18
20).

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

ANALISIS COMPARATIVO DE METODOS MATEMATICOS PARA LA CLASIFICACION DE OBJETOS

T E S I S

QUE PARA OBTENER EL TITULO DE:

M A T E M A T I C A

P R E S E N T A

SUSANA HEDDING SALGADO

DIRECTOR DE TESIS: DR. PEDRO MIRAMONTES VIDAL

1997

M. en C. Virginia Abrín Batule
Jefe de la División de Estudios Profesionales de la
Facultad de Ciencias
P r e s e n t e

Comunicamos a usted que hemos revisado el trabajo de Tesis:
ANALISIS COMPARATIVO DE METODOS MATEMATICOS PARA LA CLASIFICACION DE OBJETOS

realizado por HEDDING SALGADO SUSANA

con número de cuenta 8514212-1 , pasante de la carrera de MATEMATICAS

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis
Propietario Dr. PEDRO MIRAMONTES VIDAL
Propietario Dr. FAUSTINO SANCHEZ GARDUÑO
Propietario M. en C. MARIA GUADALUPE ELENA IBARGUENGOITIA
Suplente Dra. MARIA de LOURDES ESTEVA PERALTA
Suplente M. en C. JOSE LUIS GUTIERREZ SANCHEZ

Consejo Departamental de Matemáticas
DR. MANUEL FALCONI MAGAÑA

MATEMÁTICAS

Dedico este trabajo que significa una meta más lograda en mi carrera académica a mis padres y hermanos que me han apoyado en todo momento, además de darme su cariño, ayudándome así a seguir adelante y a Ismael que también ha significado una fuerte guía e inspiración para alcanzar lo que ahora tengo.

También lo dedico con mi más profundo respeto y admiración a Lupita a Faustino y a Javier que son ejemplo a seguir porque impulsan a valorar el significado de ser matemática.

Agradecimientos

Deseo agradecer a Pedro Miramontes Vidal, que me asesoró en el desarrollo conceptual de la tesis, guiándome sabiamente en esta etapa universitaria.

También agradezco a Faustino Sánchez Garduño, Lupita Ibargüengoitia González, Lourdes Esteva Peralta y José Luis Gutiérrez Sánchez, sinodales, que dieron su opinión para mejorar este trabajo.

Por lo que respecta a la ayuda para lograr la impresión de este escrito a Eduardo Mora, Javier Páez, Georgina González, Francisco Tenorio, Carmen Ramos, Patricia Hernández, César Suosa, Luis Angel, y Mirna Ocampo quienes de una manera desinteresada me otorgaron lo que estaba a su alcance para que yo continúe adelante.

Contenido

Introducción

En labores muy sencillas puede haber implícita una serie de datos que son útiles para interesantes aplicaciones dentro de las matemáticas. Un etnógrafo, de acuerdo a rasgos fisonómicos de las personas (color de piel, estatura, tipo de ojos, labios, nariz, etc.) determina el grupo racial al que pertenece una persona; un astrónomo, observa el tamaño, luminosidad, color, tipo de movimiento, etc. para indicar qué clase de cuerpo celeste está estudiando; un médico, guiándose por las respuestas al cuestionario acerca de los síntomas que presenta el enfermo, por la auscultación, análisis clínicos, etc. concluye cual es la enfermedad que padece el paciente. Incluso en algo tan sencillo como distinguir varios libros entre sí, sólo necesitamos enlistar su grosor, color, ancho, largo, tema, etc. para separarlos en, digamos, diccionarios y atlas.

Las actividades anteriores tienen en común que son procedimientos en donde dado un *objeto* (una persona, un cuerpo celeste, un paciente o un libro respectivamente en cada uno de los casos anteriores) se reúne información de dicho objeto (las *características o rasgos* del mismo) para diferenciarlo de otros objetos y asignarlo a cierta *clase* (en los ejemplos anteriores serían el tipo racial, el cuerpo celeste del que se trata, la enfermedad que se tiene o el tipo de libro); es decir, que identificamos si se sigue un patrón de comportamiento permitiendo así un *reconocimiento de patrones* y una *clasificación* de los objetos estudiados.

Para lograr la *clasificación y reconocimiento de patrones (RP)* se han desarrollado varios métodos matemáticos, algunos de ellos son: *teoría de testores, clasificación bayesiana, redes neuronales (RN), análisis discriminante y análisis de cúmulos* (los últimos dos métodos no serán vistos en este trabajo).

Los fundamentos y estilo de cada uno de los métodos mencionados son muy diferentes; la teoría de testores se basa en un formalismo lógico - com-

binatorio para lograr la *reducción de variables* y la *clasificación e identificación de objetos*. En cambio la clasificación bayesiana se basa en el cálculo de probabilidades y, específicamente, en el teorema de Bayes para llevar a cabo la separación de objetos en sus respectivas clases. Por otro lado, las redes neuronales surgen de la modelación de la organización básica de neuronas biológicas y constan de las llamadas *"neuronas artificiales"* que interconectadas de diversas formas, llevan a cabo, entre otras actividades, la clasificación de objetos.

Los objetivos del presente trabajo son: dar a conocer los principios básicos de los tres métodos de clasificación anteriores, enfatizando en cierto tipo de redes, los perceptrones multicapa (PMC); observar las marcadas diferencias entre los métodos; conocer qué es la separabilidad lineal entre clases ya que esta es una información de mucha ayuda para saber cuál método utilizar en determinados problemas; mostrar la mayor capacidad para clasificar de un PMC comparado con un perceptrón simple dando como ejemplo la solución de la función XOR (O- exclusivo) y dar la lista de varias aplicaciones al realizar la clasificación de objetos.

La organización de este trabajo es como sigue.

En el Capítulo 1, la primera sección contiene conceptos e ideas básicas del reconocimiento de patrones, la segunda sección menciona un aspecto esencial del mismo como lo es determinar la *separabilidad lineal y no lineal* de un grupo de objetos con la explicación de cada una de ellas.

El Capítulo 2 se refiere a métodos para el reconocimiento de patrones; en la primera sección se da la definición del método de teoría de testores y algunos comentarios del mismo y la segunda sección trata sobre la clasificación bayesiana; a partir de la tercera se explica un método de red neuronal (RN), el perceptrón simple: su definición, historia, un modelo de RN, tipos de RN, el aprendizaje de RN, capacidades y limitaciones, lo que hay que tomar en cuenta al realizar una RN y algunas aplicaciones.

El Capítulo 3 en su primera sección contiene la historia en la investigación de PMC; la segunda, es una descripción de arquitectura y lenguaje matemático, algoritmos, capacidades y limitaciones acerca de los PMC.

En el Capítulo 4, finalmente, se comentará acerca de las capacidades y limitaciones de los métodos anteriores para tener un criterio que nos ayude a elegir el más adecuado para problemas que se solucionan mediante el uso de los mismos, además de algunas aplicaciones de los PMC.

Capítulo 1

Reconocimiento de patrones

1.1 Descripción del problema

Partiendo de un conjunto inicial de objetos Ω, a cada objeto lo llamaremos $O_i, i = 1, ..., n$ y cada uno será representado por un *vector de m rasgos* $(X_1, ..., X_m)$, donde los componentes de este vector sólo serán números, por ejemplo, -1, 0, 1, 4, etc.,(aunque pueden tomar valores no numéricos como: alto, verde,mamífero, etc.). Convengamos en que a cada objeto le corresponde uno y sólo un vector. Cada vector puede ser identificado como un punto en el espacio *m-dimensional* (pues hay m rasgos) y a tal espacio se le nombra *espacio de rasgos*. La detección de categorías o clases en Ω, estudiando los vectores $(X_1, ..., X_m)$ es la primera etapa del problema de *RP*. Dicha etapa pertenece a la estadística multivariada y se realiza mediante el análisis de cúmulos, sólo que no es el objetivo de este trabajo, para consultarlo (ver [Ralston y Reilly, 1993; Beale y Jackson, 1991, Helmut Späth, 1979]).

Si se da un nuevo objeto O_r que no pertenece al conjunto inicial Ω el problema que nos planteamos es el de decidir a cuál de las clases detectadas pertenece. Esta labor es la segunda etapa de un problema de *RP* y se le llama *clasificación*.

Describamos de otra forma lo anterior: El *reconocimiento de patrones (RP)* es el proceso mediante el cual dada una colección de objetos, cada uno de ellos identificado por sus características, sean separados los unos de los otros en grupos o clases donde cada objeto pueda pertenecer solamente a una de ellas. Si los objetos son del mismo grupo es porque tienen ciertas características en común; a esas *características* se les llama también *rasgos*

o *atributos*. Los *rasgos* son la medida tomada sobre el patrón de entrada que se va a clasificar. Además, los rasgos describen a un objeto dando las características que pueden ser o no de utilidad para distinguirlo de cualquier otro.

Un sistema de RP tiene dos etapas o problemas a resolver:

1a.. Extracción de rasgos.

2a.. Clasificación de objetos.

Uno de los objetivos principales de un sistema de reconocimiento de patrones es clasificar objetos.

1.2 Separabilidad lineal

Pensemos en el caso de que los objetos son representados en su vector de rasgos por componentes sólo con valores numéricos. Entonces, cada objeto puede representarse como un punto en un espacio de m dimensiones mediante su vector de rasgos. La manera en que los objetos quedan representados en el espacio de rasgos, divide los problemas de RP en dos grandes grupos: los que tienen *separabilidad lineal* y los que carecen de ella. Los problemas con separabilidad lineal son aquellos en los que con una estructura lineal (una línea recta, un plano o un hiperplano) se pueden separar completamente las regiones del espacio que contienen a los representantes de clases ajenas; de otro modo son problemas sin separabilidad lineal. La Figura 1.1 muestra ejemplos para ambos casos cuando $m = 2$, es decir cuando los objetos están en un plano bidimensional.

Los clasificadores que sólo son capaces de resolver problemas con separabilidad lineal son llamados *clasificadores lineales*.

En general, si las clases son linealmente separables, entonces las clases son separadas por un hiperplano *(m-1) dimensional*, sólo que las clases están dentro de un espacio de dimensión m.

En un problema de clasificación de objetos, una vez que se tienen determinadas las clases, lo siguiente es averiguar si hay o no separabilidad lineal, para saber qué tipo de método utilizar en la clasificación de un nuevo objeto. La demostración de dicha afirmación no se verá aquí. (ver [Minsky y Papert, 1969]).

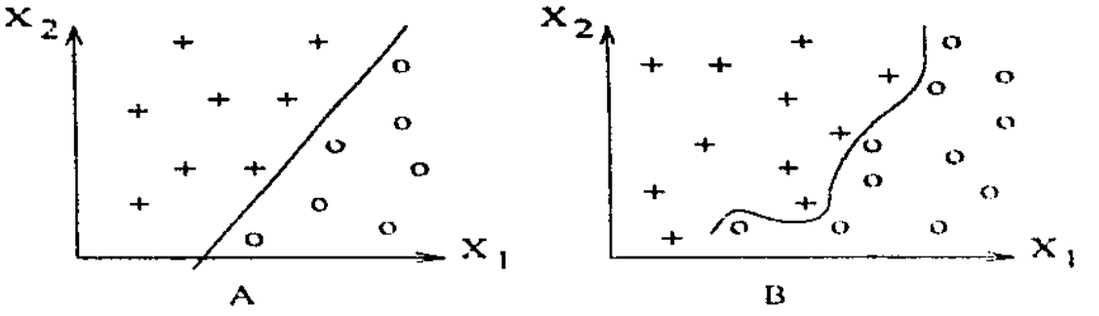

Figura 1.1: Problemas con diferentes tipos de separabilidad. A)Con separabilidad lineal. B)Sin separabilidad lineal. Las cruces y los círculos representan objetos que pertenecen a clases diferentes.

1.2.1 Aspectos geométricos de la separabilidad lineal

Para determinar si clases de objetos tienen separabilidad lineal o carecen de ella, se utiliza como criterio la *envoltura convexa*, un ejemplo de estas envolturas se puede ver en la Figura 1.2.

Definición 1.1 *La envoltura convexa de un conjunto de puntos S en R^p, es la frontera del dominio convexo más pequeño en R^p que contiene a S.*

Dos clases son linealmente separables sólo si sus envolturas convexas son ajenas.

Los objetos pueden ser vistos como puntos en el espacio Euclidiano d-*dimensional*, R^p, cuyas coordenadas están dadas por los rasgos. Así, podemos definir a una envoltura convexa en los siguientes términos:

El siguiente resultado indica cuando un problema es linealmente separable:

Teorema 1.1 *([Beale y Jackson, 1991; Preparata y Shamos, 1988]) Sean C_1 y C_2 dos envolturas convexas, si $C_1 \cap C_2 = \emptyset$, entonces el problema es linealmente separable.*

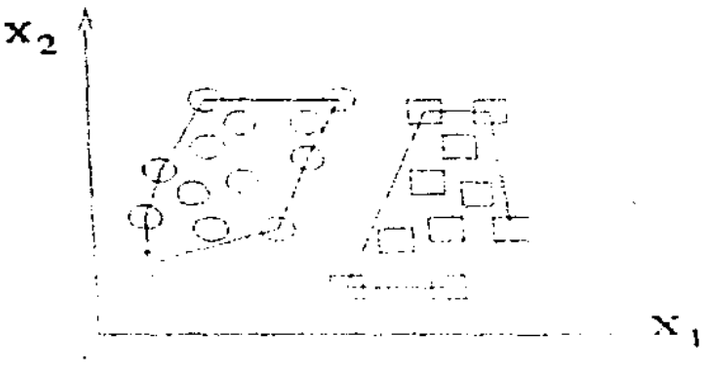

Figura 1.2: Dos clases son linealmente separables sólo si sus envolturas convexas son disjuntas. Los círculos y los cuadrados representan objetos que pertenecen a clases diferentes. Las líneas continuas son la envoltura convexa de su clase respectiva.

La demostración de este teorema se puede consultar en [Beale y Jackson, 1991; Preparata y Shamos, 1988].

Hay distintos métodos para saber cuál es la envoltura convexa de un determinado conjunto de puntos sólo que éste es un problema difícil de resolver cuando se tienen más de dos dimensiones y no es el objetivo principal del presente trabajo dar estos algoritmos; para ello se puede consultar [Preparata y Shamos, 1988].

Existe un algoritmo diseñado en el Centro de Geometría Computacional de la Universidad de Minnesota llamado QHull que permite conocer la envoltura convexa para problemas multidimensionales, lo cual es de mucha ayuda. El programa es público y se puede obtener en el internet:
(http://www.geom.UMN.Edu/Software/download).

Capítulo 2

Métodos para reconocimiento de patrones

2.1 Teoría de testores

Al describir objetos o fenómenos de un cierto universo, para diferenciarlos, existe el problema de determinar la importancia de distintas variables que los caractericen. Este problema es conocido como el problema de *reducción de variables* y para resolverlo hay, entre otros, métodos estadísticos y probabilísticos; además, dentro de la computación se cuenta con otras técnicas de la matemática discreta como la llamada *teoría de test o de testores* que no emplea métodos estadísticos sino argumentos de lógica y combinatoria.

La teoría de testores además de resolver el problema de reducción de variables es útil para la *clasificación e identificación de objetos*. Esta teoría fue propuesta originalmente por Zhuravliov en la ex-Unión Soviética y fue extendida por el grupo de Ruiz Shulcloper en el Instituto de Matemáticas Aplicadas y Cibernética de la Academia de Ciencias de Cuba. Este problema es importante pues tiene aplicaciones en: clasificación, *RI*, diagnóstico automatizado, comprensión de información, etc.,(ver [Morales y Miramontes, 1988]).

A los elementos que intervienen en el problema de la *selección de variables* los denotaremos como:

Sea $\{O_j\}_{j=1}^{n}$ una colección de n objetos distintos y donde algún subconjunto de este conjunto básico de objetos lo denotaremos por sus índices $\{J_i\}$.

Sea $\{V_k\}_{k=1}^{m}$ un conjunto de m variables medidas en cada O_j, y en una forma similar, denotemos algún subconjunto de dichas variables como $\{K_i\}$; podemos pensar que un examen clínico (químico u otro semejante) es aplicado a cada objeto y V_k es la respuesta del objeto. Considere aplicar un conjunto de m pruebas o exámenes a cada uno de los objetos en $\{O_j\}$ y registrar las respuestas en una matriz A_{nm}. La entrada $a_{jk} = O_j(V_k)$ en A_{nm} es la $k-$*ésima* variable medida en el objeto j-*ésimo*. Denotemos por $A_{nm}/\{K_i\}$ a la submatriz obtenida considerando sólo las columnas $\{K_i\}$ de A_{nm}.

Con la notación anterior y las siguientes definiciones, en este capítulo veremos cómo resolver el problema de la selección de variables y un ejemplo sencillo de la aplicación del método de la teoría de testores.

2.1.1 Definiciones

Supongamos, por simplicidad, que las entradas de A_{nm} sólo son 0 ó 1 (la teoría de testores puede manejar sólo datos en un conjunto de valores discreto y finito), que significa que cada examen en nuestro conjunto obtiene de cada objeto una respuesta: 0 ó 1. Formalmente, supongamos la siguiente hipótesis:

Hipótesis 2.1 *A_{nm} no tiene renglones repetidos.*

Si dos objetos distintos tienen la misma respuesta para cada una de las pruebas, no podremos distinguirlos entre sí. Entonces, dos renglones idénticos significan dos objetos iguales. La Hipótesis 2.1 excluye esta situación. Por otro lado, si todos los objetos muestran el mismo valor para una variable, entonces esta variable no es de ayuda para apartar los objetos entre sí; la variable tiene un valor informativo de cero. No decimos que la variable no es importante; en una situación médica puede ser muy importante, por ejemplo, la presencia de un virus letal en todos los objetos, pero su prevalencia en todos ellos no es de ayuda para distinguirlos.

Definición 2.1 *Un testor es un subconjunto $\{K_i\}$ de $\{V_k\}$ con la propiedad de que $A_{nm}/\{K_i\}$ no tiene renglones repetidos. Así, podemos decir que un testor es un subconjunto de las variables originales que basta para distinguir*

a objetos diferentes.

Lema 2.1 *A_{nm} tiene al menos un testor.*

La demostración del lema anterior es la siguiente:
De la Hipótesis 2.1 se tiene que el conjunto $\{V_k\}$ es un testor. Esto es así pues A_{nm} no tiene renglones repetidos. Por otro lado, $\{V_k\}$ es subconjunto de $\{V_k\}$ y, $A_{nm}/\{K_i\}$ es exactamente A_{nm} en el caso de que $\{K_i\}$ es igual a $\{V_k\}$. Por lo tanto $\{V_k\}$ cumple con los requisitos de la Definición 2.1 para ser testor.

A continuación definimos un tipo de testores.

Definición 2.2 *Un testor típico es aquél que, siendo un testor, no tiene un testor como un subconjunto propio.*

En otras palabras, si sustraemos una sola variable de un testor típico (si borramos una columna completa a la matriz A_{nm}) éste dejará de ser un testor. Un testor típico es un subconjunto especial de variables ya que es un subconjunto mínimo de variables suficientes para distinguir diferentes objetos y, si quitamos una sola variable de un testor típico, al menos dos objetos diferentes aparecerán como el mismo.
En la literatura podemos encontrar a los testores típicos llamados también como testores mínimos.
Sea $\{\tau_i\}$ el conjunto de todos los testores típicos de una matriz. Si una variable no está contenida en algún elemento de este conjunto, entonces dicha variable es inútil debido a que no se requiere al diferenciar los objetos entre sí. En el otro caso, si una variable está contenida en todos los elementos de $\{\tau_i\}$, entonces esta variable es esencial para la separación de los objetos y, consecuentemente, en su clasificación.
Así, para los casos intermedios, la siguiente definición surge naturalmente:

Definición 2.3 *El peso informativo de la $i-$ ésima variable lo definimos como:*

$$\rho_i = \frac{T_i}{T} \ ,$$

donde T_i es el número de testores típicos que contienen a la variable $i-$ésima y T es el número total de testores típicos.

