EL SABER DE MIS HIJOS
HARÁ MI GRANDEZA

UNIVERSIDAD DE SONORA

DIVISIÓN DE CIENCIAS EXACTAS Y NATURALES

DEPARTAMENTO DE MATEMÁTICAS

Introducción a las Redes Neuronales Artificiales

TESIS

Que para obtener el Grado de:

LICENCIADO EN MATEMÁTICAS

Presenta:

LUIS OMAR FAVELA HERNÁNDEZ

Hermosillo, Sonora **27 de Agosto de 2004**

Universidad de Sonora

Repositorio Institucional UNISON

Hermosillo, Sonora a 20 de agosto de 2004

DR. OSCAR VEGA AMAYA,
Coordinador del Programa de la
Licenciatura en Matemáticas
Presente.-

Los abajo firmantes, miembros del Comité Revisor de tesis intitulado **"Introducción a las Redes Neuronales Artificiales"**, que para obtener el título de Licenciado en Matemáticas realiza el **P.L.M. Luis Omar Favela Hernández** y a quienes tuvo Usted a bien designar, le comunicamos que hemos concluido el seminario de exposición de tesis, en el cual se han realizado las correcciones y sugerencias pertinentes; consideramos que el mencionado pasante está ya en condiciones de presentar su examen profesional, por lo cual solicitamos a Usted determine la fecha de dicho examen, inmediatamente después de que el interesado entregue la cantidad de ejemplares convenida de su tesis, en las que se incluyan las correcciones y sugerencias arriba mencionadas.

Sin más por el momento, nos despedimos de Usted.

ATENTAMENTE

Comité Revisor de la tesis

M.C. Pedro Flores Pérez

M. G. García A.

Dr. Martín Gildardo García Alvarado

M.C. Sonia Guadalupe Sosa León

M.C. Israel Segundo Caballero

C.c.p. Comité Revisor
C.c.p. Interesado
C.c.p. Expediente

UNIVERSIDAD DE SONORA
DEPTO. DE MATEMÁTICAS
RECIBIDO
25 AGO 2004
RECIBIO:

AGRADECIMIENTOS

A mis padres Epifanio Favela y Alicia Hernández, por haberme dado la vida y todo el amor que me mostraron así como el apoyo que me brindaron para estudiar en la Universidad de Sonora y lograr mis metas.

A mis hermanos, Bertha, Javier, César y Epifanio, por todo el apoyo recibido durante mis estudios y realización de mi trabajo de tesis.

A mi tío Felix y su esposa Soledad, mis primos, por haberme apoyado durante mis estudios de licenciatura.

A mi director de tesis M.C. Israel Segundo Caballero, por dirigir este trabajo, por todo el apoyo que me brindo en este trabajo y en la vida. Mis más sinceras GRACIAS.

A mi asesor Dr. Martín García Alvarado, por su apoyo que me brindo durante mis últimos semestres de la carrera y en este trabajo, así como sus sugerencias y comentarios.

A mis sinodales M.C. Pedro Flores Pérez y M.C. Sonia Sosa León, por el apoyo que me brindaron durante la revisión de este trabajo y sus sugerencias.

A mis Maestros que me ayudaron a forjarme en la carrera con sus enseñanzas.

A la Lic. Yadira Jiménez Ramos, por todo el apoyo que me brindó.

A los profesores DR. Adolfo Minjáres Sosa, DR. Oscar Vega Amaya, DR. Juan Antonio Nido Valencia, M.C. José María Bravo Tapia, M.C. Miguel ángel Moreno Núñez, M.C. Carlos Robles Corbalá, M.C. Edelmira Rodriguez Alcantar, M.C. Gerardo Gutiérrez Flores, LM. Héctor Hernández Hernández, LM. Rene César Leyva Contreras, LM. Martín Manosalvas Leyva, LM. Irene Rodríguez Castillo, por la ayuda que me brindaron.

A mis amigos que siempre estuvieron apoyándome y dándome ánimos, y esa gran amistad que me demostraron.

La vida es un camino que hay que recorrer. Para poder realizarlo, hay que terminar las metas que nos proponemos.

Para mi hermana Bertha.

Gracias por tus regaños y consejos que me diste, así como el apoyo que me brindaste.

Índice General

Índice de Figuras

Introducción

El hombre se ha caracterizado siempre por su búsqueda constante de nuevas vías para mejorar sus condiciones de vida. Estos esfuerzos le han servido para reducir el trabajo en aquellas operaciones en las que la fuerza juega un papel importante. Los progresos obtenidos han permitido dirigir estos esfuerzos a otros campos, como por ejemplo, a la construcción de computadoras que ayuden a resolver de forma automática y rápida determinadas operaciones que resultan tediosas cuando se realizan a mano.

Las computadoras de hoy en día son capaces de realizar millones de operaciones por segundo, resolver problemas matemáticos y científicos, crear, manipular y mantener bases de datos, procesar textos, gráficos, realizar autoedición y comunicaciones electrónicas. Sin embargo, aún no es posible que una computadora resuelva un problema que no admita un tratamiento algorítmico y que sea capaz de aprender a través de la experiencia, en lugar de realizar un conjunto explicito de instrucciones dadas por el ser humano.

Las investigaciones actuales de los científicos se dirigen al estudio de las capacidades humanas como una fuente de nuevas ideas para el diseño de las nuevas computadoras. Así, la inteligencia artificial es un intento por descubrir y describir aspectos de la inteligencia humana que pueden ser simulados mediante máquinas. Esta disciplina se ha desarrollado fuertemente en los últimos años teniendo aplicación en algunos campos como visión

artificial, procesamiento de información expresada mediante lenguajes humanos, etc.

Las *redes neuronales* son un intento de simular el funcionamiento del cerebro humano. Sin embargo, debido a la complejidad de éste, es imposible construir un modelo que realice completamente todas sus funciones, pero sí es posible emular ciertas características, como la capacidad de memorizar, asociar y clasificar objetos o formas y resolver situaciones acudiendo a la experiencia acumulada.

Una *red neuronal artificial* (que abreviaremos como RNA) es un modelo simplificado del cerebro humano que consiste en un conjunto de elementos procesadores o nodos interconectados de acuerdo a arquitecturas específicas y cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano: la neurona.

En los últimos años el desarrollo de las redes neuronales artificiales ha sido sobresaliente y han sido utilizadas en diferentes áreas como son:

(a) Biología.

Reconocimiento de patrones.

Visión por computadora.

(b) Empresas:

Evaluación de probabilidad de formaciones geológicas y petrolíferas.

Explotación de bases de datos.

Optimización de plazas y horarios en líneas de vuelo.

Reconocimiento de caracteres escritos.

(c) Medio ambiente:

Análisis de tendencias y patrones.

Previsión del tiempo.

(d) Finanzas:

Previsión de la evolución de los precios.

Valoración del riesgo de los créditos.

Interpretación de firmas.

Identificación de falsificaciones.

(e) Manufacturación:

Robots y sistemas de control (visión artificial y sensores de presión, temperatura, etc).

Control de producción en líneas de proceso.

(f) Medicina:

Analizadores del habla para la ayuda de audición de sordos.

Monitorización en cirugía.

Predicción de reacciones adversas a los medicamentos.

Lectores de rayos X.

Diagnóstico y tratamiento a partir de síntomas y/o datos analíticos.

(g) Militares:

Clasificación de las señales de radar.

Creación de armas inteligentes.

Reconocimiento y seguimiento en el tiro al blanco.

El objetivo de este trabajo es dar un panorama introductorio a las RNA's, desde su nacimiento hasta los principales algoritmos básicos.

En el Capítulo I se describen las Redes Neuronales Biológicas y su relación con las RNA, y su estructura. Además de mencionar un poco de la historia de las RNA's, estudiaremos el *perceptrón* y su limitante.

En el Capitulo II describiremos las *redes con propagación hacia atrás* (Back Propagation Networks). Mostraremos también los conceptos de *la regla de mínimos cuadrados* y *regla delta*, que nos ayudaran para desarrollar y entender el algoritmo del Back Propagation, así como también un ejemplo aplicando dicho algoritmo.

En el Capitulo III se hará una descripción de las *redes de Hopfield* que se basa en la ley de Hebb, la cual nos permite determinar la manera de almacenar la información correspondiente a una clase. En este capitulo se mostrará también el algoritmo de Hopfield y su funcionamiento mediante un ejemplo realizado usando el lenguaje de cálculo simbólico *Maple* 8 .

Capítulo 1

Historia

1.1 Historia breve de las redes neuronales

El desarrollo de teorías sobre el aprendizaje y del procesamiento neuronal se produjeron aproximadamente al final de la década de 1940. En la actualidad, la computadora ha sido una herramienta indispensable para modelar neuronas individuales, así como agrupaciones de neuronas que se denominan *redes neuronales*. En similitud con el cerebro humano, una red neuronal es un agregado de células nerviosas interconectadas, como se muestra en la Figura 1.1. En la antigüedad los investigadores utilizaban modelos físicos de los órganos y músculos del hombre y animales y explicaban sus hipótesis acerca del funcionamiento por medio de modelos fabricados con los materiales disponibles en su época. En ese tiempo, anatomistas, neurólogos, neurocirujanos y psiquiatras conocían la anatomía del cerebro, pero nadie imaginaba el hacer un modelo físico de un cerebro utilizando, por ejemplo, una esfera de gelatina y a partir de ahí, elaborar una teoría que explicara la epilepsia, los sueños, etc.

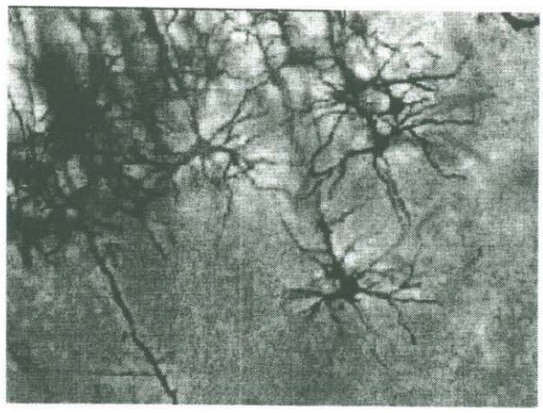

Figura 1.1: *Esquema de una red neuronal biológica*

Fueron Luigi Galvani (ver apéndice A, página 69) y sus colaboradores, en el año de 1780, quienes nos dieron las herramientas para ver el cerebro humano de una forma distinta. Observaron experimentalmente que la actividad muscular es fundamentalmente de tipo eléctrico: el cerebro emite pequeños impulsos eléctricos los cuales provocan una contracción muscular cuando se realiza alguna actividad física. Décadas más tarde, en 1920, Hans Berger(ver apéndice A, página 70) utilizó la tecnología médica de su tiempo e inventó la encefalografía, mediante la cual se registran señales eléctricas producidas por el cerebro humano, dando la pauta para modelar su funcionamiento.

1.2 Las neuronas y sus componentes

Don Santiago Ramón y Cajal (ver apéndice A, página 69) demostró que el tejido nervioso está constituido por elementos individuales, células nerviosas o *neuronas*, como se les llamó. El cerebro humano tiene aproximadamente 12 billones neuronas y cada una tiene de 56,000 a 60,000 conexiones provenientes de otras neuronas.

Una neurona está compuesta por el *axón*, las *dendritas* y *sinapsis*. El axón es

el componente que le permite comunicarse con otras mediante las corrientes iónicas responsables del potencial eléctrico que se desplaza a lo largo del axón (ver Figura 1.2) y las dendritas son conexiones que transportan los impulsos enviados desde otras neuronas y están conectadas a una membrana de la neurona. El contacto de cada axón con una dendrita se realiza a través de la sinapsis. Tanto el axón como las dendritas transmiten la señal en una única dirección. La sinapsis (ver Figura 1.3) consta de un extremo postsináptico de una dendrita existiendo normalmente entre éstos un espacio denominado *espacio sináptico*.

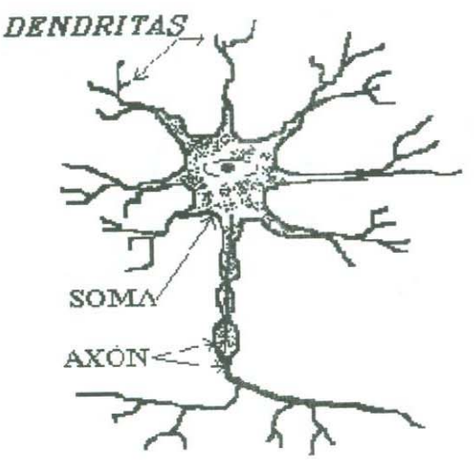

Figura 1.2: *Esquema de una neurona*

Las neuronas son eléctricamente activas e interactúan entre ellas mediante un flujo de corrientes eléctricas locales. Estas corrientes se deben a diferencias de potencial entre las membranas celulares de las neuronas. Un impulso nervioso es un cambio de voltaje que ocurre en una zona localizada de la membrana celular. El impulso se trasmite a través del axón hasta llegar a la sinapsis, produciendo la liberación de una sustancia química denominada *neurotransmisor* que se esparce por el fluido existente en el espacio sináptico. Cuando este fluido alcanza el otro extremo, trasmite la señal a la dendrita.

Los impulsos recibidos desde la sinapsis se suman o se restan a la magnitud de las variaciones del potencial de la membrana. Si las contribuciones totales alcanzan un valor determinado (denominado *umbral*) se dispara un impulso, que se propaga a lo largo del axón. Cada neurona recibe muchas señales de entrada y envía una única señal de salida.

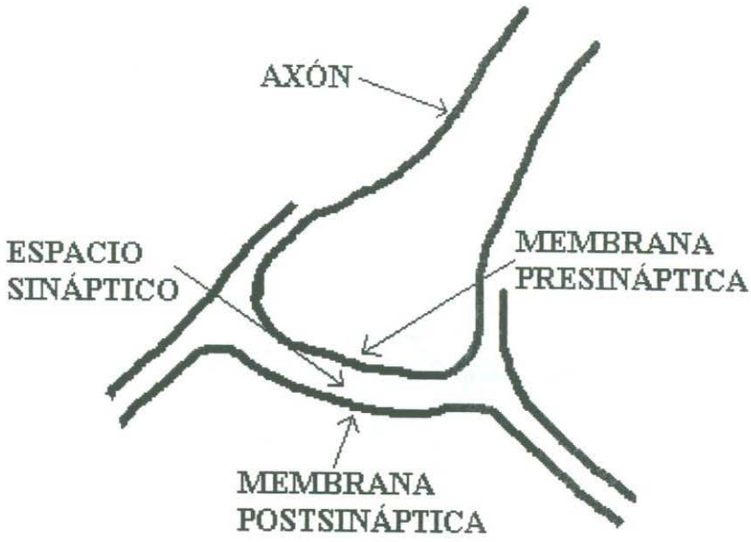

Figura 1.3: *Esquema del espacio sináptico*

En la década de 1930 había un número considerable de investigadores que se interesaban en el funcionamiento del cerebro, sin que éste fuera su disciplina principal de trabajo. Tanto matemáticos como psicólogos e ingenieros estaban interesados, por diversas razones, en modelar aspectos del funcionamiento del cerebro. En este tema destacaron dos psicólogos: Donald Hebb y Warren McCulloch (ver apéndice A, página 71), quienes a sus conocimientos del cerebro desde el campo de la psicología añadieron el aspecto matemático. Propusieron modelos muy sencillos de las neuronas y con ellos trataron de construir una teoría del cerebro basada en las redes neuronales y sus interconexiones. Su propuesta tuvo una respuesta aceptable por parte de la comunidad científica.

Como se mencionó en la Introducción, una red neuronal artificial es un conjunto de elementos procesadores o nodos (neuronas), conectados unos con otros de acuerdo a arquitecturas específicas y con pesos o intensidades modificables en la conexión (sinapsis), de una neurona a otra, que tratan de emular ciertas características de una red neuronal biológica.

Los primeros ejemplos de estos sistemas aparecieron al final de la década de 1950. La referencia histórica más trascendente es el trabajo realizado por Frank Rosenblantt (ver apéndice A, página 72), que consistió en el desarrollo un dispositivo denominado perceptrón, pero a principios de 1969 se perdió el interés debido a que M. L Minsky y S. Papert (ver apéndice A, página 73) en aquella época, seguidores de la inteligencia artificial y considerados como fundadores de la inteligencia artificial moderna, pusieron al descubierto graves deficiencias del perceptrón en su libro *Perceptrons; an introduction to computational geometry* (Cambridge, Mass., MIT Press, 1969).

