no es invariante en el tiempo, puede requerirse una re-identificación periódica del modelo del proceso. Además, si el proceso no es lineal respecto a las condiciones de operación, se puede requerir de varios modelos lineales para abarcar todo el rango de operación del proceso [POLL 92].

Las consideraciones que se hacen en la identificación de procesos mediante RR.NN.AA son considerablemente menores que las hechas mediante una identificación clásica. En cualquiera de los dos tipos de identificación es el usuario quien define las entradas-salidas del proceso. En la identificación clásica el usuario especifica una estructura de la naturaleza de la relación entre las entradas y salidas; mientras que en la identificación mediante RNA el usuario sólo especifica la topología de la red. La RNA no trata de asumir o determinar una relación funcional entre las variables de entrada y salida. En su lugar, la red simplemente crea un mapa de la entrada hacia la salida, basada en los ejemplos con los cuales fue entrenada. De esta manera, la primera ventaja que se aprecia al utilizar una RNA para identificar un proceso dinámico es evitar

especificar la estructura del modelo del proceso [SHAW 97]. Los modelos de regresión, error o respuesta a una señal (ARMAX, OER, FIR) requieren de un experimento cuidadosamente controlado sobre el rango de la línea operativa del proceso. Para procesos no lineales, el conjunto de datos debe ser separado en subconjuntos de datos donde el proceso sea lineal. Las RR.NN.AA permiten que el mapa entrada-salida sea detectado sin alguna inferencia de la relación funcional. La desventaja de esta aproximación es la necesidad de especificar un conjunto de entradas que incluya la región de interés, ya que sin un verdadero conocimiento del proceso los resultados de un experimento no pueden ser extrapolados fuera de la región en que el experimento fue desarrollado [NARE 90] [POLL 92] [ZAMA 98].

La identificación mediante RNA puede requerir menos conocimiento del proceso que una identificación clásica; pero puede necesitar de experimentos demasiado largos, quizás ordenes de magnitud más grandes que los experimentos desarrollados para una identificación clásica [POLL 92].

2.3 Extracción de Conocimiento en BD (KDD)

La técnica de Data Mining o Knowledge Discovery in Databases (KDD) es un proceso iterativo en el que se van produciendo progresivos descubrimientos y cuantificaciones de relaciones predictivas en los datos y se permite transformar la información disponible en conocimiento útil. El KDD surgió como una integración de múltiples tecnologías tales como la estadística, el soporte a la toma de decisiones, el aprendizaje automático y la gestión y almacenamiento de bases de datos. Para la realización de éstos procesos se aplican técnicas procedentes de muy diversas áreas, como pueden ser los AA.GG, las RR.NN.AA, los árboles de decisión, etc.

Una de las mejores definiciones del proceso de descubrimiento del conocimiento es: "la extracción no trivial de información implícita, desconocida y potencialmente útil de los datos" [FRAW 91].

Sin embargo, existe una diferencia clara entre el proceso de extracción de datos y el descubrimiento de conocimiento. Fayyad et al. [FAYY 96a] establecen que el

proceso de descubrimiento del conocimiento toma los resultados tal como vienen de los datos (proceso de extraer tendencias o modelos de los datos) cuidadosamente, y con precisión los transforma en información útil y entendible. Esta información no es típicamente recuperable por las técnicas normales, pero es descubierta a través del uso de técnicas de IA.

KDD es un campo en constante crecimiento, hay muchas metodologías del descubrimiento del conocimiento en uso y muchas en desarrollo. Algunas de estas técnicas son genéricas, mientras otras son de dominio específico. En KDD se emplean varias técnicas para la obtención de resultados y análisis de datos. Para ello, el análisis se realiza en prueba de hipótesis, para lo cual es necesario un conocimiento a priori de cuáles van a ser los resultados importantes. Los resultados obtenidos se aproximan a describir tendencias generales de los grupos de datos, diferencias generalizadas y amplias categorizaciones.

2.3.1 Características generales

Aunque hay muchos enfoques al KDD, existen seis elementos comunes y esenciales que califican al descubrimiento del conocimiento como una técnica. Las características básicas que comparten todas las técnicas de KDD son [FRAW 91] [FAYY 96a]:

1. Se requiere de grandes cantidades de datos que proporcionen información suficiente para derivar un conocimiento adicional.
2. Dado que se requieren grandes cantidades de datos, es esencial el proceso de la eficiencia.
3. La exactitud es requerida para asegurar que el descubrimiento del conocimiento es válido.
4. Los resultados deberán ser presentados de una manera entendible para el ser humano.

5. Una de las premisas mayores de KDD es que el conocimiento es descubierto usando técnicas de aprendizaje inteligente que van examinando los datos a través de procesos automatizados.

El KDD proporciona la capacidad para descubrir información nueva y significativa usando los datos existentes. El KDD excede rápidamente la capacidad humana para analizar grandes cantidades de datos. Así, la cantidad de datos que requieren procesamiento y análisis en grandes bases de datos exceden las capacidades humanas y la dificultad de transformar los datos con precisión es un conocimiento que va más allá de los límites de las bases de datos tradicionales. Por consiguiente, la utilización plena de los datos almacenados depende del uso de técnicas del descubrimiento del conocimiento.

2.3.2 Técnicas de KDD

Hay muchos métodos diferentes que son clasificados como técnicas de KDD. Hay métodos cuantitativos, como los probabilísticos y los estadísticos, y métodos de clasificación como el Bayesiano, la lógica inductiva, el descubrimiento, el modelado de datos y el análisis de decisiones. Otros métodos incluyen la desviación y tendencia del análisis, los AA.GG, las RR.NN.AA y los métodos híbridos que combinan dos o más técnicas.

Esta clasificación de técnicas de extracción de conocimiento ha sido proporcionada por Peggy Wright [WRIG 98]. Posteriormente se mostrarán las características más relevantes de cada una de ellas.

2.3.2.1 Métodos probabilísticos

Esta familia de técnicas de KDD utiliza modelos de representación gráfica para comparar las diferentes representaciones del conocimiento. Estos modelos están basados en las probabilidades e independencias de los datos. Estos son útiles para aplicaciones que involucran incertidumbre y aplicaciones estructuradas tal que una probabilidad puede asignarse a cada uno de los "resultados" o a una pequeña cantidad

del descubrimiento de conocimiento. Las técnicas probabilísticas pueden usarse en los sistemas de diagnóstico y sistemas de control [BUNT 96b].

2.3.2.2 Métodos estadísticos

El método estadístico usa la regla del descubrimiento y se basa en las relaciones de los datos. El algoritmo de aprendizaje inductivo puede seleccionar automáticamente trayectorias útiles y atributos para construir las reglas de una base de datos con muchas relaciones [HSU 96]. Este tipo de inducción es usada para generalizar los modelos en los datos y construir las reglas de los modelos nombrados. El proceso analítico en línea (OLAP) es un ejemplo de un método orientado a la estadística.

Dentro de los métodos estadísticos se pueden distinguir varias técnicas:

- ✓ ANOVA: se basa en el análisis de la varianza
- ✓ Chi cuadrado: contrasta las hipótesis de independencia entre variables
- ✓ Componentes principales
- ✓ Análisis de clusters
- ✓ Análisis discriminante
- ✓ Regresión lineal
- ✓ Regresión logística

2.3.2.3 Métodos de clasificación

La clasificación es probablemente el método más viejo y más usado de todos los métodos de KDD [QUIN 93]. Este método agrupa los datos de acuerdo a similitudes o clases.

El **método Bayesiano** [BUNT 96a] de KDD, que aunque usa los medios probabilísticos y gráficos de representación, también es considerado un tipo de clasificación. Se usan muy frecuentemente las redes Bayesianas cuando la incertidumbre que se asocia a un resultado puede expresarse en términos de una

probabilidad. Este método cuenta con un dominio del conocimiento codificado y ha sido usado para los sistemas de diagnóstico. Otras aplicaciones de reconocimiento de patrones, incluyendo el modelo Markov Oculto, pueden ser modelados usando un método Bayesiano [BUNT 96b].

El **descubrimiento de patrones** y de datos es otro tipo de clasificación que sistemáticamente reduce una base de datos grande a unos cuantos archivos informativos [GUYO 96]. Si el dato es redundante y poco interesante se elimina, de forma que la tarea de descubrir los patrones en los datos se simplifica. Este método trabaja en la premisa de un viejo dicho: "menos es más"'.

El método del **árbol de decisión** usa las reglas de producción, construidas como figuras gráficas basadas en la clasificación de los datos según sus atributos. Es una herramienta analítica empleada para el descubrimiento de reglas y relaciones mediante la ruptura y subdivisión sistemática de la información contenida en el conjunto de datos Según Fallad et al [FAYY 96b] el principal uso de este método es predecir modelos que pueden ser apropiados para cualquier clasificación o técnica de regresión.

2.3.2.4 RR.NN.AA

Se pueden utilizar las RR.NN.AA para la obtención y clasificación de los datos en reglas basadas en rangos. Se debe, por tanto, entrenar una RNA para que reconozca situaciones en donde a partir de patrones entradas / salidas determine los diferentes rangos de funcionamiento y, a partir de ahí, determinar los grados de pertenencia a estos rangos. Esto supone una determinación "a priori" del número de rangos a utilizar y un proceso de entrenamiento adecuado para ellos.

La ventaja de éste método es que puede manejar datos continuos y discretos, lineales y no lineales, simultáneamente. Pero aplicar esta técnica a la extracción de reglas de otra RNA se convierte en una tarea ciertamente recursiva pues, de nuevo, es necesario generar una ecuación o modelo que explique el comportamiento del sistema y es muy difícil determinar la influencia de cada variable en el comportamiento global del sistema.

2.3.2.5 Lógica difusa

Consiste en una generalización del concepto de estadística. La estadística clásica se basa en la teoría de probabilidades, a su vez ésta en la teoría de conjuntos, en la que la relación de pertenencia a un grupo es dicotómica (sí o no). Si se establece una definición de conjunto borroso donde la pertenencia al grupo tiene una cierta graduación se dispone de una estadística más amplia y con resultados más cercanos al modo de razonamiento humano.

2.3.2.6 Métodos evolutivos

La utilización de AA.GG como técnica de extracción de reglas suele enfocarse a problemas de optimización. Se comienza con una población de partida y se va alterando y optimizando su composición (material genético) para la obtención de conocimiento mediante mecanismos tomados de la teoría de la evolución (selección, cruce y mutación).

Como ejemplo se puede citar GABIL [DEJO 91], que realiza una búsqueda incremental para un conjunto de reglas de clasificación representadas por una cadena de bits de longitud fija.

Una de las mayores ventajas aportadas por estos métodos es que el tipo de estructuras de datos con las que se va a trabajar no influye en el algoritmo, simplemente supone cambios en la estructura del cromosoma y no en el proceso de búsqueda [FREI 01].

2.3.2.7 Métodos híbridos

Un método híbrido para KDD combina más de un método en el proceso de extracción, también es llamado método multi-paradigmático. Aunque la implementación puede ser más difícil, las herramientas híbridas son capaces de combinar la potencia de varios métodos. Algunos de los métodos comúnmente usados combinan técnicas de visualización, inducción, RR.NN.AA y sistemas basados en

reglas para llevar a cabo el descubrimiento del conocimiento deseado. También se han usado bases de datos deductivas y AA.GG en los sistemas híbridos.

2.4 Programación Genética

El origen de la Computación Evolutiva (CE) y las ideas evolucionistas que popularizó Charles Darwin [DARW 59][DARW 95], no se originaron con él, sino que estuvieron presentes en las mentes de una serie de científicos y pensadores que no se sentían satisfechos con la idea de que había un Dios originador de todas las especies del planeta (las cuales habían sido creadas de forma separada) y de que las especies estaban jerarquizadas por Dios de tal manera que el hombre ocupaba el rango superior, al lado del creador [COEL 00][COEL 02].

2.4.1 El origen de la Computación Evolutiva

Georges Louis Leclerc [LECL 49] fue tal vez el primero en especular que el ambiente también influía en la evolución de los organismos.

A partir de 1801, el zoólogo francés Jean Baptiste Pierre Antoine de Monet comienza a publicar detalles de su propia teoría evolutiva. Conocedor del trabajo de Lamarck enfatizó la importancia de la naturaleza en los cambios de las especies.

A diferencia de Leclerc, Lamarck sí explicó un mecanismo responsable de los cambios en las especies. Lamarck creía que los organismos no son alterados de forma pasiva por su ambiente, tal y como afirmaba Ëtienne Geoffroy Saint-Hilaire en su Philosophie Anatomique, sino que más bien un cambio en el ambiente produce cambios en las necesidades de los organismos, lo que hace que, en consecuencia, éstos cambien su comportamiento. Estos cambios de comportamiento conducen al mayor uso (o al desuso) de ciertos órganos o estructuras corporales de un individuo, los cuales harán que dichos órganos o estructuras crezcan (ante un mayor uso) o se reduzcan (ante el menor uso) con el paso de las generaciones. Además, Lamarck creía que estos cambios eran hereditarios, lo que implicaba que los organismos se van adaptando gradualmente a su ambiente.

Es interesante hacer notar que aunque el mecanismo evolutivo propuesto por Lamarck difiere notablemente del propuesto varios años después por Darwin, los resultados a los que ambas teorías conducen son los mismos: las especies sufren cambios adaptativos debido a la influencia del ambiente a lo largo de periodos de tiempo considerables. Sin embargo, las ideas de Lamarck no fueron muy populares en su época, y sólo sirvieron para desacreditarlo con sus contemporáneos, incluyendo a Leclerc.

Hasta el re-descubrimiento de las leyes de Gregor Mendel [MEND 65] (enunciadas originalmente en 1865) a principios del siglo XX, nadie entendía de forma precisa los mecanismos de la herencia, y la teoría de Lamarck (con diversas variantes) gozó de gran popularidad entre varios científicos destacados del siglo XIX como el alemán Ernest Haeckel y el norteamericano Edward Drinker Cope, considerándose como una alternativa viable al mecanismo de selección natural que Charles Darwin propusiera en su libro "El Origen de las Especies" [DARW 59].

Aunque Mendel descubrió las leyes de la herencia, realmente no llegó a entender el mecanismo detrás de ellas. Los genes y su mecanismo de transmisión, de generación en generación, no fue descubierto hasta varios años después. Asimismo, su trabajo permaneció largamente ignorado por la comunidad científica, debido a que los dió a conocer originalmente en dos conferencias dictadas ante la Sociedad de Ciencias de Brünn, el 8 de febrero y el 8 de marzo de 1865. El manuscrito original fue publicado en las memorias de dicha Sociedad en 1866, en alemán [MEND 65], y no se tradujeron al inglés hasta 1901 [MEND 01].

2.4.2 La teoría de la mutación

El botánico danés Hugo De Vries creyó haber descubierto una nueva especie de planta al encontrar (alrededor del año 1900) una flor roja entre una gran cantidad de flores amarillas. Según De Vries, esto se debía a una mutación abrupta e infrecuente de las flores amarillas. Accidentalmente, De Vries re-descubrió nuevamente las leyes de la herencia que enunciara varios años atrás Gregor Mendel, y puso de moda la teoría de las

"mutaciones espontáneas" [VRIE 06]. Según De Vries, los cambios en las especies no eran graduales y adaptativos como afirmaba Darwin, sino más bien abruptos y aleatorios (es decir, al azar).

2.4.3 La teoría cromosomática de la herencia

En 1903, Walter Sutton (entonces un estudiante de postgrado en la Universidad de Columbia), leyó el trabajo de De Vries, y determinó correctamente (y sin la ayuda de experimentos genéticos) que los cromosomas en el núcleo de las células eran el lugar donde se almacenaban las características hereditarias. También afirmó que el comportamiento de los cromosomas durante la división de las células sexuales era la base para las leyes de la herencia de Mendel. Un poco después indicó que los cromosomas contenían genes, y que los genes de un mismo cromosoma estaban ligados y, por tanto, se heredaban juntos. A esto se le llamó la "teoría cromosomática de la herencia" [SUTT 02].

Thomas Hunt Morgan confirmaría algunos años más tarde experimentalmente las hipótesis de Sutton, con lo que pasó a ser uno de los pioneros más importantes de la genética moderna.

2.4.4 La evolución natural

La evolución natural fue vista como un proceso de aprendizaje desde el año 1930. Por ejemplo, Cannon [CANN 32] plantea que el proceso evolutivo es algo similar al aprendizaje por ensayo y error que suele manifestarse en los humanos.

El célebre matemático inglés Alan Mathison Turing reconoció también una conexión obvia entre la evolución y el aprendizaje de las máquinas en el artículo "Computing Machinery and Intelligence" [TURI 50] que es considerado hoy en día como un clásico de lectura en el área de IA.

A finales de los años 50, el biólogo Fraser publicó una serie de trabajos sobre la evolución de sistemas biológicos en una computadora [FRAS 57a][FRAS 57b][FRAS 60], dando la inspiración para lo que más tarde se convertiría en el conocido Algoritmo

Genético (AG). El trabajo de Fraser incluye, entre otras cosas, el uso de una representación binaria, de un operador de cruce probabilístico, de una población de padres que generaban una nueva población de hijos tras recombinarse y el empleo de un mecanismo de selección. Su trabajo, de más de 10 años, resumido en el libro [FRAS 70], anticipó la propuesta del AG simple de Holland.

2.4.5 El Algoritmo Genético

John H. Holland empezó a estudiar en los años 60 los procesos lógicos involucrados en la adaptación e inspirado por los estudios de los autómatas celulares [BURK 60] y las redes neuronales [SELF 58], se percató de que el uso de reglas simples podría generar comportamientos flexibles, y visualizó la posibilidad de estudiar la evolución de comportamientos de un sistema complejo [HOLL 95] (conversación personal en su visita a nuestro laboratorio).

Holland vio el proceso de adaptación en términos de un formalismo en el que los programas de una población interactúan y mejoran en base a un cierto ambiente que determina lo apropiado de su comportamiento [HOLL 92]. El combinar variaciones aleatorias con un proceso de selección (en función de qué apropiado fuese el comportamiento de un programa dado), debía conducir a un sistema adaptativo general [HOLL 95].

Este sistema fue desarrollado hacia mediados de los años 60 y se dio a conocer en el libro que Holland publicó en 1975 [HOLL 75] donde denominó el sistema como "plan reproductivo genético" y que después se popularizó bajo el nombre de "Algoritmo Genético".

Aunque concebido originalmente en el contexto de aprendizaje de máquina, el AG se ha utilizado mucho en optimización [QUAG 98], siendo una técnica sumamente popular en los últimos años [HOLL 95].

2.4.6 La Programación Genética

Aunque los primeros intentos por evolucionar programas se remontan a los años 50 y 60 [FRIE 59][FOGE 64] no fue hasta los 80 cuando se obtuvieron resultados satisfactorios. Hicklin [HICK 86] y Fujiki [FUJI 86] usaron expresiones en LISP para representar programas cuyo objetivo era resolver problemas de teoría de juegos. Cramer [CRAM 85] y posteriormente Koza [KOZA 89] propusieron de forma independiente el uso de una representación en árbol donde se implementó el operador de cruce intercambiando sub-árboles entre diferentes programas de una población generada al azar.

John R. Koza en su libro "Genetic Programming. On the Programming of Computers by means of Natural Selection" [KOZA 92] sienta las bases de lo que a partir de ese momento se conoce como Programación Genética (PG) y que originariamente se ha implementado en lenguaje LISP [KOZA 96][KOZA 97]. Su técnica es casi independiente del dominio y ha sido utilizada en numerosas aplicaciones como la compresión de imágenes, el diseño de circuitos electrónicos, el reconocimiento de patrones, los movimientos de robots, etc.

Posteriormente [KOZA 94] [KOZA 99] Koza extendió su técnica mediante la incorporación de lo que ha denominado Funciones Definidas Automáticamente (FDA), las cuales pueden ser reutilizadas como las subrutinas y sirven para incrementar de forma notable el poder de la PG para generar programas automáticamente.

2.4.6.1 Ventajas de la PG

Son varias las razones porqué la PG se ha revelado como una técnica de resolución de problemas potente y sencilla. De ellas se pueden destacar:

- ✓ Implementa un **proceso de búsqueda estocástico**, realizando búsquedas simultáneas en varias regiones del espacio de estados, con continuas exploraciones en nuevos sectores. Se revela así como un eficiente algoritmo de búsqueda, apto para aquellos problemas en los

que su espacio de estados tenga muchos máximos locales, valles, etc. en los que los otros algoritmos, como puedan ser de minimización de gradiente, no llegarían a la solución o le sería muy costoso llegar.

- ✓ Al contrario que en los AA.GG tradicionales, la codificación de cada solución particular (que suele tomar el nombre de *cromosoma*) se realiza en términos del propio problema. En los AA.GG tradicionales era necesario realizar una tarea de codificación / decodificación generalmente en cadenas de bits, pero en la PG esto ya no es necesario, puesto que la solución buscada va a ser encontrada en forma de algoritmo, de una forma similar a la que entienden los compiladores: en forma de árbol.
- ✓ Al estar trabajando con este tipo de estructuras y codificación, se minimiza y facilita el análisis necesario para la codificación, puesto que ahora se va a realizar de una forma próxima al problema a tratar, lo que también proporciona una gran versatilidad y variedad de problemas que va a poder resolver.
- ✓ Son algoritmos inherentemente paralelos, puesto que se basan en realizar cálculos de forma masiva e independiente (evaluar muchas soluciones distintas e independientes).
- ✓ Permite la obtención de soluciones de forma jerárquica.

2.4.6.2 Funcionamiento de la PG

El funcionamiento es similar al de los AA.GG y se basa en la iteración de sucesivas generaciones a partir de las anteriores. Este algoritmo se puede ver en la figura 12 [KOZA 92].

Tras la creación inicial de árboles, que generalmente serán aleatorios, se construyen sucesivas generaciones a partir de copias, cruces y mutaciones de los individuos de cada generación anterior.