El peso informativo de las variables nos da un orden en la escala de cero a uno para todas las variables. Podemos rechazar aquellas variables que tienen valores ρ iguales a cero o cercanos a cero; lo cual da un criterio para reducir las variables en un experimento.

Los objetos heredan una clasificación del peso informativo de las variables. Esto se precisa en la siguiente definición.

Definición 2.4 *El peso informativo del $j-$ésimo objeto es:*

$$W(O_j) = \sum_{k=1}^{m} \rho_k \cdot O_j(V_k) \ .$$

El objeto con el más alto peso informativo es llamado el objeto típico.

Ahora supongamos que la matriz de datos está dividida en dos clases ajenas, por ejemplo, si $\{O_j\}$ representa personas, podemos dividirlas en una primera clase de personas enfermas y en otra segunda clase (independiente de la primera clase) de personas sanas. Después usemos el primer renglón en la matriz para escribir a las personas sanas y el resto para las personas enfermas. Por lo tanto, podemos reescribir la Definición 2.1 como sigue:

Definición 2.5 *Un testor es un subconjunto $\{K_i\}$ de $\{V_k\}$ con la propiedad de que en $A_{nm}/\{K_i\}$ cada objeto en la primera clase es diferente de todos los objetos en la segunda clase.*

Como es fácil de imaginar, es posible generalizar esto al caso en que la matriz es construida por objetos pertenecientes a r clases ajenas; de hecho; la Definición 2.1 es el caso particular de la Definición 2.5 cuando $n = r$ y cada clase tiene un solo objeto y así no tenemos que preocuparnos del cuidado del nombre testor en ambas. La generalización de las Definiciones 2.3 y 2.4 es directa: cambiar la ρ por una τ y sustituir la frase "peso informativo" por *peso diferenciante (PD)* de la $i-ésima$ variable. El peso diferenciante mide la capacidad de una variable particular para separar entre clases ajenas de objetos; una variable con cero PD es innecesaria para distinguir entre clases mientras, en el otro extremo, una variable con una calificación de PD igual a uno, separa rigurosamente a las clases. Si lo explicado en las últimas líneas es claro, podemos usar informativo y diferenciante como sinónimos. Entonces, la Definición 2.4 cuando es aplicada por separado a cada clase ajena da una información para un *objeto representativo* de la $s-ésima$ clase. O, en otros términos, un *objeto ideal (típico)* para la $s-ésima$ clase.

2.1.2 Clasificación de objetos

Consideremos que la matriz de datos A_{nm} está dividida en dos clases ajenas K_1 y K_2, donde los objetos $\{O_i\}_1^r$ pertenecen a la clase K_1 y los objetos $\{O_j\}_{j=1}^n$ pertenecen a la clase K_2.

Recordemos que al generalizar la Definición 2.3 y la Definición 2.4 el peso diferenciante de la $i-ésima$ variable es:

$$\tau_i = \frac{T_i}{T},$$

donde T_i es el número de testores típicos que contienen a la variable $i-ésima$, $i = 1, ..., n$, y, T es el número total de testores típicos.

A continuación explicamos lo que se entiende por un *problema de clasificación*.

Sea un objeto nuevo O_α (que no pertenece a la matriz A_{nm}) y que está determinado por m variables que forman su vector de rasgos, es decir, $O_\alpha = (X_1, ..., X_m)$. Para clasificar O_α, es necesario comparar la distancia entre cada uno de los rasgos de dicho objeto con cada uno de los rasgos de los objetos pertenecientes, primero, a la clase K_1 y después a la clase K_2, para así saber con cuál de las dos clases comparte mayor número de atributos iguales y, en consecuencia, a qué clase pertenece.

Formalmente, lo anterior se denota como sigue:

Definamos al operador

$$d(\xi, \delta) = \begin{cases} 1 & \text{si } \xi \neq \delta \\ 0 & \text{en otro caso.} \end{cases}$$

La distancia entre el objeto nuevo, O_α, y cada uno de los objetos, O_i, de la clase K_1 con respecto a cada uno de sus rasgos, la calculamos de acuerdo al operador anterior aplicándolo rasgo por rasgo en cada objeto. De forma semejante se calcula la distancia entre el objeto nuevo y los objetos de la clase K_2 en cada uno de sus rasgos.

Definición 2.6 *La distancia entre el objeto nuevo y la clase K_1 es:*

$$D\left(O_\alpha\right)_{k_1} = \frac{1}{p} \sum_{i=1}^{p} \sum_{k=1}^{m} \tau_k \; d\left(O_{\alpha k}, O_{ik}\right) \;,$$

donde τ_k es el peso diferenciante de la $i-$ ésima variable.

Luego de terminar con la clase K_1 hay que hacer un procedimiento análogo para la clase K_2.

Primero, en el caso de los objetos O_j que pertenecen a la clase K_2, hay que obtener la distancia entre el objeto nuevo, O_α, y cada uno de los objetos O_j con respecto a cada uno de sus rasgos

Extendiendo la Definición 2.6 al caso de la distancia entre el objeto nuevo y la clase K_2 tenemos:

$$D\left(O_\alpha\right)_{k_2} = \frac{1}{n} \sum_{j=1}^{n} \sum_{k=1}^{m} \tau_k \; d\left(O_{\alpha k}, O_{jk}\right) \;.$$

Una vez que ya fueron obtenidas las distancias entre el objeto nuevo con respecto a cada una de las dos clases, dicho objeto pertenece a la clase con la que tenga menos disimilitudes, es decir, a aquella clase en que la distancia entre ambos sea menor.

Las distancias anteriores indican el total de las diferencias del objeto desconocido con cada una de las clases ponderado al peso discriminante de cada una de las variables, es decir indica el número de disimilitudes de las variables del objeto O_n con respecto a cada una de las variables de las clases K_1 y K_2.

Como un caso específico, si las dos distancias anteriores quedan empatadas esto significa que el algoritmo fue incapaz de clasificar al objeto debido a que no hubo una clase que tuviera una distancia menor que la otra con respecto al objeto nuevo O_n.

Además de la forma anterior de clasificar un nuevo objeto existe otra. Esta la explicamos enseguida.

Recordemos que al objeto con el más alto peso informativo se le llama el objeto típico. Obtengamos el objeto típico de la clase K_1 y el objeto típico de la clase K_2 denotándolos como Ot_1 y Ot_2 respectivamente. Calculemos ahora, en una forma similar a la descrita en el método anterior, la distancia entre el objeto nuevo O_n y el objeto típico de la clase K_1, Ot_1; después la distancia entre el objeto nuevo y el objeto típico de la clase K_2, Ot_2. Comparando ambas distancias, el objeto nuevo pertenecerá a la clase que corresponda a la distancia más pequeña. De este modo se concluye la clasificación del objeto nuevo en su clase correspondiente.

Ahora daremos un ejemplo pequeño, pero ilustrativo, de la forma en que se aplica la teoría de testores para clasificar un objeto.

Supongamos que tenemos cinco libros, denotados por O_i, $i = 1, \ldots, 5$. Dichos libros están divididos en dos clases: una de libros de matemáticas, a la cual llamaremos K_1 y otra de libros infantiles a la que nombraremos K_2. De los cinco libros O_1 y O_2 pertenecen a la clase de matemáticas, es decir a K_1, y O_3, O_4 y O_5 pertenecen a la de infantiles, es decir a K_2. Cada libro está descrito de acuerdo a las siguientes variables:

X_1 indica que tiene caricaturas,
X_2 indica que tiene fórmulas matemáticas,
X_3 indica que tiene pasta dura,
X_4 indica que es de papel fino,
X_5 indica que es a colores.

Si el libro satisface lo que indica una variable dada, entonces la variable tendrá el valor de 1 y sino la satisface valdrá 0.

Los valores de las variables en el vector de rasgos para el objeto O_1 son: (0,1,0,0,0), es decir, el libro O_1 no tiene caricaturas, sí tiene fórmulas matemáticas, no tiene pasta dura, no es de papel fino ni es a colores. El vector de rasgos para el objeto O_2 es: (0,1,1,0,0), para el objeto O_3 es: (0,0,1,1,0), para O_4 es: (1,0,0,0,1), para O_5 es: (1,0,1,1,0).

Observemos que en el caso de los libros infantiles la presencia de las variables para clasificar a un objeto dentro de la misma es:

X_1 muy importante,
X_2 poco importante,
X_3 irrelevante,
X_4 casi importante,
X_5 casi importante.

Si se trata de libros de matemáticas la presencia de las variables para clasificar a un objeto dentro de ella es:

X_1 poco importante,
X_2 muy importante,
X_3 irrelevante,
X_4 irrelevante,
X_5 poco importante.

Así, la matriz A_{55} que registra a las variables X_m, $m = 1, ..., 5$ de cada objeto O_i, $i = 1, ..., 5$ es:

$$\begin{array}{c} \\ O_1 \\ O_2 \\ O_3 \\ O_4 \\ O_5 \end{array} \begin{array}{c} \begin{array}{ccccc} X_1 & X_2 & X_3 & X_4 & X_5 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Ahora el problema es que dado un nuevo objeto O_6, no perteneciente a la muestra, hay que clasificarlo en su clase correspondiente. El vector de rasgos de O_6 es (0,1,1,1,0).

Primero obtengamos todos los testores de la matriz A_{55}.

$$\begin{array}{c} \\ O_1 \\ O_2 \\ O_3 \\ O_4 \\ O_5 \end{array} \begin{array}{c} \begin{array}{ccccc} X_1 & X_2 & X_3 & X_4 & X_5 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Matriz A_{55}

$$\begin{array}{c} \\ O_1 \\ O_2 \\ O_3 \\ O_4 \\ O_5 \end{array} \begin{array}{c} \begin{array}{cccc} X_1 & X_2 & X_3 & X_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

Matriz $A_{55}/\{K_4\}$

$$\begin{array}{c} \\ O_1 \\ O_2 \\ O_3 \\ O_4 \\ O_5 \end{array} \begin{array}{c} \begin{array}{ccc} X_1 & X_2 & X_3 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} \end{array}$$

Matriz $A_{55}/\{K_3\}$

$$\begin{array}{c} \\ O_1 \\ O_2 \\ O_3 \\ O_4 \\ O_5 \end{array} \begin{array}{c} \begin{array}{cc} X_1 & X_2 \end{array} \\ \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} \end{array}$$

Matriz $A_{55}/\{K_2\}$

$$\begin{array}{c} \\ O_1 \\ O_2 \\ O_3 \\ O_4 \\ O_5 \end{array} \begin{array}{c} X_1 \\ \left(\begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{array} \right) \end{array}$$

Matriz $A_{55}/\{K_1\}$

De los testores anteriores $A_{55}/\{K_1\}$, $A_{55}/\{K_2\}$ y $A_{55}/\{K_3\}$ son testores típicos.

Después calculemos el peso informativo de cada una de las variables:

$\tau_1 = \frac{T_1}{T} = \frac{3}{3} = 1$

$\tau_2 = \frac{T_2}{T} = \frac{3}{3} = 1$

$\tau_3 = \frac{T_3}{T} = \frac{3}{3} = 1$

$\tau_4 = \frac{T_4}{T} = \frac{2}{3}$

$\tau_5 = \frac{T_5}{T} = \frac{1}{3}$.

Ahora obtengamos la distancia entre O_α y la clase K_1:

$$\begin{aligned} D(O_\alpha)_{K_1} &= \frac{1}{2}\left[\sum_{i=1}^{2}\{\tau_1 d(O_{\alpha 1}, O_{i1}) + \ldots + \tau_5 d(O_{\alpha 5}, O_{i5})\}\right] \\ &= \frac{1}{2}[\tau_1 d(O_{\alpha 1}, O_{11}) + \ldots + \tau_5 d(O_{\alpha 5}, O_{15}) + \tau_1 d(O_{\alpha 1}, O_{21}) + \ldots \\ &\quad + \tau_5 d(O_{\alpha 5}, O_{25})] \\ &= \frac{1}{2}\left[1(0) + 1(0) + 1(1) + \frac{2}{3}(1) + \frac{1}{3}(0) + 1(0) + 1(0) + 1(0)\right. \\ &\quad \left. + \frac{2}{3}(1) + \frac{1}{3}(0)\right] \\ &= \frac{1}{2}\left(\frac{7}{3}\right) \\ &= \frac{7}{6}. \end{aligned}$$

Ahora obtengamos la distancia entre O_α y la clase K_2:

$$
\begin{aligned}
D(O_\alpha)_{K_2} &= \frac{1}{3}\left[\sum_{j=1}^{3}\{\tau_1 d(O_{\alpha 1}, O_{j1}) + \ldots + \tau_5 d(O_{\alpha 5}, O_{j5})\}\right] \\
&= \frac{1}{3}[\tau_1 d(O_{\alpha 1}, O_{11}) + \ldots + \tau_5 d(O_{\alpha 5}, O_{15}) + \tau_1 d(O_{\alpha 1}, O_{21}) + \ldots \\
&\quad + \tau_5 d(O_{\alpha 5}, O_{25}) + \tau_1 d(O_{\alpha 1}, O_{31}) + \ldots + \tau_5 d(O_{\alpha 5}, O_{35}] \\
&= \frac{1}{3}\left[1(0) + 1(1) + 1(0) + \frac{2}{3}(0) + \frac{1}{3}(0) + 1(1) + 1(1) + 1(1)\right. \\
&\quad \left. + \frac{2}{3}(1) + \frac{1}{3}(1) + 1(1) + 1(1) + 1(0) + \frac{2}{3}(0) + \frac{1}{3}(0)\right] \\
&= \frac{1}{3}(7) \\
&= \frac{7}{3}.
\end{aligned}
$$

Debido a que O_α pertenece a la clase cuya distancia entre ambos sea menor, entonces O_α es de la clase K_1.

Recordemos que hay otra forma para clasificar a O_α, así:
Primero obtengamos a los objetos típicos.

$W(O_1) = \sum_{k=1}^{5} \rho_k O_1(V_k) = 1(0) + 1(1) + 1(0) + \frac{2}{3}(0) + \frac{1}{3}(0) = 1$

$W(O_2) = \sum_{k=1}^{5} \rho_k O_2(V_k) = 1(0) + 1(1) + 1(1) + \frac{2}{3}(0) + \frac{1}{3}(0) = 2$

$W(O_3) = \sum_{k=1}^{5} \rho_k O_3(V_k) = 1(0) + 1(0) + 1(1) + \frac{2}{3}(1) + \frac{1}{3}(0) = \frac{5}{3}$

$W(O_4) = \sum_{k=1}^{5} \rho_k O_4(V_k) = 1(1) + 1(0) + 1(0) + \frac{2}{3}(0) + \frac{1}{3}(1) = \frac{4}{3}$

$W(O_5) = \sum_{k=1}^{5} \rho_k O_5(V_k) = 1(1) + 1(0) + 1(1) + \frac{2}{3}(1) + \frac{1}{3}(0) = \frac{5}{3}$.

Por lo tanto, el objeto típico de la clase K_1 es O_2 y el de la clase K_2 es O_5, a los cuales denotaremos como O_{t_1} y O_{t_2} respectivamente.
Ahora, la distancia entre O_α y O_{t_1} es:

$$
\begin{aligned}
d(O_\alpha, O_{t_1}) &= \frac{1}{2}\sum_{k=1}^{5} \tau_k d(O_{\alpha k}, O_{t_1 k}) \\
&= \frac{1}{2}\left(\tau_1 d(O_{\alpha 1}, O_{t_1 1}) + \ldots + \tau_5 d(O_{\alpha 5}, O_{t_1 5})\right)
\end{aligned}
$$

$$= \frac{1}{2}\left(1(0)+1(0)+1(0)+\frac{2}{3}(1)+\frac{1}{3}(0)\right)$$
$$= \frac{1}{2}\left(\frac{2}{3}\right)=\frac{1}{3}\ ,$$

y la distancia entre O_a y O_{t_2} es:

$$\begin{aligned} d(O_a, O_{t2}) &= \frac{1}{2}\sum_{k=1}^{5}\tau_k d(O_{ak}, O_{t2k}) \\ &= \frac{1}{2}\left(\tau_1 d(O_{a1}, O_{t21}) + \ldots + \tau_5 d(O_{a5}, O_{t25})\right) \\ &= \frac{1}{2}\left(1(1)+1(1)+1(0)+\frac{2}{3}(0)+\frac{1}{3}(0)\right) \\ &= \frac{1}{2}(2)=1\ . \end{aligned}$$

Entonces O_a pertenece a K_1.

2.1.3 Comentarios

Hay una situación difícil que conduce al peso informativo más alto para una variable: si los r renglones correspondientes a dos objetos son el mismo excepto por una entrada, entonces la variable correspondiente aparecerá en todos los testores típicos y tendrá una calificación de PD de uno. Pero esta variable sólo nos informará que los objetos tienen características cercanas. Si este es el caso, se sugiere se anule la variable y uno de los objetos, cuya única deferencia con los demás objetos es dicha variable.

La Definición 2.4 amerita una explicación: esta definición depende demasiado de la información que aporten las variables para clasificar a los objetos en sus clases respectivas. Para extraer un objeto representativo tenemos que definir a las variables teniendo en mente que el objeto que acumule mas respuestas "sí" (o "uno") es el "mejor" con respecto a un rasgo para representar a una clase; por ejemplo, si deseamos averiguar acerca del esquimal típico, entonces nuestras preguntas pueden ser: ¿es menor de cinco pies de estatura?, ¿vive de la cacería?, ¿vive arriba del paralelo 80?, etc. Las anteriores serían variables elegidas adecuadamente para realizar una buena clasificación de esquimales. En cambio, variables que no ayudarían a lograr la

clasificación pueden ser: ¿tiene brazos?, ¿es de cabello largo?, ¿ es carnívoro?, ¿tiene orejas?, etc.

La cantidad de los cálculos a fin de obtener los testores típicos de una matriz usando sólo la definición, es muy grande. La tarea real de la teoría de testores es dar un algoritmo rápido y eficiente para los cálculos de testores típicos de una matriz dada. El algoritmo incluye una técnica para distinguir los testores mínimos (o típicos) donde para lograr la clasificación de objetos la cantidad de descripciones que impide diferenciar a algunas variables de entre todas las que describen al objeto se considera como una medida de su "error", siendo un testor aquel subconjunto de variables cuyo error es igual a cero y un testor mínimo aquel que no solamente tiene error nulo, sino que lo alcanza con un número mínimo de variables. El incluir el algoritmo completo llevaría demasiado tiempo, además de que no es el objetivo principal de esta tesis, por lo que si fuera de interés para el lector, puede encontrarlo en [Morales y Miramontes, 1988].

Por otro lado, si se tiene un problema con n objetos o fenómenos y si los cálculos necesarios para clasificarlos en clases es exactamente n, entonces se dice que el problema es de *orden lineal*. Si se encuentra un caso cuyo orden no es polinomial, $O(NP)$, al cual se le llama NP - *completo*, ni siquiera es posible saber cuántos cálculos son necesarios para resolver el problema pues no existe un número p, entero y finito tal que el número de cálculos para clasificar a los objetos sea menor o igual a n^p. [Freeman y Skapura, 1992]. Algunas de las limitaciones en el uso de la teoría de testores son:

1. Entre más grande es la matriz, aumenta el grado de dificultad para realizar los cálculos y se llega al caso de orden no polinomial siendo imposible resolver el problema.

2. Para encontrar *todos* los testores típicos de una muestra de aprendizaje son 2^n subconjuntos diferentes para n variables, lo cual es un gran espacio de búsqueda.