En el año de 1986 las redes neuronales revivieron con un vigor extraordinario, cuando Rumelhart y McClelland(ver apéndice A, página 74), mostraron que algunas de las dimensiones que son imposibles para los perceptrones simples pueden ser resueltos por redes multi-capas con funciones no lineales, usando un procedimiento simple de entrenamiento, al que se le llamó *Back-Propagation* o *Propagación hacia Atrás*, el cual, como su nombre lo indica, tiene la función de propagar los errores producidos en la capa de salida hacia las capas anteriores. De aquí siguieron las neuronas formales de McCulloch y Pitts (ver apéndice A, página 71), los *adalines* de Widrow, hasta llegar a los modelos de Hopfield y los de Anderson (ver apéndice A, página 74).

Una de las preguntas que siempre han surgido es: ¿pueden las computadoras aprender a resolver problemas a partir de experiencias? Esta cuestión que bordeaba no hace

mucho tiempo la frontera de la ciencia ficción, es actualmente objeto de profundos estudios. Las redes de neuronas artificiales formales son estructuras que permite crear algoritmos que poseen esta capacidad de aprendizaje.

1.3 Aprendizaje de Hebb

Los sistemas neuronales biológicos no nacen preprogramados con todo el conocimiento y las capacidades que llegarán a tener eventualmente. Un proceso de aprendizaje, que tiene lugar a lo largo de un período de tiempo, modifica de alguna forma la red para incluir la nueva información.

En esta sección exploraremos una teoría relativamente sencilla del aprendizaje. La teoría básica procede de un libro escrito por Donald O. Hebb (ver apéndice A, página 71), *The organization of behavior; a neuropsychological theory* (New York, Wiley, 1949) y la llamó *Ley de Aprendizaje*, que establece lo siguiente:

*Cuando un axón de la célula **A** está suficientemente próximo para excitar a una célula **B** o toma parte de su disparo de forma persistente, tiene lugar algún proceso de crecimiento o algún cambio metabólico en una de las células, o en las dos, de tal modo que la eficiencia de **A**, como una de las células que desencadena el disparo de B, se ve incrementada.*

Para ilustrar esta idea básica, consideremos el célebre ejemplo de Pavlov, (ver apéndice A, página 70) representado en la Figura 1.4. En ella se muestran tres neuronas idealizadas que toman parte del proceso. Supongamos que la excitación de **C**, causada por la visualización de la comida, es suficiente para excitar a **B**, que da lugar a la

salivación. Además, supongamos que, en ausencia de estimulación adicional, la excitación de **A**, que consiste en oír una campanilla, no basta para excitar a **B**. Permitamos que **C** dé lugar a que **B** se active mostrando comida al sujeto, y, mientras **B** sigue activada, estimulemos **A** haciendo sonar una campanilla. Dado que **B** sigue estimulada, **A** participa ahora en la excitación de **B**, aun cuando por sí sola **A** no sería suficiente para dar lugar a que **B** se active.

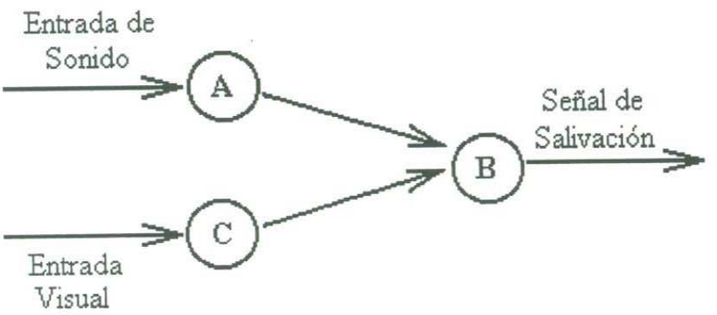

Figura 1.4: *Dos neuronas, **A** y **C**, son estimuladas por las entradas sensorial de sonido y visión, respectivamente. La tercera neurona, **B**, da lugar a la salivación.*

En esta situación, la suposición de Hebb determina que se produce algún cambio entre **A** y **B**, de tal modo que la influencia de **A** sobre **B** se ve incrementada. Si el experimento se repite con suficiente frecuencia, **A** será capaz de lograr finalmente que se dispare **B** incluso en ausencia de la estimulación visual de **C**. Esto es, si se hace sonar la campanilla, pero no se muestra la comida, seguirá produciéndose la salivación, porque la excitación debida únicamente a **A** ahora es suficiente para lograr que **B** se dispare. Dado que la conexión entre neuronas se hace a través de la sinapsis, es razonable suponer que cualquier cambio que pueda tener lugar durante el aprendizaje deberá producirse en ellas.

Hebb observó que la sinapsis se reforzaba si la neurona de entrada (presináptica) y la neurona de salida (postsináptica) eran activadas de manera continua; de esta forma, las conexiones que se usan son las que se refuerzan. Así, según este método de aprendizaje

aplicado a las redes neuronales artificiales, las conexiones entre las neuronas de entrada activas y las neuronas de salida activas se refuerzan durante el entrenamiento.

1.4 El Elemento General de Procesamiento

Los elementos individuales de cálculo que forman los modelos de sistemas neuronales artificiales se les denomina *nodos*, *unidades* o *elementos de procesamiento* (PEs, por sus siglas en Ingles, *Processing Elements*).

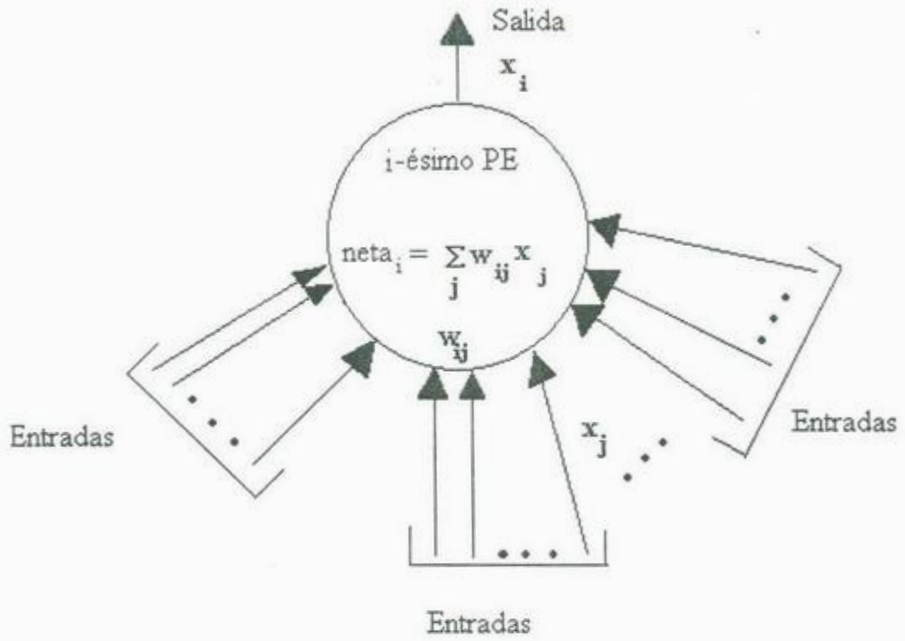

Figura 1.5: *Esta estructura representa un único PE de una red.*

La Figura 1.5 muestra el modelo general de un PE. Al igual que una neurona verdadera el PE tiene muchas entradas pero una sola salida, recibidas por el i-ésimo PE procedente del j-ésimo PE, el cual es representado por X_j (obsérvese que este valor es también la salida del j-ésimo nodo, del mismo modo que la salida generada por el i-ésimo

nodo se denota X_i). Cada conexión con el i-ésimo PE tiene asociada a él una magnitud llamada *peso o intensidad de conexión.* El peso de la conexión procedente del j-ésimo nodo que llega al i-ésimo nodo se denota mediante W_{ij}.

Todas estas cantidades tienen sus análogos en modelos de una neurona: la salida del PE se corresponde con la frecuencia de disparo de la neurona y los pesos corresponden a la intensidad de las conexiones sinápticas entre las neuronas. En los modelos estas cantidades se van a representar mediante valores reales.

Cada PE determina un valor de entrada neto basándose en todas las conexiones de entrada, el cual se calcula con la suma de los valores de entrada multiplicados por sus pesos correspondientes. En otras palabras, la entrada neta de la i-ésima unidad se puede representar de la forma

$$neta_i = \sum_j w_{ij} x_j \tag{1.1}$$

donde el índice j recorre todas las conexiones que posea el PE.

Una vez que la entrada neta ha sido calculada, se transforma en el valor de activación. Se puede escribir ese valor de activación de la forma

$$a_i(t) = F_i(a_i(t-1), neta_i(t)), \tag{1.2}$$

para denotar la activación como una función explícita de la entrada neta y del estado de la activación en el instante anterior.

Una vez que se ha calculado la activación del PE, se puede determinar el valor de salida aplicando la *función de salida*:

$$X_i = f_i(a_i). \tag{1.3}$$

Dado que normalmente $a_i = neta_i$, esta función podemos escribirla de la forma

$$X_i = f_i(neta_i). \tag{1.4}$$

La *función de activación* f_i, transforma el valor de la entrada $neta_i$, en el valor de salida del nodo X_i. Utilizaremos siempre el término función de salida para aludir a la función f_i de las ecuaciones 1.3 y 1.4.

1.5 Formulación Vectorial

En muchos de los modelos de redes neuronales resulta útil describir ciertas magnitudes en forma vectorial. Considérese una red neuronal compuesta por varias capas de elementos de procesamiento idénticos como se muestra en la Figura 1.6. Si una capa contiene n unidades, las salidas de esa capa se pueden considerar como un vector n-dimensional, $\boldsymbol{x} = (x_1, x_2, ...x_n)^t$. Los vectores escritos en negrita denotaran vectores columna. Denotaremos con $\boldsymbol{x}^t$ al vector transpuesto de $\boldsymbol{x}$.

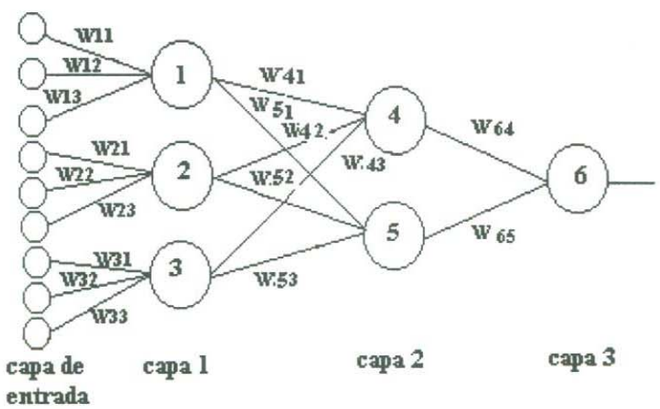

Figura 1.6: *Esquema de una red neuronal de varias capas.*

Supongamos que el vector n-dimensional de salida proporciona los valores de entrada

de todas las unidades de la siguiente capa m-dimensional. Cada una de las unidades de la capa m-dimensional poseerá n pesos asociados a las conexiones procedentes de la capa anterior. De esta manera, hay m vectores de pesos n-dimensionales asociados a esta capa; es decir, hay un vector de pesos n-dimensional para cada una de las m unidades. El vector de pesos de la i-ésima unidad se puede escribir en la forma $\boldsymbol{w_i} = (w_{i1}, w_{i2}, ...w_{in})^t$. Se añade un índice a la notación de los pesos para distinguir entre los pesos de diferentes capas.

La entrada neta de la i-ésima unidad se puede escribir en términos del producto interno o producto escalar del vector de entradas por el vector de pesos como lo ilustra la Ecuación 1.5:

$$neta_i = \sum_{j=1}^{n} w_{ij} x_j, \tag{1.5}$$

donde n es el número de conexiones en la i-ésima unidad. Esta ecuación se puede escribir de la forma resumida en notación vectorial de la forma

$$neta_i = \boldsymbol{w}_i^t \boldsymbol{x}. \tag{1.6}$$

1.6 Entrenamiento de la red

El método de aprendizaje que utilizan las redes neuronales pueden ser *supervisado* o *no supervisado*. En el caso supervisado se le presentan a la red unos datos de entradas con las correspondientes salidas que deseamos que aprenda, se calcula la salida y el error correspondiente y se ajustan sus pesos proporcionales al error que ha cometido. Para el caso no supervisado sólo se tienen los datos de entrada, por lo que los pesos se adaptan utilizando los patrones de entrada anteriores.

Utilizaremos una regla de entrenamiento que sí tenga en cuenta la eficacia de la red en cada momento,es decir, supondremos que la red está supervisada. Consideremos una red con 3 neuronas (ver Figura 1.7):

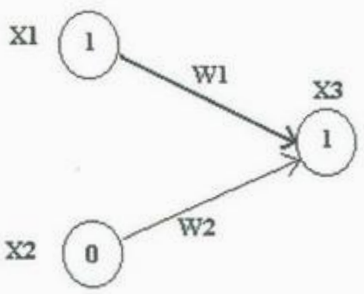

Figura 1.7: *Esquema de la intensificación del peso sináptico entre dos neuronas activas.*

1 Si la salida generada por la neurona 3 es la correcta, no se realizan ajustes en los pesos sinápticos w.

2 Si la salida es 1 pero debería ser 0, se reducen sólo los pesos de las conexiones activas en 1.

3 Si la salida es 0 pero debería ser 1, entonces se aumentan sólo los pesos de las conexiones activas en 1.

Así en cada ciclo de entrenamiento:

1.- Se presenta un dato de entrada (formado por los valores de las neuronas $\boldsymbol{X}_1$ y $\boldsymbol{X}_2$) del conjunto de datos de entrenamiento.

2.- La red, a partir del dato de entrada, generará un dato de salida.

3.- Se aplica la regla anterior, la cual mide la eficacia de la red, y se actúa en consecuencia.

Se realizan diferentes ciclos con los valores de entrenamiento hasta que la red responda correctamente a todos los datos de entrada (en todos los casos de entrenamiento).

1.7 El perceptrón

El perceptrón es un dispositivo de aprendizaje que permite clasificar objetos, con la limitante de que son separables únicamente por una línea recta, al igual que una neurona el perceptrón tiene varias entradas pero una sola salida que puede ser binaria o bipolar.

En la Figura 1.8 se muestra el esquema de un perceptrón donde $\boldsymbol{X} = \{x_1, x_2, ..., x_n\}$ es el conjunto de entradas provenientes de otras neuronas, con x_i=1 ó 0. Las entradas se ponderan con el conjunto $\boldsymbol{W} = (w_{i1}, w_{i2}, ...w_{in})^t$ que se denomina *conjunto de pesos de conexión*. La entrada neta esta dada por $\sum w_{ij}x_j$ y el valor de la salida es

$$f(neta) = \begin{cases} 1 & \text{si} \quad \text{neta} \geq \theta \\ 0 & \text{si} \quad \text{neta} < \theta \end{cases}, \tag{1.7}$$

donde θ es el valor umbral y $f(neta)$ es la función de activación.

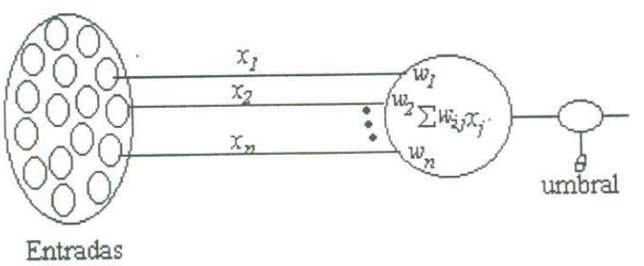

Figura 1.8: *Estructura idealizada de un perceptrón.*

Las dos funciones de activación más utilizadas son la función *escalón* y la *sigmoide*, que se muestran en la Gráfica 1.9.

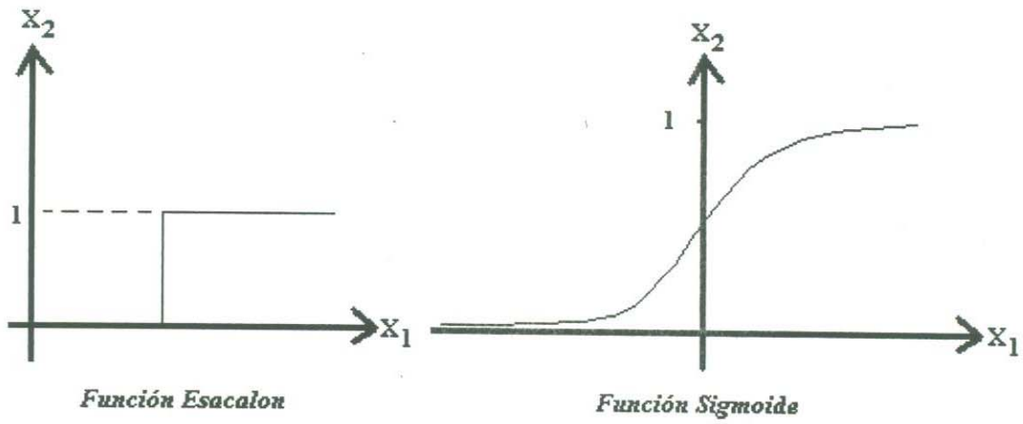

Figura 1.9: *Esquema de las funciones de activación.*

Como lo mencionamos anteriormente, la entrada neta es el producto interno del vector de entradas por el vector de pesos. A continuación analizaremos la relación

$$\theta = w_1 x_1 + w_2 x_2. \tag{1.8}$$

La Ecuación 1.8. representa una línea recta en el plano X_1, X_2. Esta línea recta parte al plano en dos regiones. Entonces se pueden clasificar los puntos de una región como pertenecientes a la clase que posee una salida de 1 y los de la otra región como la pertenecientes a la clase que posee una salida 0, así como se muestra en la Figura 1.10.