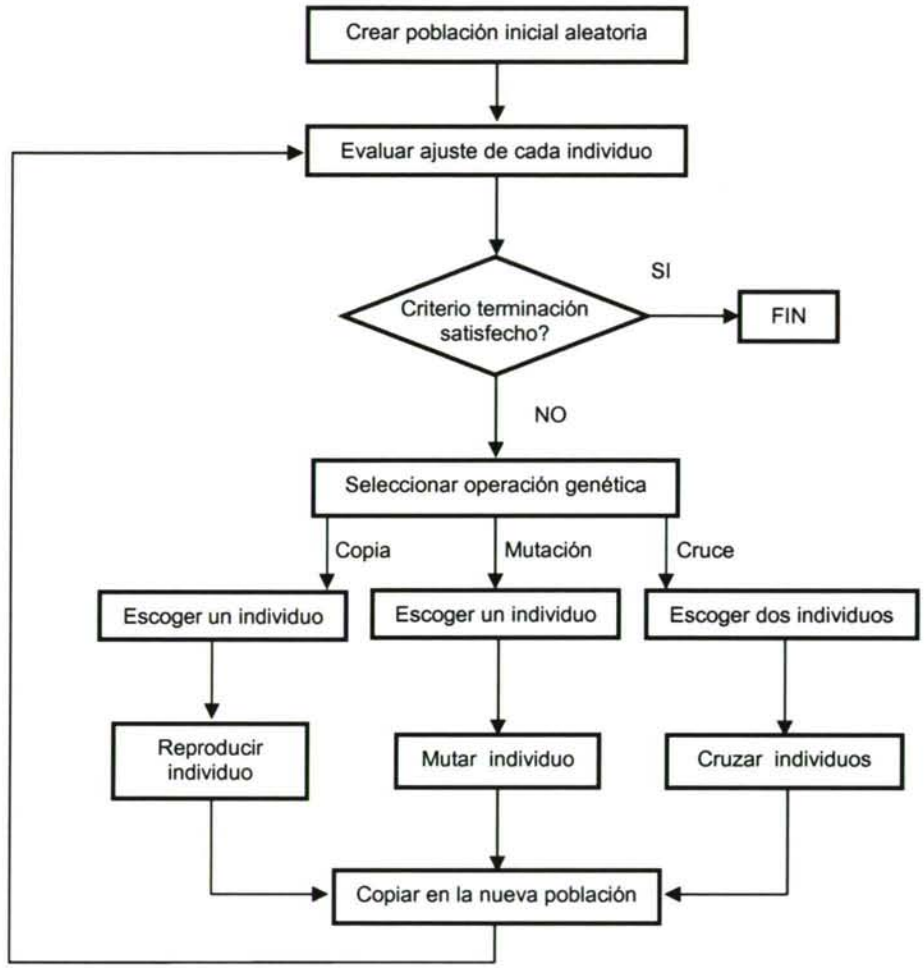


***Fig. 12** - Diagrama de flujo de la Programación Genética*

El primer paso en el funcionamiento del algoritmo es la generación de la población inicial. En la creación de la generación 0, cada árbol se creará de forma más o menos aleatoria, dependiendo del algoritmo, teniendo en cuenta las restricciones que existen en los árboles. Dado que los árboles son aleatorios, los individuos de esta población en general representan soluciones malas al problema.

Para la creación de un árbol existe una gran variedad de algoritmos, pero son tres los más utilizados [KOZA 92]: parcial, completo e intermedio.

- ✓ El algoritmo de creación parcial genera árboles cuya altura máxima no supera la especificada.

- ✓ El algoritmo completo genera árboles cuyas hojas están todas a un determinado nivel, ya que genera árboles completos.
- ✓ El algoritmo de creación intermedio es una mezcla de los dos anteriores, creado para que exista mayor variedad en la población inicial, y con ello mayor diversidad genética. Este algoritmo se basa en ejecutar los anteriores alternándolos y tomando distintas alturas para crear todos los elementos de la población.

Estos algoritmos se basan fuertemente en el azar, y la única intervención del usuario está en la introducción de los elementos terminales y no terminales. Sin embargo, existen muchos más algoritmos en los que la creación de árboles no es tan aleatoria. Por ejemplo, Luke [LUKE 00] asigna una probabilidad de aparición a cada nodo y de esta forma se reduce el carácter aleatorio de la creación y se orientan los árboles a que contengan más nodos de una clase que otra. Los algoritmos serán muy similares, con la salvedad de que la elección de los elementos seguirá siendo aleatoria, pero estará ponderada por esa probabilidad asignada.

2.4.6.3 Elementos del árbol

Al haber una representación en árbol, existirán dos tipos de nodos:

- ➢ **Terminales, u hojas del árbol**. Son aquellos que no tienen hijos. Normalmente se asocian con valores constantes o variables.
- ➢ **No terminales**. Son aquellos que tienen uno o más hijos. Generalmente se asocian con operadores del algoritmo que se quiere desarrollar.

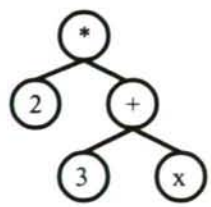


***Fig. 13** - Árbol para la expresión 2*(3+x)*

En la figura 13 se puede ver un ejemplo de árbol, que representa el programa f(x) = 2*(3+x). Tiene como nodos no terminales los correspondientes al producto y a la suma, y como terminales los correspondientes a los valores 2 y 3 y la variable x.

Una parte fundamental del funcionamiento de la programación genética es la especificación del conjunto de elementos terminales y no terminales antes del inicio del proceso evolutivo. Con los nodos que se le especifique, el algoritmo construirá los árboles. Por tanto, es necesario un mínimo proceso de análisis del problema para configurar el algoritmo, puesto que hay que decirle qué operadores puede utilizar. Como regla general, es conveniente ajustar el número de operadores sólo a los necesarios, puesto que la adición de elementos que no sean necesarios no provocará que no se encuentre la solución, pero sí que el algoritmo tarde más en encontrarla.

A la hora de especificar los conjuntos de elementos terminales y no terminales, es necesario que estos conjuntos posean dos requisitos, que son *suficiencia* y *completitud*. El requisito de suficiencia dice que la solución al problema debe poder ser especificada con el conjunto de operadores especificados. El requisito de completitud dice que debe ser posible construir árboles correctos con los operadores especificados.

Dado que el proceso de construcción de árboles es un proceso basado en el azar, muchos de los árboles construidos no serán correctos, no por no seguir las reglas de la gramática, sino por la aplicación de operadores (nodos no terminales) a elementos que no están en su dominio. Por esta razón no se aplican estos operadores directamente, sino una modificación de ellos en la que se amplía su dominio de aplicación. El ejemplo más claro es el operador de división, cuyo dominio es el conjunto de números reales excepto el valor cero. Ampliando su dominio, se define un nuevo operador (%):

$$\%\,(a,b) \begin{cases} 1 & \text{si } b = 0 \\ a/b & \text{si } b \neq 0 \end{cases}$$

A esta nueva operación se denomina *operación de división protegida*, y en general cuando se crea una nueva operación que extiende el dominio de otra se denomina *operación protegida*.

2.4.6.4 Restricciones

Existen dos tipos principales de restricciones:

- Tipado.
- Altura máxima del árbol.

Para establecer reglas sintácticas en la creación de árboles, es posible especificar reglas de tipado [MONT 95]. Se establece el tipo de cada nodo (terminales y no terminales), y para los no terminales el tipo que debe tener cada hijo. De esta forma, se especifica la estructura que deben seguir los árboles.

Al especificar el tipo de cada nodo, se está especificando una gramática que va a ser la que siga el algoritmo para la construcción de árboles, esta gramática permitirá que los árboles tengan la estructura deseada.

Los tipos más usados son aquellos que tienen que ver con la realización de operaciones aritméticas: reales y enteros. Sin embargo, son también muy usados otros como el tipo booleano, o el tipo sentencia, este último se utiliza cuando se quiere desarrollar un programa que sea una secuencia de mandatos, y que designa nodos que no devuelven nada, sino que su evaluación se basa en los efectos laterales que provoca su ejecución.

Por su parte, la restricción de altura evita la creación de árboles demasiado grandes y fuerza una búsqueda de soluciones cuyo tamaño se acota de antemano. Con esta restricción se evita que los árboles posean mucho código redundante y el crecimiento excesivo de los árboles [SOUL 97][SOUL 98].

2.4.6.5 Operadores genéticos

En la creación de una nueva generación se aplican operadores que generan y modifican los árboles. Estos operadores son resultado de adaptar los existentes en los AA.GG tradicionales a la PG, modificándolos para adaptarlos a la codificación en forma de árbol. Los operadores más utilizados son:

- Cruce.
- Reproducción.
- Selección.
- Mutación.

2.4.6.5.1 Cruce

Es el principal operador. En él, dos individuos de la antigua población se combinan para crear otros dos individuos nuevos. Después de seleccionar a dos individuos como padres, se selecciona un nodo al azar en el primero y otro en el segundo, de forma que su intercambio no viole ninguna de las restricciones: los nodos deben ser de igual tipo y los árboles nuevos deben seguir manteniendo la altura máxima. El cruce entre los dos padres se efectúa mediante el intercambio de los subárboles seleccionados en ambos padres.

Existen variantes adaptativas de este operador [ANGE 96] en las que el propio algoritmo se modifica, así como numerosas variantes que tienen como base este algoritmo [AGUI 99][PERE 99].

2.4.6.5.2 Reproducción

La reproducción simplemente es la copia de individuos en la nueva generación. Esta operación es asexual en el sentido en que se genera un individuo a partir de un individuo anterior.

Esta operación, junto con la de cruce, forman los operadores que se utilizan más frecuentemente, y entre ellos tiene una especial predominancia la de cruce. De hecho, el

porcentaje de individuos nuevos generados a partir de cruces suele ser superior al 90%, mientras que el resto son generados mediante copias. El aumento del número de individuos generados por copias aumenta el peligro de predominancia de un individuo sobre el resto de la población, y que finalmente tras varias generaciones toda la población converja hacia ese individuo. Esta es una situación indeseable, puesto que se ha perdido por completo la diversidad genética que se tenía al principio y ello conlleva que la búsqueda en el espacio de estados sólo se lleve a cabo en una determinada zona, lo cual es lo contrario que se pretende con AG.

El operador de cruce es por tanto el principal operador utilizado para la generación de los nuevos árboles y exploración del espacio de estados [POLI 98].

2.4.6.5.3 Selección de individuos

Para evitar una predominancia de un individuo en la reproducción, ya sea por cruce o copia, es necesario regular bien la selección de los individuos a los que se aplican estos operadores. Existen muchos algoritmos de selección, pertenecientes a los AA.GG tradicionales, que realizan la tarea de escoger qué individuos se van a reproducir y cuáles no.

En general todos los algoritmos de selección se basan en la misma idea: que los individuos más aptos tengan más posibilidades de ser escogidos para reproducirse pero sin eliminar por completo las posibilidades de los menos aptos, puesto que de ser así la población convergería en pocas generaciones. Para medir la bondad de un individuo, éste está valorado con un nivel de ajuste o aptitud, y en base a ese nivel se puede decidir la elección de individuos. Algunos de los algoritmos más utilizados son la selección por torneo y la ruleta.

En la selección por torneo [WETZ 83], se escoge un número de individuos al azar de la población (típicamente dos individuos) y es seleccionado el mejor de ellos. Este método de selección es muy usado, y puede regular la presión de selección que se ejerce sobre la población variando el número de individuos que participan en el torneo. De esta forma, si participa un número bajo, se ejerce poca presión y se dan más oportunidades

de ser seleccionados a los menos aptos. Conforme crece el número de individuos, serán seleccionados los mejores más frecuentemente. Además, es posible seleccionar un individuo varias veces.

En la selección por ruleta [DEJO 75], todos los individuos de la población son dispuestos en una ruleta ocupando cada uno una parte proporcional al nivel de ajuste del individuo comparado con el nivel de ajuste de toda la población (suma de ajustes); es decir, que ocupan más espacio en la ruleta los mejores individuos. Para realizar la selección, simplemente se hace girar la ruleta y se devuelve el individuo seleccionado por ella. Este método es muy utilizado por su simplicidad y sus buenos resultados. Sin embargo, presenta el problema de que al poder seleccionar el mismo individuo varias veces, el mejor individuo sea escogido muchas veces y acabe predominando en la población.

Existen otros muchos algoritmos, en los que el número de veces que un individuo es seleccionado se realiza de forma determinística. Esto evita problemas de predominancia de un individuo. Cada uno de estos algoritmos presentan variaciones respecto al número de veces que se tomarán los mejores y los peores. Se impondrá una presión en la búsqueda en el espacio de estados en la zona donde se encuentra el mejor individuo, o bien se tiende a repartir la búsqueda por el espacio de estados, pero sin dejar de tender a buscar en la mejor zona. Algunos de estos algoritmos son sobrante estocástico [BRIN 81][BOOK 82], universal estocástica [BAKE 87] o muestreo determinístico [DEJO 75].

2.4.6.5.4 Mutación

El operador de mutación provoca la variación de un árbol de la población. Este operador suele usarse con probabilidad muy baja (menos de 0.1) antes de introducir un individuo en la nueva generación.

Existen dos tipos principales de mutación: mutación en la que se varía un solo nodo y mutación en la que se varía una rama entera del árbol.

En el primer caso, conocida por mutación puntual, la mutación actúa de la siguiente manera:

1. Se escoge un nodo al azar del árbol.
2. Se escoge al azar un nodo del conjunto de terminales o no terminales, del mismo tipo que el seleccionado, con el mismo número de hijos y de forma que sus hijos sean del mismo tipo.
3. Se intercambia el nodo antiguo del árbol por el nuevo, manteniendo los mismos hijos que el antiguo.

Dado que cada rama del árbol representa una solución a un subproblema y el no terminal que las une representa la forma de combinar esas soluciones, si se realiza este tipo de mutación sobre un elemento no terminal se estará provocando que las soluciones se combinen de distinta forma. Este tipo de mutación apenas se usa.

El operador de mutación provoca en el individuo un salto en el espacio de estados, comenzando una búsqueda distinta en otra zona. La mayoría de las mutaciones son destructivas, es decir, el individuo empeora, y por eso se utilizan con una probabilidad muy baja, para conseguir variedad genética. Existen estudios sobre la evolución sin el uso de cruces, en los que la mutación juega un papel fundamental [CHEL 97], en los que se utilizan distintos tipos de mutaciones, pero los resultados siguen siendo peores que utilizando cruces.

2.4.6.6 Evaluación

La cuantificación de la bondad de un determinado individuo se realiza por medio del ajuste de ese individuo. Este valor representa lo bien que el árbol soluciona el problema actual.

El ajuste es probablemente el principal concepto en la evolución darwiniana, se refiere a la habilidad que tiene un individuo de competir en un entorno por los recursos disponibles. Goldberg describió la función de ajuste como "una medida de beneficio, utilidad o bondad que queremos maximizar" [GOLD 89].

En el algoritmo genético, esta competición se basa en la actuación del cromosoma dentro del dominio del problema. Se determina una escala adecuada a la tarea como "tiempo antes de fallo" [RAND 94], o "tiempo antes de estabilizarse" [KOZA 90]. Después de haber aplicado un cromosoma al problema, se le asigna un valor de ajuste que refleje su actuación. De esta manera, cuando la población entera haya sido probada, la habilidad relativa de cada cromosoma puede ser identificada.

Para valorar esta medida de ajuste, existen tres tipos fundamentales [KOZA 92]:

- ✓ **Estandarizado.** Este tipo de ajuste s(i,t) mide la bondad de un individuo i en la generación t, de tal forma que valores próximos a cero indican un buen valor de ajuste y valores lejanos un mal individuo. Por tanto, en una generación t un individuo i será peor que otro j si s(i,t) > s(j,t). Esta medida es muy útil en problemas en los que la cuantificación del nivel de ajuste de los individuos se basa en penalizaciones, como puede ser el error en inducción de fórmulas, error cuadrático medio, número de ejecuciones necesarias para encontrar la solución, etc.

- ✓ **Ajustado**. Este valor se obtiene del estandarizado de la siguiente forma:

$$a(i,t)=\frac{1}{1+s(i,t)}$$

  Con esta medición la bondad se cuantifica entre 0 y 1, siendo el valor 1 correspondiente al mejor individuo.

- ✓ **Normalizado**. Es un valor de ajuste comparativo del individuo con toda la población. Se obtiene de la siguiente expresión, dado el tamaño de población M:

$$n(i,t)=\frac{a(i,t)}{\sum_{k=1}^{M} a(k,t)}$$

  Este valor está comprendido entre 0 y 1. Este tipo de ajuste indica el nivel de bondad dentro de la población. En este caso desaparece el

componente de objetividad de evaluación de los tipos anteriores, y un valor cercano a 1 ya no indica que ese individuo represente una solución buena al problema, sino que ese individuo representa una solución destacada y notablemente mejor que la del resto de la población. Este valor es utilizado para las selecciones proporcionales al ajuste, como la ruleta. En este caso, la proporción de ruleta ocupada por un individuo será este valor, ya que la suma de todos será la unidad.

Para evitar redundancia en el código y un crecimiento excesivo de éste, puede incluirse un factor de parsimonia en el cálculo del ajuste [SOUL 97] [SOUL 98]. Esta técnica se puede usar para reducir la complejidad del cromosoma que está siendo evaluado, y funciona mediante la penalización en el ajuste del individuo i de la siguiente forma:

$$f(i) = P(i) + \alpha \cdot s_i$$

Donde P(i) es una medida de la bondad del individuo (en este caso, peor cuanto más positivo), α es el nivel de parsimonia y S_i es el tamaño (número de nodos) del individuo. Con este coeficiente se está penalizando el número de nodos de un árbol, y su valor máximo suele ser de 0.1. Con este valor, se necesitará que el árbol tenga 10 nodos para incrementar en una unidad el valor de ajuste. Sin embargo, un valor tan alto ya es muy dañino en la evolución, y se suelen tomar valores menores (0.05, 0.01, etc.), dependiendo del rango de valores en los que se espera que estén los ajustes de los individuos.

"¿ Podrías decirme, por favor, hacia donde debo ir desde aquí ?"

"Eso depende bastante de a donde quieras llegar"

Lewis Carrol

CAPÍTULO

3 ESTADO DEL ARTE

3.1 Motivación

Las RR.NN.AA han sido aplicadas satisfactoriamente a problemas de aprendizaje y generalización en una gran variedad de situaciones del mundo real [SEJN 86][WAIB 89][POME 89][LECU 90][TESA 92]. Esto ha permitido a las RR.NN.AA abarcar un amplio rango de dominios de aplicación. Sin embargo las salidas que producen las RR.NN.AA no siempre son comprensibles por los usuarios humanos, aunque sean las correctas. Esto es debido a que las soluciones que producen las RR.NN.AA son, la mayor parte de las veces, un gran número de elementos no-lineales, caracterizados por grandes conjuntos de valores numéricos que son muy difíciles de interpretar. Además, las representaciones internas distribuidas en los elementos de proceso, que típicamente ocurren durante el proceso de entrenamiento, hacen muy difícil entender qué ha aprendido exactamente la red, y dónde fallará, en el caso de que lo haga, en la generación de una respuesta.

A continuación se presentarán las diversas técnicas que hasta el momento se han venido aplicando a la tarea de extracción de conocimiento de las RR.NN.AA. Se expondrán algunos métodos que utiliza la computación evolutiva, ya que estos comparten una base común con las RR.NN.AA y ambos son emulaciones de comportamientos biológicos compartiendo, por tanto, numerosas características. Se empezará dando una somera explicación histórica del por qué del proceso evolutivo y del proceso de aprendizaje. Y debido a que en esta tesis se ha desarrollado un sistema inductivo de extracción de reglas utilizando PG se referenciarán trabajos sobre esta técnica evolutiva. Posteriormente, se mostrará toda una serie de técnicas, métodos y sistemas existentes para la extracción de conocimiento de las RR.NN.AA y de las bases de datos. Debido a que los métodos de extracción de reglas de base de datos utilizan como núcleo el tratamiento sobre los datos, las técnicas existentes para este ámbito son de relevante importancia para esta tesis, pues al tratar una RNA como una caja negra (para aislar del tratamiento la arquitectura de la red) hay que centrarse en el tratamiento de los patrones entrada-salida de ella, y por consiguiente estas técnicas están intrínsecamente relacionadas con el tema de esta tesis.

3.2 Sistemas evolutivos

A lo largo del período de evolución de los seres vivos, se han ido seleccionando conductas que son adecuadas para la supervivencia. Las especies y los individuos que sobreviven son aquellos que están mejor adaptados al ambiente que les rodea. En la lucha por sobrevivir, los individuos mejor adaptados sobreviven más fácilmente y por tanto pueden reproducirse y transferir a sus descendientes las cualidades beneficiosas que les permitieron estar mejor adaptados, lo cual constituye el proceso de selección natural explicado por Charles Darwin en su libro "El origen de las especies" [DARW 59]. Todos aquellos rasgos que faciliten la supervivencia del individuo tenderán a mantenerse, mientras que lo que constituya una dificultad o debilidad para la adaptación tenderá a desaparecer, pues su poseedor no tendrá oportunidad de legarlo a su descendencia.

Ya en la década de los 50, Arthur Samuel [SAMU 59] intentando superar el conocido como régimen de Lovelace en el que se establecía que los computadores sólo hacen aquello que se les dice cómo han de hacerlo,planteó la cuestión que ha llevado al desarrollo de la Computación Evolutiva: "¿Cómo pueden los ordenadores aprender a resolver problemas sin ser explícitamente programados?". La computación evolutiva (CE) abarca una serie de técnicas como la Programación Genética (PG) [KOZA 89], los Algoritmos Genéticos (AA.GG) y la Vida Artificial, entre otras. Todas ellas son formas de aprendizaje basadas en una evolución simulada [COEL 00][COEL 02].

El considerado "padre" de la PG es John R. Koza. Él acuñó el término que da título a la publicación del libro "Programación Genética" [KOZA 92]. Este libro es el que establece de modo formal las bases de la PG sobre la que se trabaja hoy en día. Posteriormente, el mismo autor ha publicado "Programación Genética II" [KOZA 94] y, muy recientemente, "Programación Genética III" [KOZA 99], que exploran nuevas posibilidades para la PG.

Dentro de la PG se están desarrollando distintas ramas. Una de las más prometedoras en cuanto a la extracción de conocimiento (Knowledge Discovery KD) son las reglas difusas [FAYY 96a][BONA 96]. Esta rama es fruto de la unión de la lógica difusa (fuzzy logic) y de los sistemas basados en reglas (SBR). La obtención de reglas difusas se puede realizar mediante computación evolutiva con la técnica de las Funciones Definidas Automáticamente (Automatically Defined Functions ADF) [KOZA 94], las cuales representan una evolución del concepto de lo que se ha denominado "Programación Genética Clásica" (si se puede considerar como clásica una materia de estudio con apenas 15 años de vida), desarrollado por John R. Koza y James P. Rize.

3.3 Evolución histórica de la extracción de reglas de RR.NN.AA

Los diferentes algoritmos que existen para la extracción de reglas de RR.NN.AA básicamente se diferencian entre sí por alguna o algunas de las siguientes características [DUCH 00]:

- ✓ La "potencia expresiva" de las reglas extraídas (tipos de reglas)
- ✓ La "calidad" de las reglas extraídas (exactitud, comparación fidedigna del funcionamiento de la RNA, comprensibilidad y consistencia de las reglas)
- ✓ La "translucidez" del método: análisis de las neuronas individuales versus análisis de la función de la RNA al completo
- ✓ La complejidad algorítmica del método
- ✓ El nivel de dependencia del método de extracción a los esquemas de entrenamiento de la red
- ✓ El tratamiento de las variables lingüísticas

La equivalencia entre RNA y reglas difusas ha sido estudiada por diferentes autores [JANG 92][BUCK 93][BENI 97]. La mayoría de los resultados establecen dicha equivalencia a través de un proceso de sucesivas aproximaciones. Además de ser soluciones puramente teóricas requieren un número de reglas muy elevado para aproximar el funcionamiento de la RNA [BENI 97]. El trabajo de Jang y Sun [JANG 92] proporciona una equivalencia entre RR.NN.AA radiales y sistemas difusos donde se requiere un numero finito de reglas o de neuronas, pero en este caso se limita a una arquitectura fija de RNA.