3. Otro inconveniente de esta técnica es que no hay posibilidad, en parte por la falta de investigación al respecto, de saber si es aplicable en problemas linealmente separables o en los no linealmente separables.

En conclusión, la teoría de testores sólo es viable para matrices pequeñas.

En la siguiente sección se presenta otra forma de clasificación que también se ha usado en la literatura acerca de este tema.

2.2 Clasificación bayesiana

El profundizar en la forma de obtener las probabilidades no es el objetivo principal de esta tesis, por ello sólo se dará una visión rápida del tema, (véase [Beale y Jackson, 1991; Bernardo, 1981]).

La clasificación bayesiana es una técnica estadística que utiliza la probabilidad. Se usa para hacer una estimación de la semejanza o probabilidad de que un patrón pertenezca a una clase en especial.

Pensemos en que tenemos clases ya determinadas y deseamos clasificar un nuevo objeto.Desde el punto de vista estadístico esto se puede hacer obteniendo la mayor información posible. Así, en un problema de decisión específico, quien tiene que tomar las decisiones inicia precisando la información con la que dispone inicialmente; luego obtiene nuevos datos que den información relevante y finalmente, combina esta información aquella con la que incialmente ya disponía para entonces tomar la decisión más apropiada.

A continuación enunciaremos el *teorema de Bayes* que nos ayudará a realizar la clasificación de objetos, siguiendo el enfoque estadístico:

Teorema 2.1 *([Beale y Jackson, 1991]). Sean una colección de clases posibles G_i, $i = 1,...,n$, y un conjunto de características cuantificables $X = (X_1,...,X_n)$, (X es el vector rasgo de un patrón conocido). Si $P(G_i)$ es la probabilidad de que un patrón pertenezca a una clase G_i, $i = 1,...,n$, con $(0 \leq P(G_i) \leq 1)$ y $P(X/G_i)$ es la probabilidad de obtener el vector patrón X en cada una de las clases posibles, entonces,*

$$P(G_i/X) = \frac{P(X/G_i)P(G_i)}{\sum_j P(X/G_j)P(G_j)} ,$$

donde $P(G_i/X)$ es la probabilidad de que el patrón tome el valor X, dado que está en la clase G_i.

El teorema de Bayes describe la información que se posee sobre los sucesos inciertos luego de añadir a la información inicial la que dan los resultados experimentales y esto lo hace relacionando $P(G_i/X)$ y $P(X/G_i)$.

Como el vector rasgo X es de un patrón conocido, entonces indica la forma de calcular la probabilidad de que pertenezca a cada una de las clases $G_1, \ldots G_n$; el teorema de Bayes determina a cuál de esas clases pertenece. En este enfoque el, problema se plantea así:

Decidir si X pertenece a la clase i para $P(G_i/X) > P(G_j/X)$, $i = 1, \ldots, n$; $i \neq j$.

Es decir, que asignamos un patrón a la clase que tiene la probabilidad condicional más alta de que el vector X pertenezca a ella. Se puede probar que ésta es la mejor estimación que se pudo haber esperado (si medimos esto en términos del cociente de error más pequeño). Pero en términos prácticos hay dificultades para obtener dichas probabilidades pues las probabilidades condicionales necesarias en el teorema de Bayes frecuentemente son desconocidas y, para estimarlas, se hacen hipótesis acerca del patrón de datos y en el dato se describen distribuciones desconocidas usando modelos de probabilidad.

Dado que conocemos el patrón, éste debería pertenecer a uno de los n grupos, que es la probabilidad $P(X/G_i)$ mencionada anteriormente. Aunque no conocemos el valor de esta probabilidad lo podemos aproximar usando el teorema de Bayes.

$P(X/G_i)$ se estima suponiendo que sigue una distribución normal debido a que es una buena aproximación para muchas otras distribuciones y es fácil de trabajar pues ha sido bien investigada. Debemos tomar en cuenta que hay otro tipo de distribuciones de probabilidad además de la normal (como la Poisson, por ejemplo), pero para el caso del teorema de Bayes sólo se manejan los problemas en los que las variables que aparecen en ellos siguen una distribución (normal), teniendo, en consecuencia, un campo restringido en el cual es útil.

$P(G_i)$ a la cual habíamos nombrado la probabilidad de que un patrón pertenezca a la clase G_i se puede hallar fácilmente adoptando modelos para la probabilidad condicional $P(X/G_i)$ basados en una distribución normal, y aplicando el teorema de Bayes se puede definir un clasificador estadístico relativamente directo. La ejecución dependerá de qué tan cerca se ajusta el patrón de datos al modelo elegido, pero generalmente los clasificadores bayesianos pueden optimizarse para hacerlo extremadamente bien en el sentido de que se puede disminuir la dificultad para resolver el problema usando

métodos aparentemente no estadísticos.

El teorema de Bayes sólo es aplicable en problemas linealmente separables, (véase [Beale y Jackson, 1991]).

Los clasificadores bayesianos tienen muchas dificultades para realizar el reconocimiento no lineal de patrones y esto se debe a que imponen muchos requisitos a los datos que se clasificarán. Dentro de éstos se tienen: regularidad, distribución gaussiana, etc., (véase [Beale y Jackson, 1991]).

Para distribuciones en dos dimensiones, con dos clases, es demostrable (véase [Beale y Jackson, 1991]) que el espacio patrón se divide más sencilla y eficientemente utilizando una *superficie de decisión cuadrática*, es decir, una gráfica de una función polinomial de grado dos. Esta superficie puede ser modificada al caso más simple de un clasificador lineal, consiguiendo hacer la hipótesis de que ambas distribuciones de clase tienen iguales matrices de covarianza, lo cual indicaría que las distribuciones tienen ambas la misma forma y extensión y, en consecuencia, la partición más exacta del espacio patrón se obtiene mediante una línea recta ($y = mx + c$). Véase la Figura 2.1.

2.3 Redes neuronales

2.3.1 Cerebro y computadoras

La idea de que una máquina llegue a tener las mismas capacidades que el cerebro es muy atractiva, es por ello que desde antes de 1930 se ha investigado al respecto y aunque con un avance relativamente lento dada la dificultad del proyecto, se continúa trabajando dentro de la inteligencia artificial para alcanzar ese objetivo. Para ello es necesario observar ciertas características del cerebro que lo hacen más ventajoso que las computadoras:

1.Procesa la información visual que le permite reconocer objetos eficazmente.

2.Es robusto y tolerante a fallas, ya que células nerviosas cerebrales mueren todos los días sin afectar la eficacia de la conectividad significativamente. Falla de una manera tal que se recupera sin grandes daños y se puede autoreparar. Si el cerebro recibe un golpe y se daña seriamente, entonces los sistemas distribuidos paralelamente manifiestan lo que se conoce como *degradación graciosa*, donde la ejecución del sistema cae (baja) de un

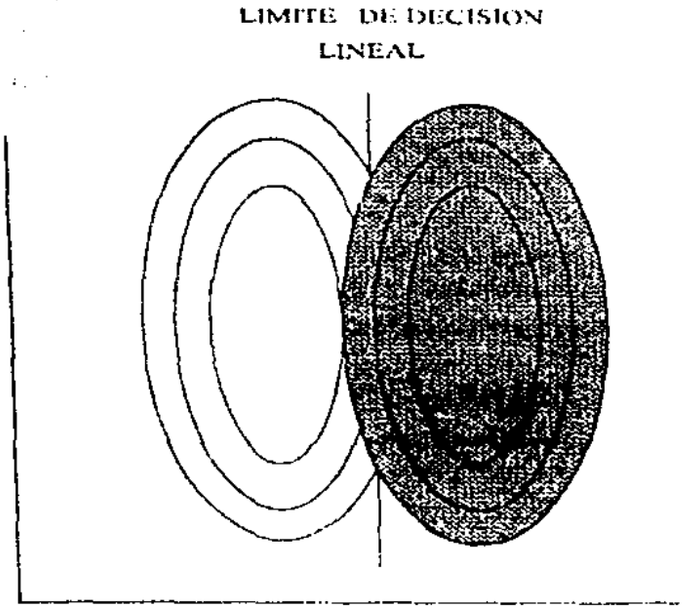

Figura 2.1: La clasificación bayesiana se reduce a una clasificación lineal bajo ciertas condiciones.

nivel alto para reducir el nivel, pero sin caer catastróficamente a cero.

3.Es flexible pues se puede ajustar a un ambiente nuevo por "aprendizaje". Además se programa por sí mismo adaptándose a medios cambiantes.

4.Puede manejar información confusa, probabilística, ruidosa o inconsistente.

5.Tiene un alto grado de paralelismo entre sus neuronas, es decir que puede desempeñar diferentes tareas al mismo tiempo.

6.Es pequeño, compacto y requiere poca energía.

7.Por otro lado, sucede que la computadora ejecuta una instrucción tras otra rápidamente, al grado de poder calcular millones de operaciones por segundo, mientras que el cerebro es mucho más lento comparado con ella.

De lo anterior concluimos que, aunque las computadoras manejen muy bien operaciones en serie, eso no implica que sean tan efectivas para llevar a cabo tareas simultáneas como la visión, control motor, decisiones sobre las bases de datos ruidosas e incompletas, o reconocimiento del habla; que involucran gran cantidad de tipos de datos, una amplia memoria, además

de muchas y diferentes ideas complejas. Nuestro cerebro ejecuta fácilmente tareas paralelas, mientras que una computadora las realiza con dificultad. Lo importante es esa capacidad de paralelismo cerebral, no la velocidad del mismo.

Aunque el diseño del cerebro es en paralelo, es posible *simular* ciertas actividades que éste hace usando el diseño en serie de una computadora. Dicha simulación se realiza *modelando* a las redes de neuronas cerebrales, es decir, haciendo una versión simplificada del sistema de las neuronas que tenga el mismo comportamiento general; esto se logra extrayendo pocos rasgos que se consideren importantes, aunque para simplificar la implementación se tengan que ignorar muchos otros, y aún así realizar tareas útiles. Este campo es conocido como *neurocomputación, computación neuronal, redes asociativas, computación colectiva, conexionismo*, etc.

La simulación por computadora tiene al menos dos aplicaciones en este campo:

1- Estudio de modelos de sistemas biológicos.

2- Estudio de redes de neuronas artificiales usadas para computación y control, tratando de lograr la eficiencia de una arquitectura electrónica basada en neuronas.

Nos enfocaremos solamente hacia las RN. Las RN simuladas por computadora sirven para tomar decisiones y son llamadas *neurocomputadoras*. Primero veamos al elemento biológico que dio origen a este tipo de diseño: las neuronas.

2.3.2 Neuronas biológicas

El sistema nervioso cerebral de los seres humanos está compuesto por, cerca de 10^{11} neuronas (células nerviosas) de varios tipos, una neurona típica es como la ilustración de la Figura 2.2.

Una *neurona* es una unidad funcional y estructural del sistema nervioso que se encarga de intercambiar información con otras neuronas a través de cambios químicos en la sinapsis. La *sinapsis* es el punto de contacto entre neuronas adyacentes donde los impulsos nerviosos son transmitidos de una a la otra (las neuronas se unen excepto por un espacio microscópico, la *hendidura sináptica*). La neurona puede tener muchas sinapsis de entrada que

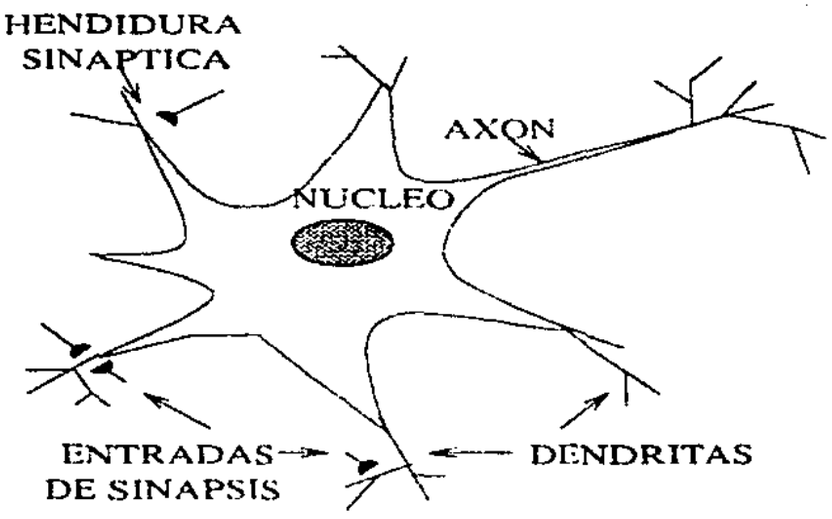

Figura 2.2: Dibujo esquemático de una neurona biológica.

llegan al cuerpo celular o *soma* a través de las dendritas. Las *dendritas* son redes de fibras nerviosas (partes eléctricamente pasivas de la neurona), que se encargan de recibir la sinapsis de entrada. Las sinapsis de salida de la neurona se llevan a cabo por el *axón* (parte eléctricamente activa de la neurona) que es una fibra larga individual a través de la cual los impulsos recogidos y sumados por el soma son transferidos a otras neuronas. El axón realiza un procesamiento no lineal, y ejecuta la labor de *umbral* (nivel mínimo de excitación de energía).

La transmisión de la señal de una neurona a otra en la sinapsis es un complejo proceso químico en el cual ciertas sustancias transmisoras son liberadas del lado de la transmisión de la unión celular. Esto provoca un aumento o disminución del potencial eléctrico interior de la neurona receptora. Si este potencial alcanza un umbral, entonces el axón envía una descarga de la neurona a otras neuronas, sino alcanza el umbral no hay descarga. Luego de descargar, la célula tiene que esperar por un tiempo llamado *periodo refractario* antes de que pueda descargar otra vez. [Abercrombie y Hickman, 1990].

2.3.3 Neurocomputación

En la vida cotidiana realizamos tareas como ver y reconocer objetos, oír sonidos y asociarlos con eventos del pasado, escuchar nuestra melodía favorita y reconocerla con sólo percibir una parte de ella, etc. Para una máquina no es así; aunque son mucho más eficaces en la exactitud y rapidez para llevar a cabo algunas operaciones lógicas y, por ejemplo, para operar con inmensas cantidades aritméticas; sin embargo son más las tareas demasiado complejas y tardadas para ellas, mientras que para un ser humano (incluso para algunos animales irracionales, digamos para una mascota, el recordar a su dueño) son sencillas. Dentro de los ejemplos anteriores están descritas algunas características como: la asociación, la memoria, la clasificación y el reconocimiento de patrones que son parte de la capacidad cerebral en muchas especies animales.

La computación puede clasificarse como la *algorítmica* y la *no algorítmica*. [Beale y Jackson, 1991]. La computación convencional es la que utiliza algoritmos indicando a la máquina paso a paso lo que debe hacer. Pero sucede que hay tareas para las que los algoritmos no existen ni existirán, o para las que siguiendo una serie de pasos lógicos o aritméticos se lleve a la respuesta. Y hay muchos de estos casos. Actualmente no hay "software algorítmico" para un piloto de automóvil, un lector de caracteres manuscritos, un traductor de lenguaje hablado, un sistema que pueda identificar aeronaves enemigas, o uno que reconozca un discurso continuo sin importar quien habla.

Estas tareas tienen características comunes: los seres humanos somos capaces de hacerlas, se pueden generar grandes grupos de ejemplos de las metas que se persiguen y cada tarea involucra *asociación de objetos* de un grupo con objetos en otro grupo. Por ejemplo, una computadora que pueda leer en voz alta debe asociar grupos de letras escritas con sonidos específicos, pausas e interrogaciones, etc.

Por otro lado, el cerebro puede *aprender cosas y autoasociarse*. Aprendiendo de ejemplos es como un niño habla, aprende a escribir, comer, beber, etc.

Del deseo de lograr realizar las tareas anteriores y muchas otras de una forma no algorítmica y aprendiendo (igual que lo hace el cerebro) es que surgió la *neurocomputación*, la cual está basada en el funcionamiento de las *RN* cerebrales. La neurocomputación utiliza *RN artificiales* que desarrollan asociaciones (transformaciones o mapeos) entre objetos de acuerdo al medio en que estén. La *RN* artificial genera sus propias reglas internas utilizando

sobre todo la asociación y refina éstas reglas comparando sus resultados con los ejemplos. La red se autoenseña a realizar tareas aprendiendo de sus errores, lo cual hace más atractivo usarla como opción a los programas convencionales donde usualmente hay un largo y complicado programa que entre más largo es, da más probabilidad a que se cometan errores y por ello es más difícil encontrarlos, exigiendo así, de mucho tiempo de dedicación.

En el tipo de procesamiento de las *RN* tenemos muchos procesadores, cada uno ejecutando un programa muy simple, en lugar de la situación convencional donde uno o a lo más unos pocos procesadores ejecutan programas muy complicados. Y, en contraste con la robustez de una *RN*, un cálculo secuencial ordinario podría fácilmente ser arruinado por el error de un sólo bit (es decir, por el error en un solo paso todo se arruina, por ejemplo, al realizar una operación aritmética ésta se arruina si un único cálculo falla). El ciclo de tiempo típico de las actividades entre las neuronas es de unos cuantos milisegundos, el cual es unos millones de veces más lento que el de sus contrapartes de silicón, las entradas de semiconductores. Sin embargo, esto es posible sólo porque billones de neuronas operan simultáneamente.

La neurocomputación es aplicable en cualquier campo porque se pueden desarrollar nuevas capacidades de procesamiento de información, y los costos de desarrollo y tiempo frecuentemente disminuyen comparados con los de la computación algorítmica. Destacan sus aplicaciones a la ingeniería, la computación, la psicología sensorial y del conocimiento en donde puede servir como un generador de modelos.

Sin embargo, la neurocomputación no reemplaza a la programación algorítmica debido a que hay cierto tipo de problemas que la neurocomputación no puede resolver y la programación algorítmica sí (un ejemplo de estos problemas es el procesamiento masivo de datos aritméticos). Más bien, la neurocomputación y la programación algorítmica se pueden usar conjuntamente para resolver problemas de un modo más eficiente.

2.3.4 Aprendizaje en las redes neuronales

Cuando se conocen de antemano los patrones de entrada y salida, las memorias asociativas pueden ser diseñadas con o sin *RN*. Incluso hay algoritmos de entrenamiento de *RN* complicados y lentos que pueden ser utilizados en algunas aplicaciones donde la red es *entrenada* de una vez por todas y entonces

sirve para la asociación de patrones sólo para "recordar".

Pero las *RN* tienen una utilidad más acentuada cuando la memoria asociativa debe *adaptarse* a patrones nuevos que cambian. Hay entonces dos problemas: el diseño óptimo de una memoria asociativa para "recordar" patrones conocidos, y la formulación de algoritmos de aprendizaje que se aproximen al óptimo patrón, mientras las redes en el proceso de tomar decisiones se adaptan a nuevas condiciones. El aprendizaje adaptativo, en las máquinas, así como en los organismos es, a veces, el método más práctico para optimizar o afinar un diseño.

El tipo de enlaces de retroalimentación de la *RN* determina qué tan factible es de ser entrenada; el grado de interconexión (es decir, el número de conexiones entre los nodos de la RN) determina su *paralelismo*. A cada una de las conexiones de la red se le asigna un valor dentro de los números reales, llamado *peso*. La función de todos los pesos de la red, en conjunto, es lograr que la misma aprenda a dar la salida correcta.

A una *RN* no se le programa como a una computadora digital; se le "enseña" a dar respuestas aceptables; mide su velocidad no en "instrucciones" por segundo sino en interconexiones por segundo, y su "memoria" no se mide en bytes sino en interconexiones.