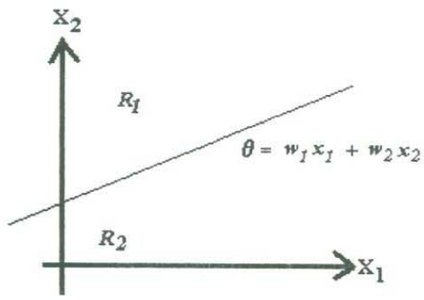

Figura 1.10: *Esquema de separabilidad de regiones.*

En el espacio tridimensional el plano puede descomponer el espacio en dos regiones distintas, dos planos pueden dar lugar a tres o cuatro regiones distintas, dependiendo de su orientación, y así sucesivamente. En general en un espacio n-dimensional, los hiperplanos que son objetos de dimensión $n-1$ con una colocación adecuada nos permiten descomponer el espacio n-dimensional en varias regiones diferentes, asícomo lo muestra la Figura 1.11.

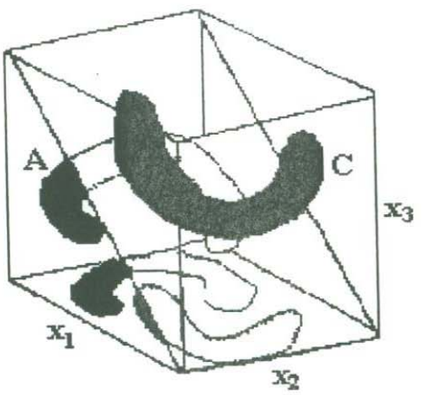

Figura 1.11: *Esquema del Espacio de Dimensión 3.*

Veamos un ejemplo sencillo utilizando la función AND y la función OR.

FUNCIÓN AND.

Los datos de entrada de la función AND se muestran en la Tabla 1.12.

x_1	x_2	S
0	0	0
0	1	0
1	0	0
1	1	1

Figura 1.12: *Datos de entrada de la función AND.*

Los vectores X_1, X_2 son las entradas que se le presentaran al perceptrón y S es la

salida que deseamos. La Figura 1.13. muestra el esquema del perceptrón para resolver el problema AND.

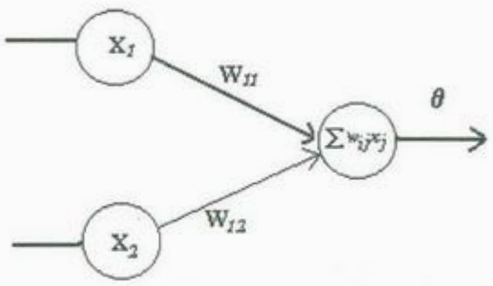

Figura 1.13: *Esquema de un perceptrón para la función AND.*

En la Figura 1.14 se muestra gráficamente que el perceptrón tiene la capacidad de clasificar grupos que son linealmente separables, es decir, divide al plano en dos regiones. Se le presentan los datos de entrada al perceptrón y mediante un proceso de aprendizaje se obtiene que los valores de los pesos $w_{11} = 1$ y $w_{12} = 2$ con un valor umbral de 1.2. En el Apéndice **B** se muestra el programa del perceptrón hecho en $C++$ que resuelve el problema AND.

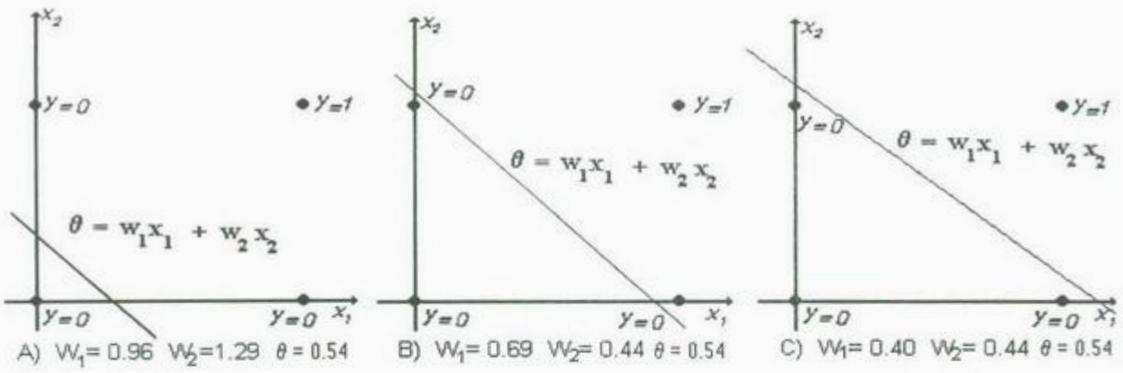

Figura 1.14: *Esquema de clasificación de la función AND.*

FUNCIÓN OR.

En la Tabla 1.15. se muestran los datos de entrada de la función OR.

x_1	x_2	S
0	0	0
0	1	1
1	0	1
1	1	1

Figura 1.15: *Datos de entrada de la función OR*

X_1, X_2 son los vectores de entrada y S es la salida que deseamos. En la Figura 1.16 se muestra el esquema del perceptrón para resolver el problema OR.

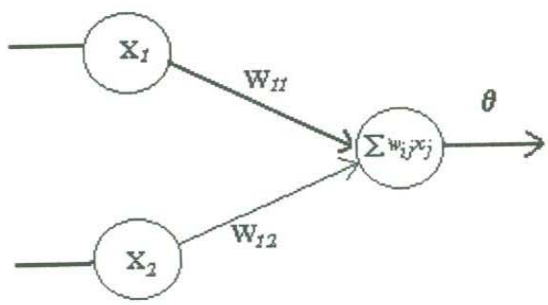

Figura 1.16: *Esquema de un perceptrón para la función OR.*

La figura 1.17. se muestra el esquema del perceptrón clasificando al problema OR, que también es linealmente separable. Al presentársele los datos de entrada al perceptrón, este empieza a realizar el aprendizaje y se obtienen los pesos adecuados $w_{11} = 1$ y $w_{12} = 1$ y un valor umbral igual a 0.5, con estos datos se puede resolver el problema OR, en el Apéndice **C** se muestra el programa del perceptrón hecho en $C++$ que resuelve el problema OR.

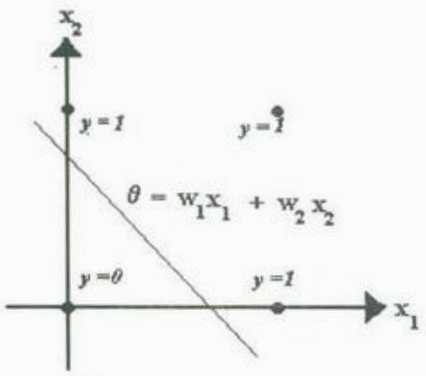

Figura 1.17: *Esquema de clasificación de la función OR.*

Con la función XOR mostraremos la limitante del algoritmo de aprendizaje del perceptrón.

FUNCIÓN XOR.

Los datos de entrada de la función XOR se muestran en la tabla 1.18.

x_1	x_2	S
0	0	0
0	1	1
1	0	1
1	1	0

Figura 1.18: *Datos de entrada de la función XOR*

Los vectores X_1, X_2 son las entradas que se le presentaran al perceptrón y S es la salida que deseamos. Al presentársele los datos de entrada al perceptrón y la salida que deseamos el aprendizaje que realiza el perceptrón no es capaz de obtener los pesos adecuados.

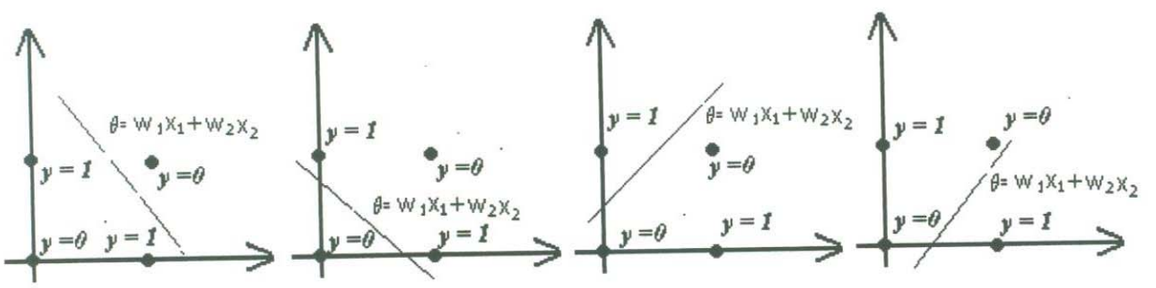

Figura 1.19: *Esquema de la limitante del perceptrón del XOR.*

Obsérvese en la Figura 1.19 la limitante del perceptrón, que no es posible clasificar mediante una sola línea recta la región solución. La única manera de encontrar la región solución es dividir el plano en tres regiones a sí como lo muestra la Figura 1.20.

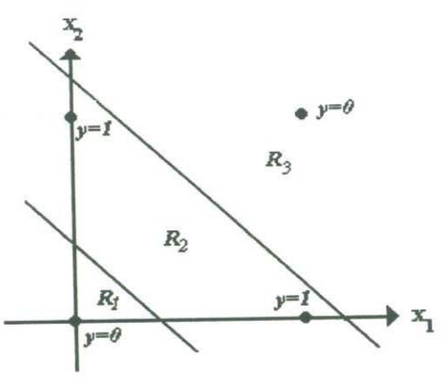

Figura 1.20: *Clasificación del plano en tres regiones.*

Esta solución se puede encontrar agregando una capa oculta de neuronas al perceptrón así como lo muestra la Figura 1.21.

En el siguiente capitulo mostraremos un algoritmo que nos ayuda a tratar este problema de clasificación y así poder resolver el problema de la función XOR.

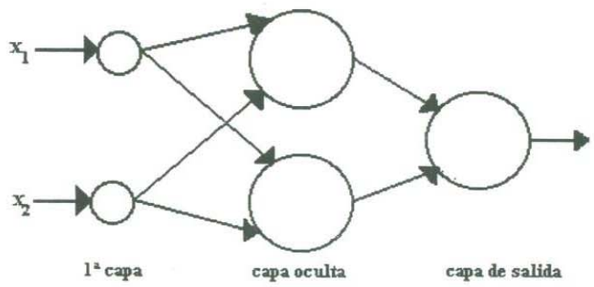

Figura 1.21: *Esquema de un perceptrón con capa oculta.*

Capítulo 2

Back Propagation

2.1 Red con Back Propagation

Una red con propagación hacia atras (BPN por sus siglas en ingles, *Back Propagation Network*) es una red multicapa, como se muestra en la Figura 2.1. esta compuesta por

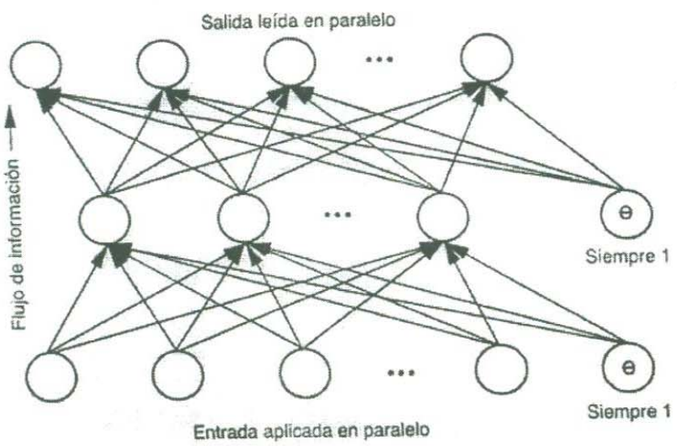

Figura 2.1: *Esquema de una red multicapa.*

una capa intermedia o capa oculta, la cual tiene varias entradas y una salida, donde cada

nodo es un perceptrón. Esta red es también conocida como *perceptrón multicapa.* Su entrenamiento se realiza mediante el algoritmo de Back Propagation.

2.2 Funcionamiento de una BPN

Al entrenar un perceptrón multicapa, se busca minimizar los errores de la capa de salida y obtener un valor muy parecido a la salida deseada. Para ello se realizan dos grandes pasos que consisten en lo siguiente.

Paso 1: Se le presenta un patrón como entrada a la red que estimula a la primera capa y este estímulo se propaga a todas las capas superiores hasta generar una salida. Se compara con la salida deseada y se calcula el error. Este se propaga hacia atrás a cada una de las capas intermedias y se actualizan los pesos de la conexión de cada unidad en cada capa. Esto se repite mediante un proceso iterativo hasta obtener la salida deseada.

Paso 2: Una vez entrenada la red, cuando se le presente un patrón de entrada arbitrario que contenga ruido, las unidades de las capas ocultas de la red responderán a una salida activa si la nueva entrada contiene un patrón que se asemeje a aquella característica que las unidades individuales hayan aprendido a recordar durante su entrenamiento.

El problema central cosniste en encontrar el vector de pesos para el cual el error entre la señal producida por la red y la señal deseada sea mínima. A continución mostraremos dos maneras de resolver este problema en una red monocapa con el fin de hacer una extensión a la red multicapa.

2.3 La regla de aprendizaje de mínimos cuadrados (LMS).

La regla de aprendizaje de mínimos cuadrados (o LMS, por sus siglas en Inglés *Least Mean Square*) consiste en lo siguiente.

Sean x_k los vectores de entrada asociados a la salida deseada d_k, con $k = 1, 2, \ldots, L$ donde x_k es un vector de dimensión n, d_k es un escalar. Sea y_k la salida obtenida. El objetivo consiste en encontrar el vector de pesos $\boldsymbol{w}$ para el cual el error sea mínimo.

Sea $\epsilon_k = d_k - y_k$, donde $y_k = \boldsymbol{w}^t \boldsymbol{x}_k$ Definimos el valor esperado del error de la siguiente forma:

$$\begin{aligned}
\xi &= \frac{1}{L}\sum_{k=1}^{L} \epsilon_k^2 \\
&= \frac{1}{L}\sum_{k=1}^{L} (d_k - y_k)^2 \\
&= \frac{1}{L}\sum_{k=1}^{L} (d_k - \boldsymbol{w}^t \boldsymbol{x}_k)^2 \\
&= \frac{1}{L}\sum_{k=1}^{L} ((d_k)^2 - 2d_k \boldsymbol{w}^t \boldsymbol{x}_k + (\boldsymbol{w}^t \boldsymbol{x}_k)^2) \\
&= \frac{1}{L}\sum_{k=1}^{L} (d_k)^2 - \frac{1}{L}\sum_{k=1}^{L} 2d_k \boldsymbol{w}^t \boldsymbol{x}_k + \frac{1}{L}\sum_{k=1}^{L} (\boldsymbol{w}^t \boldsymbol{x}_k)(\boldsymbol{w}^t \boldsymbol{x}_k).
\end{aligned}$$

Utilizando la propiedad

$$\boldsymbol{w}^t \boldsymbol{x}_k = \boldsymbol{x}_k^t \boldsymbol{w}$$

obtenemos

$$\begin{aligned}\xi &= \frac{1}{L}\sum_{k=1}^{L}(d_k)^2 - 2\frac{1}{L}\sum_{k=1}^{L} d_k \boldsymbol{x}_k^t \boldsymbol{w} + \frac{1}{L}\sum_{k=1}^{L} \boldsymbol{w}^t(\boldsymbol{x}_k \boldsymbol{x}_k^t)\boldsymbol{w} \\ &= \frac{1}{L}\sum_{k=1}^{L}(d_k)^2 - 2\frac{1}{L}[\sum_{k=1}^{L} d_k \boldsymbol{x}_k^t]\boldsymbol{w} + \boldsymbol{w}^t[\frac{1}{L}\sum_{k=1}^{L}(\boldsymbol{x}_k \boldsymbol{x}_k^t)]\boldsymbol{w}.\end{aligned}$$

Definamos

$$\boldsymbol{R} = \frac{1}{L}\sum_{k=1}^{L} x_k x_k^t, \qquad \boldsymbol{P} = \frac{1}{L}\sum_{k=1}^{L} d_k x_k^t.$$