Andrews [ANDR 95][ANDR 96] identifica tres técnicas de extracción de reglas: "decompositional", "pedagogical" y "eclectic". La primera hace referencia a la extracción a nivel de cada neurona. La segunda trata la RNA como una caja negra donde aplicando entradas a la red se analiza de atrás hacia adelante a través de las neuronas de las capas ocultas y se extraen las reglas pertinentes. Y la tercera usa la arquitectura de la RNA y los pares entrada-salida como complemento a un algoritmo de entrenamiento simbólico.

Towell y Shavlik [TOWE 94] aplican la primera técnica utilizando las conexiones entre neuronas como reglas basándose en transformaciones simples. Esto limita la

extracción a redes con una cierta arquitectura multicapa y de pocos elementos de proceso [ANDR 95].

La principal aproximación utilizando la segunda técnica ha sido desarrollada por Thrun [THRU 95] y se titula "Validity Interval Analysis" (VI-A). El algoritmo utiliza programación lineal (SIMPLEX) aplicando intervalos de valores a las activaciones de cada neurona en cada una de las capas. Por propagación de esos intervalos hacia delante y hacia atrás a través de la RNA el sistema extrae reglas "posiblemente correctas". Este algoritmo, debido a que utiliza programación lineal, en el peor de los casos tiene complejidad exponencial. Además, cuando se utiliza un numero elevado de elementos de proceso el tiempo para alcanzar la solución es excesiva.

Otras aproximaciones utilizando la segunda técnica son los algoritmos RULENEG [POP 94] y DEDEC [TICK 96] los cuales utilizan una RNA para extraer reglas de otra RNA a partir del conjunto de entrenamiento. Pero las técnicas de extracción de reglas que se centran sólo en los datos de entrenamiento pierden la facultad de generalización que tienen las RR.NN.AA. Otras técnicas de extracción de reglas son las desarrolladas por Chalup et al, Visser et al y Tickle et al [CHAL 98][VISS 96][TICK 98] que se basan en unas u otras aproximaciones que se han visto anteriormente.

Más recientemente y debido a las ventajas que incorporan los métodos evolutivos para la búsqueda en sistemas complejos, se han usado los AA.GG para la búsqueda y extracción de reglas de RR.NN.AA. Utilizando la segunda técnica (pedagogical), Keedwell [KEED 00] utiliza un AG donde los cromosomas son reglas multicondición basadas en intervalos o rangos de valores aplicados a las entradas de la RNA. Estos valores se obtienen de los patrones de entrenamiento.

3.4 Clasificación de las técnicas de extracción de reglas

Hasta ahora se han utilizado diversas técnicas para caracterizar el resultado del entrenamiento de las RR.NN.AA, siendo una de las pioneras la utilización de la estadística. Estos métodos, siempre basados en regresiones lineales, realizan un proceso

de inferencia, una vez se ha entrenado la red, en el proceso de evaluación de la misma, donde se "mide" su capacidad de generalización con valores que no se han utilizado en el proceso de entrenamiento. Aunque estos métodos estadísticos proveen un significado (en ratio) a los futuros fallos de clasificación, supone disponer de una gran cantidad de datos de test, que incluso en muchas situaciones es imposible de obtener. Además, estos métodos no explican la función de clasificación realizada por la RNA.

Es entonces cuando se ha pensado la opción de utilizar reglas simbólicas que reflejen el ajuste de los datos entrenados [TOWE 90][FRAS 91][HANS 91][GILE 92a][WATR 92][FU 94]. Se ha visto que el utilizar reglas redunda en un beneficio mayor que el de explicar el funcionamiento de la red, las reglas simbólicas permiten interactuar con los sistemas basados en el conocimiento como por ejemplo los SE o las bases de datos inteligentes. La importancia de la verificación y extracción de reglas de las RR.NN.AA ha sido entonces reconocida como de gran importancia. Han aparecido muchas técnicas de extracción de reglas, pero casi todas se centran en una arquitectura o en un tipo de RNA determinado. La mayoría de las técnicas existentes se centran en unas RR.NN.AA alimentadas hacia delante, conocidas en la literatura como Perceptron Multicapa (MLP). Existen pocas publicaciones de técnicas de extracción de reglas de otras arquitecturas de RR.NN.AA. Así, por ejemplo, técnicas para la extracción de reglas de redes neuronales de base radial (Radial Basis Function, RBF) expuestas por Tresp [TRES 93] donde utilizan neuronas gaussianas, Berthold [BERT 95] utiliza redes con funciones locales especializadas (RecBF), Abe [ABE 95] describe un método recursivo para construir hiper-cajas de las que extrae reglas difusas, Duch [DUCH 99] describe un método para extraer, optimizar y aplicar un conjunto de reglas difusas que representa las funciones que se forman en el interior de una RNA entrenada y Andrews [ANDR 00] expone un método de extracción (RULEX) de reglas simbólicas de una RNA con clusters locales (LC). Todos estos métodos de extracción se basan en una análisis exhaustivo de las activaciones de las neuronas y, partiendo de complejas fórmulas matemáticas, alcanzan una serie reducida de reglas simbólicas basadas en rangos de valores numéricos acotados.

3.4.1 Autómatas de estado finito

Desde los primeros momentos se ha podido identificar a las RR.NN.AA como autómatas de estados finitos. El primer trabajo en este sentido ha sido de McCulloch y Pitts mostrando que algunas RR.NN.AA se pueden comportar como algunos tipos de autómatas de estado finito. Posteriormente, Minsky demuestra que un perceptron multicapa se puede representar por medio de un autómata de estado finido. Pero el proceso de identificar estados en una RNA se vuelve complejo cuando su arquitectura es recurrente, de todas formas ha habido trabajos que han seguido esta línea. El primero de todos ha sido Giles [GILE 92b] el cual propone un algoritmo para extraer un autómata de estado finito de una RNA recurrente. Este algoritmo intenta agrupar las salidas en clusters e identificarlos como estados de un autómata. El algoritmo de extracción divide las salidas de cada una de las N neuronas en q intervalos de igual tamaño, esto supone q^N particiones en el espacio de salidas. El resultado del algoritmo de extracción es un árbol de búsqueda donde las uniones entre nodos corresponden a transiciones entre estados del autómata. El autómata de estado finito extraído de la red es por tanto un árbol con numerosos nodos y transiciones entre nodos. Este autómata (árbol) suele ser muy difícil de entender por seres humanos.

Un segundo trabajo que intenta extraer reglas usando lenguajes basados en fórmulas lógicas es el presentado por Shavlik [SHAV 92]. Partiendo del concepto de que las RR.NN.AA.RR tienen la habilidad de mantener estados de información de un ciclo al siguiente en las neuronas ocultas, se pueden representar como unidades de alto nivel donde residen las características importantes de representación. Se basa entonces, el algoritmo, en "entender" estas unidades de alto nivel. Pero este proceso es complejo debido a la interrelación entre las unidades.

Craven [CRAV 97] introduce los conceptos de lenguaje de representación y estrategia de extracción. Lenguaje de representación es el lenguaje usado por el método de extracción. Los lenguajes que se han usado por diversos métodos de extracción son las reglas de inferencia (IF-THEN), las reglas M-N, las reglas difusas, los autómatas de estado finito y los árboles de decisión. La estrategia de extracción es el método usado

para representar la RNA entrenada en el lenguaje de representación. Específicamente es la explicación de cómo el método explora un espacio de características posibles. Cuando se extraen las reglas éstas deben describir el comportamiento de la RNA.

Schellhammer [SCHE 96] se centra en la extracción de reglas gramáticas de las RR.NN.AA.RR de tipo Elman. Omlin [OMLI 96] usa una RNA para simular un autómata de estado finito que representa una gramática regular. Extraen un autómata de una RNA entrenada con sintaxis en lenguaje natural. La salida de la RNA es convertida en un diagrama de transición de estados que representa la gramática que ha aprendido la red.

3.4.2 Basados en estrategias de búsqueda

Muchos algoritmos de extracción de reglas comienzan como un problema de búsqueda sobre un espacio de reglas candidatas comprobando cuáles son válidas y cuáles no. Muchos de estos algoritmos usan un espacio de reglas "IF-THEN" que representan un árbol de decisión. Cada nodo de este árbol con su antecedente corresponde con una posible regla. Y cada una de ellas es validada considerando las restricciones que aparecen en las entradas-salidas de la RNA.

El proceso de validación de las reglas en una RNA multicapa es ciertamente problemática. Una aproximación para validar las reglas es tratar las RR.NN.AA como una colección de "perceptrons", y de esta forma poder extraer reglas de cada neurona oculta y de salida de forma separada [CRAV 93]. Esto es lo que se considera como una **aproximación descomposicional**. Una ventaja de esta estrategia es que produce "términos intermedios" que resultan en simples descripciones. Pero, por el contrario, el algoritmo requiere tratar las neuronas ocultas como neuronas umbrales y esto supone una pérdida de capacidad de generalización en la red.

Uno de los principales problemas de los métodos de extracción basados en búsqueda es que el espacio de búsqueda puede ser enorme. Para una tarea que tenga "n" características binarias existen 3^n posibles reglas "IF-THEN" (una característica puede aparecer como un literal positivo, negativo o no aparecer). Para limitar ésta explotación

combinatoria del proceso de exploración de reglas han aparecido varias estrategias basadas en la heurística. Towell [TOWE 93] propone una técnica de búsqueda de reglas que incluye "M-N" expresiones. Una "M-N expresión" es una expresión booleana que está especificada por un valor de umbral entero "M" y un conjunto de "N" literales booleanos. Cada expresión es satisfecha cuando al menos "M" de sus "N" literales son satisfechos. La extracción de "M-N" reglas ofrecen mucha más exactitud y unas reglas más entendibles que las "IF-THEN".

3.4.3 Basados en estrategias de aprendizaje

El algoritmo de extracción llamado TREPAN [CRAV 96] trata la búsqueda de reglas como una tarea de aprendizaje inductivo. TREPAN realiza conjuntos de test que introduce como entradas a la RNA y observando las salidas induce un árbol de decisión que aproxima la función representada en la RNA. Este algoritmo es similar al árbol de decisión que explora "M-N expresiones", pero trabaja directamente sobre un conjunto de datos que introduce a la RNA. Esta forma de extracción presenta dos grandes ventajas: la primera es que el método es independiente de la arquitectura de la red y del método de entrenamiento y la segunda es que el algoritmo le da al usuario un control estricto sobre la complejidad de los modelos extraídos.

Al igual que la gran mayoría de los algoritmos desarrollados hasta el momento, son diseñados específicamente para trabajar sobre tareas de clasificación donde las salidas de la RNA son, booleanas o de carácter discreto, extrapolables a valores de clasificación. Por eso, utilizar árboles de decisión favorece la discriminación de conocimiento residente en la RNA porque, en definitiva, las RR.NN.AA clasificadoras deben aprender a decidir, ante una secuencia de valores de entrada, la salida dentro de unas clasificaciones preestablecidas. Por lo tanto, cuando se afronta la extracción de conocimiento de RR.NN.AA que trabajen sobre salidas continuas, estos algoritmos no son aplicables. Y, por regla general, los basados en árboles de decisión no son los adecuados para tareas en las que se trabaje con salidas de tipo continuo.

3.4.4 Métodos de aprendizaje de hipótesis

Jackson [JACK 96] presenta el algoritmo llamado "Boosting Based Perceptron" (BBP), basado en un método de "estimulación de hipótesis". Es aplicable a RR.NN.AA de tipo "perceptron" con muy pocas conexiones. Este algoritmo no es en sí una estrategia de extracción de reglas sino una técnica de construcción de pequeñas RR.NN.AA multicapa cuyo funcionamiento y operación es comprensible por seres humanos. El algoritmo BBP tiene dos grandes limitaciones: por un lado está diseñado para sólo aprender tareas de clasificación binarias y, por otro, supone que las entradas de la red son funciones booleanas.

3.4.5 Métodos descomposicionales

Son presentados por Krishnan [KRIS 96] como un métodos descomposicionales de extracción de reglas "IF-THEN" por medio de un proceso de búsqueda de combinaciones de pesos que hagan activar una neurona de la RNA. Las reglas son obtenidas de las neuronas de capas ocultas y de salida. De cualquier forma es inherentemente combinacional. Propone una estrategia heurística para reducir el espacio de búsqueda y limitar el algoritmo a RR.NN.AA alimentadas sólo hacia delante y con entradas de tipo booleano. El algoritmo se ha denominado COMBO.

3.4.6 Métodos auto-organizativos

Lawrence [LAWR 00] entrena RR.NN.AA.RR para predecir valores de varias series temporales y, posteriormente, extrae autómatas de estado finito de esas redes. Esta tarea involucra tres componentes principales: el primero es una RNA llamada SOM (Self-Organizing Map) [KOHO 88] la cual es entrenada por un método no supervisado, el segundo es una RNA que tiene un conjunto de conexiones recurrentes de cada neurona oculta a todas las demás neuronas ocultas y el tercero es el componente de extracción de reglas, un autómata de estado finito que se extrae de la RNA recurrente. Los estados del autómata se corresponden a regiones del espacio de activaciones de las neuronas. Cada estado es etiquetado por la correspondiente

predicción de la red y cada estado de transición es etiquetado por el valor de la variable discreta que caracteriza la serie temporal. Se han comparado los resultados de las predicciones con los valores de los estados de los autómatas y han observado que sólo eran mínimamente acertados, pero esto implica un punto de partida en el desarrollo de autómatas de estado finito a partir de RR.NN.AA recurrentes utilizadas en procesos dinámicos [LAWR 00].

El algoritmo GMDH (Group Meted of Data Handling) ha sido propuesto por Ivakhnenko [IVAK 95]. Dicho algoritmo entrena una RNA donde la función de transferencia entre neuronas se modifica por técnicas evolutivas. Es, por tanto, un método que utiliza principios de auto-organización y usa reglas simbólicas para modelizar todo el proceso. Esta aproximación asume que los componentes elementales están predefinidos (modelo base) codificados genéticamente.

Aplicar el algoritmo GMDH para entrenar una RNA limita las posibilidades del usuario en el proceso de extracción de conocimiento haciendo el proceso demasiado automático [LEMK 97]. Las ventajas del algoritmo de auto-organización es que trabaja de una forma muy rápida, sistemática y buscando objetivos concretos. Por el contrario, los algoritmos GMDH sólo se han aplicado para entrenar RR.NN.AA multicapa donde la estructura son polinomios distribuidos, fracasando a la hora de producir reglas lógicas directamente del conjunto de entrenamiento.

3.4.7 Aproximaciones mediante computación evolutiva

De Jong [DEJO 91] propone por vez primera en 1991 un algoritmo basado totalmente en AA.GG utilizando técnicas inductivas. El algoritmo, llamado GABIL, realiza una búsqueda incremental para un conjunto de reglas clasificadoras. Estas reglas están representadas por cadenas de bits de longitud fija. Mientras Opitz [OPIT 94] propone un método de aprendizaje inductivo basado totalmente en AA.GG realmente competitivo. Este método llamado REGENT actúa sobre el proceso de aprendizaje de la RNA modificando su topología para incrementar el conocimiento que reside en ella. El algoritmo REGENT es una variante del método TOPGEN, basado en técnicas

heurísticas del mismo autor [OPIT 93], pero incorporando estrategias evolutivas para mejorar la capacidad de crear conocimiento en el proceso de aprendizaje de las RR.NN.AA.

Otros algoritmos de extracción de conocimiento basados en AA.GG son el REGAL, propuesto por Giordana [GIOR 94], GA-MINER propuesto por Flockhart [FLOC 95] y el SET-GEN propuesto por Cherkauer [CHER 96]. Al igual que los anteriores, utilizan los AA.GG para obtener reglas clasificadoras basadas en expresiones del tipo "IF-THEN-ELSE". Estos algoritmos han sido pensados para técnicas de KDD y ser aplicados sobre grandes BD, por lo que se limitan a obtener reglas genéricas de patrones de comportamiento de los datos. Su aplicabilidad a la extracción de conocimiento de RR.NN.AA se limita, por tanto, a la extracción de reglas simples.

En un capítulo posterior (6.3.2.1) se analizarán los algoritmos evolutivos basados en la PG más recientes como son el LOGENPRO y el BGP. Se presentarán en el capítulo 6 debido a las similitudes con el sistema propuesto en esta tesis.

3.4.8 Aproximaciones mediante inducción simbólica

La técnica denominada DEDEC (Decision Detection by rule extraction from neural networks) es presentada por Tickle [TICK 94]. Dicha técnica es una aproximación genérica a la extracción de reglas de RR.NN.AA entrenadas. Utiliza la capacidad de generalización de la RNA para generar un conjunto de ejemplos del dominio del problema. Estos ejemplos son procesados en un algoritmo de inducción simbólica que extrae un conjunto de reglas que representa una serie de requisitos que deben tener los datos procesados en la RNA. De cualquier forma, la característica más innovadora de esta técnica es que el DEDEC usa información extraída del vector de la matriz de pesos de la RNA para obtener las reglas que deben seguir las variables de entrada. Esta información adicional es usada para centrar el proceso de búsqueda en ciertas zonas delimitadas por las reglas anteriores. Dicho proceso de búsqueda está realizado con técnicas evolutivas utilizando restricciones en el AG propuesto.

Andrews [ANDR 95] desarrolla el "CEBP" (Constrained Error Back-Propagation) aplicable a RR.NN.AA multicapa alimentadas hacia delante, que es capaz de producir aprendizaje inductivo sobre las mismas. Además, introduce el "RULEIN/RULEX" [ANDR 96][ANDR 99] donde propone una técnica automática para refinar un conocimiento base. "RULEIN" construye una "RNA RBP" (Rapid BackPropagation) a partir de un conocimiento inicial que es formulado por una serie de reglas. La "RBP" es entonces entrenada y se aplica el "RULEX" sobre los pesos de la red entrenada para extraer un conjunto de reglas "refinadas" y entendibles por los seres humanos. Las reglas refinadas representan el conocimiento base inicial modificado según el proceso de entrenamiento y de acuerdo al problema a resolver. El algoritmo "RULEX" utiliza reglas "IF-THEN" para representar el conocimiento extraído.

Estos algoritmos están desarrollados sobre RR.NN.AA alimentadas sólo hacia delante y con comportamientos especiales. Las neuronas usan respuestas locales combinadas con funciones sigmoide conformando funciones de activación en forma de cresta. La forma de entrenar estas RR.NN.AA es mediante "Backpropagation" restringido manteniendo algunos pesos fijos. Por lo tanto, no sólo depende de la arquitectura de las redes sino también de su entrenamiento.

3.5 Algoritmos de extracción existentes más importantes

A continuación se va a exponer una serie de los algoritmos desarrollados en la teoría de extracción de reglas de RR.NN.AA. La mayoría de ellos ya han sido expuestos en los puntos precedentes. Asociado a cada algoritmo se indican los autores que inicialmente lo han propuesto.

- ✓ **Métodos Inductivos:** uno de los pioneros en proponer un algoritmo inductivo ha sido Michalski en 1983 [MICH 83]
- ✓ **CART:** (Classification and Regresión Tree) propuesto por Breiman et al [BREI 84]
- ✓ **PVM:** propuesto por Weiss [WEIS 90]

- ✓ **GABIL:** propuesto por De Jong et al [DEJO 91]
- ✓ **KT:** propuesto por Fu [FU 91] [FU 94]
- ✓ **RULENET:** propuesto por McMillan et al en 1992 [MCMI 92] y modificado posteriormente por Alexander et al en 1995 [ALEX 95]
- ✓ **C 4.5:** propuesto por Quinlan [QUIN 93] y posteriormente mejorado por Zheng [ZHEN 98]
- ✓ **Método M-N:** propuesto por Towell et al [TOWE 93]. Recientemente ha sido redefinido por Setiono [SETI 02] proponiendo el algoritmo M-N3
- ✓ **SUBSET:** propuesto por Towell et al [TOWE 93]
- ✓ **BRAINNE:** propuesto por Sestito et al [SEST 94]
- ✓ **DEDEC:** propuesto por Tickle et al [TICK 94]
- ✓ **OC** (Oblique Classifier) y variantes CART-LC, CART-AP, OC-1, OC-LP, OC-1AP ..., propuesto inicialmente por Murthy et al [MURT 94].
- ✓ **REAL:** (Rule Extraction As Learning) propuesto por Craven et al [CRAV 94]
- ✓ **REGENT:** propuesto por Opitz et al [OPIT 94]
- ✓ **REGAL:** propuesto por Giordana et al [GIOR 94]
- ✓ **RISE:** (Rule Induction from a Set of Exemplars) propuesto por Domingos [DOMI 94]
- ✓ **RULENEG:** propuesto por Pop et al [POP 94]
- ✓ **RULEX:** propuesto por Andrews et al [ANDR 94]
- ✓ **GA-MINER:** propuesto por Flockhart et al [FLOC 95]
- ✓ **CEBP:** propuesto por Andrews et al [ANDR 95]
- ✓ **GMDH:** propuesto por Ivakhnenko et al [IVAK 95]

- ✓ **VIA:** (Validity Interval Análisis) propuesto por Thrun [THRU 95]
- ✓ **BBP:** propuesto por Jackson et al [JACK 96]
- ✓ **COMBO:** propuesto por Krishnan et al [KRIS 96]
- ✓ **SET-GEN:** propuesto Cherkauer et al [CHER 96]
- ✓ **TREPAN:** propuesto por Craven et al [CRAV 96]
- ✓ **Successive Regularization:** propuesto por Ishikawa [ISHI 96]
- ✓ **NEUROLINEAR:** propuesto por Setiono et al [SETI 97]
- ✓ **ANN-DT:** (Decisión Trees from Neural Networks) propuesto por Schmitz et al [SCHM 99]
- ✓ **C-MLP2LN:** propuesto por Duch et al [DUCH 00]
- ✓ **FERNN:** propuesto por Setiono et al [SETI 00]
- ✓ **GANN:** propuesto por Keedwell et al [KEED 00]
- ✓ **LRA:** propuesto por Tsukimoto [TSUK 00]
- ✓ **LOGENPRO:** propuesto por Wong et al [WONG 00]
- ✓ **BGP:** propuesto por Engelbrecht et al [ENGE 01]

3.6 Consideraciones

Es importante resaltar que la mayoría de las técnicas y algoritmos presentados en este capítulo para extraer reglas de RR.NN.AA son específicos de una implementación o de una arquitectura de RNA. Aunque la técnica "decompositional", según la clasificación establecida por Andrews [ANDR 95], y el algoritmo "KT" [FU 91] han tenido un gran éxito en los inicios de la teoría de extracción de conocimiento, la técnica "pedagogical" como los algoritmos "VIA" [THRU 95] y "REAL" [CRAV 94] y la técnica "eclectic" como el algoritmo "DEDEC" [TICK 94], pronto se han ido imponiendo a las anteriores. Sin embargo, se seguía centrando la elicitación de conocimiento sobre RR.NN.AA con una arquitectura y estructura muy restrictiva. Más

concretamente, se han desarrollado algoritmos de extracción específicos de una única determinada arquitectura de RNA, como ejemplo de ello se puede citar el "RULEX" [ANDR 94], "M-N" [TOWE 93] y "BRAINNE" [SEST 94]. Posteriormente, se han desarrollado algunos algoritmos que han pretendido aplicarse a un amplio espectro de implementaciones y arquitecturas de RR.NN.AA combinando técnicas "pedagogical" y "eclectic" como el algoritmo "TREPAN" [CRAV 96].