El *entrenamiento* de una red es un proceso donde ésta aprende a asociar un patrón de entrada con la respuesta correcta. Dicho aprendizaje se realiza introduciendo a la red datos nuevos, que obligan a que los pesos (que pueden ser números reales) se ajusten, ya sea manual o automáticamente. Durante el aprendizaje sólo cambian los valores de los pesos de las conexiones sinápticas. Una *matriz ponderada (o de pesos)* entre conexiones permite a la *RN* aprender y recordar. Una vez que la red aprende, los valores de los pesos asociados a los nodos dejan de cambiar. Si se le da una nueva entrada que no esté almacenada, la red aún puede dar respuestas adecuadas.

El aprendizaje de una red puede ser: supervisado, no supervisado o autosupervisado (o automatizado).

Aprendizaje supervisado. La red tiene que comparar sus salidas con las respuestas correctas ya conocidas, luego procesa a este patrón y lo compara con la salida deseada, corrige cualquier diferencia y repite la actividad si es necesario, hasta que, si hay error, éste sea mínimo o desaparezca. Este es a veces, llamado *aprendizaje con maestro*; el maestro dice a la red cuales son sus respuestas correctas o, al menos (en el caso especial de *aprendizaje reforzado*), si sus respuestas son correctas o incorrectas y no qué respuesta

es la correcta. El algoritmo de retropropagación es uno de los métodos más utilizados y que requieren este tipo de aprendizaje.

Aprendizaje no supervisado. Los datos entran sin intervención humana. Aquí no hay maestro ni respuestas correctas o incorrectas; la red debe descubrir por sí misma las categorías o características interesantes en el dato de entrada, por ejemplo, los patrones que ocurren frecuentemente. Este proceso conduce a un agrupamiento de datos internos, que es el resultado deseado.

En lugar de tener que especificar paso a paso un cálculo que debe ejecutarse, sólo hay que compilar un conjunto de entrenamiento de ejemplos representativos. Esto significa que podemos tratar problemas donde reglas apropiadas son muy difíciles de conocer por adelantado, como en los sistemas expertos y robóticos; además nos puede ahorrar mucho del tedioso y caro diseño de software y programación aún cuando tengamos reglas explícitas. Este es a menudo usado para problemas de clasificación.

Aprendizaje autosupervisado. Sucede cuando la red se autocontrola y corrige errores en la interpretación de los datos por retroalimentación.

2.4 El perceptrón simple

En resumen, un *modelo de RN o RN artificial* está basado en la simulación de las neuronas biológicas. El conjunto de neuronas que funcionalmente se asemejan al cerebro que pueden reconocer patrones, reorganizar datos y, lo más interesante aprender. Las RN artificiales constan de objetos llamados unidades (o neuronas artificiales, nodos, elementos de proceso, elementos procesadores,o incluso neuronas matemáticas). Las unidades están conectadas por uniones, las cuales actúan como los axones y las dendritas de una neurona biológica. La unión multiplica la salida de una unidad por un factor de peso, un valor análogo a las fuerzas de conexión en la sinapsis (las uniones y los valores de peso son elementos discretos cuya característica principal es que tienen pocos grados de libertad, interactuando entre sí de forma no lineal, generando propiedades no deducibles de las neuronas biológicas). Entonces la unión transfiere el valor peso de la salida a otra unidad, la cual suma los valores que pasaron a ella con todas las otras uniones que entran. Si el valor de la entrada total excede algún valor umbral, la unidad se enciende.

Las *RN retroalimentadas* (este concepto se definirá posteriormente) más

sencillas son los *perceptrones simples (PS)* también llamados *perceptrones unicapa*, estos consisten en una fila de neuronas de entrada llamada *retina* y en una capa de salida. Rosenblatt en su libro "Principles of Neurodynamics" describe a los perceptrones como redes simplificadas en las cuales ciertas propiedades de sistemas nerviosos verdaderos son exagerados, mientras que otros son ignorados. Rosenblatt señaló que las RN no intentan ser copias detalladas de algún sistema nervioso verdadero.

McCulloch y Pitts (1943) propusieron un modelo simple de una neurona como una *unidad umbral binaria*. Este no da una explicación de los patrones complejos y sincronización de la actividad nerviosa en sistemas de neuronas verdaderas ni tiene características complicadas halladas en el cuerpo de neuronas biológicas, de manera que se puede simular en una computadora digital.

El modelo de McCulloch y Pitts es el siguiente:

Para modelar la eficiencia de las conexiones sinápticas se asigna un peso sináptico a cada una de las entradas a la neurona.

Llamemos X al conjunto de *entradas* a la neurona, donde cada elemento de ese conjunto es un rasgo X_i, $i = 1, \ldots, n$ y sea W el conjunto de las conexiones, donde los componentes, W_i, son los *pesos sinápticos*, es decir, los pesos entre las conexiones ($W_i \in \mathbb{R}$); esto significa que si hay n entradas, entonces hay n pesos asociados en las líneas de entrada. Sea E al *estado de la neurona*. En la Figura 2.3 podemos ver un PS.

El modelo de las neuronas de McCulloch y Pitts calcula la suma de sus entradas para obtener la entrada total: toma la primera entrada, la multiplica por el peso en esa línea de entrada, y luego hace lo mismo en la siguiente línea de entrada; así, sucesivamente. La suma de todas da el valor total que la neurona reciba.

Esto se escribe como:

$$W_1X_1 + W_2X_2 + \ldots + W_nX_n = \sum_{i=1}^{n} W_iX_i .$$

La suma, llamada *suma ponderada* se compara con un cierto umbral en la neurona, el *valor umbral*, al que llamaremos θ. Si la suma es mayor o igual al valor umbral, entonces la neurona "descargará" una salida al axón y diremos

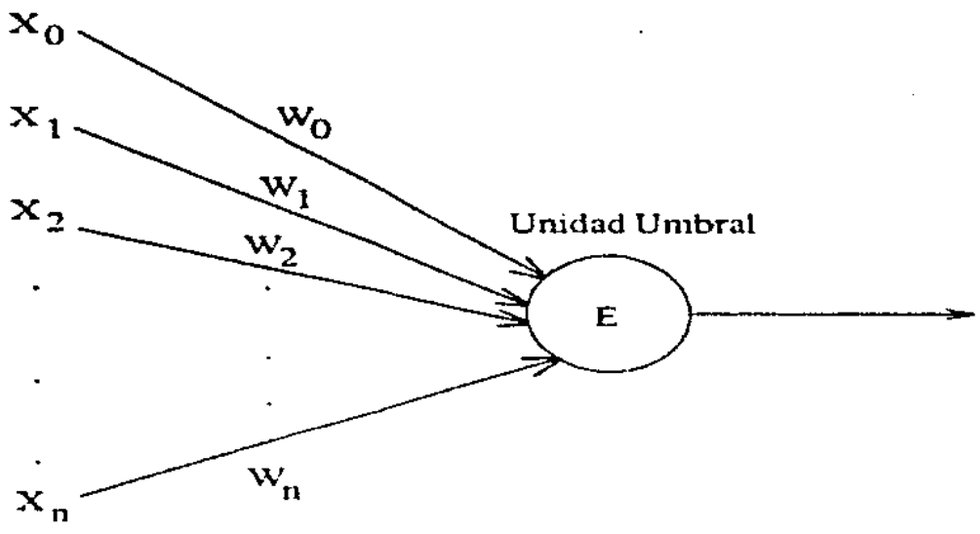

Figura 2.3: Modelo de la estructura de una neurona en la red. E es el estado de la neurona. Este estado es una función de las entradas X_i y de los pesos sinápticos W_i asociados a cada conexión con la neurona.

que la salida es 1 si la suma es menor al umbral, la neurona "no descargará" señal alguna y diremos que la salida es 0. Esto se muestra en la Figura 2.4:

Equivalentemente, el valor umbral se puede restar de la suma ponderada, y el resultado compararlo con cero: si el valor es positivo, entonces la salida es 1, sino la salida es 0, como se puede ver en la Figura 2.5.

Nótese que la forma de la función de la Figura 2.4 es la misma que la de la Figura 2.5, pero ahora el salto ocurre en cero en lugar de que suceda en θ.

Se puede conseguir el mismo efecto mediante el *sesgo de la neurona*, dejando al umbral fuera del cuerpo del modelo neuronal y luego conectándolo a un cierto valor de entrada extra que se deja fijo para estar "encendido" todo el tiempo. En este caso, en lugar de restar el umbral de la suma ponderada, la entrada extra de $+1$ es multiplicada por un peso igual al negativo del valor umbral, $-\theta$, y sumada como todas las entradas. El valor de $-\theta$ es llamado el sesgo de la neurona para hacer que la neurona descargue.

Si llamamos Y a la salida, podemos escribir:

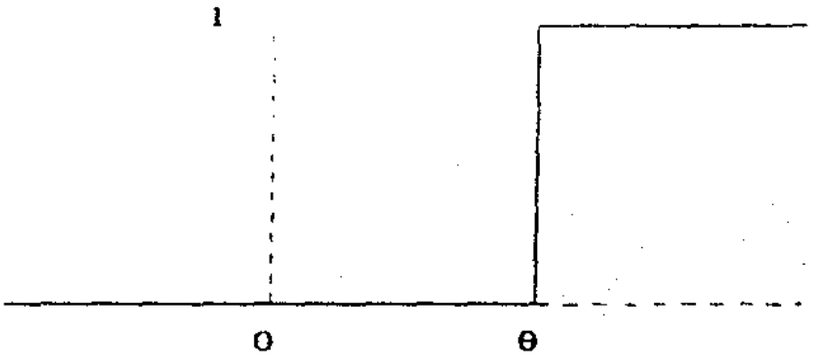

Figura 2.4: Función umbral con umbral en θ, (el eje X representa la entrada y el eje Y la salida).

$$Y = f_h(\sum_{i=1}^{n} W_i X_i - \theta) .$$

La función f_h es la función de Heaviside que obliga al estado de la neurona (encendido o apagado), y en consecuencia, a la salida a permanecer en el conjunto $\{0,1\}$. La expresión para Y puede simplificarse si usamos la aproximación del sesgo de la neurona podemos definir una entrada extra, la entrada X_0, la cual siempre está encendida (es decir, $X_0 = 1$), con un peso que represente el sesgo aplicado a la neurona (es decir, $W_0 = -\theta$). La ecuación que describe la salida es entonces

$$Y = f_h(\sum_{i=0}^{n} W_i X_i) .$$

Observe que el límite inferior de la suma ha cambiado de 1 a 0 para así incorporar $W_0 X_0 = -\theta$ a la suma, dejando en ella implícito al sesgo de la neurona.

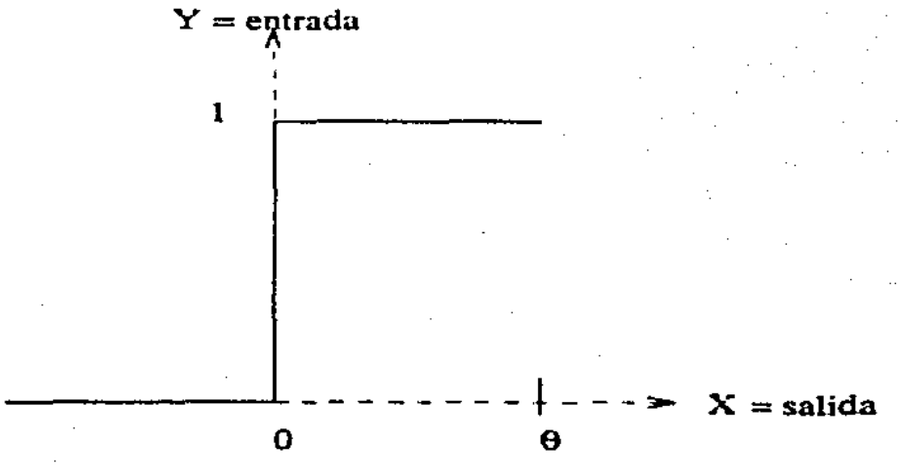

Figura 2.5: Función umbral con umbral en cero.

Como las entradas pasan a través del modelo neuronal para dar la salida, el sistema es conocido como una *retroalimentación*.

Comúnmente la tecnología de neurocomputadoras se basa en la suposición de que la actualización de las señales dentro de cada nodo ocurre en forma discreta (es decir, no hay niveles intermedios) en lugar de usar una manera continua o concurrente.

Cuando cada neurona calcula una suma ponderada de las entradas de otras neuronas y obtiene un número, ésta salida es enviada a otras neuronas, las cuales utilizan el mismo tipo de cálculo. Ellas están usando diferentes pesos y posiblemente diferentes *funciones ganancia* que se encargan de controlar la adaptación de pesos, es decir, ayudan a que los valores de los pesos cambien hasta obtener los pesos que clasifiquen correctamente a los objetos. Los coeficientes W_i son, en general, diferentes para diferentes i y podríamos también hacer a la función ganancia denotada por $g(x)$, localmente dependiente. Estas funciones ganancia y pesos pueden ser pensadas como datos locales almacenados por los procesadores. Se ampliará la información acerca de la función ganancia en el capítulo 4.

2.4.1 Entrenamiento del perceptrón simple

Recordemos que el objetivo principal es, utilizando los perceptrones, clasificar objetos en su clase respectiva y para ello es necesario entrenar a la red.

En los procesos de aprendizaje, ya sea que se trate de humanos o de algunos animales, con frecuencia se usa el *reforzamiento positivo*. Dicho brevemente este consiste en que si la tarea por aprender se realiza correctamente, entonces ésta se refuerza (se premia), en caso opuesto se "reprende" por el error. Usando dicha idea en la red, hay que reforzar el comportamiento que deseamos se repita y desalentar aquel que es indeseable.

Para que la red sea entrenada es necesario que aprenda de sus errores. Si el vector de entrada es la codificación de un objeto conocido y si la salida lo es de la codificación de una clase o categoría, entonces pueden suceder dos cosas:

1. Si la red da una salida incorrecta (es decir, que la red indique que el objeto conocido pertenece a una clase que no le corresponde, por ejemplo, dar un 0 cuando debió ser un 1 o viceversa), se desea que esto no vuelva a suceder.

2. Si la red da la salida correcta, entonces es innecesario hacer cambios.

Entonces para que la red sea entrenada *el algoritmo de aprendizaje para el PS* sería:

1. Iniciar a la red colocando los pesos y umbrales aleatoriamente en sus entradas.

2. Presentar como entrada un objeto (codificado a través de su vector de rasgos).

3. Calcular la salida (en el caso más sencillo, usando la función Heaviside).

4. Cambiar los pesos el número de veces necesario para obtener todos los pesos sinápticos que clasifiquen correctamente al conjunto llamado *conjunto de aprendizaje*. Los pesos se van variando para resolver el problema de encontrar el *error* E, definido como:

$$E = \min_{w_{ij}}(d(S, S_c)) \ .$$

donde S es el vector de salida y S_c es el vector que codifica la salida correcta.
Para el caso del PS, el error es:

$$\Delta = d(t) - Y(t) \ .$$

donde $d(t)$ es la respuesta deseada y $Y(t)$ es la respuesta de la red; el error es importante en el proceso de aprendizaje. La función escalón no se usa aquí, excepto para que la clasificación de la red produzca $+1$ ó 0.
Si la salida deseada es 1 y la salida de la red es 0, entonces $\Delta = +1$ y así los pesos son incrementados.
Si la salida deseada es 0 y la salida de la red es 1, entonces $\Delta = -1$ y así los pesos son decrementados.
Si la salida deseada es igual a la salida de la red (es decir, es la correcta), $\Delta = 0$ y los pesos no cambian.

5. Presentar la entrada siguiente.

Algoritmo de aprendizaje del perceptrón simple

Ahora usaremos el algoritmo de aprendizaje anterior aplicándolo a un ejemplo en particular:

Si hay diferentes tipos de letras, por ejemplo, A y B deseamos sean clasificados las A en la clase de la letra A y las B en la clase de la letra B.

1. **Iniciar los pesos y el umbral.**

Definamos $W_i(t)$, con $0 \leq i \leq n$, como el peso de entrada i en el tiempo t y Θ como el valor umbral en el nodo de salida.

Asignar $W_0 = -\Theta$, que es la desviación y dejar fija a $X_0 = 1$. Asignar $W_i(0)$ a valores aleatorios pequeños, iniciando así todos los pesos y el umbral.

2. Presentar la entrada y la salida deseada.

Presentar la entrada $X_0, X_1, X_2, \ldots, X_n$ y la salida deseada $d(t)$.

3. Calcular la salida de la red.

$$Y(t) = f_h(\sum_{i=0}^{n} W_i(t)X_i(t)) \ .$$

4. Adaptar pesos usando el algoritmo Delta Widrow - Hoff.

$\Delta = d(t) - Y(t)$.
$W_i(t+1) = W_i(t) + \eta \Delta X_i(t)$.
$d(t) = +1$ si la entrada pertenece a la clase A.
$d(t) = 0$ si la entrada pertenece a la clase B.
donde $0 \leq \eta \leq 1$, es llamada la *función ganancia (o término ganancia)* positiva que controla la razón de adaptación.

5. Presentar la entrada siguiente.

Una *RN* que use este algoritmo de aprendizaje se llama ADALINE (ADAptative LInear NEurons) y la unión en una sola red de varias ADALINES es llamada MADALINE (MAny- ADALINE). [McCord e Illingworth, 1991].

Otra manera de resolver el problema de clasificación es hacer una modificación del algoritmo anterior alterando los pesos para reforzar las decisiones correctas y desalentando las incorrectas, es decir *reducir el error*. Si la red se

equivoca (caso 1 anterior) hay que incrementar la suma ponderada la siguiente vez para que rebase el umbral y dé la salida correcta, un 1 (o disminuirla para que no exceda el umbral y produzca la salida correcta, un 0). Esto se logra sumando los valores de entrada a los pesos para que la salida esté encendida (es decir, en 1) o restando los valores de entrada de los pesos para que la salida esté apagada (es decir, en 0):

$W_i + X_i$ para que la salida sea 1
$W_i - X_i$ para que la salida sea 0.

Nótese que las entradas que están activas al mismo tiempo serán afectadas; pero las inactivas no contribuyen a la suma ponderada: por ello, el cambiarlas no afectaría el resultado para la entrada particular en cuestión y sí puede desordenar lo que ya se había aprendido.

El algoritmo de aprendizaje anterior es una variante del *aprendizaje Hebbiano* (1949), el cual sólo es de conexiones de reforzamiento *activo*.

La versión usada sólo afecta conexiones activas pero, además, son robustecidas o debilitadas debido a que podemos alterar los pesos conforme conocemos cuál resultado debería dar la red, es decir usamos *aprendizaje supervisado*.

Esta idea para aprendizaje fue probada en 1959 en una "red neuronal" construida por M. Minsky y D. Edmonds, usando para esto una máquina muy grande. [Beale y Jackson, 1991]

Retomando lo anterior, otra versión del paso 4 sería:

4'. Adaptar pesos

Si la salida es correcta, entonces $W_i(t+1) = W_i(t)$ (es decir, no hay cambio).

Si la salida es 0, y debería haber sido 1 (clase A), entonces

$$W_i(t+1) = W_i(t) + X_i(t) \ .$$

Si la salida es 1, y debería haber sido 0 (clase B), entonces

$$W_i(t+1) = W_i(t) - X_i(t) \ .$$

Los pesos no se ajustan en las líneas de entrada, lo cual no contribuye a la respuesta incorrecta, ya que cada peso se ajusta por el valor de la entrada en esa línea X_i, la cual sería cero.

Otra alternativa para adaptar los pesos de tal modo que se logre la clasificación de los objetos es introducir un factor multiplicativo menor que uno en el término de adaptación de pesos para alentar el cambio en los pesos, haciendo que la red dé pequeños pasos, lo cual implicaría que el paso 4 fuera:

4". Adaptar pesos (versión modificada).