Nótese que $\boldsymbol{R}$ es una matriz simétrica de $n \times n$. Por lo tanto:

$$\xi = \frac{1}{L}\sum_{k=1}^{L}(d_k)^2 - 2\boldsymbol{P}\boldsymbol{w} + \boldsymbol{w}^t \boldsymbol{R}\boldsymbol{w}.$$

Para encontrar $\boldsymbol{w}^*$ óptimo se deriva ξ y se iguala a cero. Para ilustrar como se obtiene la derivada de ξ, derivaremos individualmente cada termino de la expresión de ξ.

$$\frac{\partial[\frac{1}{L}\sum_{k=1}^{L}(d_k)^2]}{\partial w} = 0$$

$$\frac{\partial}{\partial w}(\boldsymbol{P}\boldsymbol{w}) = \begin{bmatrix} \frac{\partial}{\partial w_1}(P_1 w_1 + P_2 w_2 + \cdots + P_n w_n) \\ \frac{\partial}{\partial w_2}(P_1 w_1 + P_2 w_2 + \cdots + P_n w_n) \\ \vdots \\ \frac{\partial}{\partial w_n}(P_1 w_1 + P_2 w_2 + \cdots + P_n w_n) \end{bmatrix}$$

$$= \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_n \end{bmatrix} = \boldsymbol{P}.$$

El producto $\boldsymbol{w}^t \boldsymbol{R}\boldsymbol{w}$ es una forma cuadrática de la forma

$$\boldsymbol{w}^t \boldsymbol{R}\boldsymbol{w} = \sum_{i}^{n}\sum_{j}^{n} R_{ij} w_i w_j.$$

Derivando la ecuación con respecto a w obtenemos:

$$\frac{\partial}{\partial w}(\boldsymbol{w}^t\boldsymbol{R}\boldsymbol{w}) = \begin{bmatrix} \frac{\partial}{\partial w_1}(\sum_{i=1}^{n}\sum_{j=1}^{n} R_{ij}w_iw_j) \\ \frac{\partial}{\partial w_2}(\sum_{i=1}^{n}\sum_{j=1}^{n} R_{ij}w_iw_j) \\ \vdots \\ \frac{\partial}{\partial w_n}(\sum_{i=1}^{n}\sum_{j=1}^{n} R_{ij}w_iw_j) \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{n} R_{1j}w_j + \sum_{j=1}^{n} R_{i1}w_i \\ \sum_{i=1}^{n} R_{2j}w_j + \sum_{j=1}^{n} R_{i2}w_i \\ \vdots \\ \sum_{i=1}^{n} R_{nj}w_j + \sum_{j=1}^{n} R_{in}w_i \end{bmatrix};$$

$$= \boldsymbol{R}\boldsymbol{w} + \boldsymbol{R}^t\boldsymbol{w} \tag{2.1}$$

Como $\boldsymbol{R}$ es una matriz simétrica, la Ecuación 2.1 se reduce a

$$\frac{\partial}{\partial w}(\boldsymbol{w}^t\boldsymbol{R}\boldsymbol{w}) = 2\boldsymbol{R}\boldsymbol{w}.$$

Por lo tanto:

$$\frac{\partial \xi}{\partial w} = -2\boldsymbol{P} + 2\boldsymbol{R}\boldsymbol{w}. \tag{2.2}$$

Igualando a cero y sustituyendo $\boldsymbol{w}$ por el vector de pesos óptimo $\boldsymbol{w}^*$ obtenenmos

$$\begin{aligned} -2\boldsymbol{P} + 2\boldsymbol{R}\boldsymbol{w}^* &= 0 \\ 2\boldsymbol{R}\boldsymbol{w}^* &= 2\boldsymbol{P} \\ \boldsymbol{w}^* &= \boldsymbol{R}^{-1}\boldsymbol{P} \end{aligned}$$

Como puede observarse, el cálculo de w^* depende de encontrar la inversa de la matriz $\boldsymbol{R}$, que no necesariamente existe (por ejemplo, si los vectores de entrada son $x_1 = (1,1)^t$ y $x_2 = (2,2)^t$, entonces la matriz $\boldsymbol{R}$ es de la forma $\boldsymbol{R} = \begin{pmatrix} 5 & 5 \\ 5 & 5 \end{pmatrix}$ y esta matriz no es invertible).

A continuación utilizaremos un método iterativo, llamado *Método de Descenso más pronunciado o regla Delta*, el cuál consiste en, primero, asignar valores arbitrarios a los pesos y a partir de este punto se determina la dirección de la pendiente más pronunciada en dirección hacia abajo. Luego se modifican ligeramente los pesos para que el nuevo vector de pesos se deslice un poco más abajo en la superficie. Este proceso se repite hasta haber alcanzado el mínimo, ver Figura 2.2. Inicialmente el vector de pesos no se desliza

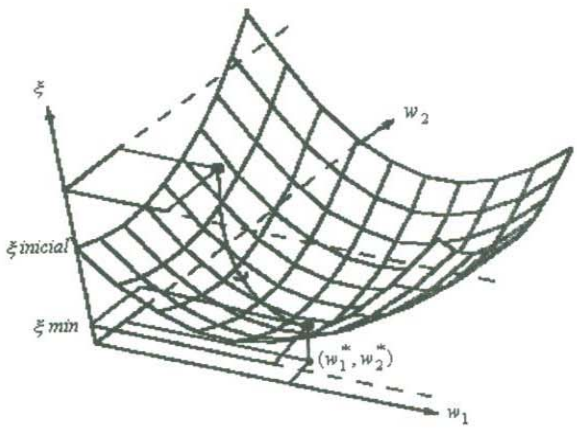

Figura 2.2: *En esta figura se muestra la visualización del método más pronunciado.*

directamente hacia el punto mínimo. La sección transversal de la superficie de pesos suele ser elíptica, de tal forma que el gradiente negativo puede no apuntar directamente hacia el punto mínimo, al menos al principio, así como se muestra en diagrama de contornos de la superficie de pesos de la Figura 2.3. Dado que el vector de pesos en este procedimiento varía, lo describiremos como una función explícita del tiempo t. Al vector inicial lo denotaremos con $w(0)$, y el vector de pesos en el instante t como $w(t)$. En el instante

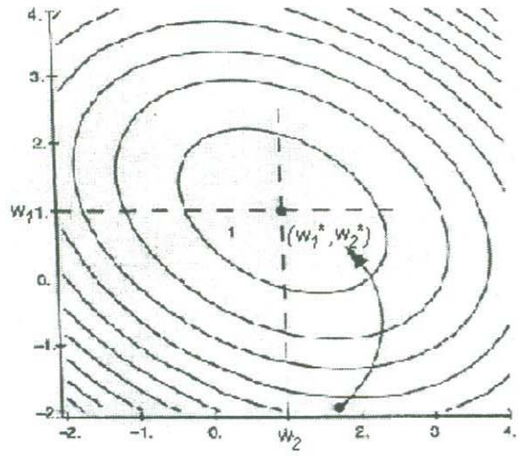

Figura 2.3: *Esquema del diagrama de contorno de pesos.*

$t+1$ el vector w es de la siguiente manera:

$$w(t+1) = w(t) + \Delta w(t), \tag{2.3}$$

donde $\Delta w(t)$ es el cambio que sufre w en el instante t.

El objetivo es encontrar w tal que el error sea mínimo. Por lo tanto el cambio que sufre $w(t)$ es proporcional al $\nabla \epsilon_k^2(w(t))$. Se considera el gradiente negativo debido que se esta buscando la dirección de descenso más pronunciado en cada punto de la superficie, lo que da lugar a la siguiente expresión.

$$w(t+1) = w(t) - \mu \nabla \epsilon_k^2(w(t)) \tag{2.4}$$

donde μ se conoce como la *tasa de aprendizaje.*

Considerando el error cuadrático medio

$$\epsilon_k^2(w(t)) = \frac{1}{2}(d_k - y_k)^2 = \frac{1}{2}(d_k - w^t(t)x_k)^2,$$

obtenemos

$$\nabla \epsilon_k^2(w(t)) = -(d_k - w^t(t)x_k)x_k$$

$$= -\epsilon_k(w(t))x_k.$$

Por lo tanto

$$w(t+1) = w(t) + \mu\epsilon_k(w(t))x_k \tag{2.5}$$

A continuación mostraremos un ejemplo.

Dados un vector de entrada y una salida deseada veamos cómo este proceso iterativo encuentra el vector de pesos óptimo para el cual el error es mínimo:

Entradas			*Salida*
x_1	x_2	x_3	D
1	0	0	2

Tabla 2.1 *Ejemplo del algoritmo LMS*

Supongamos que todos los pesos inicializan con un valor de 0.1. Aplicando el patrón de entrada obtenemos la siguiente salida

$$y = 0.1(1) + 0.1(0) + 0.1(0) = 0.1$$

El error es

$$\epsilon = 2 - 0.1 = 1.9.$$

Dando un valor arbitrario muy pequeño a μ , por ejemplo $\mu = 0.1$, obtenemos el siguiente resultado:

$$\mu\epsilon_k(t)x_k = (0.1)(1.9)\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.19 \\ 0 \\ 0 \end{bmatrix}.$$

Por lo tanto el vector de pesos actual es:

$$w = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} + \begin{bmatrix} 0.19 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.29 \\ 0.1 \\ 0.1 \end{bmatrix}.$$

Volvemos a calcular la salida

$$y = 0.29(1) + 0.1(0) + 0.1(0) = 0.29.$$

El error obtenido es:

$$\epsilon = 2 - 0.29 = 1.71.$$

$$\mu\epsilon_k(t)x_k = (0.1)(1.71)\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.171 \\ 0 \\ 0 \end{bmatrix}.$$

Por lo tanto el nuevo vector de pesos es:

$$w = \begin{bmatrix} 0..29 \\ 0.1 \\ 0.1 \end{bmatrix} + \begin{bmatrix} 0.171 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.461 \\ 0.1 \\ 0.1 \end{bmatrix}.$$

La salida actual es:

$$y = 0.461(1) + 0.1(0) + 0.1(0) = 0.461.$$

Resultados del ejemplo de la tabla 2.1.									
Iterac.	X_1	W_1	X_2	W_2	X_3	W_3	Y	D	ϵ
1	1	0.1	0	0.1	0	0.1	0.1	2	1.9
2	1	0.29	0	0.1	0	0.1	0.29	2	1.71
3	1	0.461	0	0.1	0	0.1	0.461	2	1.539
4	1	0.6149	0	0.1	0	0.1	0.6149	2	1.3851
5	1	0.75341	0	0.1	0	0.1	0.75341	2	1.24659
6	1	0.878069	0	0.1	0	0.1	0.878069	2	1.121931
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
28	1	1.889535	0	0.1	0	0.1	1.889535	2	0.110466
29	1	1.900581	0	0.1	0	0.1	1.900581	2	0.099419
30	1	1.919471	0	0.1	0	0.1	1.919471	2	0.080529
31	1	1.927524	0	0.1	0	0.1	1.927524	2	0.072476
32	1	1.934771	0	0.1	0	0.1	1.934771	2	0.065229
33	1	1.941293	0	0.1	0	0.1	1.941293	2	0.058707

Tabla 2.2 *Resultados de la tabla 2.1 utilizando el método de la regla Delta.*

En la Tabla 2.2 muestra como están cambiando los vectores de pesos en cada iteración, que después de 33 iteraciones se observa que el método está convergiendo a la salida deseada.

Como hemos utilizado una aproximación del gradiente en la Ecuación 2.5, el camino que sigue el vector de pesos al bajar por la superficie de pesos hacia el mínimo no será directamente. En la Figura 2.4 se muestra la ruta que va siguiendo el vector en busca del error mínimo utilizando el algoritmo LMS, la cual no es una curva suave, por que se está aproximando el gradiente en cada punto. Obsérvese también que el tamaño del paso se vuelve cada vez más pequeño a medida que nos aproximamos a la solución de error mínimo.

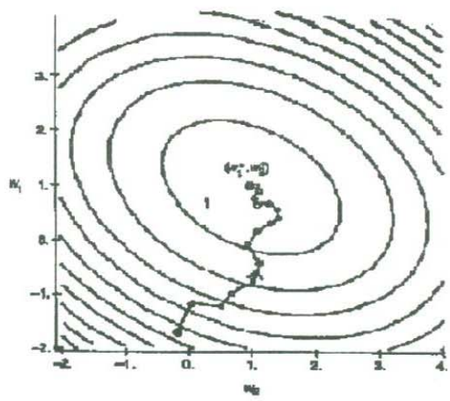

Figura 2.4: *Esquema de la búsqueda del error utilizando el algoritmo de mínimos cuadrados LMS.*

El método antes visto no es útil para resolver el problema de separabilidad no-lineal, que se nos presentó en el primer capitulo para resolver el problema XOR. Sin embargo utilizaremos los conceptos aquí vistos para resolver el problema con el método de Backpropagation.

2.4 Back Propagation

La Figura 2.5 nos muestra la arquitectura de una red multicapa sin conexiones de realimentación ni conexiones que salten de una capa para ir directamente a una capa exterior. A continuación desarrollaremos el algoritmo de Back Propagation. Supongamos que se

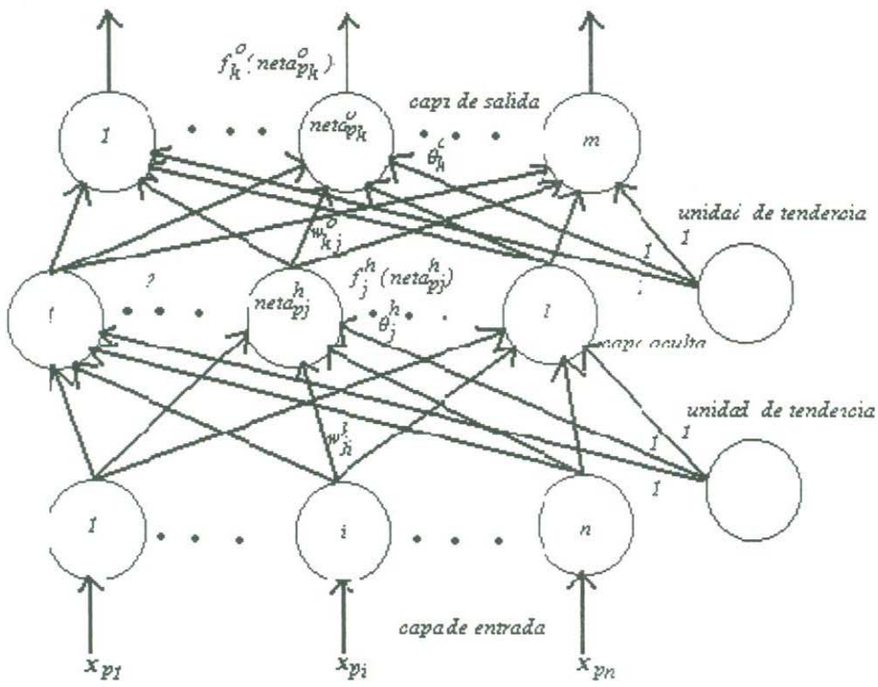

Figura 2.5: *Esquema de la arquitectura de una Red Multicapa.*

tiene un conjunto de P entradas, $\boldsymbol{X}_k = \{x_1, x_2, ..., x_n\}^t$ donde $\boldsymbol{X}_k \in \boldsymbol{R}^n$, con $k = 1, 2, ..., p$. Vamos a desarrollar un método de entrenamiento suponiendo que los pares de vectores de entrenamiento se hayan seleccionado adecuadamente y que haya un número suficiente de ellos. Veamos el funcionamiento de una BPN, revisando las ecuaciones que utiliza el procesamiento de información que hay en la red de tres capas de la Figura 2.5. Se aplica un vector de entrada $\boldsymbol{X}_p = (x_{p1}, x_{p2}, ..., x_{pn})^t$ en la capa de entrada de la red. Las unidades de entrada distribuyen los valores a las unidades de la capa oculta, donde la

entrada neta de la j-ésima unidad oculta es:

$$neta^h_{pj} = \sum_{i=1}^{n} w^h_{ji} x_{pi} \tag{2.6}$$

en donde w^h_{ji} es el peso de conexión procedente de la i-ésima unidad de entrada, x_{pi} es la salida de la i-ésima neurona, x_{p0} es una entrada mas y es igual a 1. El índice h se refiere a la capa oculta. La salida de la capa oculta esta dada por:

$$s_{pj} = f^h_j(neta^h_{pj}). \tag{2.7}$$

La entrada neta a la capa siguiente está dada por:

$$neta^o_{pk} = \sum_{j=0}^{L} w^o_{kj} s_{pj} \tag{2.8}$$

$$o_{pk} = f^o_k(neta^o_{pk}) \tag{2.9}$$

La Ecuación 2.8, indica la entrada que tiene la capa de salida y la Ecuación 2.9 indica la salida, donde o significa el nivel de la capa de salida.