Muchos algoritmos utilizan la técnica de las reglas difusas y algunos presentan unos resultados ciertamente prometedores [HALG 94][DUCH 95][ZURA 96][NAUC 96][NAUC 97]; sin embargo, tienden a una excesiva parametrización provocando una excesiva dificultad de encontrar las soluciones óptimas [KASA 96][KASA 98], incluso con la ayuda de los AA.GG u otros métodos de optimización. Muchos métodos de extracción han sido probados con conjuntos de datos bastante raros, por lo que sus ventajas son difíciles de catalogar. La mayoría de los artículos relativos a la materia de extracción de reglas se limitan, básicamente, a la descripción de nuevos algoritmos presentando solamente una solución parcial al problema de la extracción del conocimiento. El control del equilibrio entre la comprensibilidad y la exactitud, entre la optimización de las variables lingüísticas y de las reglas finales, y la estimación de la fiabilidad de las reglas obtenidas casi nunca se han discutido. En aplicaciones prácticas puede ser muy útil tener exactitud, alta o baja, siempre determinada por el usuario. También es muy útil poder tener un nivel de descripción simple de los datos, o poder proporcionar una descripción más exacta, pero de igual manera controlada por el usuario.

La gran mayoría de los algoritmos expuestos en este capítulo tratan la elicitación de conocimiento como técnicas aplicadas a datos. Esto es debido a que están concebidos para trabajar con BB.DD y partiendo de estos datos buscar relaciones entre ellos. Es lo que se conoce como KDD o "data mining". Todos estos algoritmos deben ser altamente eficientes y poder trabajar con ingentes cantidades de datos sin grandes costos computacionales. Es por ello que el conocimiento que puedan extraer se limita a simples relaciones debidas a clasificaciones y detección de patrones de comportamiento.

De ahí la constante aparición de técnicas que se centran en árboles de decisión y en reglas difusas. En la gran mayoría de los casos se centran en obtener reglas del tipo "IF-THEN-ELSE". Por ello, se vuelve necesario la existencia de una metodología que trate el desarrollo de los algoritmos de extracción de una forma clara y concisa, determinando de antemano la aplicabilidad del algoritmo a desarrollar para poder comparar los logros que pueda conseguir con otros existentes y que sean aplicables a casos similares. La falta de una metodología ha dado lugar a la existencia de trabajos de investigación que resuelven la tarea de extracción de conocimiento de una manera poco clara y centrados en aspectos muy particulares de la misma. De esta forma, se dificulta enormemente la elaboración de estudios comparativos entre los algoritmos desarrollados y, por consiguiente, su posible aplicabilidad por otros investigadores y desarrolladores.

El objetivo de la metodología que se presentará en los capítulos posteriores será el establecer de forma clara y evidente las diferentes fases por las que un investigador debería guiarse para desarrollar un algoritmo de extracción de conocimiento, ya sea sobre RR.NN.AA o sobre datos procedentes de BB.DD. Además, se expondrá un desarrollo de un algoritmo siguiendo las diferentes fases propuestas que sea comparable a otros desarrollados y donde se procure evitar la principal carencia vista hasta el momento de los algoritmos existentes: **"la limitación a tareas clasificadoras"**. El algoritmo desarrollado y propuesto en los siguientes capítulos trata la tarea de extracción de conocimiento como un proceso en el que la complejidad de las reglas obtenidas las define el usuario y, por consiguiente, es aplicable a tareas donde se requiera una alta eficiencia en el proceso (búsqueda de patrones en BB.DD) y también es aplicable a donde se requiera un gran nivel de extracción, como en el caso de la búsqueda de conocimiento residente en las RR.NN.AA.

Una vez sentadas las bases de cómo funcionan y qué ventajas e inconvenientes tienen las RR.NN.AA se han analizado diferentes disciplinas de estudio relacionadas con la tarea que trata la presente tesis. En primer lugar, es preciso disponer de una metodología de trabajo, aportada en este caso por la Identificación de Sistemas.

Posteriormente, se han analizado diferentes técnicas existentes en otra materia de estudio similar a la tratada en esta tesis, la minería de datos, de donde se han podido entresacar aquellas técnicas más prometedoras y comprobar que las basadas en métodos evolutivos son las que mejor encajan en el ejemplo de desarrollo presentado en esta tesis, la PG. Una vez determinado que es una de las más idóneas para el presente estudio, se han indicado sus líneas de funcionamiento básicas. Como se verá en los capítulos siguientes, se han aplicado estos conceptos aquí vistos para el análisis, definición y desarrollo de la metodología y sistema expuestos en la presente tesis.

"El Conocimiento y el poder humano son sinónimos"

Francis Bacon

CAPÍTULO

4 HIPÓTESIS DE TRABAJO

4.1 Ámbito de las RR.NN.AA

Hay algunas áreas en las que se han aplicado las RR.NN.AA con éxito: en el análisis financiero, el procesamiento de señales, en la automatización y robótica, en los diagnósticos médicos [JABR 92], en problemas de clasificación, en el reconocimiento de patrones, en el control de procesos, en la optimización, etc. Pero las RR.NN.AA, de momento no son capaces o eficientes para resolver todo tipo de problemas y, por tanto, no pretenden ser el sustituto de las computadoras convencionales; por ejemplo no serían adecuadas para un sistema de gestión de empleados.

Dependiendo del área de aplicación, es conveniente aplicar una arquitectura u otra. Así, para los problemas de clasificación se aplicarán arquitecturas alimentadas solo hacia delante y para procesamiento de señales temporales es preferible la aplicación de arquitecturas recurrentes.

Nace entonces la inquietud de saber: ¿las RR.NN.AA, en qué tipo de aplicaciones trabajan adecuadamente y en dónde su aplicación es de especial relevancia?, y por ende

¿cuáles son los alcances y limitaciones de las RR.NN.AA?. La respuesta es que trabajan con éxito en tareas donde no hay reglas bien definidas las cuales parecen fáciles para los humanos y difíciles para las computadoras. Este campo es tan amplio como el de las computadoras convencionales. La IA ha estado generalmente dominada por las áreas de manipulación lógica y simbólica, pero algunos piensan que las RR.NN.AA reemplazarán la IA tradicional actual. Mas bien parece que se combinarán en sistemas apoyándose mutuamente, como sucede en los seres humanos, pues éstos se apoyan en sistemas rápidos soportados por reconocimiento de patrones. Otros sistemas realizan actividades que requieren más tiempo cuando el reconocimiento falla o cuando se requieren niveles superiores de decisión [ZURA 92][CAST 00].

Pero el problema más importante y que ha conllevado multitud de críticas a las RR.NN.AA es su falta de habilidad para "explicar" cómo resuelven el problema. Esta falta de habilidad se debe a que la representación interna generada en la red puede ser demasiado compleja aún en los casos más sencillos [ZURA 92][CAST 00]. Por tanto, se hacen necesarios los métodos que traten la extracción de reglas de las RR.NN.AA, y además, tratando las RR.NN.AA como sistemas independientes de su arquitectura, entrenamiento y distribución interna de pesos, conexiones y funciones de activación. Y aún más necesaria una metodología para el diseño de estos métodos.

4.2 Hipótesis

El primer paso, antes de afrontar la problemática del desarrollo de un sistema de explicación del conocimiento en RR.NN.AA, es establecer una serie de conceptos sobre la forma y la metodología de trabajo de las RR.NN.AA. Para ello, todo debe girar sobre el concepto de aprendizaje y la forma en la que las RR.NN.AA afrontan la resolución de problemas.

El estudio del aprendizaje desde siempre ha constituido uno de los temas fundamentales de la IA. Se pretende tanto comprender los procesos de aprendizaje como su implantación en sistemas artificiales. Los computadores actuales tienen muchas limitaciones en cuanto a aprender a hacer una tarea basándose en ejemplos o en

la analogía con tareas similares previamente resueltas y adquirir nuevas habilidades observando e imitando a los expertos.

La investigación sobre el aprendizaje, dentro de la IA, busca dotar a las máquinas de estas capacidades para facilitar la programación haciéndola automática. Una de las técnicas de la IA que ha sido creada para tal efecto son los SS.EE.

Una de las tareas claves de los SS.EE es la adquisición de conocimiento y la dificultad para expresar de forma adecuada y precisa este conocimiento adquirido porque, como ya expresó el filósofo Claude Bernard: "Podemos más de lo que sabemos, sabemos más de lo que comprendemos y comprendemos más de lo que podemos explicar". Es decir, para codificar el conocimiento es necesario poder explicarlo de forma apropiada y precisa en el entorno adecuado.

***Fig. 14** - Ámbito del conocimiento adquirido*

La dificultad de poder poner los conocimientos de forma explícita y la posibilidad de que la máquina pueda adquirir directamente conocimientos que están implícitos en la solución de casos anteriores, y no formulados explícitamente, es sumamente atractiva para el desarrollo de Sistemas Basados en Conocimiento (SBC).

El proceso de aprendizaje se puede considerar como la adquisición de conocimiento de o sobre algo. Y se pueden distinguir dos formas básicas de aprendizaje [GOME 97]:

- Adquisición de conocimiento: adquisición de conceptos, sus significados, sus relaciones entre sí y con el mundo externo y modelos de comportamiento. Puede darse en multitud de representaciones, desde modelos intuitivos, imágenes y ejemplos, hasta ecuaciones matemáticas y leyes. Por tanto, se adquiere un conocimiento que explica un conjunto amplio de situaciones.
- Refinamiento de habilidades: mejora gradual y progresiva de habilidades. Por ejemplo, aprender a conducir o detectar situaciones peligrosas. En estos casos el proceso de aprendizaje se realiza mediante la práctica repetida de situaciones y la corrección de desviaciones respecto al comportamiento deseado.

La investigación clásica de la IA ha estado más dirigida al proceso simbólico, y por esto puede parecerse más al primer tipo de aprendizaje. En cambio, el segundo tipo ha conllevado al desarrollo de modelos no simbólicos como las RR.NN.AA.

Por tanto, paralelamente con la investigación típica de la IA, que se centra en el desarrollo de modelos conceptuales y simbólicos, se han desarrollado sistemas análogos a las neuronas y sus redes que resultan particularmente aptos para procesos de aprendizaje del tipo "mejora de habilidades" o "reconocimiento de patrones"

Como se ha comentado más en detalle en el capítulo 2, el proceso de aprendizaje de una RNA consiste en un "mapeado" continuo desde un espacio n-dimensional (entradas) a uno m-dimensional (salidas), basándose en ejemplos de la acción del mapeado.

Por el contrario, el proceso simbólico trata el aprendizaje como una creación y manipulación de símbolos.

Algunas técnicas de aprendizaje que utilizan el modelo simbólico son:

- Modificación de descripciones de conceptos u objetos: las descripciones están representadas como redes de símbolos

interrelacionados y el aprendizaje se hace modificando las conexiones o interrelaciones entre los símbolos.

- Modificación de los diversos caminos de un árbol de decisión o búsqueda según las experiencias que ha tenido el sistema.
- Modificaciones en las representaciones basadas en reglas. Adición de nuevas reglas deducidas de casos resueltos, modificación de reglas existentes (generalización, especialización y combinación de reglas) y modificación del uso de las reglas incorporándolas como Metareglas o reglas sobre como usar las reglas del dominio.

Cuando se usan RR.NN.AA el dominio y la tarea no requieren ser tan bien comprendidos como cuando se utilizan los SS.EE. No se requiere que el experto detalle cómo llega a una solución ya que el sistema se adapta en base a las relaciones estímulo/respuesta, reconfigurándose para resolver el problema. Pero la principal ventaja de usar las RR.NN.AA para emular los SS.EE es que no requiere algoritmos específicos del dominio, así como que es mucho más tolerante a fallos (por ejemplo, si falla una conexión) que los SS.EE tradicionales. En cambio, la principal desventaja es que no se tiene manera de examinar los conocimientos del sistema ya que es una especie de "caja negra" en la cual sólo se puede confiar por su comportamiento entrada/salida.

La IA dispone de tareas de aprendizaje de conocimiento basadas en objetos simbólicos. Dichas tareas son la deducción y la inducción. Mediante la deducción se parte de reglas generales y se obtienen casos particulares. Sin embargo, la inducción trabaja en sentido contrario, desde casos particulares hacia reglas generales. Es decir, partiendo de muestras de casos inducir reglas [BALD 00].

La tarea de inducción consiste pues en hallar una regla de clasificación o decisión a partir de un conjunto de casos de ejemplo de entrenamiento, donde cada uno de ellos posee un conjunto de valores de ciertas características y de un conjunto de clases que representan las categorías en que un "experto" clasifica cada caso.

Esta tarea de inducción es realizada por un algoritmo inductivo y la calidad de lo inducido depende tanto de dicho algoritmo como de la riqueza y la representatividad del conjunto de datos de ejemplo.

4.2.1 Expectativas de desarrollo

Según lo visto anteriormente, los algoritmos inductivos se diferencian entre sí por la manera en la cual exploran en el conjunto de datos de ejemplo, por la manera cómo generalizan y por la manera cómo se enfrentan a los errores y el ruido en los casos entrenados. Algunos algoritmos se diseñan pensando en un dominio específico y otros pretenden ser de aplicación general. Estos últimos pueden resultar más ineficientes y producir resultados menos espectaculares, pero sirven para todos los casos de problemas.

En definitiva, lo que se desea es poder desarrollar una metodología para el diseño de algoritmos inductivos de aplicación general sobre RR.NN.AA, independiente de su estructura y arquitectura interna. Para ello, aplicando la metodología de aplicación global procedente de la disciplina de IS explicada en detalle en el capítulo 2, adaptarla y reestructurarla para su aplicación al entorno Conexionista, y más concretamente a las RR.NN.AA.

Hasta ahora se han desarrollado numerosas técnicas de extracción de conocimiento y de extracción de reglas de RR.NN.AA, pero ninguna establece una metodología de operación ni son consecuencia directa de la aplicación de alguna metodología existente. Es, por tanto, un punto de especial importancia el desarrollo de una metodología nueva que aporte formalismo y disciplina a todo el proceso y rigor científico al establecimiento de pruebas del sistema obtenido.

La necesidad de una metodología para los procesos de extracción de conocimiento es tan necesaria como cualquier desarrollo informático de cierta relevancia. La falta de una clara metodología da lugar a la existencia de múltiples algoritmos que hacen énfasis en múltiples facetas de la extracción de reglas. Desde los primeros sistemas de extracción se han tomado dos líneas claramente diferenciadas. Por un lado se han

desarrollado algoritmos para la elicitación de conocimiento de grandes cantidades de datos almacenados en BB.DD en los que la tarea fundamental es la búsqueda de relaciones entre los mismos expresados en forma de reglas simbólicas. Esta línea ha desembocado en la teoría denominada "KDD", "data mining" o "minería de datos". Por otro lado, se han desarrollado algoritmos para extraer reglas que simulen el comportamiento de RR.NN.AA. Dentro de estos algoritmos se diferencian varias técnicas que fundamentalmente se pueden clasificar en dos:

1. Las técnicas que analizan los diferentes elementos de proceso (EP) y su interrelación con los demás para producir reglas. Los algoritmos que aplican esta técnica suelen resultar muy eficientes pero tienen el problema de la gran dependencia del tipo de arquitectura de la RNA, número de neuronas, tipo de las mismas, e incluso algoritmo de entrenamiento.
2. Las técnicas que tratan las RR.NN.AA como "cajas negras" donde, a partir de una serie de entradas y sus correspondientes salidas analizan las relaciones existentes y producen reglas que simulan el comportamiento de dicha "caja negra".

Esta última línea de desarrollo de extracción de reglas de RR.NN.AA es la que hoy en día está más en auge debido a que se pueden desarrollar algoritmos que sean independientes de la arquitectura de las RR.NN.AA. Además, esta línea de desarrollo coincide en cierta manera con la forma de trabajar de los algoritmos de KDD, buscando, en definitiva, relaciones entre los datos. Sin embargo, los algoritmos de KDD al trabajar con grandes volúmenes de datos deben ser computacionalmente rápidos para poder procesar en tiempos razonables el máximo número de datos. Esto supone que el nivel de complejidad de las reglas buscadas es mínimo. Por el contrario, si se trata las RR.NN.AA, se desea que las reglas emulen lo más fielmente posible su comportamiento por lo que los algoritmos deben realizar búsquedas complejas de reglas y expresiones.

Un importante objetivo de la metodología a desarrollar debe ser unificar los desarrollos de algoritmos que agrupen ambas categorías: **"eficientes versus**

complejos". Con esto en mente se pretende desarrollar una metodología aplicable a ambas facetas, la minería de datos y la extracción de reglas de RR.NN.AA, técnicas que han seguido caminos parejos pero separados y que en la presente tesis se pretende que sigan un camino común aprovechando las ventajas que ambas pueden proporcionar.

Un algoritmo ideal de extracción de conocimiento, por tanto, debe poder balancear los niveles de complejidad y eficiencia al tratar la exploración de los datos y las reglas obtenidas de los mismos.

4.2.2 Estudio de metodologías

Antes de afrontar el desarrollo de una metodología y sus etapas es necesario conocer las características más relevantes de aquellas desarrolladas hasta el momento.

En el ámbito de la ingeniería informática existen básicamente dos grandes grupos de metodologías:

1. Metodologías de ingeniería del software
2. Metodologías de ingeniería del conocimiento

Las metodologías desarrolladas para la ingeniería del software se centran en el desarrollo de un producto software que cumpla una serie de requisitos del cliente que lo solicita. Un desarrollo metodológico en ingeniería exige la definición y estandarización de un ciclo de vida que va desde la especificación de requisitos al mantenimiento del producto final. Esta exigencia, perfectamente definida en la ingeniería del software, presenta lagunas en la ingeniería del conocimiento debido a que en este campo muchos de los problemas a resolver presentan un difícil análisis de los requisitos "a priori". Por ello, se han desarrollado metodologías específicas para esta rama de la ingeniería.

La metodología que se pretende desarrollar está en el ámbito de la IA, ya que se aplicaría al desarrollo de algoritmos basados en técnicas de IA como son las RR.NN.AA, la inducción y las reglas simbólicas.

Algunas de las principales metodologías desarrolladas en la IA [GOME 97] son la ICOT desarrollada en 1986, la POLITE desarrollada en 1991, la KADS desarrollada en

1992 y la más interesante desde nuestro punto de vista de todas, desarrollada en 1995, denominada IDEAL. Esta metodología consta de cuatro etapas fundamentales:

1. Identificación del problema
2. Desarrollo de prototipos y del sistema completo
3. Transferencia de tecnología
4. Mantenimiento

Estas metodologías están desarrolladas con el objetivo de la construcción de un SE, por lo que existen las etapas de transferencia de tecnología y de mantenimiento. Lo que se trata en esta tesis no está centrado en los SS.EE por lo que estas últimas fases de la metodología no son precisas para el desarrollo del presente estudio.

Sin embargo en la Identificación de Sistemas se tratan más adecuadamente los objetivos del desarrollo de sistemas y relaciones funcionales entre patrones de datos de un proceso. Los usos de esta relación funcional son la simulación y el control del proceso identificado.

Básicamente, las metodologías de simulación aplican las siguientes fases:

1. Realización de experimentos
2. Desarrollo del modelo
3. Calibración del modelo
4. Verificación del modelo
5. Validación del modelo

Concretamente, Norton [NORT 86] y posteriormente Söderström y Stoica [SÖDE 89] definen una serie de etapas que son consideradas dentro del área como la metodología de aplicación genérica en todo proceso de IS y que han sido expuestas en el capítulo 2.2. Partiendo de estas, se desarrollará la presente metodología adaptándola al proceso de inducción simbólica y a las RR.NN.AA.

En definitiva, la metodología a desarrollar será una mezcla entre las metodologías de ingeniería de conocimiento y las que se pueden denominar metodologías de simulación donde se trata de elaborar modelos a partir de la realización de experimentos.

"Nada puede ser más amargo
que no ser comprendido"

Herik Ibsen

CAPÍTULO

5 METODOLOGÍA

5.1 Premisas

Como se ha comentado en el capítulo 1, en un sistema de extracción de reglas ideal es conveniente que se cumplan una serie de cuatro características fundamentales: ser **independiente de la arquitectura** de la RNA, ser **independiente del algoritmo de entrenamiento** de RNA, ser **correcto** y tener un **alto nivel expresivo**. Ciñéndose a estas características, según las dos primeras el sistema debe poder tratar la RNA como una estructura de "caja negra" y, por lo tanto, debe ser un algoritmo de aplicación general. Según las dos últimas, el algoritmo debe poder trabajar con símbolos de alto poder representativo de conocimiento expresado en forma de reglas. Estas características hacen que el algoritmo deba utilizar modelos simbólicos.

Con estas características en mente, y basándose en la metodología de IS, se debe afrontar la adaptación de las diferentes etapas de dicha metodología.

5.2 Establecimiento de las etapas de la metodología

A continuación, se van a exponer las diferentes etapas de las que consta la metodología propuesta. Que, como se ha comentado anteriormente, se basa en la metodología de IS.

5.2.1 Diseño del experimento y Conocimiento previo

Esta es la primera etapa y una de las más importantes, pues según la "calidad" de los datos seleccionados para la experimentación y las pruebas, dependerán los resultados finales. Cuando se aplica el proceso de aprendizaje supervisado a una RNA se está construyendo un conjunto de datos que constituyen los ficheros de entrenamiento. Según el nivel de aprendizaje la RNA, después del proceso de entrenamiento, habrá obtenido un conocimiento de esos datos y los almacenará de forma distribuida en sus conexiones.

Por tanto, estos ficheros de entrenamiento deben constituir una referencia fundamental en el proceso de extracción de reglas de la RNA, ya que de ellos obtienen el conocimiento y, lo que es más importante, **la capacidad de generalización**. Ésta es una de las características más importantes de la RNA y que le ha dado todos sus éxitos en el mundo real.

Como conclusión de esta primera etapa, lo fundamental es poder obtener los ficheros que han sido utilizados por el proceso de entrenamiento de la RNA. En caso de no ser posible, habría que construirlos basándose en la teoría de aprendizaje de RR.NN.AA vista en el capítulo 2.