Si es correcto, entonces

$$W_i(t+1) = W_i(t) \ .$$

Si la salida es 0, y debería haber sido 1 (clase A), entonces

$$W_i(t+1) = W_i(t) + \eta X_i(t) \ .$$

Si la salida es 1, y debería haber sido 0 (clase B), entonces

$$W_i(t+1) = W_i(t) - \eta X_i(t) \ .$$

donde $0 \leq \eta \leq 1$ es un término ganancia positivo que controla la razón de adaptación.

Una opción más (parecida a las anteriores) es, en lugar de entradas binarias (0 ó 1), usar entradas bipolares, es decir, 1 ó +1. Si se usan entradas binarias las que tienen 0 no están entrenadas; en cambio, usando bipolares, todas las entradas serán entrenadas, apresurando así el proceso de convergencia. Aquí sólo usaremos entradas binarias, [véase Hertz *et al.* 1993].

Con un perceptrón unicapa es posible resolver únicamente problemas de clasificación de objetos que sean linealmente separables por un hiperplano (plano generalizado) en un espacio patrón *n-dimensional*, de lo contrario sin importar cómo sea entrenada, la red no podrá ejecutar la tarea. (ver [Minsky y Papert, 1969]).

Un ejemplo de esta limitación del perceptrón es tratar de resolver el problema XOR, el cual veremos a continuación.

El problema XOR

La función O-exclusivo (XOR) tiene dos entradas y una salida que se simbolizan en la Figura 2.6.

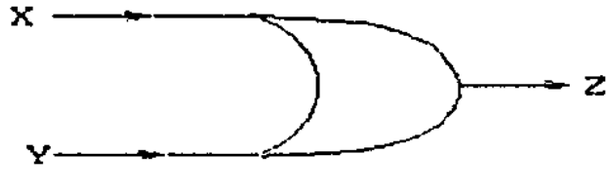

Figura 2.6: El símbolo lógico del O-exclusivo.

Esta produce una salida sólo si una u otra de las entradas está encendida, pero no si ambas están encendidas o si ambas están apagadas. Representando encendido con 1 y apagado con 0, podemos escribir esto como en la Tabla 2.1.

Entrada	Entrada	Salida
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 2.1. Función O - exclusivo.

Queremos que el perceptrón aprenda a resolver este problema: la salida es 1 si X está encendida y Y está apagada, o si Y está encendida y X está apagada, en otro caso la salida es 0. Es decir, deseamos que el perceptrón

considere a las entradas X y Y que den una salida igual a 1 como miembros de una clase y a las entradas X y Y que den una salida igual a 0 como miembros de otra clase (distinta a la primera clase), y que además separe a una clase de la otra correctamente. Así el perceptrón habrá clasificadora las Xs y Ys en su clase respectiva.

Podemos dibujar esto en un espacio patrón como el de la Figura 2.7.

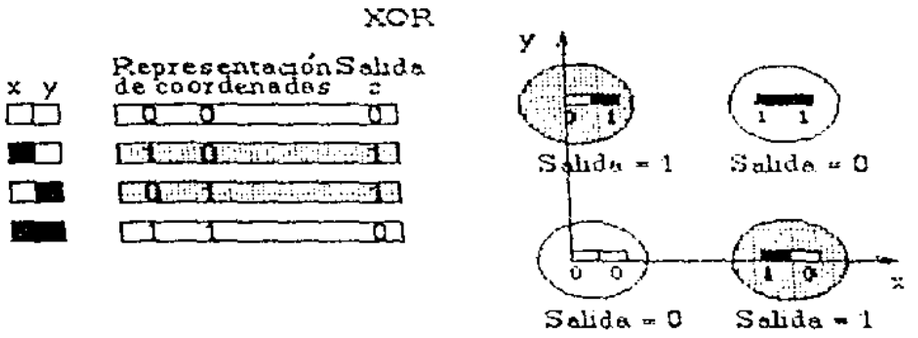

Figura 2.7: Representación del espacio patrón en el problema XOR.

Ahora, si el eje X representa el valor de X, el eje Y el valor de Y. Los círculos oscuros representan las entradas que producen una salida de 0. Considerando los círculos oscuros como elementos de una clase y los círculos claros como elementos de otra clase ajena a la anterior, no podemos hallar una línea recta que separe a las dos clases. Por lo tanto, tales patrones no son linealmente separables (sección 1.2) y por ello el perceptrón unicapa no puede resolver el problema.

Capítulo 3

Perceptrones multicapa

3.1 Historia

El estudio de *RN* aplicado a modelos computacionales se originó en el escrito de McCulloch y Pitts publicado en 1943. En éste se introdujo el modelo descrito en el capítulo anterior.

Durante los siguientes quince años se trabajó mucho en la lógica completa (ésta puede hacer cualquier cálculo que una computadora ordinaria digital puede ejecutar, aunque no necesariamente así de rápido o convenientemente) de las redes umbral. Las redes son capaces de cálculos de todo tipo y se analizaron como máquinas de estados finitos, con Marvin Minsky en 1967. El problema de hacer una red que diera resultados acertados o lo más aproximados posibles a la verdadera respuesta utilizando ya sea elementos con cálculos inexactos, con interferencias, etc. fue resuelto por Von Neumann quien usó la redundancia. Esto llevó, tiempo después, a representaciones de *distribución redundante* con Vinograd y Cowan, (ver [Beale y Jackson, 1991]).

En el extremo opuesto a la lógica, se desarrollaron teorías continuas. Este enfoque usó ecuaciones diferenciales para describir patrones de actividad en grandes masas neuronales y se le conoce como *neurodinámica o teoría de campo neuronal* estudiada por Rashevsky, Beurle, Wilson, Cowan y Amari. (ver [Beale y Jackson, 1991]).

Alrededor de 1960 hubo una ola de actividades centradas en el grupo de Frank Rosenblatt, enfocándose en el problema de cómo hallar pesos apropiados, W_i $(i = 1..., n)$, para tareas computacionales particulares. Ellos usaron redes llamadas *perceptrones*, donde las unidades estaban organizadas en ca-

pas con conexiones retroalimentadas entre una capa y la siguiente.

Redes muy similares llamadas ADALINES fueron inventadas casi al mismo tiempo por Widrow y Hoff. (ver [Beale y Jackson. 1991]).

Para los *perceptrones unicapa o simples*, que son aquellos sin capas intermedias, Rosenblatt demostró la convergencia de un *algoritmo de aprendizaje*, una forma de cambiar los pesos repetitivamente para realizar el cálculo deseado. Muchas personas creyeron que tales máquinas serían la base de la inteligencia artificial, y efectivamente así sucedió.

Sin embargo Minsky y Papert en su libro "Perceptrons" señalaron una limitación del algoritmo de aprendizaje: el algoritmo sólo se aplica a los problemas cuya estructura. En el libro se demuestran fallas del perceptrón unicapa al resolver problemas con separabilidad no lineal. Pero la demostración de que es incapaz de resolver problemas de este tipo es difícil y se abandonó. En el libro se analizan las capacidades y limitaciones del perceptrón unicapa. Minsky y Papert mostraron que algunos cálculos elementales no podrían ser hechos por el perceptrón simple *(PS)* de Rosenblatt (aunque hay autores que a un PS de tres capas lo nombran así, pues consideran las entradas como una capa además de la capa de salida. Pero aquí convendremos en nunca contar las líneas de entrada como una capa. Así la Figura 3.1 será una red de dos capas). El ejemplo más simple para mostrar los límites del PS es el problema del O - exclusivo (XOR): se necesita una unidad individual de salida para encender (n = +1) si una o la otra de dos líneas de entrada está encendida, pero no cuando ninguna o ambas entradas están prendidas.

Rosenblatt estudió estructuras con más capas de unidades y creyó que podrían sobrepasar las limitaciones de los *PS*. Sin embargo, no encontró un algoritmo de aprendizaje para determinar los pesos necesarios para implementar un cálculo dado. Minsky y Papert dudaron que pudiera ser encontrado alguno y mejor exploraron otras aproximaciones a la inteligencia artificial. Con esto, la mayoría de los científicos en computación dejaron de investigar el algoritmo de aprendizaje en una *RN* para resolver cualquier tipo de problema por casi 20 años.

Pese a ello, algunas personas continuaron desarrollando la teoría de la *RN* en los años 70's. En esta labor se estudió la *memoria asociativa de contenido direccionable*, (ver [Kohonen, 1977]), en la cual diferentes patrones de entrada son asociados con algún otro, es decir, dan la misma respuesta si son muy similares. Esto lo propusieron mucho antes Taylor y Steinbuch, y fueron redescubiertos después por Anderson, Willshaw, Marr y Kohonen. Grossberg reformuló el entendimiento del problema general del aprendizaje en una red.

Marr desarrolló teorías de red del cerebelo, la neocorteza cerebral (cubierta en capas de células que cubre la superficie del cerebro en la parte frontal), y del hipocampo (estructura prominente del suelo del ventrículo lateral del cerebro), asignando funciones específicas para cada tipo de neurona. Varias personas, incluyendo a Marr, Vonder Malsburg, y Cooper, estudiaron el desarrollo y funcionamiento del sistema visual, (ver [Beale y Jackson, 1991]).

Cragg y Temperley reformularon la red McCulloch - Pitts como un sistema de giro (magnético), (ver [Hertz y et al, 1993]) muy conocido dentro de la física. Se creyó que la memoria residía en la histéresis (ver [Trigg, 1991; Michels, 1956]) de los patrones dominantes esperados para dicho sistema. Luego, Caianiello construyó una teoría estadística, usando ideas de mecanismos estadísticos, utilizando aprendizaje con las ideas de Hebb acerca del aprendizaje en el cerebro. El mismo tema fue retomado en los años 70's por Little y de nuevo en 1981 por Hopfield. Hopfield ayudó a comprender mejor el aprendizaje en las RN al utilizar, desde un punto de vista de la física, una *función energía* y enfatizar memorias como atractores dinámicamente estables. Hinton, Sejnowski y Peretto usaron *unidades estocásticas* que siguen la función $Y = f_h(\sum_{i=1}^{n} W_i X_i - \Theta)$ sólo cometiendo "errores" de aproximación con una cierta probabilidad similar a la temperatura en mecánica estadística. El poder verdadero de la mecánica estadística fue entonces traído para mostrar el problema de red estocástica por Amit, usando métodos desarrollados en la teoría de sistemas magnéticos desordenados llamados *cristos de espín*, (ver [Hertz y et al, 1993]).

Sin embargo, quizás el desarrollo más sobresaliente en ésta década se basa en los perceptrones de Rosenblatt interrumpidos hace 20 años. Varias personas han desarrollado un algoritmo que trabaja eficientemente para ajustar las unidades de conexión de peso en capas sucesivas de *perceptrones multicapa (PMC)*. Conocido como *retropropagación* parece que su fundador fue primero Werbos a mediados de los años 70's, e independientemente lo redescubrieron en 1986 Rumelhart, Hinton y Williams, y Parker en 1982. Le Cun también propuso un algoritmo relacionado con el de retropropagación. Aunque la retropropagación no da a un algoritmo general, la capacidad completa de enseñar una tarea computacional arbitraria a una red, sí puede resolver muchos problemas, tales como XOR, que los perceptrones de una capa simple no podrían solucionar. La mayoría de la actividad actual se centra en la retropropagación y sus extensiones, (ver [Beale y Jackson, 1991]).

El desarrollo tecnológico de las *RN* lo han realizado actualmente univer-

sidades como Caltech y compañías de alta tecnología como TRW, General Electric, y Texas Instruments para luego llevarse a la práctica (por ejemplo, Nestor y Hecht - Nielsen). Esta tecnología se abre paso en muchas disciplinas, incluyendo psicología, biología, fisiología, matemáticas, física, ciencias de la computación y lingüística.

3.2 ¿Qué son los perceptrones multicapa?

Para aumentar la utilidad de los perceptrones unicapa surgieron los *perceptrones multicapa (PMC)* que son similares a los primeros sólo que añadiéndoles una o más capas intermedias.

Los *PMC* son redes jerárquicas antealimentadas en capas que consisten de una *retina* (cuyo único papel es alimentar patrones de entrada dentro de la red, tomando así información desde el mundo exterior), de una *capa de salida* (donde el resultado es el cálculo leído, el cual es una salida visible para el mundo externo) y de una o varias capas intermedias llamadas *capas ocultas* porque no tienen una conexión directa con el mundo exterior, ni entrada o salida y actúan como mediadores entre las unidades de entrada y las unidades de salida. Las capas ocultas permiten al sistema formar una representación interna del problema, dando así más potencia a los *PMC* que a los unicapa, y para ello es indispensable que las relaciones entre las activaciones de las capas no sean lineales. Además, si el número de unidades en las capas ocultas es muy pequeño, la red necesitará de muchas interconexiones para entrenarse y le costará mucho esfuerzo recordar con exactitud.

Como las "unidades" de entrada (las que están en la retina) no juegan un papel significativo, es usual que la *capa de entrada* no se cuente; así, para una red con una capa oculta diremos que es de dos capas (la capa oculta y la capa de salida). De esta manera se tiene que una red de N capas tiene N capas de conexiones y $N-1$ capas ocultas. El número de capas ocultas y la cantidad de neuronas en ellas están estrechamente vinculados con la complejidad del problema a resolver. Así, usando estas magnitudes como parámetros, es posible modificar la capacidad de la red para resolver diferentes tipos de problemas. Se sabe que un perceptrón de dos capas puede resolver el caso más general de *RP*: el de la clasificación no lineal.

En la Figura 3.1 se da la representación gráfica del *PMC*.

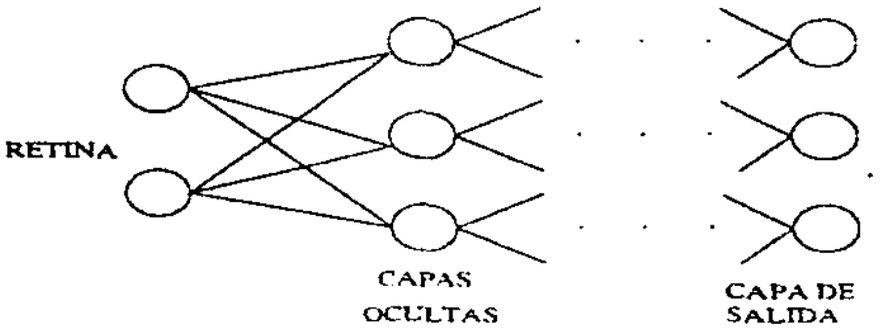

Figura 3.1: Arquitectura del PMC.

3.2.1 Arquitectura del PMC

Este tipo de *RN* está formado, como ya se mencionó, por una primera fila de neuronas que es llamada retina, varias capas ocultas y una capa de salida.

En la retina hay n neuronas de entrada que forman un vector $X \in R^n$ de rasgos o atributos que se encarga de dar la información a la red; la capa de neuronas de salida da la información ya procesada como un vector en R^r.

Un *estado del perceptrón* es un punto en el espacio multidimensional Ω en el cual se representan los pesos W, asignados a las conexiones sinápticas.

Para los *PMC* usaremos una función escalón más suave que para el caso de los perceptrones simples. En este cambio haremos que en mayor o menor cantidad encienda o apague, como antes, pero además, en dicha función existe una pendiente entre el momento de encendido y el de apagado que dará alguna información de las entradas para así saber cuándo necesitamos fortalecer o debilitar los pesos relevantes. De este modo la red aprenderá. Dos posibilidades para la nueva función umbral son mostradas en las Figuras 3.2 y 3.3.

En ambas funciones, el valor de la salida es muy cercano a uno si la suma ponderada excede demasiado al umbral, y es muy cercano a cero si la suma ponderada es mucho menor que el umbral. Cuando el umbral y la suma ponderada son casi iguales, la salida de la neurona tendrá un valor entre los dos extremos. Esto significa que la salida de la neurona es capaz de ser afín a sus entradas en un modo más útil e informativo que en el caso de la función umbral para un perceptrón simple, donde sólo ocurre que si la suma

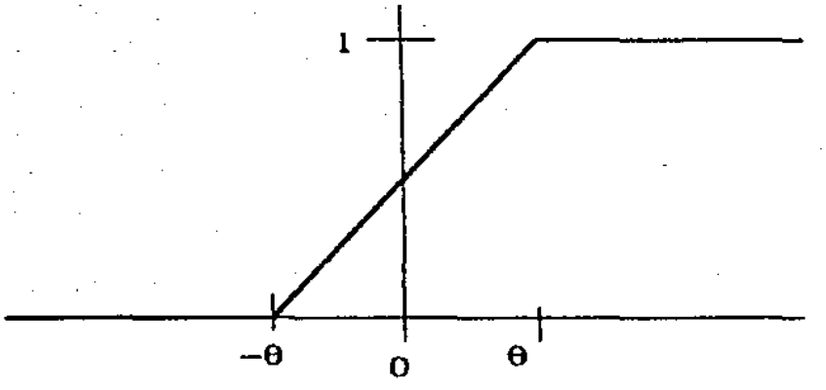

Figura 3.2: Función umbral con límites entre 0 y 1.

ponderada excede al umbral la salida será 1 y en caso de no excederlo será 0. Así la entrada ya no es sólo encendido o apagado, sino que se mantiene en un cierto rango.

Note que se alteró el modelo para anular la dificultad del proceso límite de alcanzar el umbral que disfraza las entradas de las salidas, y entonces al ajustar el modelo esto puede resolverse.

El *PMC* usa una función no lineal, que le permite resolver problemas más complicados que los posibles de solucionar con un perceptrón simple, y tiene capas ocultas, mientras que los perceptrones simples usan una función escalón lineal y no tienen capas ocultas. Por estas diferencias es que la regla de aprendizaje del *PMC* es diferente de la del perceptrón simple.

De las dos funciones usaremos la *función umbral* de tipo *sigmoide logística* para que la red aprenda a clasificar objetos.

3.2.2 El nuevo aprendizaje

La regla de aprendizaje para los *PMC*, que es la generalización de la regla delta para perceptrones simples, es llamada la *regla delta generalizada* y da las representaciones internas necesarias sobre las unidades ocultas; fue desarro-

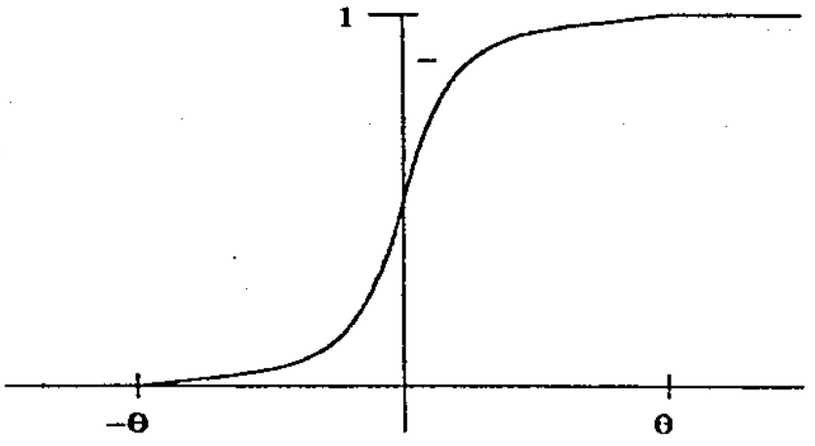

Figura 3.3: Función umbral sigmoide.

llada en 1986, principalmente por Rumelhart y McClelland, quienes además, produjeron redes multicapa a las cuales investigaron y caracterizaron. Su libro "Parallel Distributed Processing" es uno de los más importantes en el campo. (ver [Rumelhart y McClelland, 1986]).