El conjunto inicial de valores de pesos representa una primera aproximación de los pesos correctos para el problema. El procedimiento básico para entrenar a la red es la siguiente:

1 Se aplica un vector de entrada a la red, y se calculan los valores correspondientes de salida.

2 Se comparan las salidas obtenidas con las salidas deseadas, y se determina una medida de error.

3 Se determina en qué dirección (+ ó -) debe de cambiar cada peso con objeto de reducir el error.

4 Se determina la cantidad en que en que es preciso cambiar cada peso.

5 Se cambian los pesos.

6 Se repiten los pasos del 1 al 5 con todos los vectores de entrenamiento hasta que el error para todos los vectores del conjunto de entrenamiento quede reducido a un valor aceptable.

2.5 Actualización de pesos de la capa de salida

Para el cambio de los pesos se utilizará la regla Delta:

$$w_i(t+1) = w_i(t) + \mu\epsilon_k x_{ki}$$

en donde μ como se menciono anteriormente, es la tasa de aprendizaje. x_{ki} es la i-ésima componente del k-ésimo vector de entrenamiento y ϵ_k es el error entre la salida y el valor correcto, $\epsilon_k = (d_k - y_k)$.

Cuando tenemos más de dos capas ocultas o la salida no es lineal, utilizaremos las siguientes ecuaciones que resultan de derivar la regla delta. El error del k-ésimo vector de entrada es $\epsilon_k = (d_k - y_k)$, en donde d_k es la salida deseada y y_k es la salida real, como en una capa hay varias unidades y en una BPN no basta con un único valor de error ϵ_k, se utilizara una notación distinta a la que se uso en la regla Delta. Definamos el error de una unidad de salida de la forma δ_{pk} =$(y_{pk} - o_{pk})$, en donde el subíndice p se refiere al p-ésimo vector de entrenamiento, y k se refiere a la k-ésima unidad de salida, donde y_{Pk} . es el valor de salida deseado, y o_{Pk} es la salida obtenida a partir de la k-ésima unidad. El error que se minimiza es la suma de los cuadrados de los errores de todas las capas de

salida:

$$E_p = \frac{1}{2}\sum_{k=1}^{M} \delta_{pk}^2. \tag{2.10}$$

El factor $\frac{1}{2}$ de la ecuación 2.10, se pone por conveniencia para facilitar los cálculos mas adelante.

Para determinar el sentido en el que se debe de cambiar los pesos, calculamos el gradiente negativo de E_P ,∇E_p, con respecto a los pesos w_{kj}. Después se pueden ajustar los valores de los pesos de tal forma que se reduzca el error total. En la Figura 2.6 se muestra un esquema del gradiente ∇E_p en el punto $\mathbf{z}$, junto con el valor negativo del gradiente. Los cambios de los pesos se producen en la dirección del gradiente negativo, que es la dirección del descenso más pronunciado a lo largo de la superficie al punto $\mathbf{z}$. Además los cambios de los pesos deberían hacerse iterativamente hasta que E_P alcance el punto mínimo $z_{\min}$. Considérese por separado cada componente de ∇E_p. Partiendo de

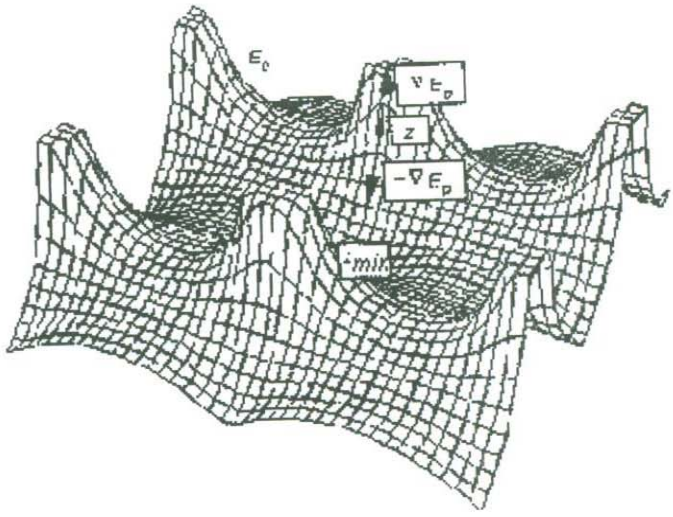

Figura 2.6: *Esquema del espacio de pesos.*

la Ecuación 2.10 y de la definición de δ_{pk}.

$$E_p = \frac{1}{2}\sum_k (y_{pk} - o_{pk})^2$$

utilizando la regla de la cadena para calcular las derivadas parciales de E_P y utilizando las Ecuaciones 2.8 y 2.9, para el valor de salida o_{Pk}, la expresión nos queda de la siguiente manera:

$$\frac{\partial E_p}{\partial w^o_{kj}} = -(y_{pk} - o_{pk})\frac{\partial f^o_k}{\partial (neta^o_{pk})}\frac{\partial (neta^o_{pk})}{\partial w^o_{kj}} \tag{2.11}$$

donde

$$\frac{\partial (neta^o_{pk})}{\partial w^o_{kj}} = \frac{\partial}{\partial w^o_{kj}}\left(\sum_{j=0}^{L} w^o_{kj} s_{pj}\right) = s_{pj} \tag{2.12}$$

y

$$\frac{\partial f^o_k}{\partial (neta^o_{pk})} = f'^o_k(neta^o_{pk}). \tag{2.13}$$

Sustituyendo las Ecuaciones 2.12 y 2.13 en la Ecuación 2.11 se obtiene la siguiente expresión:

$$\frac{\partial E_p}{\partial w^o_{kj}} = -(y_{pk} - o_{pk}) f'^o_k(neta^o_{pk}) s_{pj}.$$

Multiplicando por menos uno para obtener el gradiente negativo:

$$-\frac{\partial E_p}{\partial w^o_{kj}} = (y_{pk} - o_{pk}) f'^o_k(neta^o_{pk}) s_{pj}$$

Por lo tanto la magnitud del cambio de los pesos, consideramos que será proporcional al gradiente negativo:

$$\Delta w^o_{kj} = \mu(-\frac{\partial E_p}{\partial w^o_{kj}})$$

Así, los pesos se actualizan de la siguiente manera para la capa de salida:

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \Delta w^o_{kj}$$

en donde

$$\Delta w_{kj}o = \mu(y_{pk} - o_{pk}) f_k'^o(neta^o_{pk}) s_{pj}$$

Ahora examinemos la función $f_k'^o$.

El primer requisito que se pide es que f_k^o sea derivable. Este requisito elimina la posibilidad de utilizar una unidad umbral lineal tal como la describimos en él Capitulo I.

La salida que utilizaremos será la función *sigmoide*:

$$f_k^o(neta^o_{pk}) = (1 - e^{-neta^o pk})^{-1}$$

En la Figura 2.7. representaremos la función sigmoide.

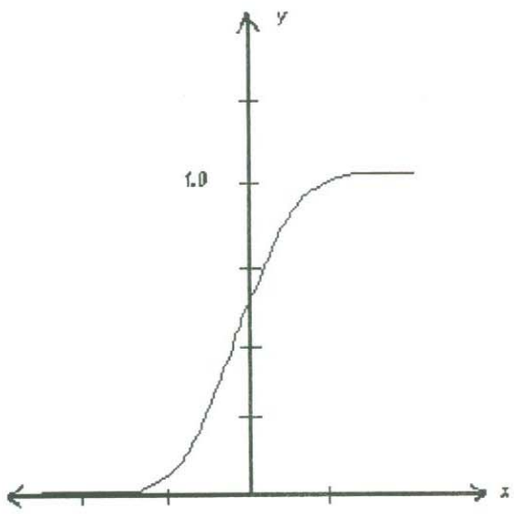

Figura 2.7: *Esquema de una función sigmoide.*

Derivando tenemos que:

$$f_k'^o(neta^o_{pk}) = f_k^o(1 - f_k^o) = o_{Pk}(1 - o_{Pk}).$$

Por lo tanto la actualización de los pesos se expresa de la siguiente manera:

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \mu(y_{ok} - o_{pk})o_{pk}(1 - o_{pk})s_{pj}$$

Resumiendo las ecuaciones de actualización de pesos, definimos el error de la siguiente manera:

$$\delta^o_{pk} = (y_{pk} - o_{pk})o_{pk}(1 - o_{pk})$$

donde $o_{Pk}(1 - o_{Pk}) = f'^o_k(neta^o_{pk})$. Por lo tanto:

$$\delta^o_{pk} = (y_{pk} - o_{pk})f'^o_k(neta^o_{pk}). \tag{2.14}$$

Sustituyendo $(y_{pk} - o_{pk}) = \delta_{pk}$, que es el error de una sola unidad de salida, la expresión es la siguiente:

$$\delta^o_{pk} = \delta_{pk}f'^o_k(neta^o_{pk})$$

Entonces podemos escribir la ecuación de actualización de pesos para la capa de salida en la forma:

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \mu\delta^o_{pk}s_{pj}$$

2.6 Actualización de pesos de la capa oculta

Podemos calcular el error de salida de la capa oculta, sin embargo no conocemos la salida correcta de estas unidades, por lo que no es posible obtener una medida del error de salida de la capa oculta. Trataremos de encontrar una relación del error E_P con los valores de salida de la capa oculta.

Analicemos la ecuación.

$$\begin{aligned} E_p &= \frac{1}{2}\sum_k (y_{pk} - o_{pk})^2 \\ &= \frac{1}{2}\sum_k (y_{pk} - f_k^o(neta_{pk}^o))^2 \\ &= \frac{1}{2}\sum_k (y_{pk} - f_k^o\left(\sum_{j=0}^{L} w_{kj}^o s_{pj}\right))^2 \end{aligned}$$

Aprovecharemos el hecho de que, de acuerdo a las Ecuaciones 2.6 y 2.7, s_{pj} depende de los pesos de la capa oculta y calcularemos el gradiente de E_p respecto a los pesos de la capa oculta.

$$\frac{\partial E_p}{\partial w_{ji}^h} = \frac{1}{2}\sum_k \frac{\partial}{\partial w_{ji}^h}(y_{pk} - o_{pk})^2$$

Derivando explícitamente E_P utilizando la regla de la cadena la ecuación queda de la forma:

$$\frac{\partial E_p}{\partial w_{ji}^h} = -\sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial (neta_{pk}^o)} \frac{\partial (neta_{pk}^o)}{\partial s_{pj}} \frac{\partial (s_{pj})}{\partial (neta_{pj}^h)} \frac{\partial (neta_{pj}^h)}{\partial w_{pj}^h} \tag{2.15}$$

Desarrollando cada uno de los factores de la Ecuación 2.15

$$\frac{\partial o_{pk}}{\partial (neta_{pk}^o)} = \frac{\partial f_k^o(neta_{pk}^o)}{\partial (neta_{pk}^o)} = f_k'^o(neta_{pk}^o)$$

$$\frac{\partial (neta_{pk}^o)}{\partial s_{pj}} = \frac{\partial\left(\sum_{j=0}^{L} w_{kj}^o s_{pj}\right)}{\partial s_{pj}} = w_{kj}^o$$

$$\frac{\partial s_{pj})}{\partial (neta_{pj}^h)} = \frac{\partial (f_j^h(neta_{pj}^h))}{\partial (neta_{pj}^h)} = f_j'^h(neta_{pj}^h)$$

$$\frac{\partial (neta_{pj}^h)}{\partial w_{pj}^h} = \frac{\partial\left(\sum_{i=0}^{L} w_{ji}^h x_{pi}\right)}{\partial w_{ji}^h} = x_{pi}.$$

Sustituyendo estas ecuaciones en la Ecuación 2.15 la expresión queda de la siguiente manera:

$$\frac{\partial E_p}{\partial w_{ji}^h} = -\sum_k (y_{pk} - o_{pk}) f_k'^o(neta_{pk}^o) w_{kj}^o f_j'^h(neta_{pj}^h) x_{pi}$$

Considerando la magnitud del cambio de peso proporcional al gradiente negativo tenemos que:

$$\Delta w_{ji}^h = \mu \sum_k (y_{pk} - o_{pk}) f_k'^o(neta_{pk}^o) w_{kj}^o f_j'^h(neta_{pj}^h) x_{pi},$$

donde μ de considera la *velocidad de aprendizaje.*

Utilizando la Ecuación 2.14:

$$\Delta w_{ji}^h = \mu \sum_k \delta_{pk}^o w_{kj}^o f_j'^h(neta_{pj}^h) x_{pi}$$

rescribimos

$$\Delta w_{ji}^h = \mu f_j'^h(neta_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o x_{pi} \tag{2.16}$$

Nótese que la actualización de los pesos de la capa oculta depende del error de la salida δ_{pk}^o. De aquí surge la noción de la *propagación hacia atrás.* Analizando la actualización de los pesos observamos que los errores de la capa de salida se propagan hacia atrás a la capa oculta. Por lo tanto definimos el *error de la capa oculta* como:

$$\delta_{pj}^h = f_j'^h(neta_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o \tag{2.17}$$

La ecuación de la actualización de los pesos de la capa oculta es la siguiente:

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \Delta w_{ji}^h(t). \tag{2.18}$$

Sustituyendo la Ecuación 2.16 en la Ecuación 2.18 queda de la forma:

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \mu f_j'^h(neta_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o x_{pi}.$$

Sustituyendo la Ecuación 2.17, obtenemos la ecuación de la actualización de los pesos:

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \mu\delta_{pj}^{h}x_{pi}.$$

Resumiendo todas las ecuaciones, representaremos un orden en que serían utilizadas durante el entrenamiento para un único par de vectores de entrenamiento para una BPN.

1 Se aplica el vector de entrada $\boldsymbol{x}_p = (x_{p1}, x_{p2}, ..., x_{pN})^t$ a las unidades de entrada.

2 Se calculan los valores netos procedentes de las entradas para las unidades de la capa oculta:

$$neta_{pj}^{h} = \sum_{i=0}^{N} w_{ji}^{h}x_{pi}$$

3 Se calculan las salidas de la capa oculta:

$$s_{pj} = f_{j}^{h}(neta_{pj}^{h})$$

4 Se pasa a la capa de salida. Se calculan los valores netos de las entradas para cada unidad:

$$neta_{pk}^{o} = \sum_{j=0}^{L} w_{kj}^{o}s_{pj}$$

5 Se calculan las salidas:

$$o_{pk} = f_{k}^{o}(neta_{pk}^{o})$$

6 Se calculan los términos de error para las unidades de salida:

$$\delta_{pk}^{o} = (y_{pk} - o_{pk})f_{k}^{\prime o}(neta_{pk}^{o})$$

7 Se calculan los términos de error para las unidades ocultas:

$$\delta_{pj}^{h} = f_{j}^{\prime h}(neta_{pj}^{h}) \sum \delta_{pk}^{o} w_{kj}^{o}$$

Obsérvese que los términos de error de las unidades ocultas se calculan antes que hayan sido actualizados los pesos de conexión con las unidades de la capa de salida.

8 Se actualizan los pesos de la capa de salida:

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \mu \delta_{pk}^{o} s_{pj}$$

9 Se actualizan los pesos de la capa oculta:

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \mu \delta_{pj}^{h} x_{ji}$$

El orden de actualización de pesos de una capa individual no es importante.

Cuando el error $E_p = \frac{1}{2} \sum_{k=1}^{M} \delta_{pk}^{2}$ resulta aceptablemente pequeño para todos los pares de vectores de entrenamiento, éste se puede dar por concluido.

A continuación mostraremos un ejemplo para ilustrar el funcionamiento de este algoritmo.

Sean los vectores de entrada x_1, x_2, x_3, con sus respectivas salidas y_1, y_2, así como lo muestra la tabla siguiente.