En el caso de no tener los ficheros originales de entrenamiento de la RNA, y suponiendo que se conoce la tarea para la que se ha desarrollado la RNA, construir unos conjuntos de entrenamiento siguiendo las tareas de un "diseñador de RR.NN.AA", seleccionando aquellos ejemplos que reflejen claramente cada uno de los patrones a reconocer y las situaciones extremas del problema que resuelve. Se puede ver en más detalle en el capítulo 2.

Otra tarea a realizar en esta etapa es recavar toda la información posible acerca de la RNA, tipos de datos con los que trabaja, si son en modo continuo o en modo discreto, si las entradas son lógicas (TRUE, FALSE) o valores en punto flotante, etc. Dependiendo de la cantidad de información que se tenga del funcionamiento de la red se podrá ajustar más o menos el espacio de búsqueda del algoritmo de extracción. Así, por ejemplo, si las entradas y las salidas de la red son lógicas se evita el uso de operaciones matemáticas y trigonométricas complejas en la simulación, lo cuál conlleva una considerable reducción del espacio de búsqueda, y por ende del tiempo de extracción de reglas.

Esta etapa es una agrupación de las etapas 1, 2 y de 3 de la metodología de IS. Como se puede comprobar la etapa 3 de IS es el análisis y tratamiento de los datos, agrupándolos en estructuras de "caja negra", "caja blanca" y "caja gris". Sin embargo, en el caso que nos atañe, la estructura siempre es de "caja gris" ya que se tiene un sistema a modelar que es una RNA y de ella es posible obtener conocimiento "a priori" como son el tipo de datos y estructura de los mismos (dinámicos o estáticos), etc. Por ello, se hace esta agrupación y simplificación de etapas.

5.2.2 Selección del modelo o algoritmo de extracción

Partiendo de las premisas o características presentadas anteriormente se debería diseñar un algoritmo inductivo de aplicación general. Además, el proceso de extracción de conocimiento, esquemáticamente representado en la figura 15, se centraría en inducir reglas a partir de patrones entrada/salida de la RNA.

Todos los métodos de extracción de reglas existentes basan su funcionamiento, en cierto modo, en lo representado en el diagrama de la figura 15. Utilizando secuencias fijas de patrones entrada-salida o la estructura propia de la RNA extraer reglas por unos u otros métodos. Por tanto, lo que en definitiva se busca es extraer conocimiento de datos. Pero se puede ir más allá. Como primer paso se puede utilizar como conjunto de ejemplos inicial los ficheros de entrenamiento de la RNA y, posteriormente, gracias a la capacidad de generalización, obtener nuevos patrones a partir de la creación de nuevas

secuencias de entradas. Estas se presentan a la RNA y las salidas que produce permiten extraer reglas de dicha capacidad de generalización. Es por tanto un proceso iterativo auto-realimentado.

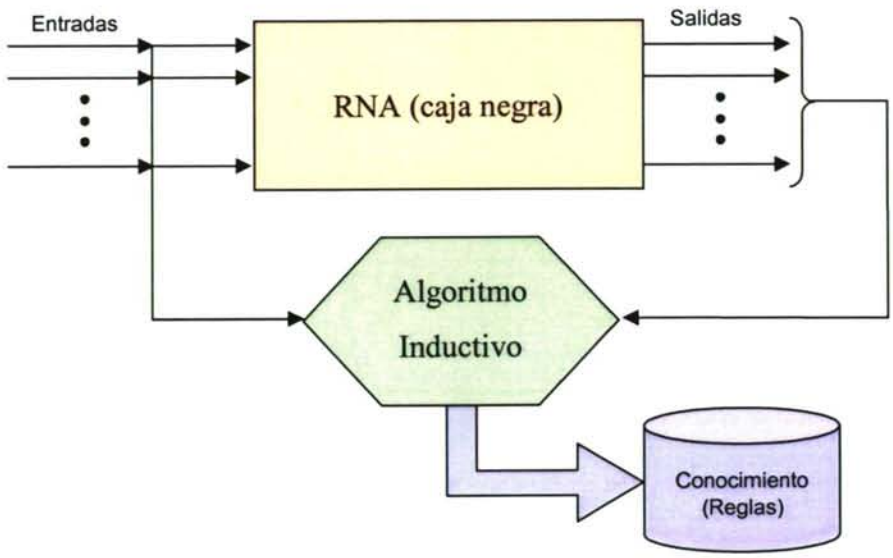


***Fig. 15** - Diagrama del proceso de Extracción de Reglas*

De esta forma, se puede plantear el desarrollo de un sistema básico donde partiendo de un conjunto de datos fijo (por ejemplo los ficheros de entrenamiento de la RNA) obtener las reglas correspondientes. En este caso la forma de proceder es darle a la RNA como entradas el fichero de entradas del entrenamiento, obtener las salidas que produce la RNA y sobre este patrón "entradas-entrenamiento / salidas-RNA" descubrir las reglas por inducción. En este caso el nivel de incertidumbre de las reglas obtenidas está limitado a la "calidad" del algoritmo de inducción y a los datos seleccionados. Para reducir el nivel de incertidumbre de las reglas obtenidas es necesario, entonces, dotar al sistema de un **algoritmo de creación de ejemplos**, donde en función de ciertos parámetros se puedan variar los valores de las entradas a la RNA. Para ello, se puede recurrir una vez más a la disciplina de IS que dispone de múltiples métodos para realizar esta tarea. Algunos de estos métodos se han podido ver en el apartado correspondiente del capítulo 2.

Otro punto interesante es poder incorporar el conocimiento extraído del "experto" al algoritmo inductivo como conocimiento "a priori" o innato. Esto es, si se conocen algunos datos de cómo está estructurada la RNA, del tipo de datos que va a tratar, del tipo de problema que va a resolver, etc., este conocimiento se puede incorporar al algoritmo inductivo como MetaReglas. Dicho conocimiento del experto se ha obtenido de la etapa anterior. De esta forma se consigue reducir el espacio de búsqueda del algoritmo inductivo en mayor o menor medida a la cantidad de conocimiento de la RNA y del tipo de problema que trata. Así, por ejemplo, si se sabe que la RNA trabaja exclusivamente sobre valores de entradas booleanas se evita, en el algoritmo inductivo, el uso de simbología que trate valores continuos, con la consiguiente reducción del espacio de búsqueda que el algoritmo necesita explorar.

5.2.2.1 Consideraciones

Se propone un proceso de extracción de reglas donde el núcleo principal sea el algoritmo inductivo y se incorporen mecanismos que permitan la obtención de las necesidades vistas anteriormente. En el diagrama de la figura 16 se puede ver que el núcleo del sistema de extracción de reglas radica en el algoritmo inductivo, éste algoritmo debe ser de carácter general, debe permitir el uso de simbología, debe funcionar por extracción de conocimiento de ejemplos, debe proporcionar un conjunto de reglas con alto nivel expresivo y debe poder incorporar conocimiento "a priori".

El hecho de trabajar sobre un conjunto de ejemplos hace pensar, por similitud, que éste algoritmo se asemeja a la técnica de "Data Mining" o "Knowledge Discovery in Databases" (KDD). El "KDD" es un proceso iterativo en el que se van produciendo progresivos descubrimientos y cuantificaciones de relaciones predictivas en los datos y se permite transformar la información disponible en conocimiento útil. Como se ha visto más en detalle en el capítulo 2 parece lógico pensar que se pueden aplicar las técnicas existentes en el KDD como posibles algoritmos inductivos a utilizar en el sistema de extracción de reglas.

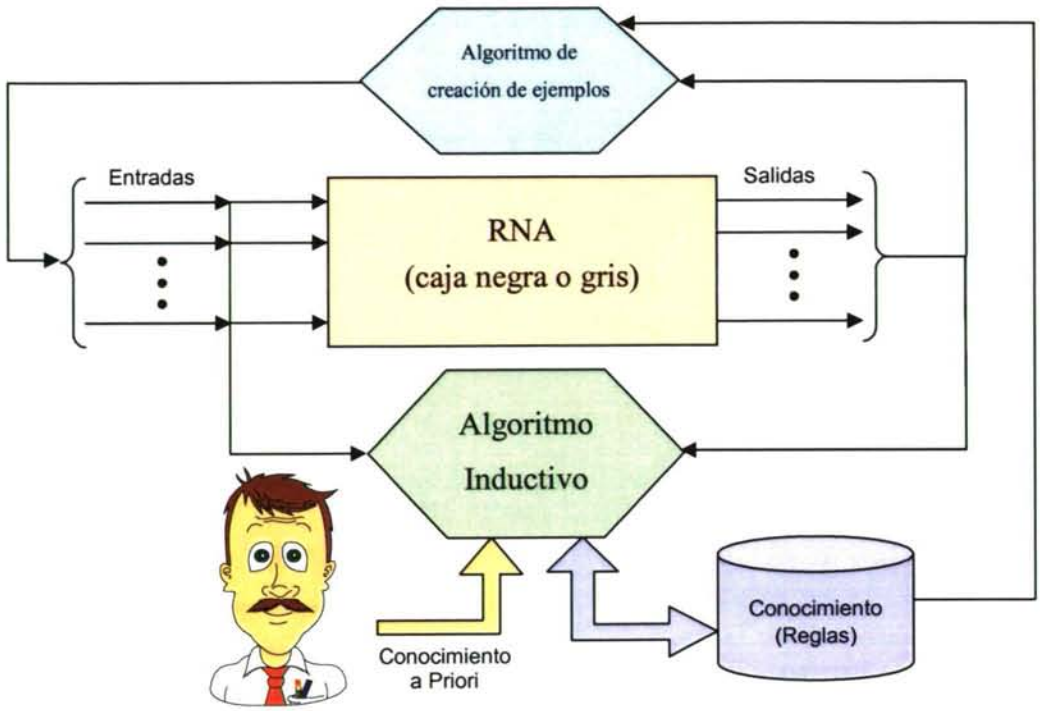


***Fig. 16** - Diagrama del proceso de Extracción de Reglas mejorado*

5.2.3 Estimación de parámetros

Una vez desarrollado e implementado el algoritmo inductivo, la siguiente etapa es determinar y cuantificar los parámetros que intervienen en el algoritmo. Para ello es necesario establecer mecanismos de obtención de los valores numéricos de estos parámetros. Esta estimación de valores numéricos se puede realizar de diferentes formas, y siempre dependiendo del tipo de parámetros y de la estructura del algoritmo:

1. **Métodos numéricos iterativos:** según el algoritmo utilizado y los parámetros involucrados se pueden utilizar métodos matemáticos para la obtención de los mejores valores de los parámetros. Pero esto puede suponer que no sean ni lineales ni convexos provocando problemas de convergencia y de mínimos locales.
2. **Analítica:** con RR.NN.AA siempre se podrán estimar los valores numéricos de los parámetros mediante experimentación sobre casos reales o casos simulados. Así, cuando los métodos numéricos sean

impracticables o no produzcan resultados satisfactorios se podrá recurrir al diseño de casos de prueba para la estimación de los mejores parámetros.

5.2.4 Diseño del algoritmo de creación de ejemplos

Esta etapa es nueva y deriva de la anterior. En el diagrama de la figura 16 se puede observar que existe un módulo que es el algoritmo de creación de ejemplos. Lo que debe realizar este módulo es crear de forma dinámica secuencias de entradas nuevas para RR.NN.AA. El objetivo es explorar nuevas zonas del espacio de estados en la búsqueda de reglas para poder comprobar el funcionamiento de la RNA con nuevos patrones de datos. El fin último es poder "refinar" las reglas obtenidas mediante nuevas secuencias de patrones y poder extraer la capacidad de **generalización** de la RNA. Por lo tanto, se incrementa la complejidad del sistema global añadiendo la importante tarea de abstraer todo el conocimiento interno de la RNA.

Se debe diseñar un algoritmo que tenga las siguientes características:

1. **No debe alterar el algoritmo inductivo:** pues en caso contrario la etapa anterior no sería válida y habría que diseñar un algoritmo en conjunto.
2. **No ser destructivo:** al manipular las secuencias de ejemplos de la RNA, los patrones de datos pueden homogeneizar los resultados obtenidos y el algoritmo inductivo eliminar reglas esenciales y caer en mínimos locales.
3. **Mantener una creación equilibrada:** debe mantener un equilibrio entre el número de patrones creado y el proceso iterativo del algoritmo inductivo. Esto es, no incrementar ni decrementar excesivamente el número total de patrones con el que va a tratar el algoritmo inductivo. Si, por ejemplo, se crean excesivas secuencias de patrones, el algoritmo inductivo puede no realizar suficientes iteraciones para extraer reglas y todo el proceso se puede estancar. Sin embargo, si se crean muy pocos patrones puede que no exista suficiente información para extraer reglas.

Estas características aquí expuestas han sido fruto de la experimentación y de los problemas encontrados cuando se ha diseñado el algoritmo de creación presentado en el capítulo 5. Como se verá en este capítulo, las anteriores características se vuelven fundamentales cuando se aplica un algoritmo realimentado como el expuesto en el diagrama de la figura 16.

5.2.5 Validación del modelo

Primero hay que proponer y realizar nuevos juegos de ensayos para comprobar el correcto funcionamiento del sistema modelado. Con esto se obtiene una medida de la fiabilidad del sistema. Típicamente en el proceso de entrenamiento de RNA se reserva una serie de conjuntos de patrones entrada-salida para la fase de test. Esto es concretamente lo que se propone realizar en esta etapa, pero aquí estos patrones de test se aplicarán a las reglas obtenidas y se comprobará si los resultados que producen estas reglas se corresponde con los resultados que produce la RNA. Con esta comparación de resultados se deben tomar varias decisiones:

1. **Si el modelo es lo suficiente bueno** para la extracción de reglas de la RNA.
2. **Cuán lejos** del comportamiento de la RNA está el modelo.
3. Si el modelo y los datos **son consistentes con las hipótesis** sobre la estructura del modelo.

Si llegados a este punto los resultados, después de estas cuestiones, no son los esperados habrá que descartar o reestructurar el algoritmo inductivo volviendo a la etapa de "Selección del modelo o algoritmo de extracción". Si, en cambio, lo que se considera que no tiene un funcionamiento correcto es el algoritmo de creación de ejemplos (en caso de utilizarse), entonces habrá que descartarlo y reestructurarlo volviendo a la etapa "Selección del modelo o algoritmo de creación de ejemplos".

En el caso de que haya que crear el fichero de entrenamiento similar al original y no se obtengan los resultados requeridos, puede ser necesario revisar el diseño de estos

ficheros e incluso modificarlos para incluir o eliminar valores que se puedan considerar necesarios o incorrectos, respectivamente. Para ello habrá que volver a la etapa "Diseño del Experimento y Conocimiento Previo".

5.3 Diagrama de flujo de la metodología propuesta

En el diagrama de la figura 17 se puede observar de forma gráfica lo expuesto en el apartado anterior. Hay que resaltar que existe una tarea que puede solapar su diseño con el de selección del modelo o algoritmo de extracción y el ajuste de parámetros, debido a que deben ser algoritmos independientes y no depende estructuralmente uno del otro. De esta forma, se aíslan los diseños y se pueden hacer pruebas por separado de ambos.

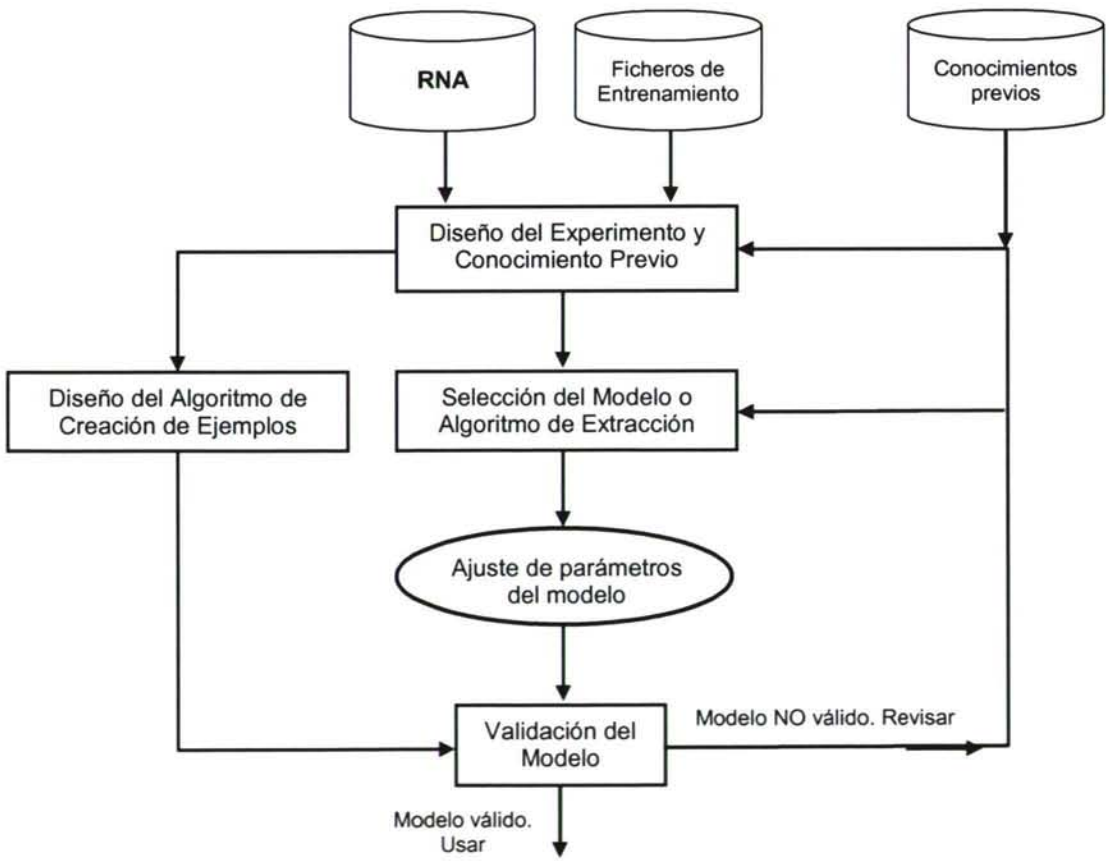


***Fig. 17** - Diagrama de flujo de la metodología propuesta.*

En el siguiente capítulo se muestra un ejemplo práctico de aplicación de esta metodología. Además, se realizarán todas las etapas y se indicarán los puntos problemáticos que se han encontrado y la forma de afrontarlos. Algunos de ellos, que han sido citado previamente, representarán los problemas que han surgido en el desarrollo del algoritmo de creación de ejemplos.

"La palabra es mitad de quien la pronuncia, mitad de quien la escucha"

Michel de Montaigne

CAPÍTULO

6 *EXTRACCIÓN DE REGLAS MEDIANTE PG*

6.1 Introducción

Se procederá a aplicar la metodología expuesta en el capítulo 4 al desarrollo de una técnica de extracción de reglas a RR.NN.AA de arquitecturas multicapa y recurrentes. Además, se procederá a extraer el conocimiento de generalización adquirido por las redes.

Para ello se explicará lo que se ha realizado en las diferentes etapas de la metodología indicando las dificultades y las facilidades encontradas en cada una de ellas. Como se ha comentado en el capítulo anterior, la metodología debe centrar su atención en la etapa de diseño del algoritmo inductivo, pues es éste el que determinará de una forma inexorable el éxito o el fracaso del proceso de extracción. Así, se hará un análisis pormenorizado de cómo se ha realizado y por qué se ha decantado la selección del algoritmo por una técnica evolutiva.

6.2 Diseño del experimento y conocimiento previo

En todo proceso de extracción de reglas de RR.NN.AA basadas en patrones entrada-salida se debe seleccionar un conjunto de datos representativos en forma de secuencia de entradas. Estas secuencias de entradas se introducen a la RNA y con las salidas que ésta produce se construyen los patrones entrada-salida. Estos patrones son los que utilizará el algoritmo inductivo para determinar y poder cuantificar el ajuste que consiguen las reglas obtenidas mediante el binomio "salidas_deseadas (RNA) / salidas_obtenidas (reglas)".

En los casos de ejemplo que se verán en el capítulo 6 no se tiene ninguna RNA que resuelva los diferentes problemas expuestos. Por lo tanto, antes de nada se construye una RNA para cada uno de los problemas. En algunos es necesario además construir los ficheros de entrenamiento (Mux, Monks, Mackey-Glass) en cambio en otros se han obtenido de diversos orígenes, referenciados en dicho capítulo.

Una vez diseñadas y entrenadas las RR.NN.AA se utilizan los mismos valores de entrenamiento y test para generar un segundo patrón de datos que será el utilizado para buscar las reglas que ha adquirido la RNA en el proceso de entrenamiento. Un diagrama general de cómo sería el proceso se puede ver en la figura 18.

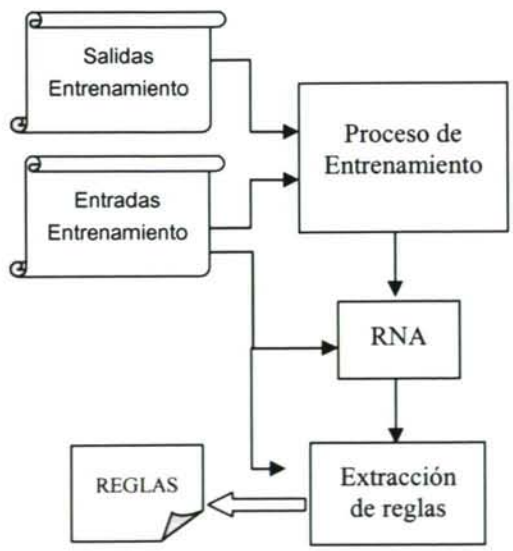


***Fig. 18** - Diagrama de flujo de la metodología propuesta.*

Para el entrenamiento de RR.NN.AA se ha dispuesto de una aplicación desarrollada como tesina por el autor del presente trabajo de investigación [RABU 99] en el que se realiza dicho proceso mediante AA.GG. Mediante esta aplicación se pueden entrenar tanto RR.NN.AA multicapa alimentadas hacia delante como recurrentes. Todas las RR.NN.AA vistas en el capítulo 7 de resultados han sido desarrolladas con esta aplicación.

6.3 Diseño del algoritmo de extracción

Como se ha visto en el capítulo 3 existen multitud de técnicas y algoritmos para la extracción de reglas de RR.NN.AA, pero la gran mayoría no son aplicables a todos los casos de arquitecturas de RR.NN.AA y otros presentan ciertos problemas de aplicabilidad a muchos problemas temporales, como por ejemplo RR.NN.AA.RR que trabajen con series temporales. En el capítulo 2 se ha visto que en la técnica de KDD existen multitud de modelos con sus ventajas y sus inconvenientes. De lo visto en ambos capítulos y del análisis expuesto a continuación se puede concluir que la Computación Evolutiva (CE) es la que presenta el mejor bagaje aplicabilidad-resultados.