La operación de la red es similar a la del perceptrón simple. Aquí utilizamos la *retropropagación* que es un algoritmo de aprendizaje para los *PMC*, en donde los pesos son modificados (utilizando la regla delta generalizada) propagando un error "hacia atrás" desde las salidas hacia las entradas.

Usar la función de tipo sigmoide logística significa que suficiente información acerca de la salida estará disponible para las unidades en capas primarias, y entonces esas unidades pueden tener sus pesos ajustados para que decrezca el error la siguiente vez.

Cuando mostremos a la red no entrenada un patrón de entrada, producirá una salida aleatoria. Conocemos el patrón "correspondiente" (es decir, tenemos un aprendizaje supervisado) y necesitamos definir una función error que dé la diferencia entre éste y la salida de la red. Para un aprendizaje adecuado necesitamos que la salida de la red se aproxime a la salida deseada, esto es, que el valor de la función error se reduzca continuamente. Esto se hace ajustando los pesos en las conexiones entre las unidades (en caso necesario) mediante la regla delta generalizada que calcula el valor de la función error para esa entrada particular, y entonces retropropaga el error de una capa

anterior.

La retropropagación consiste de dos pasos:

1. En el paso hacia adelante, las entradas proceden a través de la red y generan una salida.

2. En el paso hacia atrás, la diferencia entre las salidas deseadas y las de la red, genera una señal de error que se propaga de regreso a través de la red en forma de términos sucesivos delta para enseñarle a aproximarse y así producir la salida deseada.

La convergencia del algoritmo se puede mejorar haciendo lo siguiente:

1. Asignar valores de patrón discretos, digamos -0.9 y 0.9 en lugar de -1 y 1 para alejarse de características limitadoras de activación.

2. Limitar las activaciones de la capa de salida en un solo lado, dependiendo del signo del error.

3. Omitir la limitante de activación en la capa de salida en conjunto.

4. Añadir los términos de conexión de salto de capa dentro de capas seleccionadas.

Se puede reemplazar el criterio de minimización del error cuadrático con otro basado, por ejemplo, en errores absolutos o en la raíz cúbica del error.

Otra técnica generalmente más útil es cambiar las ganancias de optimización como procedimientos de entrenamiento; se pueden también usar ganancias de optimización individual para componentes de error diferentes.

Si la red resuelve el problema, habrá descubierto un conjunto de pesos que producen la salida correcta para cada entrada.

El ajuste de pesos para unidades en la capa de salida es sencillo, debido a que la salida de la red y la deseada son conocidas, pero para unidades en las capas ocultas el ajuste no es tan obvio. Intuitivamente, podemos conjeturar que las capas ocultas conectadas a las salidas con un gran error

tendrán que ajustar muchas veces sus pesos, mientras que las capas ocultas conectadas a salidas casi correctas tendrán que ajustar poco sus pesos. Los pesos para una unidad particular serán ajustados en proporción directa al error en las unidades a las cuales están conectados; esto sucede porque la retropropagación de estos errores a través de la red permite que los pesos entre todas las capas sean correctamente ajustados. Así la función error es reducida y la red aprende.

A continuación encontraremos condiciones sobre los pesos que aseguren que la función error es decreciente.

3.2.3 Descripción matemática del proceso de aprendizaje del PMC

Una red aprende por repeticiones sucesivas de un problema, disminuyendo los errores con cada iteración. Lo que vamos a hacer es buscar los pesos que minimicen el error para tódas las unidades de la red. La función más usada para encontrar el error es:

$$E_p = \frac{1}{2}\sum_j (t_{pj} - O_{pj})^2 , \tag{3.1}$$

donde E_p es la función error para el patrón p, t_{pj} es la salida deseada para el patrón p en la unidad j, y O_{pj} es la salida de la red en la unidad j para el patrón p.

El factor $\frac{1}{2}$ hace más simples los cálculos matemáticos, pues en ellos se tiene que calcular la derivada de E_p que tiene un término elevado al cuadrado.

La activación de cada unidad j, para el patrón p, es la suma ponderada:

$$net_{pj} = \sum_i W_{ij} O_{pi} , \tag{3.2}$$

donde W_{ij} es el peso desde la unidad i hasta la unidad j.

La salida de cada unidad j es la función umbral f_j actuando en la suma ponderada. Aquí usaremos la función de tipo sigmoide logística, aunque puede optarse por cualquier función monótona creciente, continuamente diferenciable. Entonces:

$$O_{pj} = f_j(net_{pj}) \ . \tag{3.3}$$

A fin de obtener el valor de los pesos que minimizan el error usamos las técnicas estandard del cálculo, es decir, hay que obtener la derivada de la función error, E_p, con respecto a W_{ij}, que es el peso entre las unidades i y j. Así, usando la regla de la cadena tenemos:

$$\frac{\partial E_p}{\partial W_{ij}} = \frac{\partial E_p}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial W_{ij}} \ . \tag{3.4}$$

Viendo el segundo factor de la igualdad en 3.4, y substituyéndolo en 3.2 se tiene,

$$\frac{\partial net_{pj}}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \sum_{k=1}^{N} W_{kj} O_{pk} = \sum_{k=1}^{N} \frac{\partial W_{kj}}{\partial W_{ij}} O_{pk} = O_{pi} \ . \tag{3.5}$$

el último resultado se obtiene de las siguientes igualdades:

$$\sum_{k=1}^{N} \frac{\partial W_{jk}}{\partial W_{ij}} O_{pk} = \frac{\partial W_{j1}}{\partial W_{ij}} O_{p1} + \frac{\partial W_{j2}}{\partial W_{ij}} O_{p2} + \ldots + \frac{\partial W_{ji}}{\partial W_{ij}} O_{pi} + \ldots + \frac{\partial W_{jN}}{\partial W_{ij}} O_{pN}$$

$$= O_{pi} \ .$$

debido a que si $k \neq i$ entonces $\frac{\partial W_{jk}}{\partial W_{ij}} = 0$ y si $k = i$ entonces $\frac{\partial W_{ji}}{\partial W_{ij}} = 1$.

Definamos el cambio en el error como una función, δ_{pj}, del cambio que hay en las entradas de la red a una unidad como:

$$-\frac{\partial E_p}{\partial net_{pj}} = \delta_{pj} \ . \tag{3.6}$$

y así 3.4 es:

$$-\frac{\partial E_p}{\partial W_{ij}} = \delta_{pj} O_{pi} \ . \tag{3.7}$$

El decremento del valor de E_p por lo tanto significa que el peso cambia proporcionalmente a $\delta_{pj} O_{pi}$, es decir:

$$\frac{\partial E_p}{\partial W_{ij}} = -\delta_{pj} O_{pi} \ . \tag{3.8}$$

Necesitamos conocer qué δ_{pj} es para cada una de las unidades pues si lo conocemos podemos disminuir el error, E. Usando 3.6 y la regla de la cadena:

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} = -\frac{\partial E_p}{\partial O_{pj}} \frac{\partial O_{pj}}{\partial net_{pj}} \ . \tag{3.9}$$

Ahora calcularemos a los dos últimos factores en la ecuación anterior. Para $\frac{\partial O_{pj}}{\partial net_{pj}}$ consideramos la ecuación 3.3 así:

$$\frac{\partial O_{pj}}{\partial net_{pj}} = f_j'(net_{pj}) \ . \tag{3.10}$$

Para obtener $\frac{\partial E_p}{\partial O_{pj}}$ consideremos el primer factor en la segunda igualdad de 3.9 y también a 3.1

$$\frac{\partial E_p}{\partial O_{pj}} = -\sum_j (t_{pj} - O_{pj}) \ . \tag{3.11}$$

Lo anterior lo obtenemos de la siguiente forma:

$$\frac{\partial E_p}{\partial O_{pj}} = \frac{\partial \frac{1}{2}\sum_j (t_{pj} - O_{pj})^2}{\partial O_{pj}} = 2\frac{1}{2}\sum_j (t_{pj} - O_{pj})(-1) = -\sum_j (t_{pj} - O_{pj}) \ .$$

Entonces sustituyendo 3.10 y 3.11 en 3.9:

$$\delta_{pj} = f_j'(net_{pj}) \sum_j [(t_{pj} - O_{pj})] \ . \tag{3.12}$$

Esto es útil para las unidades de salida, pues la salida deseada y la salida de la red están disponibles, pero no para las unidades ocultas, pues sus salidas deseadas son desconocidas.

Así, para el caso en que la unidad j no está en la unidad de salida (es decir, si está en la capa oculta), podemos escribir, por la regla de la cadena:

$$\frac{\partial E_p}{\partial O_{pj}} = \sum_{k=1}^{N} \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial O_{pj}} \ . \tag{3.13}$$

y ahora sustituyendo a 3.2:

$$= \left(\sum_{k=1}^{N} \frac{\partial E_p}{\partial net_{pk}}\right) \left(\frac{\partial}{\partial O_{pj}} \sum_{i=1}^{M} W_{ik} O_{pi}\right) \ .$$

Desarrollando el segundo factor de la igualdad anterior:

$$\frac{\partial}{\partial O_{pj}} \sum_{i=1}^{M} W_{ik} O_{pi} = \frac{\partial W_{1k}}{\partial O_{pj}} O_{p1} + \frac{\partial W_{2k}}{\partial O_{pj}} O_{p2} + \ldots + \frac{\partial W_{jk}}{\partial O_{pj}} O_{pj} + \ldots + \frac{\partial W_{Mk}}{\partial O_{pj}} O_{pM}$$

$$= W_{jk}$$

$$= -\sum_{k=1}^{N} \delta_{pk} W_{jk} \ . \tag{3.14}$$

Usando 3.12, sustituyendo 3.11 y a su vez sustituyendo 3.14 tenemos:

$$\begin{aligned} \delta_{pj} &= f_j'(net_{pj})(t_{pj} - O_{pj}) \\ &= f_j'(net_{pj}) \left(-\frac{\partial E_p}{\partial O_{pj}} \right) \ , \end{aligned}$$

entonces,

$$\delta_{pj} = f_j'(net_{pj}) \sum_k \delta_{pk} W_{jk}. \tag{3.15}$$

Esta ecuación representa el cambio en la función error, con respecto a los pesos en la red y da un método para que la función error sea segura de reducir. La función es proporcional a los errores δ_{pk} en unidades subsecuentes, así el error se tiene que calcular en las unidades de salida primero (3.12) y entonces pasarlo hacia atrás a través de la red hasta las primeras unidades para alterar sus pesos de conexión. Este regreso del valor del error es el que hace que se les llame redes de retropropagación.

En resumen, para entrenar a las redes multicapa fundamentalmente se realizan dos pasos:

1) Para las uniones que van a las unidades de salida el error puede calcularse directamente y la ecuación 3.12, indica cómo obtener las derivadas.

2) Para unidades ocultas, la derivada depende del valor calculado en todas las capas posteriores. Es decir, δ_{pj} , en la ecuación 3.15, debe retropropagarse para calcular las derivadas.

Ahora veremos algunas características de la función, $f(net)$ que elegimos. La función de tipo sigmoide logística:

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

$0 < f(net) < 1$, con k una constante positiva que controla la "extensión" de la función (valores grandes de k comprimen la función hasta que k tiende a infinito y $f(net)$ tiende a la función Heaviside).

Esto, además, actúa como un control de ganancia: para entradas más pequeñas la entrada es más pronunciada y la función cambia rápidamente (la ganancia es mayor), para entradas grandes la pendiente es menor y entonces la ganancia es menor. Esto significa que la red puede aceptar grandes entradas y aún permanecer sensible a cambios pequeños.

Otra razón para usar la función de tipo sigmoide logística es que su derivada es simple. Como la salida de la unidad, O_{pj} es:

$$O_{pj} = f(net) = \frac{1}{1 + e^{-knet}} ,$$

la derivada con respecto a aquella unidad, $f'(net)$ es:

$$\begin{aligned} f'(net) &= \frac{ke^{-knet}}{(1 + e^{-knet})^2} \\ &= \left(\frac{k}{1 + e^{-knet}}\right)\left(\frac{e^{-knet}}{1 + e^{-knet}}\right) \\ &= \left(\frac{k}{1 + e^{-knet}}\right)\left(\frac{1 + e^{-knet} - 1}{1 + e^{-knet}}\right) \\ &= k\left(\frac{1}{1 + e^{-knet}}\right)\left(1 - \frac{1}{e^{-knet}}\right) \\ &= kf(net)(1 - f(net)) \\ &= kO_{pj}(1 - O_{pj}) . \end{aligned}$$

Usando las ecuaciones 3.12 y 3.15 podemos dar a continuación el algoritmo de retropropagación.

3.2.4 Algoritmo de entrenamiento para el PMC usando retropropagación

Para que el PMC aprenda a clasificar objetos usaremos la función de tipo sigmoide logística

$$f(net) = \frac{1}{1 + e^{-knet}},$$

debido a que su derivada es simple (es una función umbral no lineal, continuamente diferenciable, es decir, es suave dondequiera).

Entonces, los pasos a seguir en el algoritmo de entrenamiento para el PMC usando retropropagación son:

1. Iniciar los pesos y los umbrales.

   Coloque todos los pesos y umbrales en pequeños valores aleatorios.

2. Presentar la entrada y la salida deseada.

   Presentar la entrada $X_p = X_0, X_1, \ldots X_{n-1}$ y la salida deseada $T_p = t_0, t_1, \ldots t_{m-1}$, donde n es el número de unidades de entrada y m es el número de unidades de salida.

   Asignar $-\Theta$, las diagonales,a W_0 y fijar $X_0 = 1$.

   Para la asociación de patrón, X_p y T_p representan los patrones para ser asociados.

   Para la clasificación, T_p es iniciando en cero, excepto en el elemento que corresponda a la clase que contenga a X_p, en cuyo caso T_p será igual a 1.

3. Calcular la salida de la red.

   Cada capa calcula

$$Y_{pj} = f(\sum_{i=0}^{n-1} W_i X_i) \ .$$

y pasa el valor de la función Y_{pj} como una entrada a la siguiente capa. Las salidas de la capa final evalúan O_{pj}.

4. Adaptar los pesos.

Iniciar desde la capa de salida, y trabajar hacia atrás, (retropropagar), es decir, hacia la capa de entrada.

$$W_{ij}(t+1) = W_{ij}(t) + \eta \delta_{pj} O_{pj} \ .$$

$W_{ij}(t)$ representa los pesos desde la unidad i hasta la unidad j en el tiempo t, η es un término de ganancia y δ_{pj} es un término de error para el patrón p en la unidad j. El término de error es:
para las unidades de salida

$$\delta_{pj} = kO_{pj}(1 - O_{pj})(t_{pj} - O_{pj}) \ ,$$

para las unidades ocultas

$$\delta_{pj} = kO_{pj}(1 - O_{pj}) \sum_{k} \delta_{pk} W_{jk} \ ,$$

donde la suma se hace a partir de las unidades k en la capa superior a la unidad j.

Las unidades ocultas deberían llamarse "detectores de rasgos aprendidos" o "unidades de re-representación" porque el patrón de asociación en la capa oculta es una codificación de lo que la red "piensa", son los rasgos característicos de la entrada. La representación de pesos y la de patrones de activación están relacionadas cercanamente, pero una es más informativa que la otra.

No se pueden medir los pesos entre las neuronas biológicas cerebrales pero sí se puede medir su respuesta para varias entradas. Por ello es que se probó a hacer lo mismo para analizar RN artificiales y funcionó. En problemas geométricos, Lang y Witbrock mostraron que es útil desplegar la respuesta de la unidad para muestreos sistemáticos de puntos en la región de entrada,

especialmente cuando la red tiene más de una capa oculta. Esto también se practica en las investigaciones de neurociencia clásicas del sistema visual, graficando la velocidad de disparo de las neuronas corticales mientras se varían patrones de estímulos presentados a la retina.

Ahora veremos como los PMC resuelven el problema XOR.

3.2.5 Volviendo al problema XOR

El problema XOR fue descrito en el capítulo anterior al explicar las limitaciones del perceptrón simple, el cual no lo puede resolver, pues es un problema con separabilidad no lineal. Sin embargo, para resolverlo se puede utilizar un perceptrón de dos capas, con dos unidades de entrada (ya que hay dos variables en el problema) y pueden ser una o más unidades en la capa oculta y una unidad en la salida. Los pesos están en las conexiones y el umbral dentro de la unidad. Un ejemplo se ve en la Figura 3.4.

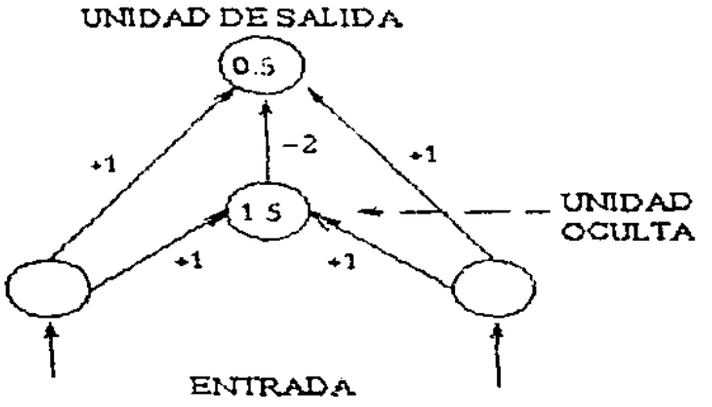

Figura 3.4: Una solución al problema XOR: la unidad de salida enciende (lo cual se representa con un 1) sólo si alguna de las unidades de entrada está encendida. Los números que aparecen en las flechas de conexión son los pesos de las uniones y los que están dentro de las unidades (círculos) son los valores umbrales.

No es una arquitectura convencional retroalimentada, pero es interesante porque sólo necesita dos unidades: la unidad oculta calcula un AND e inhibe

la unidad de salida cuando ambas entradas están encendidas.

Entrada	Entrada	Unidad Oculta	Salida
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabla 3.1. Función XOR.

La unidad de salida es alimentada con tres informaciones: si la entrada izquierda está encendida, si la entrada derecha está encendida, y si ambas entradas (izquierda y derecha) están encendidas (esto último debido a que la capa oculta enciende sólo cuando ambas están encendidas). Así, la unidad de salida trata a la unidad oculta como otra unidad de entrada y los aparentes patrones de entrada que recibe son ahora suficientemente disímiles como para que la clasificación sea aprendida.

La unidad oculta actúa como un *detector de rasgo*. Esto puede ser visto como un recodificamiento o *representación interna* de las entradas básicas para que la red pueda aprender el mapeo requerido de los patrones de entrada para las salidas. Dadas suficientes unidades ocultas es posible formar representaciones internas de cualquier patrón de entrada tal que las unidades de salida puedan dar la respuesta correcta para una entrada específica.

Es posible producir diferentes topologías de red para resolver un mismo problema.

Otra solución al problema XOR es el de la Figura 3.5.

En el caso en que no hay conexión directa desde la entrada a la salida podemos tener una red como en la Figura 3.6.

Y otra casi idéntica es la Figura 3.7.

La regla de aprendizaje no garantiza producir convergencia, sin embargo, es posible que la red caiga en una situación en la cual es incapaz de aprender la salida correcta, en cuyo caso se dice que se "estanca" en un *mínimo local* (este se explicará en la sección 3.2.9).

Otro problema menor usando la regla delta generalizada es que como los cambios de peso son proporcionales con los mismos pesos, si el sistema inicia apagado con pesos iguales, entonces los pesos desiguales nunca pueden ser desarrollados, y así la red no puede determinar la solución en caso de que los pesos no sean iguales lo cual tal vez se necesite, ver la Figura 3.8.