Entradas			*Salida*	
x_1	x_2	x_3	y_1	y_2
1	0	0	0	1
1	1	0	1	0
1	1	1	1	1

Tabla2.3 *Datos*

En la grafica siguiente se muestra la arquitectura de la red para este ejemplo:

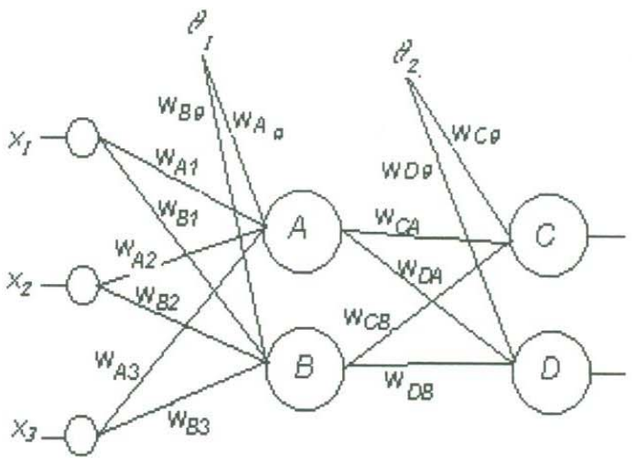

Figura 2.8: *Esquema de una red tipo BNP para el ejemplo de la tabla anterior.*

PASO 1: Se presenta el vector de entrada.

$$X_1 = (1,0,0)^t$$

PASO 2: Se calculan las entradas de cada unidad de la capa oculta.

$$neta_{pj}^h = \sum_{i=0}^{N} w_{ji}^h x_{pi};$$

con $\theta_1 = x_0 = 1$ y con un valor inicial de w =1.

$$neta_A = 1(1) + 1(1) + 1(0) + 1(0) = 2.$$

$$neta_B = 2.$$

PASO 3: Se calculan las respectivas salidas.

$$s_{pj} = f_j^h(neta_{pj}^h) = \frac{1}{1+e^{-netaj}}$$

$$s_A = 0.88.$$

$$s_B = 0.88.$$

PASO 4: Se calcula la entrada a cada unidad de la capa de salida.

$$neta_{pk}^o = \sum_{j=0}^{L} w_{kj}^o s_{pj};$$

donde $\theta_2 = s_0 = 1$.

$$neta_C = 1(1) + 1(0.88) + 1(0.88) = 2.76.$$

$$neta_D = 2.76.$$

PASO 5: Se calculan sus respectivas salidas.

$$o_C = 0.94$$

$$o_D = 0.94$$

PASO 6: Se calcula el error de la capa de salida.

$$\delta_{pk}^o = (y_{pk} - o_{pk}) f_k'^o(neta_{pk}^o);$$

donde

$$f_k'^o(neta_{pk}^o) = o_{Pk}(1 - o_{Pk}).$$

$$\delta_C = (0 - 0.94)(0.94)(1 - 0.94) = -0.05.$$

$$\delta_D = (1 - 0.94)(0.94)(1 - 0.94) = 0.003.$$

PASO 7: Se calcula el error de la capa oculta.

$$\delta_{pj}^h = f_j'^h(neta_{pj}^h) \sum \delta_{pk}^o w_{kj}^o;$$

donde

$$f_j'^h(neta_{pj}^h) = s_{Pj}(1 - s_{Pj}).$$

$$f_A' = 0.105, f_B' = 0.105$$

$$\delta_A = (0.105)[(-0.05)(1) + (0.003)(1)] = -0.004$$

$$\delta_B = -0.004$$

PASO 8: Se modifican los pesos de la capa de salida.

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \mu\delta_{pk}^o s_{pj}$$

donde $\mu = 0.1$

$$w_{C\theta}(t+1) = 0.995 \quad w_{CA}(t+1) = 1.0002 \quad w_{CB}(t+1) = 0.996$$

$$w_{D\theta}(t+1) = 1.003 \quad w_{DA}(t+1) = 0.996 \quad w_{DB}(t+1) = 1.0002$$

PASO 9: Se calculan los pesos de la capa oculta.

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \mu\delta_{pj}^h x_{ji}$$

$$w_{A\theta}(t+1) = 0.999 \quad w_{A1}(t+1) = 0.999 \quad w_{A2}(t+1) = 1 \quad w_{A3}(t+1) = 1$$

$$w_{B\theta}(t+1) = 0.999 \quad w_{B2}(t+1) = 0.999 \quad w_{B2}(t+1) = 1 \quad w_{B3}(t+1) = 1.$$

Se reinicia en el paso 1, utilizando los pesos modificados como pesos actuales.

Se le presenta el vector de entrada:

$$X_2 = (1, 1, 0)^t$$

Se calculan las entradas de cada unidad de la capa oculta.

$$neta_A = 0.999(1) + 0.999(1) + 1(1) + 1(0) = 2.99.$$

$$neta_B = 2.99.$$

Se calculan las respectivas salidas.

$$s_A = 0.95.$$

$$s_B = 0.95.$$

Se calcula la entrada a cada unidad de la capa de salida.

$$neta_C = 0.995(1) + 0.996(0.95) + 0.996(0.95) = 2.88.$$

$$neta_D = 2.9.$$

Se calculan sus respectivas salidas.

$$o_C = 0.946.$$

$$o_D = 0.947.$$

Se calcula el error de la capa de salida.

$$\delta_C = (1 - 0.946)(0.946)(1 - 0.946) = -0.0027.$$

$$\delta_D = (0 - 0.947)(0.947)(1 - 0.947) = -0.047.$$

Se calcula el error de la capa oculta.

$$\delta_A = 0.088$$

$$\delta_B = 0.088$$

Se calculan los pesos de la capa de salida.

$$w_{C\theta}(t+1) = 0.995 \quad w_{CA}(t+1) = 0.996 \quad w_{DA}(t+1) = 0.995$$

$$w_{D\theta}(t+1) = 0.998 \quad w_{CB}(t+1) = 0.996 \quad w_{DB}(t+1) = 0.995$$

Se calculan los pesos de la capa oculta.

$$w_{A\theta}(t+1) = 1.087 \quad w_{A1}(t+1) = 1.087 \quad w_{A2}(t+1) = 1 \quad w_{A3}(t+1) = 1.088.$$

$$w_{B\theta}(t+1) = 1.087 \quad w_{B1}(t+1) = 1.088 \quad w_{B2}(t+1) = 1.087 \quad w_{B3}(t+1) = 1.$$

Se empieza de nuevo en el paso 1 y se toman estos nuevos pesos como pesos iniciales.

Se le presenta el vector de entrada.

$$X_3 = (1,1,1)^t$$

Se repiten los pasos del 2 al 9 y así sucesivamente hasta que $E_p = \frac{1}{2}\sum_{k=1}^{M} \delta_{pk}^2$ sea lo suficientemente pequeño.

Capítulo 3

La Red De Hopfield

3.1 Red Neuronal Recurrente Discreta(RNRD)

Una Red Neuronal Recurrente Discreta (RNRD) es aquella en la que existen conexiones que enlazan las salidas con las entradas, lo que le proporciona gran generalidad en su operación pues la salida de la red en un instante determinado, dependerá no solo de las entradas actuales y de los pesos de conexión, sino también de las salidas ofrecidas por la red en instantes anteriores. Esta característica hace que las redes recurrentes muestren propiedades similares a la memoria humana, en la cual el estado de las salidas de las neuronas del cerebro depende, en parte, de entradas (estímulos) anteriores. Dicho de otra forma, es un sistema capaz de reconstruir información exacta y completa a partir de unos datos de entrada incompletos, ruidosos o degradados. Las redes recurrentes se usan principalmente en el desarrollo de las denominadas memorias asociativas. Las memorias asociativas son sistemas que permite imitar las funciones de la memoria humana en los siguientes sentidos:

a) Se reproducen conceptos previamente elaborados o se recuerdan impresiones pasadas;

b) Se evocan ideas desde conceptos parecidos o situaciones semejantes;

c) Por su modo de operación, esta memoria se puede describir como un sistema asociativo de procesamiento de información.

Desde otro punto de vista, una RNRD es un sistema que es capaz de almacenar en su estructura interna, es decir, en las conexiones entre las unidades del sistema, un número determinado de vectores prototipo. El estado inicial de las citadas unidades representa un vector de datos, o vector de estado inicial x(0), correspondiente a alguno de los posibles patrones μ del espacio de las clases. A continuación la red evoluciona partiendo de este estado inicial. La evolución se realiza en las neuronas; es decir, cada neurona cambia su estado de acuerdo con un cierto esquema preestablecido en una ecuación denominada ecuación dinámica. Esta ecuación es una función del nivel de activación que llega a cada neurona en el instante de tiempo actual. El sistema se estabiliza cuando se alcanza un instante de tiempo en el cual ninguna neurona cambia de estado, al aplicarle la ecuación dinámica.

3.2 Red de Hopfield

La Red de Hopfield es una implementación de memorias asociativas. Una red de Hopfield es una red recurrente, formada por un conjunto de neuronas dispuestas espacialmente, totalmente conectadas entre sí y operando concurrentemente (ver figura 3.1). En su estructura no cabe el concepto de capa ni otra caracterización de los nodos en categorías. No posee neuronas ocultas, por lo cual toda la información aprendida por la red, está directamente disponible; es una red no supervisada basada en la ley de Hebb y puede

decirse que carece de aprendizaje, entendiendo como tal, la obtención adaptativa en el tiempo de los pesos de conexión, dado que la red fija los pesos en la primera etapa de operación y los mantiene invariables hasta la convergencia de la misma.

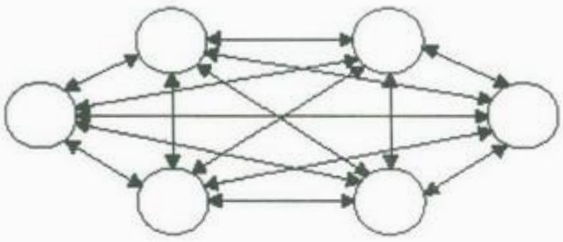

Figura 3.1: *Esquema de una Red de Hopfield.*

3.3 Ley de Hebb

Tal como hemos comentado en el apartado anterior, la solución para diseñar una RNRD que trabaje como una memoria asociativa, consiste en encontrar la matriz de pesos W y el vector de umbrales θ. Buscando la solución a este problema, algunos de los investigadores pensaron que podría encontrarse la misma, considerando a la red como un modelo del cerebro. Fue entonces cuando se estudiaron los trabajos de Donald O. Hebb(ver capitulo I), con objeto de poder aplicarlos en busca de la citada solución. Hebb mantenía que las memorias asociativas biológicas yacían en las conexiones sinápticas existentes entre las células nerviosas; es decir, las neuronas biológicas deben sus peculiares cualidades de procesamiento de la información de forma adaptativa, a su capacidad de auto organización. Hebb pensó que la capacidad de aprendizaje radicaba en la habilidad de las neuronas para modificar sus conexiones, en el establecimiento de sinápsis nuevas etc.

3.4 El caso de un solo patrón

¿Cómo se podría aplicar este resultado al problema que estamos considerando? Empezamos con el caso más sencillo, en el que se considera que el vector de umbrales θ es el vector nulo y que únicamente se quiere integrar un patrón $\xi \in \{-1, 1\}$ en la red. Considerando la ley de Hebb se puede pensar que si inciden señales del mismo signo sobre las neuronas x_i y x_j entonces el peso sináptico entre ellas debe reforzarse; es decir, el valor w_{ij} se debe incrementar. Por el contrario si inciden señales de diferente signo sobre las neuronas x_i y x_j entonces el peso sináptico entre dichas neuronas se ha de debilitar; es decir el valor de w_{ij} debe disminuir. Es importante remarcar que como las coordenadas de un patrón se toman como valores pertenecientes al conjunto $\{-1, 1\}$, puede pensarse que las señales son precisamente los signos de los elementos anteriores. Dicho de otra forma, dos señales serán iguales si son del mismo signo y serán diferentes si sus signos lo son o también, dos señales serán iguales si su producto es +1 y serán diferentes si su producto es -1. Demostraremos que con estos valores para los parámetros, $W = w_{ij}$, la red actúa como una memoria asociativa.

Aplicando la ley de Hebb y suponiendo que inicialmente asignamos un valor nulo a los pesos w_{ij}, una vez que el patrón ξ ha dejado huella sobre estos, entonces el valor de los pesos w_{ij} será $w_{ij} = \xi_i \xi_j$, ya que si ξ_i y ξ_j son del mismo signo, es decir si las señales ξ_i y ξ_j son iguales, el producto $\xi_i \xi_i$ será igual +1 y la sinápsis que une la neurona x_i con la x_j se verá reforzada. Si las señales ξ_i y ξ_j son diferentes, $\xi_i \xi_j$ será igual a -1 y la sinápsis que une la neurona x_i con la x_j se verá debilitada.

Para visualizar la matriz de pesos, podemos contemplar ésta como si se tratara de la matriz de adyacencia de un grafo completo, (ver figura 3.2). El peso w_{ij} se puede interpretar como el valor asignado a la arista w_{ij} del citado grafo.

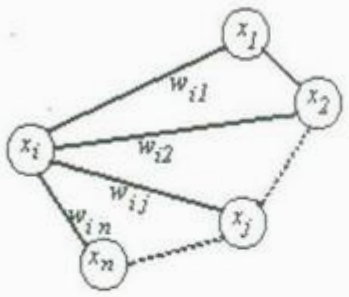

Figura 3.2: *Ley de Heebb.*

Para justificar que la matriz de pesos así obtenida responde efectivamente a los requerimientos que vamos persiguiendo, deberemos de ver que efectivamente la red así construida trabaja como una memoria asociativa.

Para ello primeramente vamos a ver que el vector ξ es un elemento estable del sistema; es decir, vamos a ver que si $x(0) = \xi$ entonces $x(0) = x(1)$. Pero esto es fácil de comprobar ya que si para uno cualquiera de los $i \in \{1, 2, ..., n\}$ se verifica por ejemplo que $x_i(0) = 1$ esto significa que $\xi_i = 1$ y por lo tanto según la ley de Hebb se verificará que

$$x_i(1) = Sgn[\sum_{j=1}^{n} w_{ij}x_j(0)] = Sgn[\sum_{j=1}^{n} \xi_i\xi_j\xi_j] \tag{3.1}$$

ya que $w_{ij} = \xi_i\xi_j$ y $x_j(0) = \xi_j$ Por lo tanto

$$x_i(1) = Sgn[\sum_{j=1}^{n} \xi_i] = Sgn[\sum_{j=1}^{n} x_i(1)] = Sgn[\sum_{j=1}^{n} 1] = Sgn[n] = 1 \tag{3.2}$$

de la misma manera si $x_i(0) = -1$, es decir si $\xi_i = -1$ se hubiese verificado que

$$x_i(1) = Sgn[\sum_{j=1}^{n} \xi_i] = Sgn[\sum_{j=1}^{n} x_i(0)] = Sgn[\sum_{j=1}^{n} -1] = Sgn[-n] = -1 \tag{3.3}$$

Esto se verifica para cada i del conjunto, por tanto el patrón ξ es un elemento estable del sistema.

Veamos ahora que también es estable el patrón $\overline{\xi}$ simétrico de ξ ; es decir, el patrón tal que $\xi_i = 1$ si y solo si $\overline{\xi_i}$ = -1 y ξ_i = -1 si y solo si $\overline{\xi_i} = 1\ \forall i \in \{1, 2, ..., n\}$. si $x_i(0) =$ 1 esto significa que $\overline{\xi_i} = 1$ y

$$x_i(1) = Sgn[\sum_{j=1}^{n} w_{ij}x_j(0)] = Sgn[\sum_{j=1}^{n} \xi_i\xi_j\overline{\xi_j}] \tag{3.4}$$

ya que $w_{ij} = \xi_i\xi_j$ y $x_j(0) = \overline{\xi_j}$. Por lo tanto, como $\xi_j\overline{\xi_j}$ = -1 para todo j, entonces

$$x_i(1) = Sgn[\sum_{j=1}^{n} -\xi_i] = Sgn[\sum_{j=1}^{n} \overline{\xi_i}] = Sgn[\sum_{j=1}^{n} x_i(1)] = Sgn[\sum_{j=1}^{n} 1] = Sgn[n] = -1 \tag{3.5}$$

De forma similar puede probarse para el caso $x_i(0)$ = -1.

Hemos calculado los parámetros de la red, matriz de pesos, que permiten memorizar un solo patrón, siguiendo un procedimiento basado en la ley de Hebb. Ahora vamos a comprobar que el sistema se comporta como una memoria asociativa, en el sentido de que cualquier otro patrón ξ^r distinto del ξ es atraído o bien por ξ o bien por $\overline{\xi}$ dependiendo de a cual de los dos está más próximo. Supongamos que el número de características de ξ^r que son coincidentes (iguales y en el mismo lugar) con las de ξ es mayor que n/2, vamos a ver entonces que si $x(0) = \xi^r$, la red se estabiliza cuando $x(t)$ es igual a ξ. Para cualquier $i \in \{1, 2, ..., n\}$ se verifica, según la ley de Hebb que

$$x_i(1) = Sgn[\sum_{j=1}^{n} w_{ij}x_j(0)] = Sgn[\sum_{j=1}^{n} \xi_i\xi_j\xi_j^r] = Sgn[\xi_i \sum_{j=1}^{n} \xi_j\xi_j^r] \tag{3.6}$$

ya que $w_{ij} = \xi_i\xi_j$ Además si el número de componentes de ξ_i^r que son coincidentes con las de ξ es mayor que n/2 se verificará que:

$$\sum_{j=1}^{n} \xi_j \xi_j^r > 1 \tag{3.7}$$

y por lo tanto

$$Sgn[\xi_i \sum_{j=1}^{n} \xi_j \xi_j^r] = \xi_i \tag{3.8}$$

y por lo tanto $x(1) = \xi$.