6.3.1 Análisis de la CE en la extracción de reglas de RR.NN.AA

Varios estudios de Cherkauer y Shavlik [CHER 96] muestran que la combinación de AA.GG con algoritmos inductivos tradicionales produce mejores resultados que la aplicación de estos últimos solamente.

Varios algoritmos genéricos inductivos tradicionales que han mostrado resultados eficientes son el CN2 [CLAR 89] y el C4.5 [QUIN 93]. Este último se basa en la construcción de árboles de decisión de los cuales se extraen reglas IF-THEN.

Sin embargo, aplicar únicamente métodos evolutivos reduce la complejidad de los mismos y produce tan buenos resultados como la combinación con otros. Como ejemplo se puede citar el sistema "GABIL" [DEJO 91], que realiza una búsqueda incremental

para un conjunto de reglas de clasificación representadas por una cadena de "bits" de longitud fija.

Otros algoritmos inductivos basados en AA.GG son: "REGAL" [GIOR 94], "GA-MINER" [FLOC 95], "SET-Gen" [CHER 96] y más recientemente "GANN" [KEED 00].

La utilización de AA.GG como única técnica de extracción de reglas está enfocada a problemas de optimización. Se comienza con una población de partida y se va alterando y optimizando su composición (material genético) para la solución del problema de ajuste de los patrones entrada-salida de la RNA mediante mecanismos tomados de la teoría de la evolución (selección, cruce y mutación).

En estos métodos, al igual que en las RR.NN.AA, se pueden manejar datos continuos y discretos, lineales y no lineales, simultáneamente según la estructura cromosomática del individuo genético. Además son métodos pensados para trabajar con datos del mundo real donde pueden ser incompletos, imprecisos e incluso inciertos. En cierto modo, similares a las RR.NN.AA, pues ambos son emulaciones de procesos biológicos y están diseñados en ámbitos comunes del mundo real.

Otra de sus ventajas es que el tipo de estructuras de datos con las que va a trabajar no influye en su algoritmo, simplemente supone cambios en la estructura del cromosoma y no en el proceso de búsqueda.

Además, son ideales para trabajar en entornos donde el espacio de búsqueda crece exponencialmente al número de variables que intervienen en el problema a resolver y donde la aplicación de otros procesos de búsqueda determinísticos se torna impracticable.

Quizás el mayor problema en la utilización de estos métodos es la forma de interpretar el material genético de un individuo como un conjunto de reglas entendibles por un ser humano.

Además, la codificación del AG para realizar la extracción de reglas de una RNA supone una implementación donde el cromosoma de los individuos sea variable, pues es muy difícil saber de antemano cuántas reglas van a obtenerse.

Estas dos últimas cuestiones suponen una complicación bastante importante a la hora de decidirse por los AA.GG como técnica a implementar en el algoritmo inductivo de extracción de reglas.

6.3.2 Aplicación de la PG como algoritmo inductivo de extracción de reglas

Existe una técnica relativamente reciente (1992), de cuyo principal exponente es John Koza, que se basa en los AA.GG y en la CE en general. Esta técnica llamada Programación Genética que se ha visto más en detalle en capítulo 2 ofrece todas las ventajas vistas de los AA.GG pero debido a que la forma de codificar los individuos en forma de árbol no limita el tamaño de éstos (depende de la profundidad del árbol), ya que por naturaleza es una estructura dinámica de tamaño variable.

Además, la representación en árbol es la forma natural de presentación de los análisis léxicos y sintácticos y de la amplia teoría de los autómatas y los compiladores. Éstos desde siempre han utilizado las reglas representadas en árbol como forma de representar el análisis léxico y sintáctico de los "programas".

La PG es vista como una especialización de los AA.GG [BÄCK 00]. De manera similar a ellos, la PG se concentra en la evolución de los genotipos. La principal diferencia radica en el esquema de representación usado. Mientras los AA.GG utilizan cadenas de representaciones, la PG representa los individuos como programas que resuelven un determinado problema de forma automática, y para ello se dispone de una serie de símbolos y funciones que estructuradas en forma de árbol representan un programa. Es por tanto una técnica pensada para que la forma de representación de los programas (reglas) sea fácilmente entendible por el ser humano [RABU 02].

La utilización, por tanto, de la PG como algoritmo inductivo puede verse como una de las técnicas potencialmente más prometedoras para alcanzar los mejores

resultados en cuanto al proceso de extracción de reglas de RR.NN.AA. Es por ello que en la presente tesis se presenta como ejemplo de desarrollo de algoritmo inductivo la PG.

De aquí en adelante se centrará este trabajo en presentar el proceso de desarrollo del algoritmo inductivo particularizado, en este caso concreto, en la PG. Obviamente, es aplicable, dentro de sus limitaciones concretas, cualquier técnica anteriormente presentada. Simplemente variará su implementación cuando se programe.

6.3.2.1 Desarrollos previos de PG para la extracción de reglas de RR.NN.AA

En el trabajo reciente realizado por Wong y Leung aplican la PG a la técnica de extracción de conocimiento y presentan el "LOGENPRO" (Logic grammar based Genetic Programming) [WONG 00]. Realizan una combinación de PG y representación del conocimiento en forma de lógica de primer orden. Esta primera aproximación demuestra las ventajas de la utilización de la PG como algoritmo inductivo.

La utilización de la PG en combinación con los árboles de decisión donde las funciones en los nodos de los árboles usan una o más variables también ha demostrado su aplicabilidad como técnica de extracción de reglas [BOT 99], pero dicha combinación complica de forma innecesaria el diseño de estos algoritmos combinados.

Más recientemente, Engelbrecht, Rouwhorst y Schoeman [ENGE 01] aplican la técnica de la PG y los árboles de decisión para extraer conocimiento de BB.DD diseñando el algoritmo llamado "BGP" (Building-Block Approach to Genetic Programming). En este algoritmo, al igual que el propuesto por Bot, se combinan árboles de decisión con métodos evolutivos de la PG pero, en este caso, centrado en el concepto de bloque constructivo, el cual representa una condición o un nodo en el árbol. Un bloque constructivo consta de tres partes: un atributo, un operador relacional y un umbral. Combinando diferentes valores de las partes de los bloques constructivos en forma de árboles de decisión se obtienen las reglas.

6.3.2.2 Consideraciones

En todas las implementaciones de PG para la extracción de reglas no se ha encontrado hasta el momento que se aproveche toda la potencia que ella puede producir. Todas se centran en obtener reglas del tipo "IF-THEN" combinados con operadores lógicos. Sin embargo, mediante la PG se puede ir más allá: utilizar regresión simbólica no sólo para obtener expresión lógicas basadas en reglas "IF-THEN" sino combinadas con expresiones matemáticas, y funciones más complejas donde trabajar con datos continuos no suponga un problema de falta de representación. Esto es debido a que, en problemas de clasificación, no se precisa usar funciones y expresiones matemáticas y trigonométricas. Sin embargo, en otros casos, como pudiera ser la predicción de series temporales, sí es necesario disponer de estos operadores como elementos no terminales.

6.3.3 Optimización de reglas

Otro de los aspectos más destacados de todo sistema de extracción de reglas es la optimización de las reglas obtenidas del análisis de la RNA. Se debe tener en cuenta que las reglas extraídas pueden contener condiciones redundantes, muchas reglas específicas pueden estar contenidas en reglas generales, y muchas de las expresiones lógicas extraídas se pueden simplificar si son escritas de otra forma. Por tanto, la optimización de reglas consiste en la simplificación y la realización de operaciones simbólicas sobre las reglas. Dependiendo del método de extracción y del tipo de reglas obtenidas se pueden aplicar diferentes técnicas de optimización. Así, se pueden clasificar en dos grandes tipos: métodos de optimización "embebidos" y métodos "a posteriori". Estos últimos son, generalmente, un algoritmo de análisis sintáctico que se aplica sobre las reglas extraídas para su simplificación. Por ejemplo Duch [DUCH 00] utiliza Prolog como lenguaje de programación para un postprocesado de las reglas obtenidas. Utiliza un programa en Prolog para la obtención de variables lingüísticas óptimas y con ellas obtener reglas simplificadas que utilicen dichas variables. Los métodos de optimización embebidos son técnicas empleadas en los algoritmos de extracción de reglas que, de manera intrínseca, hacen producir al algoritmo reglas cada vez mejores. Como ejemplo,

se puede citar la técnica de penalización por profundidad de la PG. Conceptualmente, cuando se evalúa el nivel de adaptación de un individuo de la PG (árbol) se reduce su capacidad un cierto valor según el número de nodos terminales y no terminales que tenga el árbol. De esta forma, se favorece de forma dinámica la existencia de individuos simples. Así pues, si se están buscando reglas (árboles sintácticos), se favorece de forma intrínseca la aparición de reglas simples optimizadas.

6.3.4 Modo de operación de la PG implementada

Otro punto a tener en cuenta a la hora de haber decidido la utilización de la PG para aplicar el algoritmo de extracción es su modus operandi. Como se ha comentado en capítulos anteriores, las técnicas de extracción se pueden clasificar en tres grandes grupos: "decompositional", "pedagogical" y "eclectic". La primera aproximación "decompositional", se centra en tratar cada neurona de la RNA, especialmente las de las capas ocultas y de salida. Así pues las reglas se extraen de cada neurona y de su relación con las demás. Esto hace que los métodos basados en esta técnica sean totalmente dependientes de la arquitectura de la RNA a tratar y, por tanto, limitan la aplicabilidad de forma genérica. La segunda aproximación "pedagogical", por el contrario, trata las RR.NN.AA como "cajas negras" donde sólo se observan relaciones entre lo que entra y lo que sale de la RNA. Por tanto, el objetivo fundamental de esta técnica es obtener la función computada por la RNA. Y la última aproximación "eclectic", combina elementos de las dos anteriores. Como se ha comentado anteriormente, en esta tesis se ha aplicado la PG como algoritmo de construcción de un árbol sintáctico que refleje un conjunto de reglas lo más similar posible al funcionamiento de una RNA. Para ello, se ha utilizado regresión simbólica aplicada a patrones de entrada-salida. Estos patrones son secuencias de entradas aplicadas a una RNA y salidas obtenidas de la misma. Por lo tanto, se obtiene un conjunto de reglas y fórmulas que emulan el comportamiento de la RNA. Esta técnica así mostrada se puede clasificar como "pedagogical", donde la RNA se trata como una "caja negra". Esto supone una ventaja dado que no es necesario conocer el funcionamiento interno de la RNA. Sin embargo, un algoritmo de extracción de reglas que pueda trabajar con estructuras de "caja negra" debería además poder

implementar algún mecanismo que permita la incorporación "a priori" del conocimiento que se tiene de dicha "caja negra", ya que así reducirá de forma considerable el espacio de búsqueda de las reglas del sistema. Estas estructuras se conocen como "caja gris". Con la utilización de la PG esto sí es posible debido a que se puede controlar y determinar el número y el tipo de elementos terminales y no terminales que intervienen en el proceso de búsqueda. Así, por ejemplo, si se sabe que la RNA realiza tareas de clasificación se puede limitar el tipo de nodos terminales a "booleano", prescindiendo de valores en punto flotante. Esto supone una enorme ventaja pues de antemano se eliminan todas las posibles combinaciones de operaciones matemáticas.

6.4 Ajuste de parámetros del modelo

Para el ajuste de parámetros involucrados en la PG se va a utilizar la experimentación. Se han implementado numerosas técnicas relativas a la CE para así reforzar la operativa de la PG, muchas de ellas se centran en variaciones sobre los operadores genéticos selección, cruce y mutación. Algunos de los parámetros implementados son:

- ✓ Algoritmo de creación de población
  - Completo
  - Parcial
  - Intermedio
- ✓ Algoritmo de selección
  - Torneo
  - Ruleta
  - Sobrante estocástico
  - Universal estocástica
  - Muestreo determinístico
- ✓ Algoritmo de mutación
  - Subárbol
  - Puntual
- ✓ Estrategia elitista
- ✓ Utilizar cruces no destructivos
- ✓ Tasa de cruces
- ✓ Probabilidad de mutación
- ✓ Probabilidad de selección de no terminal
- ✓ Tamaño de la población
- ✓ Nivel de parsimonia

Para ajustar los parámetros ideales se debe realizar un análisis empírico probando ciertas combinaciones e ir ajustando progresivamente los diferentes valores hasta alcanzar los mejores resultados. Como ejemplo, se puede observar en el capítulo 7 donde se muestran los resultados obtenidos relativos al problema "Mux 6 a 1" combinando de forma empírica varias configuraciones de valores de los parámetros anteriores.

Otra serie de ajuste de parámetros a establecer es el relativo a los nodos terminales y no terminales que van a intervenir en la ejecución del algoritmo. Aquí interviene el conocimiento "a priori" que se disponga de la RNA y el tipo de problemas para los que fue diseñada.

Los parámetros a seleccionar en este sentido son:

- ✓ Funciones Aritméticas
  - Suma
  - Resta
  - Producto
  - División
  - Cuadrado
  - Raíz Cuadrada
- ✓ Funciones Logarítmicas
  - Logarítmica Sigmoidal
  - Exponencial
  - Hip. Tg. Sigmoidal
  - Logaritmo Neperiano
  - Logaritmo en Base 10
- ✓ Funciones Trigonométricas
  - Seno
  - Coseno
  - Tangente
  - Arco seno
  - Arco coseno
  - Arco tangente
  - Seno hiperbólico
  - Coseno hiperbólico
  - Tangente hiperbólica
- ✓ Operadores Lógicos
  - AND

- OR
- NOT

✓ Operadores Relacionales
- >, <, =, <>, >=, <=
- Rangos

✓ Funciones de Decisión
- IF–THEN– ELSE sobre reales
- IF–THEN– ELSE sobre booleanos

✓ Funciones Umbral
- Signo
- Escalón

✓ Ventanas Temporales (problemas dinámicos)
- Número de entradas anteriores
- Número de salidas anteriores

✓ Constantes
- Selección aleatoria en rango o establecidas manualmente (indicar rango o una por una)

✓ Variables de Entrada
- Determinar una por una si es real o booleana

✓ Naturaleza de las Salidas
- Real
- Booleana
- Convertir Real a Booleana (especificando el umbral)

Como se puede observar existe una gran variedad de posibles configuraciones, así cuanta más información se disponga de la RNA más se pueden establecer de antemano los valores de los anteriores parámetros. En otro caso es necesario la experimentación para determinar cuáles son los mejores para cada RNA.

6.5 Validación del modelo

Para la comprobación del correcto funcionamiento del modelo se aplicarán problemas considerados como Test en la literatura de los sistemas de extracción de conocimiento y de aprendizaje. Para ello, se han elegido los problemas "MUX 6 a 1" y los "Monk Problems" donde, partiendo de una serie de patrones entrada-salida y de espacios de búsqueda bien definidos, se pueden realizar los procesos de extracción sobre los mismos. Sobre estos no se aplica ni se utiliza ninguna RNA, porque lo interesante en este momento es analizar el comportamiento del algoritmo ante tareas de

extracción de reglas existentes en patrones entrada-salida y comprobar dichas reglas con otras obtenidas por otros métodos existentes. Si las reglas obtenidas son satisfactorias y comparables a las obtenidas por otros métodos entonces se puede concluir que el algoritmo también va a extraer reglas de patrones de datos procedentes de las RR.NN.AA.

De los resultados presentados en el capítulo 7, se puede concluir que dichos resultados son satisfactorios y el algoritmo de inducción funciona de una forma eficiente en el proceso de extracción de reglas de patrones entrada-salida.

6.6 Algoritmo de creación de ejemplos

Para poder extraer el conocimiento relativo a la capacidad de generalización adquirida por las RR.NN.AA es necesario disponer de una estructura dinámica de patrones entrada-salida que cubran todo el amplio espacio de búsqueda.

6.6.1 Alternativas consideradas

El caso ideal es dar todas las combinaciones posibles de valores de entradas para poder analizar todas las posibles salidas generadas por la RNA. De esta forma, se puede asegurar que se han explorado todos los estados posibles de la red. Sin embargo, esto se vuelve imposible cuando el espacio de estados es elevado. Por ejemplo, en el caso del "Mux 6 a 1", sí es viable analizar todos los posibles valores de entradas posibles a la red porque se tienen 6 entradas de tipo booleano con 2 estados por entrada, lo que constituyen 2^6 posibles combinaciones. Estas 64 secuencias de valores de entrada son un número asequible para poder explorarlas todas. Sin embargo, para los "Monk problems" el espacio de estados ya aumenta a 432, que aún siendo un valor asequible aumenta de forma considerable el tiempo que precisa el algoritmo de extracción para obtener una generación de individuos y, por tanto, aumenta el tiempo global de extracción de reglas. En estos dos casos las variables son booleanas, o con niveles de clasificación bajos (2, 3 ó 4 valores a lo sumo por variable de entrada); sin embargo,

cuando alguna de las entradas es continua (en punto flotante) analizar todos los posibles valores se vuelve impensable.

Para ello se ha pensado inicialmente en establecer divisiones en el rango máximo de valores de entrada. La ventaja de trabajar con RR.NN.AA es que las entradas de la red deben estar normalizadas (ver fundamentos en el capítulo 2), por tanto estarán en el rango [0,1]. Ello supone establecer una serie de subdivisiones de este rango y presentar todas las posibles combinaciones de los valores de cada subrango. De esta forma, se puede reducir el número total de combinaciones posibles del espacio de estados. Pero ello supone perder un considerable valor de posibles combinaciones y valores representativos. Además, supone un problema decidir el nivel de granularidad de cada variable de entrada. Pero estos no son los principales problemas, por ejemplo en el caso de la detección de casos mortales de hepatitis (resultados en el capítulo 7) se tienen 19 entradas de las cuales 13 son booleanas y 6 continuas; sin embargo, de estas últimas ya hay establecidos una serie de valores o subrangos para cada variable teniendo 6 posibles valores para 4 de ellas, 8 para otra y 9 para la restante. Esto supone tener inicialmente $2^{13} = 8.192$ combinaciones posibles para las variables booleanas y $6^4*8*9 = 93.312$ para las continuas convertidas en discretas. Esas 6 variables constituyen un número de combinaciones de más de 10 veces las generadas por las 13 variables booleanas. Además, el número total de combinaciones de las 19 variables sería 8.192 * 93.312 = 764.411.904. Con esto se tiene que por cada iteración de algoritmo inductivo se deben realizar todas las combinaciones anteriores, lo cuál supone una excesiva explosión combinatoria. Por ello, conviene analizar otros posibles métodos de creación de ejemplos representativos.

Otra alternativa ha sido, partiendo de todas las combinaciones posibles seleccionar de forma aleatoria y uniformemente distribuida un número de ellas por el usuario. De esta forma, se tiene un número de patrones relativamente bajo y el proceso de extracción de reglas se puede realizar sin excesiva carga computacional por evaluación y cálculo del error cometido. Sin embargo, no se puede asegurar que se han seleccionado combinaciones que contienen todas las características relevantes que la red

realiza. De hecho, en las pruebas llevadas a cabo con el problema de detección de casos mortales de hepatitis se ha constatado que de todas las posibles combinaciones existe un alto porcentaje de ellas que inclinan la decisión de la RNA a casos mortales, por ello cuando se seleccionan casos de ejemplo al azar tendrán estos valores de porcentaje y la tendencia de las reglas obtenidas será a obtener valores de la tendencia mayor.

En el problema de la detección de casos mortales de hepatitis se ha probado a generar 10 conjuntos de entrenamiento con 1.000 secuencias de valores aleatorios cada uno. De ellos la media de que la salida de la RNA fuera la clasificación de caso mortal ha sido del 97.43% y el 2.57% en el caso opuesto. Está claro que al ejecutar la simulación del algoritmo inductivo va a obtener el resultado "caso mortal" (o valor True) sin dejar evolucionar la población ya que es la regla más corta (optimizada) y produce un ajuste del 97.43% de acierto. Por lo tanto, se puede concluir que esta técnica tampoco produce resultados satisfactorios, y al igual que las técnicas anteriores, colapsa el proceso de extracción de reglas y se detiene su ajuste.

La siguiente tentativa ha sido dividir todas las combinaciones de reglas en tantas partes como combinaciones de salidas existan. Así, en el ejemplo anterior se dividen en 2, una parte de combinaciones que producen resultados "caso mortal" (True) y otra con combinaciones que producen "caso NO mortal" (False). Una vez dividido el espacio de búsqueda el usuario decide el porcentaje de valores de cada clase que van a conformar los patrones de aprendizaje. En la figura 19 se puede ver la idea expuesta de forma esquemática.

Esto se ha probado y se han obtenido resultados importantes pero, de nuevo existe un grave problema, si existe más de una salida o tiene mas de dos clasificaciones o si las salidas son valores en continuo, entonces el proceso de división del espacio de búsqueda se complica de forma exponencial. En caso de que las salidas sean valores continuos de nuevo se debe establecer una granularidad y se perderán valores representativos, además de volverse impracticable por el gran número de combinaciones y tasas de subdivisiones creadas.

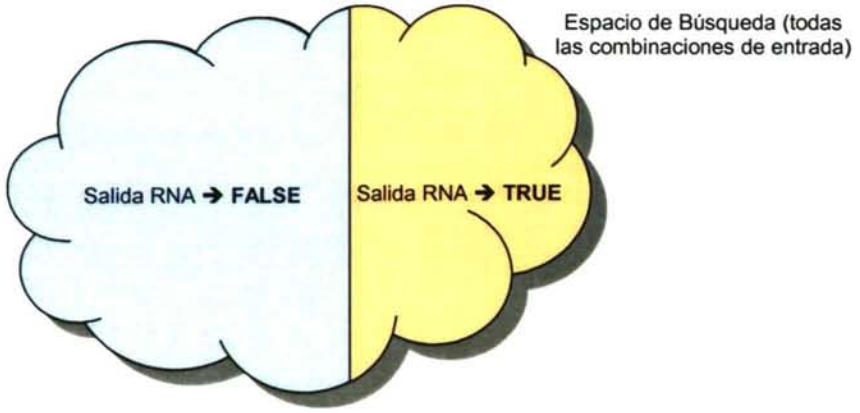


***Fig. 19** - Diagrama de operación de la división del espacio de búsqueda.*

Otro gran problema de las diferentes alternativas expuestas es que sólo valen para redes alimentadas hacia delante. Cuando se trata con redes recurrentes donde intervengan estados de activación previos es "imposible" establecer todas las posibles combinaciones de valores de entradas. Como ejemplo se puede citar una RNAR que haga la predicción a corto plazo de una serie temporal. En este caso el valor de la salida dependerá del valor de la entrada y de las N entradas anteriores que han activado neuronas internas en estados previos. En este caso no se pueden seleccionar todas las combinaciones posibles porque habrá que poner todos los posibles casos con una entrada única y reinicializar la red, luego con dos entradas, una en t y otra en t-1, y así sucesivamente. Además de crecer exponencialmente el número de casos de prueba, pueden llegar a ser de longitud "infinita".