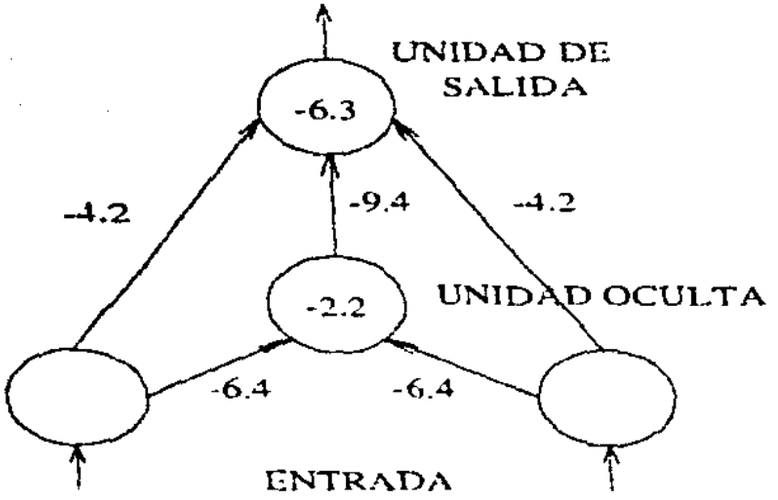

Figura 3.5: Pesos y umbrales de una red que ha aprendido a resolver el problema XOR.

El tiempo de adiestramiento de pesos con n salidas al azar es muy grande: cientos de periodos (pasando por el conjunto de entrenamiento) se requieren para obtener resultados adecuados.

Lo anterior es sencillo de hacer, en realidad lo que resulta interesante es el caso en que los pesos y umbrales no pueden ser elegidos, es decir, cuando son aleatorios. Entonces los patrones de las entradas y salidas son mostrados repetidamente y la red podría aprender los pesos necesarios para implementar cualquier problema (en particular XOR). Aún más, la red puede generalizar lo que ha aprendido. Para conjuntos de datos grandes puede reconocer patrones que nunca antes ha visto.

Para crear tal red se usa el sesgo de la neurona explicado antes en la sección.1.4.

3.2.6 Paridad

Ahora veremos una generalización de XOR, la cual es llamada paridad (igualdad o semejanza). Este problema lo realizaron Minsky y Papert, consta de N entradas y la unidad de salida necesita estar encendida si un número impar

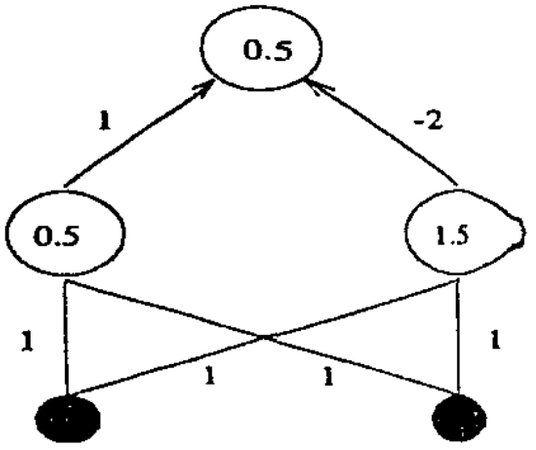

Figura 3.6: Solución al XOR.

de entradas están encendidas y apagada en otro caso. Una capa oculta de N unidades basta para resolver el problema, como se muestra en la Figura 3.9. La unidad oculta j está encendida cuando al menos las unidades j de entrada están encendidas, y uno u otro excita o inhibe la unidad de salida dependiendo de si j es impar o par. Además la retropropagación halla esta solución usando unidades de valores continuos, excepto cuando los pesos estén multiplicados por un factor muy grande.

El problema de paridad (o XOR, cuando $N = 2$) es a menudo usado para probar o evaluar los diseños de la red. Sin embargo, este es un problema muy difícil, porque la salida debería cambiar cuando alguna salida particular cambie. Esto no es típico de la mayoría de los problemas de clasificación del mundo real, el cual usualmente tiene mucha más regularidad y permite la generalización dentro de clases de patrones de entrada similares (Fahlman, 1989), véase el diagrama de la Figura 3.9.

Muchas de las características asociadas al funcionamiento de un PMC para clasificar son más fáciles de entender si se consideran en términos de la gráfica de la función error y a continuación veremos en qué consiste la misma.

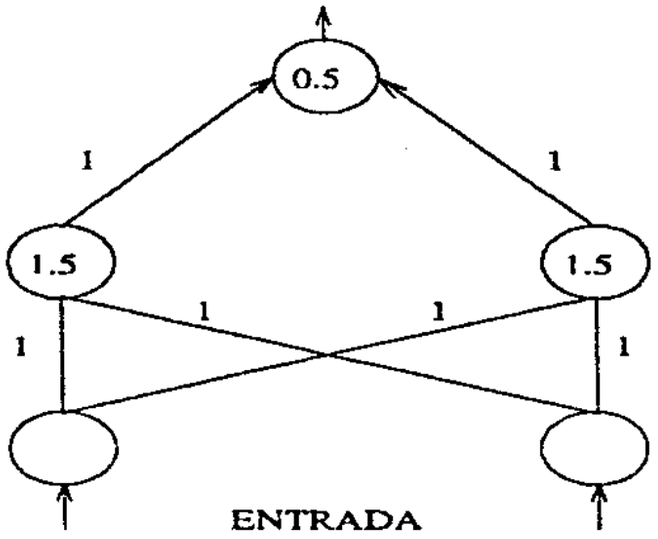

Figura 3.7: Una red resolviendo XOR con conexiones de entrada a salidas indirectas.

3.2.7 Función error o función energía

Como hemos visto, la red calcula un error o una función energía. La que hemos escrito antes es $E_p = \frac{1}{2}\sum(t_{pj} - O_{pj})^2$ y representa la diferencia entre la salida de la red y la salida deseada.

Grandes diferencias corresponden a grandes energías, pequeñas diferencias corresponden a pequeñas energías. La energía es una función de los pesos y de las entradas a la red y, de hecho, también la salida es una función de ambos.

En una red de PMC que tiene varias conexiones entre sus capas hay varios pesos asociados a cada una de estas conexiones. El comportamiento de los errores en la red se puede entender mejor si graficamos la función energía, ésta depende de n pesos, (luego, el espacio de los pesos es $n - dimensional$), por lo que su gráfica estará en un espacio de dimensión $n + 1$.

A fin de visualizar la gráfica de la función energía, conviene bosquejarla para el caso de pocos pesos.

Por ejemplo, si variamos sólo uno de todos los pesos, la gráfica de E_p tendría el aspecto de la Figura 3.10. Como puede verse en ésta, la función

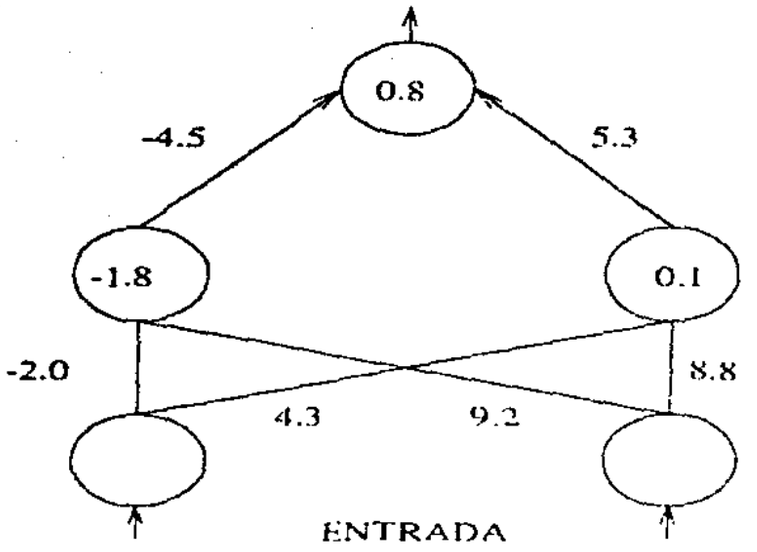

Figura 3.8: Una solución estable que no funciona.

energía tiene intervalos de crecimiento y decrecimiento. esto se debe a que el PMC da un cierto error, es decir una cierta función energía que varía de acuerdo a la elección del peso que esperamos (aunque sin garantizar que ello ocurra) dé el mínimo error posible en la siguiente iteración de la red. En la siguiente iteración el peso vuelve a cambiar para disminuir más aún el error y así obtenemos otro error. Si realizamos este proceso hasta anular o minimizar lo más posible el error la gráfica de todos los errores obtenidos es similar a la Figura 3.10.

Si ahora variamos no un solo peso sino dos pesos, tenemos un eje para cada uno de ellos y la gráfica de la función energía es una superficie en R^3 como la que se ilustra en la Figura 3.11.

Si luego consideramos tres pesos, la gráfica de la función energía estaría en R^4. En general, podemos ajustar todos los pesos en una red, dando una función energía cuya gráfica está en un espacio multidimensional. La superficie de energía es un paisaje irregular de valles y lomas, pozos y montañas. En los pozos están los puntos de energía mínima y en las cumbres está la energía máxima.

La regla delta generalizada minimiza la función de error E ajustando

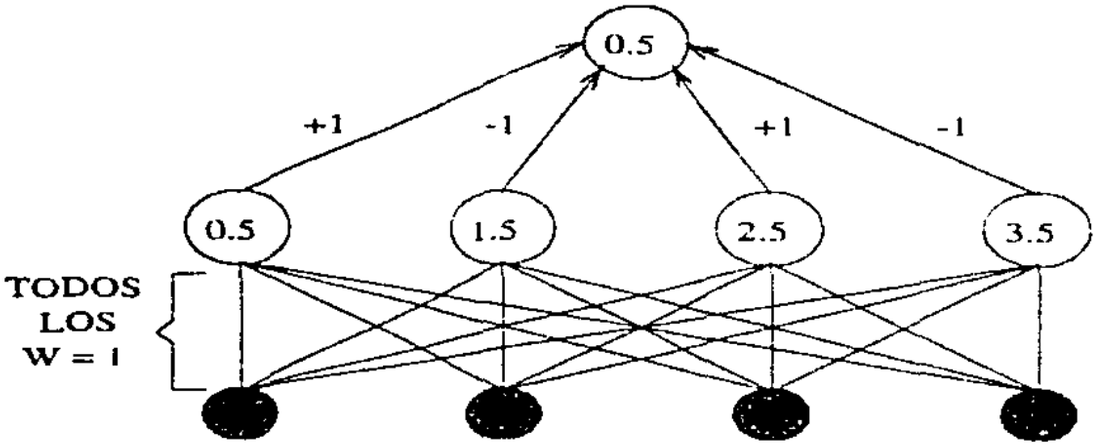

Figura 3.9: Una red que resuelve el problema de paridad para $N = 4$ con 0/1 como unidades umbrales.Los W representan los pesos de la red.

los pesos hasta que correspondan a aquéllos en los cuales la superficie de energía es la más baja. Un procedimiento para determinar el punto mínimo de la función de energía es llamado el *método del gradiente en descenso*. Dicho brevemente, éste consiste en que la función energía es calculada y los cambios son hechos en la dirección más descendente, es decir, en los pesos que indican la energía mínima.

Cada peso donde la función energía alcanza su mínimo corresponde a una solución de la red, es decir que para valores de energía mínimos, que es donde existe el error mínimo, la red ha logrado clasificar al objeto correctamente. Cada posible solución está representada como un hueco pequeño, o una cuenca en el paisaje. A estas se les llama *cuencas de atracción*, y representan las soluciones a los valores de los pesos que dan la salida correcta desde una entrada dada, es decir, la solución de la energía más baja que la red ha aprendido.

En el espacio de los pesos cada punto (que es una única combinación de valores peso) define un diferente paisaje de energía, donde las variables son los patrones y sus energías correspondientes, ver la Figura 3.12.

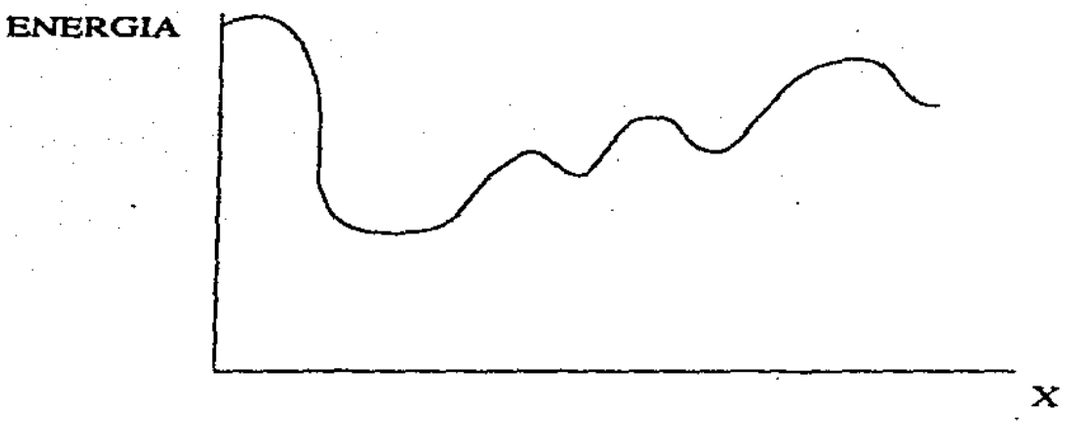

Figura 3.10: Función energía en una dimensión, variando sólo un peso, X, para un patrón fijo.

3.2.8. PMC como clasificadores

Para un perceptrón de dos capas con dos unidades en la entrada y una en la salida, si esta última tiene un umbral que enciende sólo cuando las dos unidades de entrada están encendidas, entonces está realizando el AND lógico (que es un operador diferente de XOR).

Ya que cada una de las unidades en la primer capa define una línea en el espacio patrón, la segunda unidad produce una clasificación basada en la combinación de estas líneas. De suceder que una unidad responde con 1 si la entrada está arriba de su línea de decisión, y la otra con 1 si la misma entrada está abajo de su línea de decisión, entonces la segunda capa produce una solución de un 1 si está *arriba de la línea 1 y abajo de la línea 2*, según la Figura 3.13.

Más de dos unidades pueden usarse en la primera capa, lo cual da una partición del espacio patrón que es una combinación de más de dos líneas. Todas las regiones producidas se llaman *regiones convexas*. De éstas las hay de dos tipos:

1. *Cerradas*: Tienen límite en todo su alrededor.
2. *Abiertas*: No en todo su alrededor tienen límite.

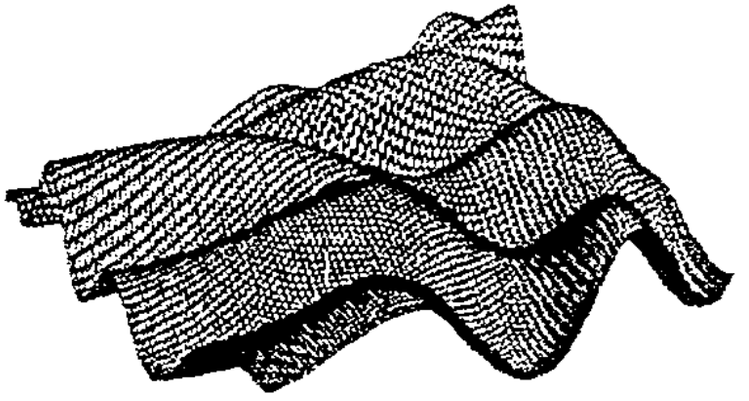

Figura 3.11: En el paisaje de energía hay varios valles que tienen muchos huecos pequeños en su fondo. Estos son mínimos locales que pueden atrapar a la solución impidiendo que se alcance el punto más profundo que, hablando globalmente y no sólo particularmente, ocurre en regiones intermedias de la misma.

La suma de más unidades en la primera capa nos permite definir más y más bordes (el número total de lados que hay en las regiones es menor o igual al número de unidades en la primera capa, y las regiones definidas serán convexas).

Sin embargo, si añadimos otra capa de perceptrones, las unidades en su capa recibirán como entradas regiones convexas, en lugar de líneas, y las combinaciones de estas regiones que no son necesariamente convexas.

Estas combinaciones se pueden intersectar, sobreponer o estar separadas unas de otras produciendo formas arbitrarias.

Entonces tres capas de unidades perceptrón pueden formar figuras arbitrariamente complejas, y separar cualquiera de todas las clases que se les presenten, ver la Figura 3.14.

Una red multicapa recibe un número de entradas que son distribuidas por una capa de unidades de entrada que no realizan alguna suma o umbralamiento (pues sólo tiene una entrada cada uno y no tiene sentido sumar

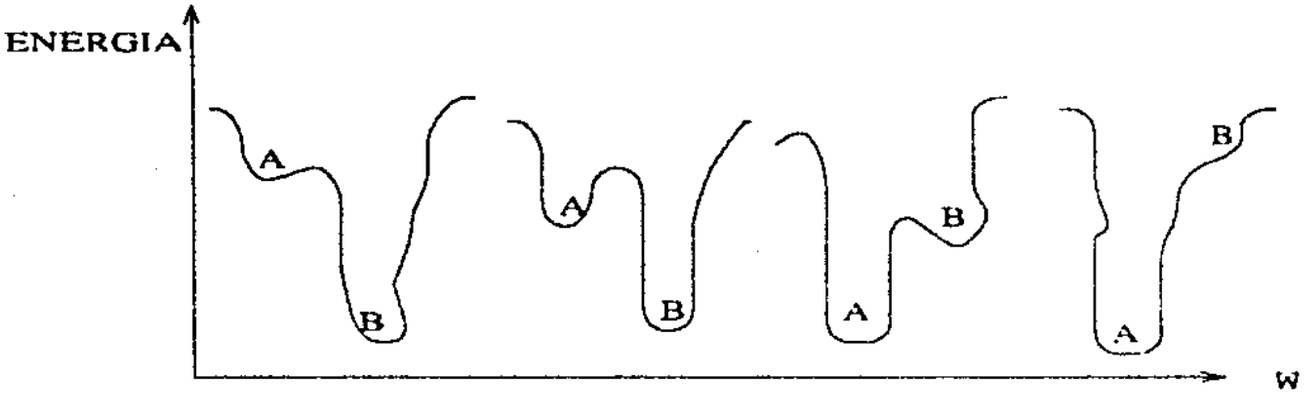

Figura 3.12: Diagrama mostrando cómo van cambiando los pesos en una red alterando el paisaje de energía.

su única entrada). Estas entradas pasan a lo largo de la primera capa de pesos adaptativos a la capa de unidades parecidas al perceptrón, la cual hace la suma y el umbral para sus entradas. Esta capa puede producir líneas de clasificación en el espacio patrón. La salida de esta capa pasa a la otra capa de unidades parecidas al perceptrón a través de pesos adaptativos y a su vez la salida de esta capa forma una región convexa en el espacio patrón. Una capa más adelante, de unidades parecidas al perceptrón, es alcanzada por otro conjunto de pesos adaptativos y la salida puede definir cualquier forma arbitraria en el espacio patrón. Contando las capas activas (de peso activo) ésta es una red de tres capas. Si se incluye el conjunto inactivo de unidades de entrada, entonces es una red de cuatro capas. Usualmente se dice que es de tres capas.

El resumen de lo anterior está en la Figura 3.15.

3.2.9 Capacidades y limitaciones de los perceptrones

McCulloch y Pitts probaron que una reunión sincronizada de PMC es capaz de un *cálculo universal* para convenientes pesos W_{ij}.

Un perceptrón de no más de tres capas es capaz de representar cualquier función sin importar qué tan compleja sea.