Es evidente que si ξ^r se hubiese parecido más a $\overline{\xi}$ que a ξ, es decir si el número de componentes coincidentes hubiese sido menor que n/2 entonces $x(1)$ hubiese sido igual al de $\overline{\xi}$ Por lo tanto hemos visto que la ley de Hebb es un buen procedimiento para calcular la matriz de pesos W, cuando tenemos una memoria asociativa de un solo patrón.

3.5 Síntesis de las redes de Hopfield para N patrones.

El problema de sintetizar mediante una red de Hopfield una memoria asociativa que almacene una serie de vectores predeterminados es complejo.

Es necesario generar la matriz de pesos (W) de la red de forma que los puntos fijos de la red se correspondan con las clases. Para que la red sea eficaz el radio de atracción de cada clase debe ser el mayor posible. En términos generales el número de clases que se pueden almacenar eficazmente es una fracción reducida de la dimensión de la red.

La forma más simple para sintetizar una red de Hopfield que tenga como puntos fijos los p vectores $x_1, ..., x_p$, es encontrar W tal que:

$$Wx_i = x_i, i = 1, ..., p$$

a) Las p clases x_i, son vectores propios de la matriz W con valor propio correspondiente igual a la unidad. Por tanto hay que construir una matriz que tenga esta propiedad. Si $X = [x_1, ..., x_p]$, se tiene como expresión para W:

$$W = X(X^tX)^{-1}X^t$$

Sin embargo, para que un prototipo x sea punto fijo de la red no es necesario exigir la identidad $Wx = x$. Basta con exigir $sgn(Wx) = x$, que se consigue imponiendo

$$Wx = \lambda \cdot x, \lambda > 0$$

b) Basta que cada prototipo sea un vector propio de la matriz W con valor propio asociado positivo.

c) De esta forma se emplea como matriz de pesos:

$$W = \frac{1}{p}XX^t$$

d) Para ajustarse a la estructura de la red de Hopfield en la que cada neurona no se autoexcita, los elementos de la diagonal principal de la matriz de pesos han de ser nulos.

$$W = \frac{1}{p}(XX^t - pI)$$

Este procedimiento tiene algunos inconvenientes, entre los que se pueden mencionar los siguientes:

a) Aparecen puntos de equilibrio diferentes a las clases especificados.

b) Si x es un punto fijo de la red también lo es -x.

c) Estos estados espúreos se pueden corregir mediante diversas modificaciones de la matriz de correlación.

d) La *capacidad* de la Red de Hopfield es el máximo número de clases que se pueden almaçenar con seguridad en la red. El problema de determinar la capacidad de una red es muy complicado y aún no resuelto de forma satisfactoria. La mayor parte de los estudios realizados ofrecen estimaciones asintóticas del máximo valor tolerable de clases cuando la dimensión de la red es muy grande. Si se tolera un cierto margen de error en la recuperación de clases, puede admitirse un número de estos proporcional a la dimensión de la red, pudiendo llegar a ser del orden del 13% del número de neuronas.

3.6 Algoritmo de Hoppfield.

PASO I. se empieza con el cálculo de los valores de los pesos que conectan los nodos utilizando la siguiente formula.

$$w_{ij} = \begin{cases} \sum_{s=1}^{M} x_i^s x_j^s, & i \neq j \\ 0, & i = j \end{cases}$$

donde w_{ij} es el peso que conecta a las neurona j con la neurona i; es el valor del i-ésimo elemento de la s-ésima clase. M es el número de clases que se desea aprender.

En forma matricial tenemos:

$$w_{ij} = \frac{1}{M} \sum_{i=1}^{M} x_i x_i^t - MI$$

donde x_i es el i-ésimo patrón a almacenar (ver figura 3.3).

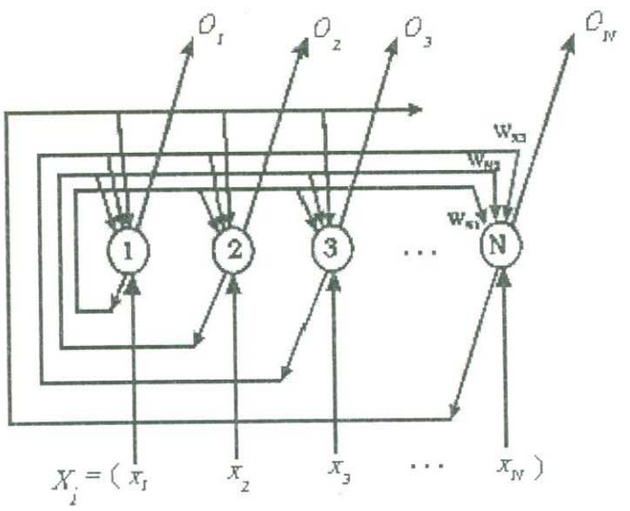

Figura 3.3: *Esquema de un patrón a almacenar, presentado a la red.*

PASO II. Sé inicializa la red con el patrón de entrada desconocido (x); es decir, $O_i(0) = x_i$ donde $x_i = \pm 1$, para $i = 1, 2, \ldots, N$ donde: $O_i(0)$ representa los elementos del vector de salida de la red en el tiempo $t = 0$.

PASO III. Iterar hasta converger de acuerdo a la fórmula:

$$O_j(t+1) = f_h[\sum_{i=1}^{N} w_i^j O_i(t)], j = 1, 2, ..., N \tag{3.9}$$

donde :

$$f_h(x) = \begin{cases} +1, & \text{si } x > 0 \\ -1, & \text{si } x < 0 \\ O_j(t), & \text{si } x = 0 \quad \text{esto es, no hay cambio en el valor de } O_j(t). \end{cases}$$

Se repite hasta que la salida para los nodos permanezcan sin cambios. Esta salida representa al patrón que más se parece al patrón de entrada dado.

Si los patrones almacenados no son suficientemente diferentes entre sí, puede ocurrir que ante una entrada la red no haga una asociación correcta y genere una salida errónea.

En la figura 3.4 se muestra como representar un patrón (una versión de la letra "A" en este caso)en términos vectoriales. Nótese que el estado de cada neurona se determina básicamente por el signo. A una neurona apagada (es decir, inactiva) se le asigna el estado -1. A una neurona encendida (es decir, activa) se le asigna el estado +1

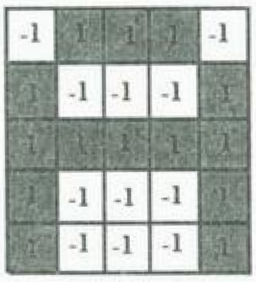

Figura 3.4: *Esquema de un patrón* $X = (-1, 1, 1, 1, -1, 1, -1, -1, -1, 1, ..., -1, -1, 1)^t$ *que representa a la letra A.*

3.7 Ejemplo en maple

La representación de las clases almacenadas se pueden ver en el apéndice **D**.

restart:

with(linalg):

Esta es una red de 120 neuronas en la que hay almacenadas 10 clases (10 dígitos).

Paso I del Algoritmo.

N:=120: M:=10:

x[0]:=array([-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,1,1,1,1,1,1,-1,-1,-1,1,1,1,-
1,-1,1,1,1,-1,-1,1,1,1,-1,-1,1,1,1,-1,-1,1,1,1,-1,-1,1,1,1,-1,-1,1,1,1,-1,-1,1,1,1,-1,-1,1,1,1,-1,-1,1,1,1,-

1,-1,1,1,1,-1,-1,1,1,1,-1,-1,-1,1,1,1,1,1,1,-1,-1,1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-
1]):

x[1]:=array([-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-
1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-
1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-
1,-1,1,1,1,1,-1,-1,-1]):

x[2]:=array([1,1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-
1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,1,1,-1,-1,-1,-
1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,1,1,-
1,-1]):

x[3]:=array([-1,-1,1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-
1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-
1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-
1,-1,1,1,1,1,1,1,-1,-1]):

x[4]:=array([-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-
1,-1,-1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-
1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-
1,-1,-1,1,1,-1,-1]):

x[5]:=array([-1,-1,-1,1,1,1,1,1,1,1,-1,-1,-1,1,1,1,1,1,1,1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-
1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-
1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,1,1,1,1,1,1,1,1

x[6]:=array([1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-

1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,-1,-1]):

x[7]:=array([1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1,1]):

x[8]:=array([-1,-1,1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,-1,-1,1,1,-1,-1,1,1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]):

x[9]:=array([-1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,1,1,1]):

xt[0]:= transpose(x[0]):

xt[1]:= transpose(x[1]):

xt[2]:= transpose(x[2]):

xt[3]:= transpose(x[3]):

xt[4]:= transpose(x[4]):

xt[5]:= transpose(x[5]):

```
xt[6]:= transpose(x[6]):

xt[7]:= transpose(x[7]):

xt[8]:= transpose(x[8]):

xt[9]:= transpose(x[9]):
```

La matriz de pesos.

```
m[0]:=evalm(x[0] &* xt[0]):

m[1]:=evalm(x[1] &* xt[1]):

m[2]:=evalm(x[2] &* xt[2]):

m[3]:=evalm(x[3] &* xt[3]):

m[4]:=evalm(x[4] &* xt[4]):

m[5]:=evalm(x[5] &* xt[5]):

m[6]:=evalm(x[6] &* xt[6]):

m[7]:=evalm(x[7] &* xt[7]):

m[8]:=evalm(x[8] &* xt[8]):

m[9]:=evalm(x[9] &* xt[9]):

w[1]:=evalm(m[0] &+ m[1]):
```

w[2]:=evalm(m[2] &+ w[1]):

w[3]:=evalm(m[3] &+ w[2]):

w[4]:=evalm(m[4] &+ w[3]):

w[5]:=evalm(m[5] &+ w[4]):

w[6]:=evalm(m[6] &+ w[5]):

w[7]:=evalm(m[7] &+ w[6]):

w[8]:=evalm(m[8] &+ w[7]):

w[9]:=evalm(m[9] &+ w[8]):

Calculo de la Matriz de pesos.

z:=Matrix(120,120,shape=identity):

Id:=evalm(-10 * z):

W:=evalm(w[9] + Id):

Paso II del algoritmo.

mu[0]:=array([-1,-1,1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,-1,1,1,1,1,1,1,1,-1,-1,-1,1,1,1,1,1,1,-1,-1]):

```
c:=matrix(12,10,mu[0]);

    Paso III del algoritmo.

    for j from 1 to 15 do

    mu[j]:=evalm(W &* mu[j-1]);

    for i from 1 to N do

    if ( mu[j][i]¿0) then

    mu[j][i]:=1 else if ( mu[j][i]¡0) then

    mu[j][i]:=-1 else

    mu[j][i]:=mu[j-1][i] fi: fi: od:od:

    for i from 1 to 15 do

    f[i]:= evalf(norm(mu[i]-mu[i-1],2))od:

    for i from 0 to 9 do q[i]:=evalf(norm(mu[15]-x[i],2)) od:

    m:=q[0]:

    for i from 0 to 9 do if (q[i]¡m) then m:=q[i] else end if od;

    m:

    for i from 0 to 9 do if abs(q[i]-m)=0 then j:=i else end if od;
```

```
j:

print ("El patrón con ruido se parece al patron",j):
```

Conclusiones

El desarrollo de las redes neuronales artificiales en los últimos años ha sido notable, ya que se ha observado el potencial que tienen en la solución de problemas en diferentes campos de la ciencia, la industria y la tecnología, aunado esto con el crecimiento de la tecnología computacional. Sin embargo aún falta mucho por descubrir y de hecho el tema de las redes neuronales artificiales es un campo de intensa investigación en la actualidad.

En el capítulo 1 se presentó una breve descripción del funcionamiento fisiológico de las neuronas en el cerebro humano para tratar de identificar aquellos aspectos que se pueden emular matemáticamente con el fin de proponer un modelo de red neuronal. También se hicieron algunos comentarios históricos sobre el desarrollo de las redes neuronales. Después se analizó el perceptrón, que es un dispositivo clasificador que modela el funcionamiento de una neurona individual, asícomo de sus principales deficiencias.

En el capítulo 2 se hizo un análisis de las redes multicapas, limitándonos al estudio del algoritmo del back propagation, que se aplica a esta clase de redes neuronales. El problema central consiste en modelar la actividad de las conexiones sinápticas entre las neuronas. Este problema se traduce en encontrar el vector de pesos para el cual el error entre la señal percibida por la red y la señal deseada sea mínimo. Primero presentamos dos maneras de resolver una red monocapa. Una consiste en usar el método de mínimos

cuadrados. Otra consiste en usar el método del gradiente descendiente. El método de mínimos cuadrados proporciona una solución cerrada al problema, mientras que la regla delta genera una sucesión de vectores que convergen al vector de pesos óptimo. Sin embargo, el primer método presenta el inconveniente de requerir la invertivilidad de una matriz, condición que no siempre se satisface. Se desarrollo el algoritmo de back propagation para resolver una red multicapa, utilizando las propiedades la regla delta.

En el capítulo 3 se estudió un tipo de redes neuronales llamadas redes de Hopfield. Estas redes se usan principalmente para modelar el problema del reconocimiento de patrones. Estas redes se conocen como memorias asociativas, que son un tipo de redes recurrentes. El principio básico de su funcionamiento es la ley de Hebb, que se usa inicialmente para determinar al matriz de pesos, que es la base para la formulación del algoritmo. El funcionamiento de una red de Hopfield se puede resumir como sigue. A partir de un conjunto de datos iniciales (clases), se construye, a partir de la ley de Hebb, la matriz de pesos, con la que se diseña el algoritmo mediante el cual, un patrón inicial es iterado hasta que la red lo asocia con una de las clases almacenadas. Terminamos este capítulo presentando la codificación del lenguaje de programación simbólica MAPLE 8 el algoritmo de Hopfield para una red de 120 neuronas en la que se han almacenado diez clases (correspondientes a los dígitos). Se observó que el código produce respuestas adecuadas, en el sentido de que si se le presentaba un carácter correspondiente a una de las clases almacenadas, el algoritmo la reconoce como tal en la primera iteración. Si el carácter presentado es una de las clases almacenadas pero con ruido (es decir, ligeramente distorsionada), después de unas cuantas iteraciones, el algoritmo lo identifica correctamente.

Si bien la tecnología ha tenido un desarrollado impresionante en los últimos años, aún no es posible contar con una computadora con la habilidad de aprender y tomar

decisiones, por lo que las redes neuronales artificiales es una alternativa para emular mediante algoritmos estos cuestionamientos.

Este trabajo como su nombre lo indica es una introducción a las redes neuronales artificiales, por lo que esperamos sea una lectura comprensible para quien inicia en este tema.

Una manera de continuar profundizando en este trabajo sería realizando un estudio detallado de otras redes neuronales especializadas en otras funciones, como son las redes de Cohen, la maquina de Boltzman, etc. Este estudio puede realizarce en dos niveles: uno computacional, en el que se haría énfasis en la implementación en los modelos correspondientes y otro matemático, en el que la preocupación sería la justificación formal de los modelos y algoritmos de las redes.

Existen muchas estructuras y algoritmos que no se mencionan en este trabajo, pero también existen resultados que se obtuvieron mediante simulación sin tener un desarrollo formal.

Apéndice A

Referencias Históricas

1.- Luigi Galvani (1737-1798) El trabajo de Galvani en anatomía comparada y fisiología incluye un estudio de los riñones de las aves y su sentido del oído. Es especialmente famoso por sus experimentos concernientes a "las fuerzas eléctricas en los movimientos musculares", que lo llevaron a su teoría de la electricidad animal. Esto comenzó en 1780 con la observación accidental de la contracción de las piernas de una rana disecada, cuando el nervio crural descubierto era tocado con un escalpelo de acero, mientras pasaban unas chispas de una máquina eléctrica que estaba cerca. Él trabajó diligentemente sobre este tema, pero esperó once años para publicar los resultados de su ingeniosa y simple teoría. Esta teoría de un fluido eléctrico nervioso, segregado por el cerebro, conducido por los nervios y almacenado en los músculos, ha sido abandonada por los científicos en vista de los nuevos descubrimientos, pero Galvani llegó a ella de una manera muy lógica y la defendió con ingeniosos experimentos que no tardaron en dar frutos. Así, descubrió que cuando un nervio y músculo tocan dos metales diferentes en contacto entre ellos, el músculo se contrae; esto llevó a sus discusiones con Volta y al descubrimiento de la pila Voltaica. El nombre Galvanismo se le da a las manifestaciones de la corriente eléctrica.