Debido a esto se ha pensado en otras alternativas y se ha llegado a una solución que es aplicable a cualquier arquitectura de RNA y que ha producido unos resultados realmente buenos, tanto en redes multicapa como en redes recurrentes aplicadas a casos temporales. En el capítulo 7, se podrán ver más en detalle estos resultados obtenidos. El algoritmo desarrollado se explica en más detalle en el siguiente apartado.

6.6.2 Algoritmo desarrollado

El algoritmo se basa en la siguiente idea: "Si el conocimiento de generalización se encuentra implícito en los ficheros de entrenamiento debido a que la RNA lo adquiere

de ellos, el centro del proceso de extracción de dicho conocimiento deben ser dichos ficheros". Siguiendo esta idea se dará al algoritmo de extracción los ficheros de entrenamiento que ha tenido la RNA y algo más: un conjunto de combinaciones de entradas seleccionadas al azar y basadas en las utilizadas en los ficheros de entrenamiento.

El número de elementos de este conjunto de ejemplos añadidos lo debe indicar el usuario en un valor en tanto por cien. Pero debe ser un valor no excesivo porque en caso contrario sucederá como en casos anteriores, se dirigirá la búsqueda de reglas hacia un mínimo local y se pueden perder reglas importantes. El número de datos nuevos deberá ser entre un 5 y un 25%, como se podrá observar en el capítulo 7, aunque obviamente dependerá de los diferentes problemas y de los ficheros de entrenamiento. Por lo tanto, se debe elegir un valor en base a la experimentación.

Pero si sólo se utiliza un 25% de datos nuevos (en 100 ejemplos de entrenamiento supone 25 nuevos) es muy poca representación para el restante espacio de posibles combinaciones de valores de entradas. Por tanto, es necesario que el proceso sea **dinámico**; es decir, vaya creando y eliminando de forma dinámica las nuevas reglas creadas, pero siempre sin modificar los valores del entrenamiento.

Entonces interviene la decisión de cuando crear y cuando eliminar. Para tomar esta decisión se usa el sentido común: *eliminar cuando se haya acertado, mantener cuando se falle y crear cuando se haya eliminado alguna.*

El algoritmo queda entonces de la siguiente forma:

1. Crear tantas secuencias de valores de entrada a la RNA como indique el usuario. Las nuevas secuencias no deben coincidir con ninguna de las existentes en los ficheros de entrenamiento.
2. Dejar evaluar el algoritmo de extracción de reglas *N* ciclos. Si después de esos ciclos las mejores reglas obtenidas aplicadas a cada una de las combinaciones nuevas coincide en sus salidas con las salidas de la RNA

(error cero para ese ejemplo) entonces ese ejemplo se elimina, en caso contrario se mantiene.

3. Generar nuevas secuencias de valores de entrada para sustituir las eliminadas en el paso anterior.

Para el proceso de creación de los valores de las series nuevas se pueden aplicar varias alternativas, por una parte generar todos los valores nuevos al azar o, por otra, partiendo de alguna secuencia del fichero de entrenamiento generar una nueva con algunos valores nuevos (conceptualmente similar a la operación de mutación en la CE).

En la figura 20 se puede observar de forma gráfica el proceso inicial de generación de secuencias nuevas.

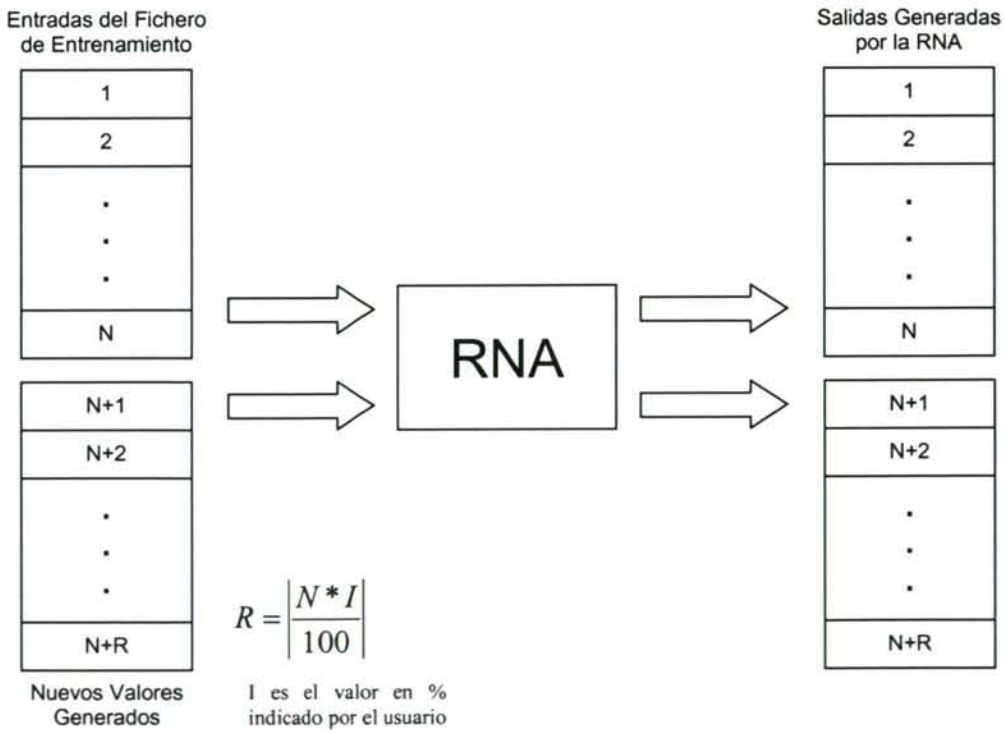


***Fig. 20** - Proceso inicial de creación de ejemplos*

Si se aplica la primera alternativa se pueden generar series con valores muy dispares que sólo provoquen un tipo de salida. Por ejemplo en el caso expuesto anteriormente en que el 97.43% producen un tipo de salida de la RNA y el 2.57% el

caso opuesto. Sin embargo, la segunda alternativa, al sólo modificar alguno o algunos valores puntuales de las secuencias de entradas del fichero de entrenamiento, los nuevos casos de ejemplo creados tienen menos probabilidad de ser tan dispares (en el espacio de búsqueda) a los casos de entrenamiento y habrá más tendencia a producir secuencias nuevas representativas.

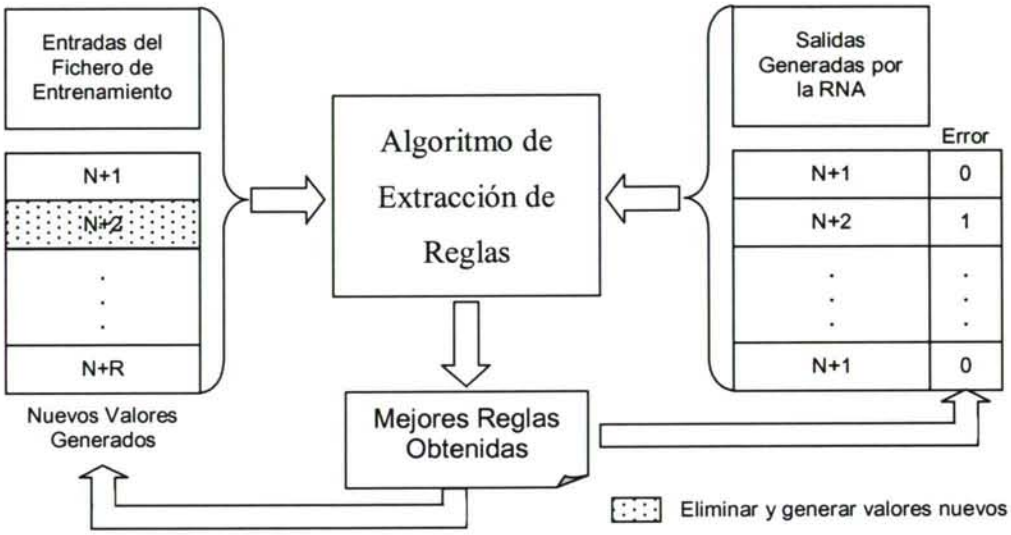


***Fig. 21** - Proceso dinámico de eliminación y creación de nuevos de ejemplos*

En la figura 21 se puede observar el diagrama de operación del algoritmo expuesto. Partiendo del estado inicial visto en la figura 20 el proceso se realiza de forma iterativa: el usuario especifica un número de generaciones o iteraciones del algoritmo inductivo de extracción de reglas, cuando se realizan se analizan las reglas obtenidas y se obtienen los resultados que producen las mismas aplicadas a los casos nuevos de prueba. Aquellos casos de prueba que coincidan en su resultado con el producido por la RNA indican que las reglas obtenidas tienen el comportamiento de esos casos, por lo que se pueden eliminar. Una vez eliminados todos los casos de prueba pertinentes se crean unos nuevos para rellenar los huecos dejados por los eliminados. Y el proceso empieza de nuevo, se ejecuta el algoritmo inductivo para *N* generaciones y se repite la eliminación y creación de casos de prueba. Terminará este proceso cuando el usuario determine que las reglas obtenidas son las deseadas. El número de generaciones se debe

determinar en base a la experiementación, ya que depende del tipo de RNA y de los ficheros de entrenamiento.

El algoritmo propuesto es aplicable para redes de estructura multicapa alimentadas hacia delante; sin embargo, al aplicarlo a redes recurrentes donde intervienen estados pasados surgen ciertos problemas debido a que las salidas dependen de varias combinaciones de secuencias de entradas. Por ello, generar combinaciones de valores aleatorios no es del todo aplicable.

La solución que se ha tomado, en el caso particular de las RR.NN.AA.RR, toma como base inicial los ficheros de entrenamiento para generar una nueva secuencia de partida que aplicada como entrada a la RNA va a producir las consecuentes combinaciones de entrada-salida. Por ejemplo, si se tiene una RNAR para la predicción de una serie temporal a corto plazo, el sistema funcionará como una realimentación, se le da el valor en el instante t y la red predice el valor del instante $t+1$, y este valor en el instante $t+1$ se suministra de entrada para predecir el valor en el instante $t+2$, y así sucesivamente.

En la figura 22 se puede observar la idea expuesta. La forma de eliminar las secuencias de ejemplos también variará. En este caso sólo se genera un valor aleatorio y éste es el que crea todos los casos siguientes. Pues bien, el criterio de eliminación será cuando las R secuencias nuevas creadas tengan un error de comparación con las producidas por la RNAR de cero; es decir, las reglas asimilen el conocimiento existente en toda la subserie de casos nuevos. Y, por consiguiente, se eliminarán todos. El siguiente paso será crear un nuevo valor aleatorio, construir a partir de él la nueva subserie y comenzar de nuevo el proceso de extracción hasta que se consiga el ajuste, y así sucesivamente.

Como se puede ver en la figura 22 en la secuencia de entradas generada se produce un solapamiento entre las entradas (X) y las salidas a raíz del primer valor generado aleatoriamente. De esta forma, se puede analizar el comportamiento de la

RNAR ante multitud de secuencias futuras posibles y se puede abstraer el conocimiento de generalización que se encuentra residente en sus conexiones recurrentes.

En el capítulo 7, se podrán comprobar los tipos de reglas y expresiones que se pueden obtener de estas redes con el algoritmo aquí expuesto.

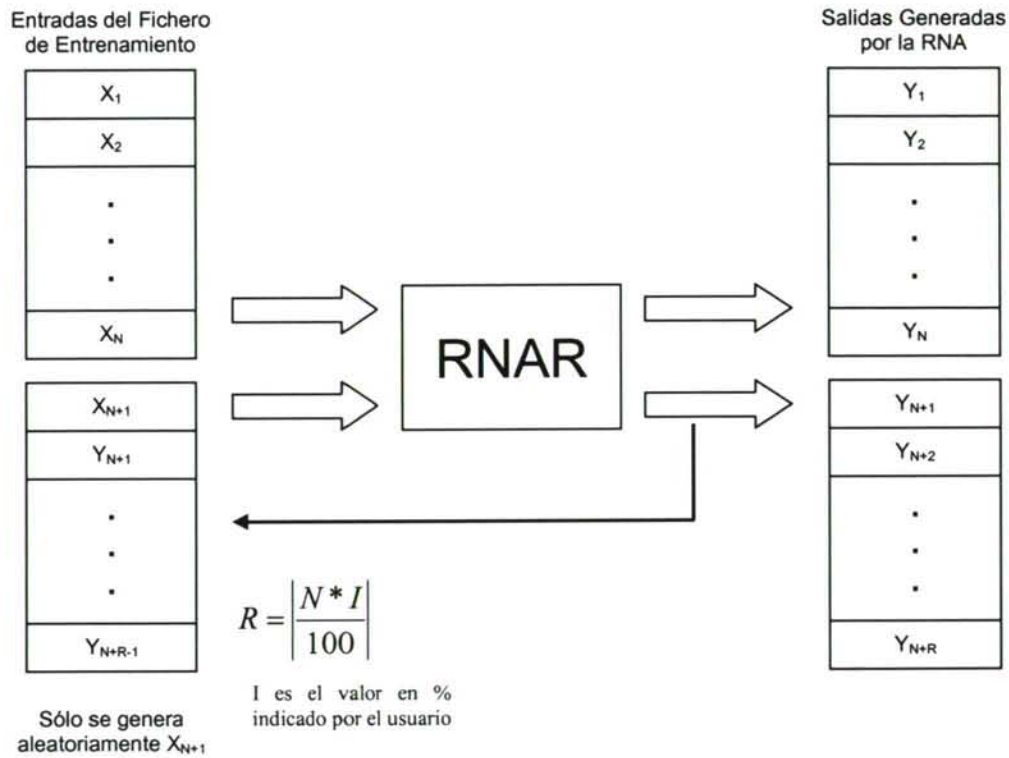


***Fig. 22** - Proceso inicial de creación de ejemplos en casos de RNAR y temporales*

6.7 Partes del sistema propuesto

La aplicación, a nivel de implementación, consta, como se puede observar en la figura 23, de dos partes bien diferenciadas: por un lado, dos módulos de representación gráfica: de interface con el usuario y de manipulación de datos y, por otro, tres módulos de procesado para las tareas de computación, simulación y gestión de memoria.

Los primeros módulos realizan toda la parte de interacción con el usuario, mostrándole todos los resultados obtenidos y los controles de los parámetros de funcionamiento de la extracción de reglas y ejecución de las RR.NN.AA. También es el

encargado de dirigir todo el proceso pasando el control a los otros módulos para realizar la simulación y solicitando los resultados de estos para su posterior representación gráfica. Los segundos módulos son los encargados de: llevar a cabo todas las tareas de cómputo relacionadas con el proceso de simulación de la RNA, construir la población de árboles sintácticos, simular el proceso biológico de la evolución de los árboles, y crear, eliminar y manipular dinámicamente los casos de ejemplo a tratar por la RNA.

En este módulo es donde se evalúan las RR.NN.AA calculando las activaciones de las neuronas en función de los pesos de las conexiones, del tipo de neuronas que contiene, de la pendiente de las rectas de activación, etc.

La ejecución de la aplicación lanza la interfaz de la simulación de la RNA que llama, para realizar el procesado, al módulo de evaluación de RR.NN.AA cuando se requiere llamar a alguna función que éste implementa. Una vez que se generan los patrones Entrada–Salida se lanza la interfaz de la simulación del algoritmo inductivo que llama al módulo de simulación de la PG y al módulo de regresión para conseguir un algoritmo de regresión simbólica mediante PG.

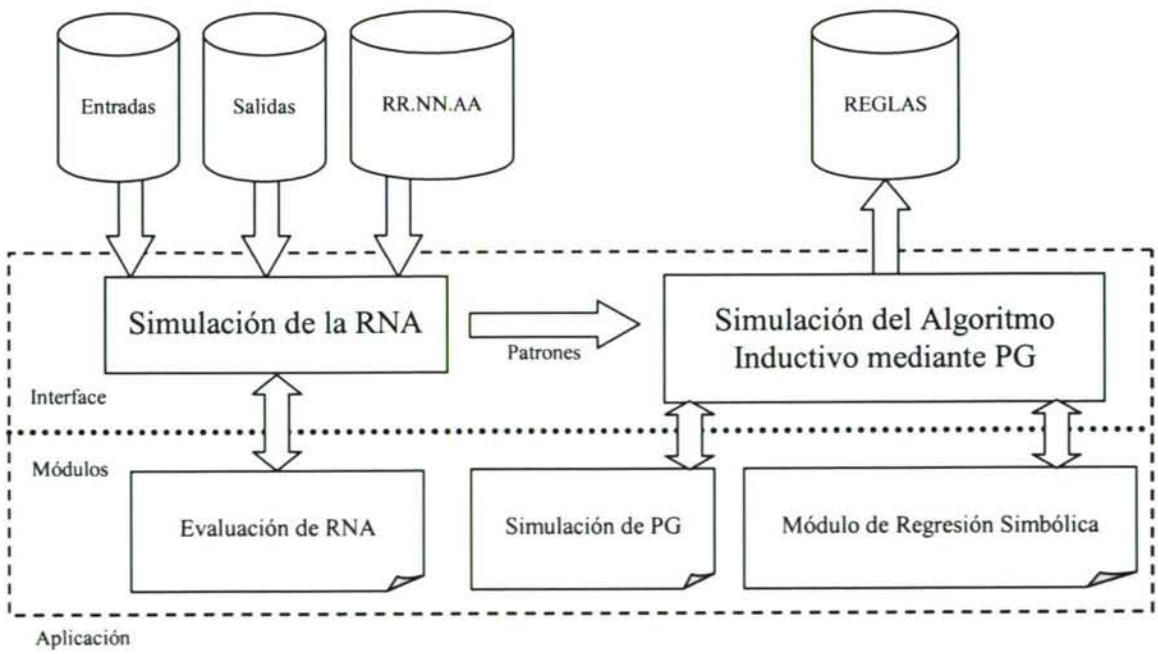


***Fig. 23** - Diagrama modular de todo el proceso de Extracción de Reglas.*

"Se puede porque
se cree poder"

Virgilio

CAPÍTULO

7 RESULTADOS

7.1 Introducción

Se ha analizado un gran número de conjuntos de datos comparando los resultados obtenidos mediante PG con los conseguidos por otros métodos, en los casos que fué posible. Aunque muchos de los conjuntos de reglas son de diferente complejidad, se expondrá la forma de obtenerlas de las diferentes RR.NN.AA según el nivel de complejidad requerido para cada clase de problema que pretenden resolver. Todos los casos que se expondrán a continuación pueden ser considerados como una referencia o "benchmark" para otros sistemas de extracción de reglas.

Se van a realizar varios casos de extracción de reglas sobre diferentes tipos de problemas. Se empezarán probando casos de laboratorio donde se establecen unos problemas clásicos en la literatura de la extracción de reglas como son la función "Mux" (multiplexador booleano) y los "Monk problems". Estos últimos son tres problemas de clasificación de carácter artificial y simple, preparados para comprobar el

correcto funcionamiento de los algoritmos de extracción de reglas. Cada uno de los tres problemas trata de determinar cuándo un objeto descrito por seis características es un robot o no.

Posteriormente se aplicarán casos reales. En todos ellos, el problema a tratar se basa en tareas de clasificación. Para ello, se dispone de unos casos reales relacionados con el campo de la medicina, donde se trata de determinar la clasificación de diferentes enfermedades. En todos estos casos se entrenarán solo RR.NN.AA con arquitectura alimentada hacia delante, y sobre ellas se aplicarán las tareas de extracción de reglas para la emulación de comportamiento de las RR.NN.AA. Posteriormente, se determinarán los rangos de operatividad de las RR.NN.AA para cada caso particular. Este aspecto es de relevante importancia debido a que se trata de verificar (comprobar mediante nuevos casos de prueba ficticios el correcto funcionamiento de la RNA) y de validar (contrastar los resultados obtenidos por la RNA ante nuevos casos de prueba) el funcionamiento de una RNA. Esto supone obtener los límites de funcionamiento "correcto" de una RNA obteniendo así su capacidad de generalización. Con esto se pone de manifiesto que la PG no sólo se puede utilizar dentro del contexto de la extracción de reglas, sino que con unas mínimas incorporaciones de ciertos operadores se puede conseguir la validación de los niveles de operación de cualquier RNA. Aspecto, éste, de especial relevancia en la aplicabilidad de las RR.NN.AA a ciertos dominios de riesgo práctico alto, como pudiera ser el control de una planta nuclear, el diagnóstico de ciertas enfermedades médicas, etc., y que hasta el momento no se ha resuelto de una forma clara en la literatura científica de las RR.NN.AA y de la extracción de reglas.

Después, se aplicará la extracción de reglas a RR.NN.AA de arquitectura recurrente. Esto supone un reto adicional debido a que éstas RR.NN.AA se caracterizan por su gran capacidad de representación y de conocimiento distribuido en sus conexiones, especialmente aplicables a problemas de carácter temporal y dinámico. Concretamente, se aplicarán a problemas de predicción de series temporales [JENK 76], el primer problema a resolver será la predicción de una serie temporal caótica de

laboratorio, muy clásica en el campo de las series temporales: la "serie de Mackey-Glass". Se verá qué tipo de reglas y expresiones se pueden obtener de RR.NN.AA que emulan el comportamiento de éstas series. Además, se probará el funcionamiento del sistema con valores continuos (punto flotante). Se utilizarán, por tanto, nodos no terminales que representen operaciones matemáticas y trigonométricas. En todos los casos las series temporales serán estructuras con una única entrada y una única salida. La entrada corresponde a un valor numérico del instante *"t"* y la salida del sistema es la predicción del valor numérico en el instante *"t+1"*. Con esto se podrá comprobar el nivel de complejidad que está tratando el algoritmo de extracción de reglas.

Los siguientes casos de prueba serán de nuevo predicciones de series temporales, pero en este caso series del mundo real. Se utilizarán las dos siguientes series: "Manchas Solares" y "Producción de Tabaco en Estados Unidos". Estas dos series son estructuralmente diferentes, mientras la primera es estacionaria en la media, la segunda es incremental creciente.

Finalmente, se procederá a extraer la capacidad de generalización de este tipo de redes. Para ello se aplicará el algoritmo de creación dinámica de patrones sobre la "función de Mackey-Glass".

De esta forma se verificará el funcionamiento del sistema de extracción de reglas mediante PG ante múltiples casos posibles y complejos de resolución de problemas mediante RR.NN.AA de diferente naturaleza y arquitectura.