Las RN tienen gran habilidad para generalizar, es decir, para clasificar

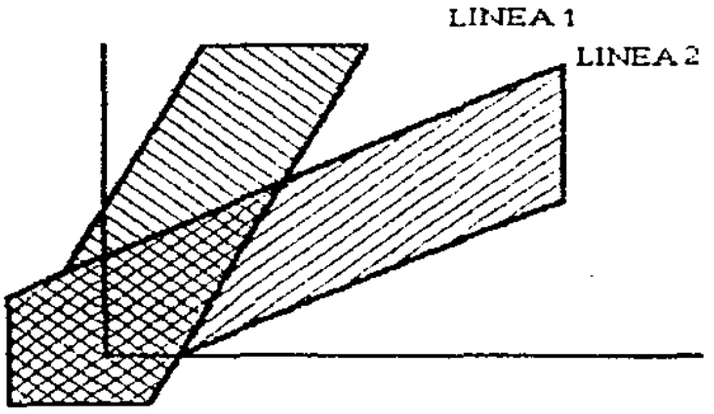

Figura 3.13: Perceptrones: la región de decisión producida por la combinación de dos perceptrones con otro perceptrón.

adecuadamente patrones que no han sido presentados previamente, es decir, los clasifica con otros que comparten los mismos rasgos distintivos.

Si a la red se le da un patrón aún no visto y este es una mezcla intermedia de dos patrones previamente enseñados lo clasificará como un ejemplo de patrón predominante. Si se le da un patrón que no corresponde a cualquier cosa semejante a lo que la red ha visto antes, entonces la clasificación será más pobre.

Las redes de *PMC* son muy tolerantes a las fallas, pues sus elementos están en paralelo. Luego si una unidad o sus pesos son dañados o perdidos, el recuerdo es dañado en su calidad, pero no puede llegar a ser una falla catastrófica sino una *degradación graciosa* pues la ejecución del sistema cae (baja) de un nivel alto para reducir el nivel, pero sin caer catastróficamente a cero.

Las *RN* son tolerantes al *ruido*, debido a que generalizan a partir de ejemplos enseñados en versiones incompletas de los patrones originales.

El *daño* a la red por pérdida de pocas unidades o por ruido en los datos de entrenamiento se puede recuperar por aprendizaje y esto es rápido frecuentemente, ver la Figura 3.16.

La convergencia a la solución original fue hacia el fondo del valle, y fue

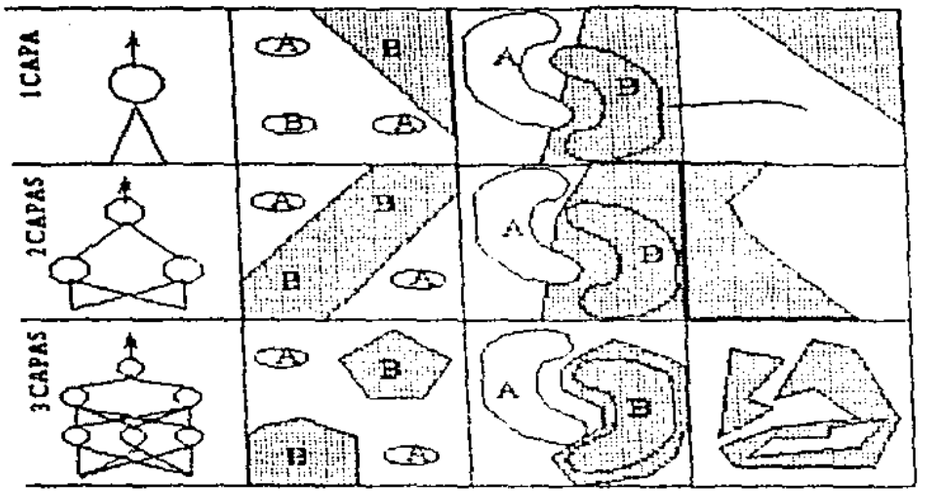

Figura 3.14: RN y sus correspondientes regiones de decisión.

bastante lenta. El daño hecho altera a la red, pero es más propenso a moverla hacia la solución correcta es decir hacia un estado que tiene un gradiente grande, y así cuando el reaprendizaje ocurre, la red se mueve hacia este gradiente pronunciado y rápidamente recobra la solución original.

Ocasionalmente la red fija una solución estable que no da la salida correcta, entonces la función energía está en un *mínimo local*. Esto significa que en cada dirección en que la red podría moverse en el paisaje de energía, la energía es más alta que en la posición actual.

Hay aproximaciones alternativas para minimizar estas dificultades de aprendizaje:

1. Disminuyendo el término ganancia.

Si la razón en la cual los pesos son alterados es progresivamente decreciente, entonces el *algoritmo del gradiente en descenso* conseguirá una mejor solución. Si el término ganancia η es grande al empezar, pasos grandes son dados a través del peso y el espacio energía hacia la solución. Conforme la ganancia decrece, los pesos de la red se fijan dentro de una configuración de

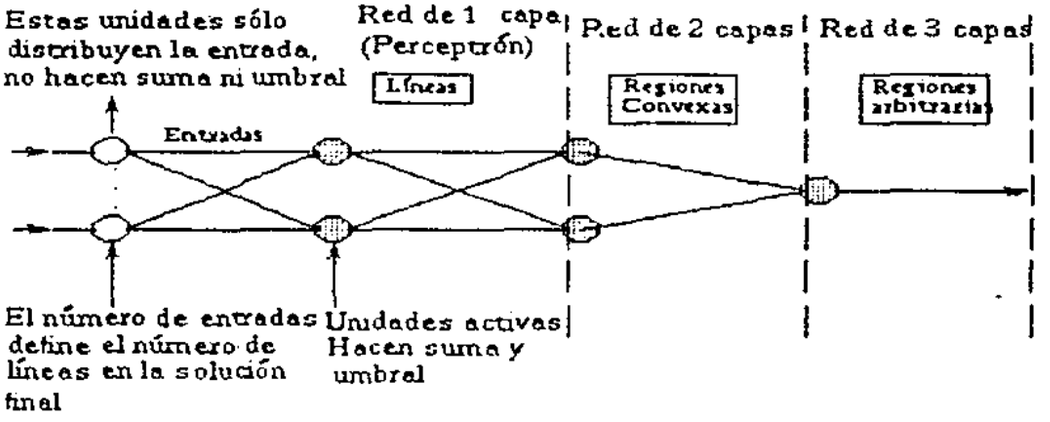

Figura 3.15: Resumen de las vecindades formadas por diferentes números de capas de perceptrón.

energía mínima sin sobrepasarse de la posición estable, conforme el gradiente desciende toma pequeños pasos cuesta abajo. Esta aproximación permite a la red rodear un mínimo local y, entonces localiza y fija algunos mínimos más profundos sin oscilar desordenadamente.

2. Adición de unidades internas.

El mínimo local ocurre cuando dos o más clases separadas son categorizadas como la misma. Esto da una pobre representación interna dentro de las unidades ocultas, y así añadiendo más unidades a esta capa tendríamos un mejor registro de las entradas y disminuiríamos la posibilidad de que estos mínimos ocurran.

3. Término momentum.

Los cambios pueden estar dando algún "*momentum*" introduciendo un término extra dentro de la ecuación de adaptación de peso que dará un gran cambio en el peso si los cambios son grandes actualmente y decrecerán conforme los cambios vayan siendo menores. Esto significa que la red es menos apropiada para fijarse en el mínimo local primeramente encendido, pues el *término momentum* empujará los cambios sobre incrementos locales en la

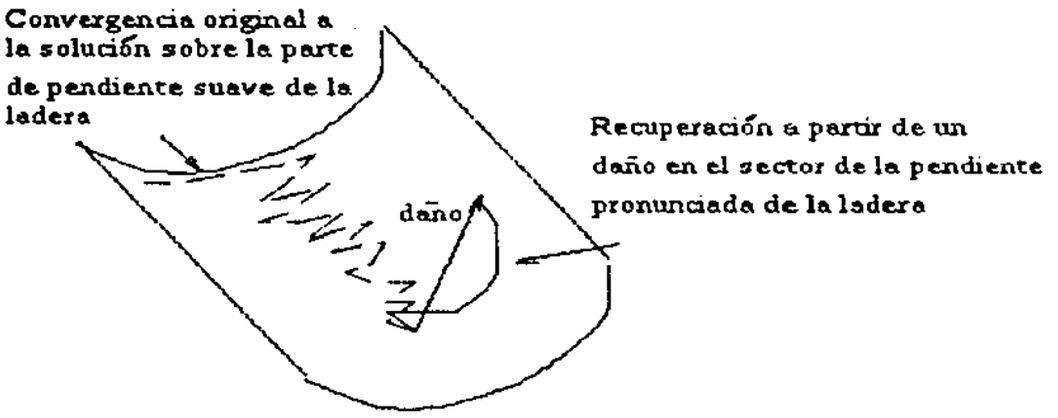

Figura 3.16: Diagrama mostrando cómo la recuperación del daño se puede lograr rápidamente.

función energía, siguiendo totalmente la dirección hacia abajo. El momentum ayuda mucho a apresurar la convergencia a lo largo de gradientes superficiales (bajos), dejando al sendero de la red llevarlo hacia la solución para tomar velocidad en la dirección cuesta abajo. El paisaje de energía podría consistir de desfiladeros de pendiente gradualmente larga las cuales terminen en mínimos. La convergencia a lo largo de estos desfiladeros es lenta, pues la dirección que tiene que ser seguida tiene un gradiente ligero, y usualmente el algoritmo oscila cruzando el valle de desfiladeros conforme serpentea hacia una solución. Esto es difícil de apresurar sin aumentar la oportunidad de sobrepasar los mínimos, pero la adición del término momentum es adecuada. Este término momentum

es:

$$\delta_p W_{ji}(t+1) = W_{ji}(t) + \eta \delta_{pj} O_{pi} + \alpha(W_{ji}(t) - W_{ji}(t-1)) \ ,$$

α es el factor momentum, $0 < \alpha < 1$, ver la Figura 3.17.

4. Adición de Ruido.

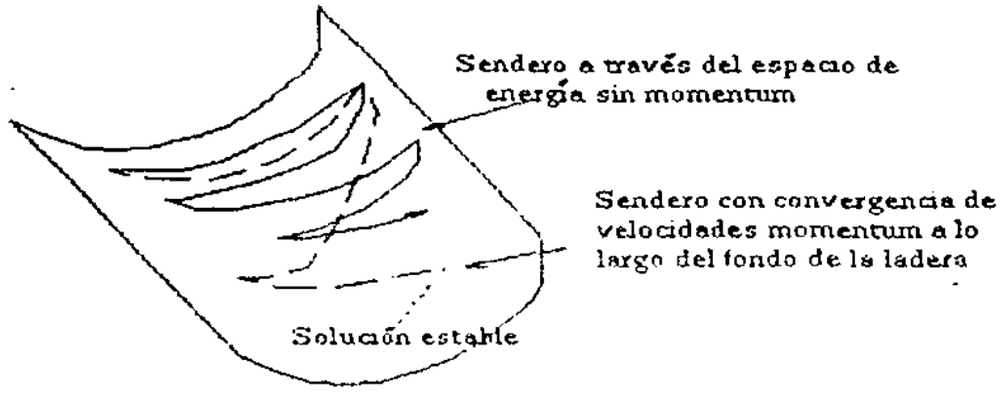

Figura 3.17: La suma de un término momentum puede apresurar la convergencia, especialmente a lo largo de la ladera.

Si el ruido aleatorio es agregado perturba el algoritmo del gradiente en descenso de la línea de descenso más empinada, y frecuentemente este ruido comunica al sistema fuera del mínimo local. Esta aproximación tiene la ventaja de que toma poco tiempo extra de cálculo, y así no es notablemente más lento que el algoritmo de gradiente en descenso.

El método de gradiente en descenso es lento para converger en un paisaje complicado, debido a la complejidad de la superficie de energía.

La adición del término momentum frecuentemente converge rápidamente, mientras que otro método altera el término ganancia η.

Otra solución que evita los falsos mínimos locales es tomar en cuenta los efectos de segundo orden en el algoritmo de gradiente en descenso. Pero el incremento de exactitud de la línea de descenso dado por esta solución es compensado por la adicional complejidad computacional implicada.

Capítulo 4

Discusión y conclusiones

Retomando lo expuesto en los capítulos anteriores podemos concluir que:

Para el método de teoría de testores:

1. La búsqueda de todos los testores típicos de la muestra de aprendizaje se realiza entre 2^n subconjuntos diferentes cuando se tienen n variables, lo cual es un espacio de búsqueda muy amplio, y por ello, quizás desventajoso de revisar.

2. Debido a lo poco utilizado de este método es imposible tener certeza de si el mismo resolverá el problema si éste es linealmente separable o carece de separabilidad lineal.

3. Entonces el caso más factible de resolver usando teoría de testores es el que tenga una matriz de objetos pequeña, entre más pequeña más posible es de solucionar.

El método de clasificación bayesiana únicamente resuelve problemas linealmente separables. En caso de que el problema sea bidimensional y con decisión cuadrática el mismo se puede reducir a un caso de clasificación lineal.

El método de perceptrones simples sólo resuelve problemas linealmente separables.

El método de RN soluciona tanto problemas linealmente separables como los que carecen de separabilidad lineal.

Los PMC:

1. Son robustos, es decir, tolerantes a las fallas, al ruido y a la información incompleta, pues aún con estos inconvenientes realizan una clasificación bastante aceptable.

2. En caso de perder información al grado de dejar de funcionar adecuadamente lo hacen lentamente sin hacerlo con brusquedad.

3. Tienen gran habilidad para clasificar patrones desconocidos para ellos.

4. Resuelven problemas linealmente separables y también los que carecen de separabilidad lineal.

Así:

1. Para resolver problemas que tienen separabilidad lineal se pueden utilizar el método bayesiano, RN, perceptrones simples y PMC.

2. Para resolver problemas que carecen de separabilidad lineal sólo se pueden usar RN y PMC.

Concluyendo, el método con más amplio rango de aplicación es el PMC; mientras que el de menor es la teoría de testores. Aunque ello puede deberse a que la teoría de testores se ha estudiado relativamente poco y ojalá más adelante sea una herramienta complementaria y quizás más útil que otros métodos para resolver ciertos problemas de clasificación.

La información de PMC es la que más me agradó debido a que es muy amplia, interesante y diversa.

Algunos temas ligados a la clasificación de objetos que no fueron abordados aquí pero merecen ser mencionados son: tipos de RN diferentes a las vistas en esta tesis (por ejemplo, la Hopfield); uno de los teoremas de Kolmogorov que asegura que si un problema es resuelto por un PMC de más de tres capas, entonces puede resolverse con uno de sólo tres capas, (ver [Beale y Jackson, 1991]); otros métodos de aprendizaje para un PMC, programas de computación que realicen clasificación de objetos con varios métodos (incluyendo los vistos en esta tesis), la aplicación de la función energía en ciertos problemas, etc.

4.1 Aplicaciones de PMC

Dentro de las RN existe un amplio campo de aplicaciones, en particular para los PMC. Algunas de ellas son:

1. *Reconocer lenguajes*, es decir distinguir entre un conjunto de palabras que un interlocutor diga, aunque en la práctica esto sólo se hace con vocabularios pequeños y palabras bien separadas entre sí.

2. *NETtalk* es un PMC que aprende a pronunciar textos en inglés y fue desarrollado por Sejnowski y Rosenberg en 1987. La principal característica de la red es que parece imitar patrones del habla como los bebés, produciendo un balbuceo incoherente primero pues los pesos son aleatorios y luego de varios periodos de aprendizaje se obtiene un lenguaje inteligible con un 90% de exactitud.

3. *Algoritmos de separación con guiones*, determina los puntos en los cuales una palabra puede ser separada usando guiones. Para los lenguajes como el danés y el alemán son escasos estos métodos y quizás una RN sea la forma más rápida para hacerlo.

4. *Reconocer objetos usando sonares*, por ejemplo para probar este PMC se reconoció una roca y metales cilíndricos en el fondo de una bahía, lo cual puede servir para localizar objetos de más interés, por ejemplo, submarinos.

5. *Conducir un carro*, Pomerleau construyó una RN controlada para conducir en carro en un camino sinuoso.

6. *Reconocer códigos escritos a mano*, con una red de retropropagación ZIP en el correo de Estados Unidos que reconoce claves o códigos de postales numéricas.

7. *Filtrar el ruido de un ECG*. Un electrocardiógrafo muestra los latidos del corazón de un paciente. Sin embargo, estos latidos no siempre son regulares y el equipo de monitoreo libera una señal a una pantalla que contiene tanto ruido que puede ser difícil ver exactamente lo que está en ella.

Además de las aplicaciones anteriores hay muchas más y para consultar las ya mencionadas con más detalle véanse [Beale y Jackson, 1991] y [Hertz y et al, 1993].

Bibliografía

[1] Abercrombie M. y Hickman M., *"Dictionary of Biology"*; Penguin. 1990.

Beale R. y Jackson T., *"Neural Computing: An Introduction"* Departament of Computer Science. University of York.. Adam Hilger. Bristol Philadelphia y New York., 1991.

Bernardo José Miguel. *"Bioestadística: una perspectiva Bayesiana"*; Vicens - Vives Universidad, 1981.

Camp Drew Van. *"Neurons for Computers"*; Scientific American. September 1992.

Freeman James A. y Skapura David M., *"Neural Networks. Algorithms, Applications, and Programming Techniques"*; Addison - Wesley, 1992.

Granino A. Korn. *"Neural Network Experiments on Personal Computers and Workstations"*; The MIT Press. Cambridge. Massachusetts. London, England. 1994.

Hecht-Nielsen Robert, *"Neurocomputing: picking the human brain"*; IEEE SPECTRUM, March 1988.

Hertz John. Krogh Anders. Palmer Richard G., *"Introduction to the Theory of Neural Computation"*; Instituto de Santa Fe. Addison-Wesley. 1993.

Helmut Späth. *"Cluster Analysis Algorithms"*; Mc Kenzie, University College. London. 1979.

Kohonen T., *"Associative Memory: a Sistem - Theoretical Approach"; Springer - Berlin, 1977.*

Kohonen T., *"Self Organization and Associative Memory"; Springer - Verlag, 1990.*

Llorente Belsis. Cedergren Robert y Miramontes Pedro. *"Las Redes de Neuronas en el Diseño de Medicamentos"*; Ciencias. Facultad de Ciencias UNAM. No. 39 Julio-Septiembre.1995.

Lernery Rita G. y Trigg George L., *"Enciclopedy of physics"*; VCH Publishers, 1991.

Mc Cord Nelson Marilyn e Illingworth W. T., "*A Practical guide of Neural Nets*"; Addison-Wesley, 1991.

Minsky M. y Paperts, "*Perceptrons*"; MIT Press, 1969.

Michels Walter C., "*The international dictionary of physics and electronics*"; DVan Nostrand, 1956.

Morales Gamboa Rafael y Miramontes Vidal Pedro, "*Theory of Testors: an alternative way of solving the problem of reduction of variables*"; Facultad de Ciencias, UNAM. 1988.

Morales Gamboa Rafael, Miramontes Vidal Pedro, "*Un Algoritmo para el Problema de Selección de Variables*"; Facultad de Ciencias, UNAM. 1989.

Obermeier Klaus K. y Barron Janet J., IN DEPTH, Neural Networks, "*Time to Get Fired Up*"; BYTE, Agust 1989.

Preparata Fanco P. e Ian Shamos Michael, "*Computational Geometry An Introduction*"; Springer-Verlag, 1985.

Ralston Anthony y Reilly Edwin D., "*Enciclopedy of Computer Science*"; Van Nostrand Reinhold; 1993.

Rumelhart D. E. y McClelland J. L., "*Parallel Distributed Processing*"; Vol. 1, MIT Bradford Press, 1986.

Tank David W. y Hopfield John J., "*Collective Computation in Neuronlinke Circuits* "; TRENDS IN COMPUTING, December 1987.

Touretzky David S.y Pomerleau Dean A., IN DEPTH, Neural Networks, "*What's Hidden in the Hidden Layers?*"; BYTE, Agust 1989.