El algoritmo de extracción de reglas, como se ha comentado anteriormente, se basa en la PG para la generación automática de algoritmos y expresiones que se codifican en forma de árbol. Para realizar este árbol es necesario que se le especifique qué nodos van a poder ser terminales (hojas) y cuales no terminales. Al especificar los operadores terminales y no terminales es necesario especificar un tipado: cada nodo tendrá un tipo y, además, los no terminales exigirán un tipo determinado a sus hijos [MONT 95]. Con esto se asegura que los árboles generados satisfagan la gramática especificada por el usuario. Además, ambos conjuntos de operadores especificados

deben cumplir dos requisitos: cerradura y suficiencia. Es decir, que debe ser posible construir árboles correctos con los operadores especificados y que la solución al problema (la expresión que se quiere encontrar) pueda expresarse por medio de esos operadores. Como operadores se han utilizado, dependiendo del problema a resolver, funciones matemáticas y trigonométricas (seno, coseno, raíz cuadrada, etc.), operadores lógicos AND, OR, NOT, IF-ELSE sobre números reales y booleanos y las operaciones típicas +,-,* y % que es la división protegida (evitar la división por cero). Así, para las tareas de clasificación se han seleccionado básicamente los operadores lógicos, prescindiendo de las operaciones matemáticas. En cambio, para la predicción sí es preciso utilizarlas, debido a la naturaleza de las salidas que tienen las RR.NN.AA que tratan estos tipos de problemas.

7.2 Función "Mux"

Como primer análisis experimental de comprobación del correcto funcionamiento del algoritmo inductivo se ha seleccionado el problema de la" multiplexación de 6 a 1". El por qué elegir este problema es debido a que el espacio de búsqueda es finito y bien conocido. Como ejemplo de utilización de este problema se puede citar a Rodríguez [RODR 99], en su tesis expone dicho problema y un análisis de resultados después de aplicar su modelo "MOGP" de análisis de multi-objetivos basado en IS. De forma similar a lo expuesto por Rodríguez se va a exponer una detallada serie de pruebas relativas a los parámetros que intervienen en el algoritmo. El problema consiste en el aprendizaje de la tarea de decodificación de una dirección binaria y la obtención del valor binario del registro indicado en dicha dirección (ver figura 24). La entrada consiste en K (en este caso 2) bits de dirección A_i y 2^k (en este caso 4) bits de datos D_j.

En este caso el espacio de búsqueda está limitado a $2^6 = 64$ combinaciones de las 6 variables de entrada. Se ha creado un fichero de patrones entrada-salida con todas estas combinaciones que será utilizado para la fase de test del sistema. De estas 64 secuencias posibles se han descartado al azar 6 de ellas (el 9.4% de los casos) y las 58 restantes conforman el fichero de patrones de aprendizaje. Se ha realizado esto para

incrementar el nivel de complejidad del problema y para comprobar, en cierta manera, la capacidad de abstracción y generalización que puede ofrecer el algoritmo implementado.

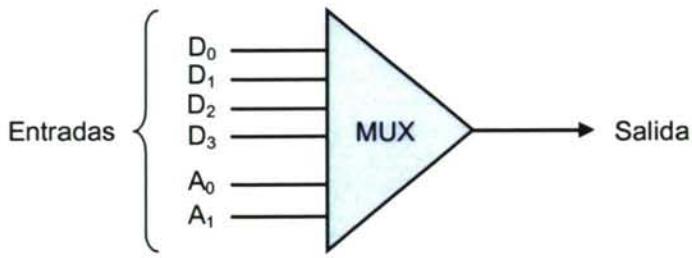


***Fig. 24** - Función MUX 6 a 1*

Para las pruebas, se han seleccionado como elementos no terminales la función de decisión "IF-THEN-ELSE" y los operadores lógicos "AND", "OR" y "NOT". Además, se ha establecido como "meta-regla" que todas las entradas y la salida sean de naturaleza lógica.

En todas las pruebas se ha establecido una altura máxima de árbol de 9, de esta forma se permiten individuos con bastante complejidad algorítmica y, si la combinación de los parámetros restantes utilizados influyen aumentando el tamaño de los individuos, aparecerán diferencias importantes en cuanto a tiempo de simulación.

En la fase de establecimiento de parámetros del algoritmo de PG se han probado varias combinaciones para comprobar cuáles producen mejores resultados.

En el fichero de entrada las variables X_i correspondientes a las entradas de la figura 24 se han codificado según las siguientes equivalencias:

$$X_1 = D_0 \quad X_2 = D_1 \quad X_3 = D_2 \quad X_4 = D_3 \quad X_5 = A_0 \quad X_6 = A_1$$

Hay que destacar que se consigue el ajuste perfecto a la solución del problema y uno de los algoritmos óptimos obtenido es el siguiente:

```
IF X5 THEN
        (IF X6 THEN
                X4
         ELSE
                X3)
 ELSE
```

(IF X_6 THEN
X_2
ELSE
X_1)

Debido al espacio de búsqueda tan reducido y bien delimitado se han probado las siguientes combinaciones de parámetros de PG:

Algoritmo de creación de población	Completo Parcial Intermedio
Algoritmo de selección	Torneo Ruleta Sobrante estocástico Universal estocástica Muestreo determinístico
Algoritmo de mutación	Subárbol Puntual
Usar estrategia elitista	Si No
Utilizar cruces no destructivos	Si No
Tasa de cruces	98% 95% 90% 50%
Probabilidad de mutación	1% 5% 10% 50%
Probabilidad de selección de no terminal	98% 95% 90% 50%
Tamaño de la población	10 100 500 1000
Nivel de parsimonia	0 0.0005 0.001 0.005 0.01

Con esto se pretende comprobar cómo afectan los diferentes parámetros a la búsqueda de reglas. Se han realizado pruebas fijando ciertos valores que producen buenos resultados y variando otros de uno en uno. Además, se han realizado, por cada combinación, 5 ejecuciones independientes y se ha tomado la media de ellas. Se han considerado el número de generaciones que ha necesitado el sistema para hallar una solución que produzca el ajuste del 100% de los casos de test, el número de generaciones adicionales para obtener el óptimo y el tiempo que ha tardado en total. En las siguientes tablas de resultados de las pruebas el campo titulado "Ajuste al 100%" se hace referencia al número medio de generaciones que ha necesitado el algoritmo para alcanzar el ajuste del 100% de los casos. El campo "Optimizado" hace referencia al número medio de generaciones adicionales que se han necesitado, partiendo del ajuste del 100%, para alcanzar un algoritmo de ajuste óptimo, mínimo número de expresiones. El campo "Tiempo Total" hace referencia al tiempo medio necesario para alcanzar un algoritmo de 100% de ajuste y óptimo.

Para ello, las pruebas se han llevado a cabo en un equipo PC con procesador Intel Pentium IV a 1.5 GHz, con 256 Mb de memoria RAM y sistema operativo Windows 2000. Además, el programa era el único proceso en ejecución.

Después de varias pruebas se han fijado los siguientes parámetros:

Algoritmo de creación:	Intermedio
Algoritmo de selección:	Torneo
Algoritmo de mutación:	Subárbol
Usar estrategia elitista:	Si
Utilizar cruces no destructivos:	No
Tasa de cruces:	95%
Probabilidad de mutación:	5%
Probabilidad selección no terminal:	90%
Tamaño de la población:	1000
Nivel de parsimonia:	0.001

Con esta configuración se consigue un número de generaciones medio de 12.45 en 31.3 segundos para el ajuste del 100% de los casos y 19.09 generaciones en 43.4 segundos para alcanzar la solución óptima.

El primer parámetro a variar va a ser el tipo de algoritmo de creación de la población.

Algoritmo de creación	Ajuste al 100%	Optimizado	Tiempo Total
Completo	11.2	28.5	1 min. 15 seg.
Parcial	18.7	27.1	51.3 seg.
Intermedio	12.4	19.1	43.4 seg.

El intermedio produce los mejores resultados y se mantiene la configuración inicial. Hay que destacar que en el caso del algoritmo parcial, de las 5 ejecuciones la evolución se ha estancado 2 veces donde se han alcanzado 1000 generaciones sin mejorar por lo que se han tenido que repetir. En cambio, el completo producía buenos resultados pero creando árboles de elevada profundidad, de ahí el tiempo medio elevado de ejecución que tiene.

Según el tipo de algoritmo de selección se consiguen los siguientes resultados:

Algoritmo de selección	Ajuste al 100%	Optimizado	Tiempo Total
Torneo	12.4	19.1	43.4 seg.
Ruleta	17.1	29.3	57.1 seg.
Sobrante estocástico	13.5	37.3	1 min. 14 seg.
Universal estocástica	12.5	20.2	46.2 seg.
Muestreo determinístico	12.2	19.0	44.4 seg.

En este caso, el torneo, la universal estocástica y el muestreo determinístico producen resultados muy similares, siendo cualquiera de los tres válido, por lo tanto se sigue manteniendo la configuración fija.

Según el tipo de algoritmo de mutación se tiene:

Algoritmo de mutación	Ajuste al 100%	Optimizado	Tiempo Total
Subárbol	12.4	19.1	43.4 seg.
Puntual	12.2	20.1	37.6 seg.

Se puede observar que ambos producen resultados similares en cuanto al número de generaciones necesarias para alcanzar el resultado, sin embargo la utilización del tipo puntual reduce la complejidad del sistema. Se obtiene aproximadamente un beneficio del 13% de reducción de tiempo de ciclo. Es por tanto lógico pensar en utilizar este tipo de algoritmo de mutación y cambiar la configuración fija. De todas formas no es necesario repetir las pruebas puesto que serán resultados muy similares pero con una reducción relativa aproximada del 13% en el tiempo empleado.

El siguiente valor a probar es la estrategia elitista:

Estrategia elitista	Ajuste al 100%	Optimizado	Tiempo Total
Si	12.2	20.1	37.6 seg.
No	19.2	39.1	1 min. 2 seg.

El no asegurar que se mantenga el mejor individuo obtenido produce peores resultados.

Se va a probar la utilización de los cruces no destructivos:

Utilizar cruces no destructivos	Ajuste al 100%	Optimizado	Tiempo Total
Si	11.9	17.6	52.1 seg.
No	12.2	20.1	37.6 seg.

Su utilización mejora el número de generaciones necesarias, pero a costa de incrementar la complejidad general del algoritmo. Se va a mantener la configuración fija no incorporando los cruces no destructivos.

El siguiente paso a realizar es la prueba de varios valores de tasa de cruces:

Tasa de cruces	Ajuste al 100%	Optimizado	Tiempo Total
98%	12.5	19.1	38.4 seg.
95%	12.2	20.1	37.6 seg.
90%	13.9	22.1	43.0 seg.
50%	21.3	32.3	59.0 seg.

De estas pruebas se puede concluir que valores de tasa de cruce entre el 98% y el 90% son los que mejores resultados producen.

A continuación, se probará la probabilidad de mutación:

Probabilidad de mutación	Ajuste al 100%	Optimizado	Tiempo Total
1%	13.1	23.4	44.9 seg.
5%	12.2	20.1	37.6 seg.
10%	14.1	21.5	40.3 seg.
50%	34.3	39.8	1 min. 11 seg.

De estas pruebas se puede concluir que valores de probabilidad de mutación entre el 1% y el 10% son las que mejores resultados producen.

Ahora se probará la probabilidad de selección de no terminal:

Probabilidad selección no terminal	Ajuste al 100%	Optimizado	Tiempo Total
98%	14.1	22.4	41.9 seg.
95%	12.2	20.1	37.6 seg.
90%	12.9	23.2	42.7 seg.
50%	21.1	27.1	50.3 seg.

De estas pruebas se puede concluir que valores de probabilidad de selección de no terminal entre el 98% y el 90% son las que mejores resultados producen.

El siguiente valor a probar es el tamaño de la población. Hasta ahora se ha utilizado un tamaño de población de 1000, debido principalmente a que se obtenían los ajustes deseados en tiempos prudenciales. Si no se han hecho todas las pruebas con valores inferiores es porque apenas se podían distinguir, en función del tiempo, las diferencias al aplicar las diferentes configuraciones de parámetros vistas anteriormente.

Tamaño de la población	Ajuste al 100%	Optimizado	Tiempo Total
10	--	--	--
100	28.3	37.4	23.3 seg.
500	10.2	15.4	21.2 seg.
1000	12.2	20.1	37.6 seg.

Aquí se puede concluir que con 500 individuos se consiguen mucho mejores resultados, tanto en número de generaciones necesarias como en tiempo empleado. En cambio, con 10 individuos no se consigue llegar a resultados aceptables, obteniendo,

por término medio, que después de 1000 generaciones en 1 min. 30 seg. no se mejoraba del 79.31% de ajuste.

El último parámetro a analizar es el nivel de parsimonia. Este es un parámetro muy importante porque es el que fija el nivel de optimización dinámica de las reglas obtenidas a medida que se van produciendo generaciones de individuos. Es, por tanto, muy importante buscar un buen valor para conseguir un equilibrio entre reglas bien y mal optimizadas.

Nivel de parsimonia	Ajuste al 100%	Optimizado	Tiempo Total
0	15.3	--	--
0.0005	13.3	29.9	45.4 seg.
0.001	12.2	20.1	37.6 seg.
0.005	13.1	20.8	40.3 seg.
0.01	--	--	--

Como se puede observar, si se elimina el nivel de parsimonia no existe optimización y las reglas obtenidas son en algunos casos de elevada profundidad, además de ralentizar el proceso ya que se tienen individuos con valores de profundidad elevados. En cambio, si el nivel de parsimonia aumenta hasta 0.01 se queda el algoritmo estancado y no se produce la evolución de la población. Concretamente, de las 5 simulaciones realizadas en todas ellas se seleccionaba una única variable de entrada y esta era la salida. De esto se puede concluir que el árbol, al tener altura 1, es el que mejor error produce, debido al nivel de parsimonia tan alto. Cualquier individuo que mejore sensiblemente el resultado de ajuste, al ser multiplicada su altura por el nivel de parsimonia, se convierte en el peor y desaparece. Por tanto, se consigue eliminar la evolución de la población.

Como conclusiones finales a las pruebas llevadas a cabo para el problema del "MUX 6 a 1" se han examinado todos los parámetros que intervienen en la ejecución del módulo de PG y la forma de funcionamiento del algoritmo inductivo, obteniendo resultados prometedores. Además, ahora se conocen aquellos parámetros más sensibles

a su funcionamiento final y limitándolos a unos pocos valores. Por tanto, el número de pruebas de los restantes problemas se reduce de forma considerable.

En todos los problemas restantes se partirá de la configuración fija expuesta para este problema y se ajustarán unos pocos valores de los parámetros más sensibles de variar el resultado final. Y siempre en los niveles vistos anteriormente.

7.3 "Monk Problems"

Se trata de determinar si un objeto definido por seis características es un robot o no. Para ello, se dispone de una secuencia de pares "entrada-salida" que determinan dicha clasificación. Se han utilizado 3 problemas diferentes, pero con la misma idea subyacente, y cada problema con diferente clase de complejidad.

Estos problemas y varios resultados comparativos han sido obtenidos del artículo de Thrun [THRU 91], donde se presenta la definición y el alcance de estos problemas artificiales así como diferentes tablas comparativas de resultados obtenidos por diferentes métodos de aprendizaje. En las tablas de resultados que se presentarán posteriormente se ha utilizado como base los resultados que Duch presenta en su artículo [DUCH 00], que han sido obtenidos del artículo de Thrun y de nuevas técnicas de extracción que él presenta, como el "método C-MLP2LN". Los métodos que se han utilizado para comparar los resultados, junto con sus autores, son los siguientes:

AQ17-DCI AQ17-HCI AQ17-FCLS AQ17-NT AQ17-GA	J. Bala, E. Bloedorn, K. De Jong, K. Kaufman, R.S. Michalski, P. Pachowicz, H. Vafaie, J. Wnek, J. Zhang
Assistant Pro.	B. Cestnik, I. Kononenko, I. Bratko
MFOIL	S. Dzeroski
ID5R IDL ID5R-hat TDIDT	W. Van de Velde
ID3 AQR CN2 CLASSWEB 0.10 CLASSWEB 0.15	J. Kreuziger, R. Hamann, W. Wenzel

CLASSWEB 0.20	
PRISM	S. Keller
ECOWEB	Y. Reich, D. Fisher
MLP MLP+regularization	S. Thrun
Cascade Corelation	S. Fahlman
C-MLP2LN rules	W. Duch, R. Adamczak, K. Grabczewski

Con estos problemas no se han utilizado patrones procedentes de ejecuciones de RR.NN.AA entrenadas para resolver estos problemas, en primer lugar porque resuelven los dos primeros problemas con un 100% de acierto y con un 97.2% el último, como se puede observar en el método MLP "Multi-Layer Perceptron" de los resultados que se mostrarán a continuación, y en segundo porque estos tres problemas están únicamente pensados para verificar el correcto funcionamiento de las técnicas de aprendizaje y de los algoritmos de extracción de reglas, y sirven como medida comparativa entre los diferentes métodos de extracción existentes.

Las seis características, y sus posibles valores, que puede tener un robot son:

- Forma de la cabeza (X_1):
  - Redonda (1)
  - Cuadrada (2)
  - Octogonal (3)
- Forma del cuerpo (X_2):
  - Redondo (1)
  - Cuadrado (2)
  - Octogonal (3)
- Está riéndose (X_3):
  - Si (1)
  - No (2)
- Está sosteniendo en la mano (X_4):
  - Espada (1)
  - Globo (2)
  - Bandera (3)
- Color del chaleco (X_5):
  - Rojo (1)
  - Amarillo (2)

- ❖ Verde (3)
- ❖ Azul (4)

➢ Tiene corbata (X_6):

- ❖ Si (1)
- ❖ No (2)

Las fórmulas que rigen el funcionamiento de cada uno de los tres problemas son las siguientes:

- ✓ Forma Cabeza (X_1) = Forma Cuerpo (X_2) ∨ Color Chaleco (X_5)
- ✓ Exactamente 2 de las 6 características tienen los primeros valores
- ✓ ¬ (Forma Cuerpo (X_2) = *Octogonal* (3) ∨ Color Chaleco (X_5) = *azul* (4)) ∨ (Sosteniendo (X_4) = *espada* (1) ∧ Color Chaleco (X_5) = *verde* (3))

En total, con la combinación de los seis atributos, existen 432 sucesos posibles de ellas. Para el primer problema existen 124 casos, para el segundo 169 y para el tercero 122, de las cuales el 5% están mal clasificados. De ésta forma, en el último problema se añade ruido en los datos, lo cuál dificulta el proceso de extracción de una forma pronunciada.

7.3.1 Caso 1

Para este caso, las expresiones obtenidas de la ejecución del algoritmo de extracción, consiguen el ajuste del 100% de los casos.

La expresión obtenida es la siguiente:

$$(X_5=1)\ \mathrm{OR}\ (X_1=X_2)$$

Una comparativa entre los resultados obtenidos para este primer problema y otros existentes obtenidos del trabajo de Duch [DUCH 00] se encuentra reflejada en la siguiente tabla.

Método	Nivel de Ajuste
Aquí propuesto	100%
AQ17-DCI	100%
AQ17-HCI	100%
AQ17-GA	100%

Assistant Pro.	100%
mFOIL	100%
ID5R	79.7%
IDL	97.2%
ID5R-hat	90.3%
TDIDT	75.7%
ID3	98.6%
AQR	95.9%
CN2	100%
CLASSWEB 0.10	71.8%
CLASSWEB 0.15	65.7%
CLASSWEB 0.20	63.0%
PRISM	86.3%
ECOWEB, ext	82.7%
MLP	100%
MLP+regularization	100%
Cascade Correlation	100%
FSM, fuzzy rules	94.5%
SSV, crisp rules	100%
C-MLP2LN rules	100%

7.3.2 Caso 2

Para este caso, las expresiones obtenidas de la ejecución del algoritmo de extracción consiguen, con una altura máxima de árbol de 50, un ajuste del 98.82% de acierto en el entrenamiento y del 91.9% en el proceso de test.

La expresión obtenida es la siguiente:

```
(IF (
 (IF (X5=1)
 THEN (X1=7)
 ELSE ((X1=1) OR
  (IF (X6=X4)
  THEN (NOT (X2=X4))
  ELSE (X2=1)))) AND
 (IF (X6=1)
 THEN (X1=2)
 ELSE (X4=1)))
THEN (X3=2)
ELSE
 (IF (
  (IF (X6=1)
  THEN
   (IF (X5=1)
   THEN
    (IF (NOT (X1=1))
```

```
      THEN (X3=2)
      ELSE (X6=X3))
    ELSE (X6=1))
  ELSE (X5=1)) AND
  (IF (X6=X4)
  THEN (X5=X1)
  ELSE (TRUE)))
 THEN
  (IF (
   (IF (X6=1)
   THEN
    (IF (X5=1)
    THEN (NOT (X5=X2))
    ELSE ((X1=1) OR
     (IF (X6=X4)
     THEN (NOT (X2=X4))
     ELSE (X2=1))))
   ELSE (X5=1)) AND (NOT
   (IF (X2=X4)
   THEN (X6=2)
   ELSE ((X2=1) AND (X1=1)))))
  THEN (X3=2)
  ELSE ((X3=1) AND (NOT (X1=1))))
 ELSE ((X3=1) AND ((X6=2) AND
  (IF (X1=1)
  THEN (X1=1)
  ELSE ((X4=1) OR
   (IF (X5=1)
   THEN (X2=X4)
   ELSE (X2=1))))))))
```

Método	Nivel de Ajuste
Aquí propuesto	91.9%
AQ17-DCI	100%
AQ17-HCI	93.1%
AQ17-GA	86.8%
Assistant Pro.	81.5%
mFOIL	69.2%
ID5R	69.2%
IDL	66.2%
ID5R-hat	65.7%
TDIDT	66.7%
ID3	67.9%
AQR	79.7%
CN2	69.0%
CLASSWEB 0.10	64.8%
CLASSWEB 0.15	61.6%
CLASSWEB 0.20	57.2%
PRISM	72.7%
ECOWEB, ext	71.3%

MLP	100%
MLP+regularization	100%
Cascade Correlation	100%
FSM, fuzzy rules	79.3%
SSV, crisp rules	80.6%
C-MLP2LN rules	100%

7.3.3 Caso 3

En este último caso, debido al 5% de errores introducidos en la secuencia de patrones de entrenamiento, el proceso de extracción debe discernir entre estos valores erróneos y los correctos, lo cual incrementa el nivel de dificultad de este tercer problema.

Se han probado varias combinaciones de elementos terminales y operadores utilizados siendo los siguientes los que mejores resultados producen:

Terminales $X_1, X_2, X_3, X_4, X_5, X_6, 1, 2, 3, 4$

Las seis variables y los cuatro valores que pueden tomar, todos de tipo real.

Funciones AND, OR, NOT, =, <>

Son operadores booleanos y relacionales para ver el valor de las variables. Todos de tipo booleano.

Para estos parámetros, las expresiones obtenidas de la ejecución del algoritmo de extracción consiguen un ajuste del 93.44% en el entrenamiento y un 97.22 en el proceso de test. Esta diferencia reside en que el proceso de entrenamiento está realizado con el 5% de valores erróneos y el proceso de test está realizado con los valores correctos. De esto se puede deducir que el algoritmo de extracción ha podido en cierta manera discernir entre los valores erróneos y los correctos. La expresión obtenida es la siguiente:

$$(X_2 <> 3) \text{ AND } (X_5 <> 4)$$