2.- Santiago Ramón y Cajal (1852-1934), histólogo y premio Nobel español conocido por su trabajo pionero sobre la estructura fina, llamada glía, del sistema nervioso; demostró la discontinuidad celular de las neuronas y anticipó el mecanismo de propagación del impulso nervioso. En 1889 descubrió los mecanismos que gobiernan la morfología y los procesos conectivos de las células nerviosas de la materia gris del sistema nervioso cerebroespinal. Durante los siguientes dos años desentrañó los cambios básicos que experimenta la neurona durante el funcionamiento del sistema nervioso. Fue también el primero en aislar las células nerviosas, llamadas células de Cajal, que se encuentran cerca de la superficie del cerebro. En 1892 se instaló en Madrid y fue nombrado catedrático de histología de la universidad de Madrid, donde trabajó y prolongó su labor científica hasta su muerte.

3.- Hans Berger (1873-1941). El psquiatra alemán Hans Berger fue el primero en demostrar, con la ayuda de un aparato amplificador (electroencefalógrafo), que existía un potencial eléctrico (oscilaciones de tensión) en el cerebro humano.

4.- Pavlov, Ivan Petrovich (1849-1936). Fisiólogo ruso discípulo de Ivan Sechenov y ganador del Premio Novel en 1904 por sus investigaciones sobre el funcionamiento de las glándulas digestivas. Trabajó de forma experimental y controlada con perros, a los que incomunicaba del exterior en el laboratorio que se pasó a llamar "las torres del silencio". Sus estudios lo llevaron a interesarse por lo que denominó secreciones psíquicas, o sea, las producidas por las glándulas salivales sin la estimulación directa del alimento en la boca. Pavlov notó que cuando en la situación experimental un perro escuchaba las pisadas de la persona que habitualmente venía a alimentarlo, salivaba antes de que se le ofreciera efectivamente la comida; no obstante, si las pisadas eran de un desconocido, el perro no salivaba. Estas observaciones le inspiraron para llevar a cabo numerosos estudios que fueron la base del Condicionamiento

Clásico. Nunca se consideró un psicólogo, y hasta el fin de sus días sostuvo que era un fisiólogo.

5.- 1936 - Alan Turing. Fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación. Sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch, un neurofisiólogo, y Walter Pitts, un matemático, quienes, en 1943, lanzaron una teoría acerca de la forma de trabajar de las neuronas (Un Cálculo Lógico de la Inminente Idea de la Actividad Nerviosa - Boletín de Matemática Biofísica 5: 115-133). Ellos modelaron una red neuronal simple mediante circuitos eléctricos.

6.- 1943 Walter Pitts (que en una de sus escapadas quinceañeras conoció por casualidad a Bertran Russell) y Warren McCulloch intentaron explicar en 1943 el funcionamiento del cerebro humano, por medio de una red de células conectadas entre sípodían ejecutar operaciones lógicas. Partiendo del menor suceso psíquico (estimado por ellos): el impulso todo/nada, generado por una célula nerviosa. El bucle "sentidos - cerebro - músculos", mediante la retroalimentacion producirían una reacción positiva si los músculos reducen la diferencia entre una condición percibida por los sentidos y un estado físico impuesto por el cerebro. También definieron la memoria como un conjunto de ondas que reverberan en un circuito cerrado de neuronas. Actualmente sabemos que las decisiones conscientes acerca de la verdad de las proposiciones lógicas se producen a un nivel más alto, y quizás participen en ellas millones de células cerebrales.

7.- 1949 - Donald Hebb. Escribió un importante libro: La organización del comportamiento, en el que se establece una conexión entre psicología y fisiología. Fue el primero en explicar los procesos del aprendizaje (que es el elemento básico de la inteligencia humana) desde un punto de vista psicológico, desarrollando una regla

de como el aprendizaje ocurría. Aun hoy, este es el fundamento de la mayoría de las funciones de aprendizaje que pueden hallarse en una red neuronal. Su idea fue que el aprendizaje ocurría cuando ciertos cambios en una neurona eran activados. También intentó encontrar semejanzas entre el aprendizaje y la actividad nerviosa. Los trabajos de Hebb formaron las bases de la Teoría de las Redes Neuronales.

8.- 1950 - Karl Lashley. En sus series de ensayos, encontró que la información no era almacenada en forma centralizada en el cerebro sino que era distribuida encima de él.

9.- 1956 - Congreso de Dartmouth. Este Congreso frecuentemente se menciona para indicar el nacimiento de la inteligencia artificial.

10.- 1957 - Frank Rosenblatt. Comenzó el desarrollo del Perceptrón. Esta es la red neuronal más antigua; utilizándose hoy en día para aplicación como reconocedor de patrones. Este modelo era capaz de generalizar, es decir, después de haber aprendido una serie de patrones podía reconocer otros similares, aunque no se le hubiesen presentado anteriormente. Sin embargo, tenía una serie de limitaciones, por ejemplo, su incapacidad para resolver el problema de la función OR-exclusiva y, en general, era incapaz de clasificar clases no separables linealmente. En 1959, escribió el libro Principios de Neurodinámica, en el que confirmó que, bajo ciertas condiciones, el aprendizaje del Perceptrón convergía hacia un estado finito (Teorema de Convergencia del Perceptrón).

11.- 1960 - Bernard Widrow/Marcial Hoff. Desarrollaron el modelo Adaline (ADAptative LINear Elements). Esta fue la primera red neuronal aplicada a un problema real (filtros adaptativos para eliminar ecos en las líneas telefónicas) que se ha utilizado comercialmente durante varias décadas.

12.- 1961 - Karl Steinbeck: Die Lernmatrix. Red neuronal para simples realizaciones técnicas (memoria asociativa).

13.- 1967 - Stephen Grossberg. A partir de sus conocimientos fisiológicos, ha escrito numerosos libros y desarrollado modelo de redes neuronales. Construyó una red: Avalancha, que consistía en elementos discretos con actividad que varía en el tiempo que satisface ecuaciones diferenciales continuas, para resolver actividades como reconocimiento continuo de habla y aprendizaje de los brazos de un robot.

14.- 1969 - Marvin Minsky/Seymour Papert. En este año surgieron críticas que frenaron, hasta 1982, el crecimiento que estaban experimentando las investigaciones sobre redes neuronales. Minsky y Papera, del Instituto Tecnológico de Massachussets (MIT), publicaron un libro Perceptrons. Probaron (matemáticamente) que el Perceptrón no era capaz de resolver problemas relativamente fáciles, tales como el aprendizaje de una función no-lineal. Esto demostró que el Perceptrón era muy débil, dado que las funciones no-lineales son extensamente empleadas en computación y en los problemas del mundo real. A pesar del libro, algunos investigadores continuaron su trabajo. Tal fue el caso de James Anderson, que desarrolló un modelo lineal, llamado Asociador Lineal, que consistía en unos elementos integradores lineales (neuronas) que sumaban sus entradas. Este modelo se basa en el principio de que las conexiones entre neuronas son reforzadas cada vez que son activadas. Anderson diseñó una potente extensión del Asociador Lineal, llamada Brain State in a Box (BSB).

15.- 1974 - Paul Werbos. Desarrolló la idea básica del algoritmo de aprendizaje de propagación hacia atrás (backpropagation); cuyo significado quedó definitivamente aclarado en 1985.

16.- 1977 - Stephen Grossberg. Teoría de Resonancia Adaptada (TRA). La Teoría de

Resonancia Adaptada es una arquitectura de red que se diferencia de todas las demás previamente inventadas. La misma simula otras habilidades del cerebro: memoria a largo y corto plazo.

17.- 1977 - Teuvo Kohonen. Ingeniero electrónico de la Universidad de Helsinki, desarrolló un modelo similar al de Anderson, pero independientemente.

18.- 1980 - Kunihiko Fukushima. Desarrolló un modelo neuronal para el reconocimiento de patrones visuales.

19.- 1985 - John Hopfield. Provocó el renacimiento de las redes neuronales con su libro: "Computación neuronal de decisiones en problemas de optimización."

20.- 1986 - David Rumelhart / J. L. McClelland. Estos dos investigadores fundaron el PDP (Parallel Distributed Processing) un grupo dedicado al estudio del conocimiento. De este grupo se editó el libro "Parallel Distributed Processing: Explorations in the Microstructures of Cognition". Este libro supuso una revolución dentro de las RNA, en él se exponía el modelo Back-Propagation que resolvía el problema de la asignación de créditos propuesto por Minsky. Después de este libro se produjo una explosión en las RNA apareciendo modelos, técnicas, campos de aplicación y fusiones híbridas de modelos. Esta explosión e interés en el tema ha durado hasta la actualidad donde las RNA son una de las herramientas matemáticas más utilizadas.

Apéndice B

Función AND

```
/*Este programa ilustra el funcionamiento de una red tipo perceptrón que resuelve el
problema de la función logica AND.*/
#include < conio.h>
#include <iostream.h>
#include < alloc.h >
#define NO_HAY_ CAMBIO 0
#define HAY_CAMBIO 1
/* Descripción de las característica y propiedades de la neurona */
class neurona{
      int *w;  /* Puntero a el arreglo de pesos */
      int neta;  /* Entrada neta */
      int inds;  /* Indicador de salida */
      float umbral; /* umbral */
      int nw; /* Numero de pesos */
      int cambio;  /* Indicador de modificación de pesos */
```

```
    int *ents; /* Puntero a un arreglo del tamaño del número de pesos para almacenar
las entradas*/
public:

    neurona(int, float);
    ~ neurona(void);
    void activacion(int *);
    int salida(void);
    void aprendizaje (int *, int);
    int mira_cambio(void);
};
neurona::neurona(int num_w,float u)
{
    neta=0;
    inds=0;
    umbral=u;
    nw=num_w;
    w=(int *)malloc (sizeof(int)*nw);
    for(int i=0;i<=nw;i++) w[i]=0;
    cambio=NO_ HAY_CAMBIO;
}
neurona::~neurona(void)
{

    free(w);
}
void neurona::activacion(int *ent)
```

```
{

    neta=0;
    for(int i=0; i<nw; i++) neta+=(ent[i] * w[i]);
}
int neurona:: salida (void)
{

    if(neta<umbral)
       inds=1;
    else
       inds=0;
    return inds;
}
/* En este bucle se realiza el aprendizaje */
void neurona:: aprendizaje (int *ent,int s)
{

    activacion(ent);
    salida();
    int error=s-inds;
    if(error){
        umbral - =error;
      for(int i=0;i<nw;i++) w[i]+=(error*ent[i]);
      if(!cambio) ents=(int *)malloc(sizeof(int)*nw);
      cambio=HAY_CAMBIO;
```

```
        memcpy(ents,ent,sizeof(int)*nw);
    }else
        if(!memcmp(ents,ent,sizeof(int)*nw)){
            cambio=NO_HAY_CAMBIO;
            free(ents);
        }
}
int neurona::mira_cambio(void)
{

    return cambio;
}
void main()
{
    int entradas[8]={0,0,0,1,1,0,1,1},salidas[4]={0,0,0,1},i;
    neurona n1(2,0.4);

    clrscr();
    do{
        for(i=1;i<=4;i++)
            n1.aprendizaje(&entradas[(i-1)*2],salidas[i-1]);
    }while(n1.mira_cambio());
    cout<<"Entradas     "<<"Salidas \ n";
    cout<<"------     "<<"-------\ n\ n";
    for(i=1;i<=4;i++){
        n1.activacion(& entradas[(i-1)*2]);
        cout << "          ";
```

```
        cout << entradas[(i-1)*2];
        cout << ","<< entradas [(i-1)*2+1];
        cout<<"                    ";
        cout << n1.salida() << "\ n";
    }
    getch();
}
```

Apéndice C

función OR

```
/*Este programa ilustra el funcionamiento de una red tipo perceptrón que resuelve el
problema de la función logica OR.*/
#include < conio.h>
#include <iostream.h>
#include < alloc.h >
#define NO_HAY_ CAMBIO 0
#define HAY_CAMBIO 1
/* Descripción de las característica y propiedades de la neurona */
class neurona{
      int *w;  /* Puntero a el arreglo de pesos */
      int neta;  /* Entrada neta */
      int inds;  /* Indicador de salida */
      float umbral; /* umbral */
      int nw; /* Numero de pesos */
      int cambio;  /* Indicador de modificación de pesos */
```

```
    int *ents;  /* Puntero a un arreglo del tamaño del número de pesos para almacenar
las entradas*/
public:

    neurona(int, float);
    ~ neurona(void);
    void activacion(int *);
    int salida(void);
    void aprendizaje (int *, int);
    int mira_cambio(void);
};
neurona::neurona(int num_w,float u)
{
    neta=0;
    inds=0;
    umbral=u;
    nw=num_w;
    w=(int *)malloc (sizeof(int)*nw);
    for(int i=0;i<=nw;i++) w[i]=0;
    cambio=NO_ HAY_CAMBIO;
}
neurona::~neurona(void)
{

    free(w);
}
void neurona::activacion(int *ent)
```

```
{

    neta=0;
    for(int i=0; i<nw; i++) neta+=(ent[i] * w[i]);
}
int neurona:: salida (void)
{

    if(neta<umbral)
      inds=1;
    else
      inds=0;
    return inds;
}
/* En este bucle se realiza el aprendizaje */
void neurona:: aprendizaje (int *ent,int s)
{

    activacion(ent);
    salida();
    int error=s-inds;
    if(error){
        umbral - =error;
      for(int i=0;i<nw;i++) w[i]+=(error*ent[i]);
      if(!cambio) ents=(int *)malloc(sizeof(int)*nw);
      cambio=HAY_CAMBIO;
```

```
        memcpy(ents,ent,sizeof(int)*nw);
    }else
        if(!memcmp(ents,ent,sizeof(int)*nw)){
            cambio=NO_HAY_CAMBIO;
            free(ents);
        }
}
int neurona::mira_cambio(void)
{

    return cambio;
}
void main()
{
    int entradas[8]={0,0,0,1,1,0,1,1},salidas[4]={0,1,1,1},i;
    neurona n1(2,0.4);

    clrscr();
    do{
        for(i=1;i<=4;i++)
            n1.aprendizaje(&entradas[(i-1)*2],salidas[i-1]);
    }while(n1.mira_cambio());
    cout<<"Entradas    "<<"Salidas \ n";
    cout<<"------    "<<"-------\ n\ n";
    for(i=1;i<=4;i++){
        n1.activacion(& entradas[(i-1)*2]);
        cout << "        ";
```

```
        cout << entradas[(i-1)*2];
        cout << ","<< entradas [(i-1)*2+1];
        cout<<"                    ";
        cout << n1.salida() << "\ n";
    }
    getch();
}
```

Apéndice D

Representación de las clases almacenadas

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	1	1	1	1	-1	-1	-1
-1	-1	1	1	1	1	1	1	-1	-1
-1	1	1	1	-1	-1	1	1	1	-1
-1	1	1	1	-1	-1	1	1	1	-1
-1	1	1	1	-1	-1	1	1	1	-1
-1	1	1	1	-1	-1	1	1	1	-1
-1	1	1	1	-1	-1	1	1	1	-1
-1	-1	1	1	1	1	1	1	-1	-1
-1	-1	-1	1	1	1	1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Clase x[0]. *Representación del Cero*

-1	-1	-1	1	1	1	1	-1	-1	-1
-1	-1	-1	1	1	1	1	-1	-1	-1
-1	-1	-1	1	1	1	1	-1	-1	-1
-1	-1	-1	1	1	1	1	-1	-1	-1
-1	-1	-1	1	1	1	1	-1	-1	-1
-1	-1	-1	1	1	1	1	-1	-1	-1
-1	-1	-1	1	1	1	1	-1	-1	-1
-1	-1	-1	1	1	1	1	-1	-1	-1
-1	-1	-1	1	1	1	1	-1	-1	-1
-1	-1	-1	1	1	1	1	-1	-1	-1
-1	-1	-1	1	1	1	1	-1	-1	-1
-1	-1	-1	1	1	1	1	-1	-1	-1

Clase x[1]. *Representación del Uno*

1	1	1	1	1	1	1	-1	-1	-1
1	1	1	1	1	1	1	-1	-1	-1
-1	-1	-1	-1	-1	1	1	-1	-1	-1
-1	-1	-1	-1	-1	1	1	-1	-1	-1
-1	-1	-1	-1	-1	1	1	-1	-1	-1
1	1	1	1	1	1	1	-1	-1	-1
1	1	1	1	1	1	1	-1	-1	-1
1	1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	1	1	-1	-1	-1
1	1	1	1	1	1	1	-1	-1	-1

Clase x[2]. *Representación del Dos*

-1	-1	**1**	**1**	**1**	**1**	**1**	**1**	-1	-1
-1	-1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	-1
-1	-1	-1	-1	-1	-1	-1	**1**	**1**	-1
-1	-1	-1	-1	-1	-1	-1	**1**	**1**	-1
-1	-1	-1	-1	-1	-1	-1	**1**	**1**	-1
-1	-1	-1	-1	-1	**1**	**1**	**1**	**1**	-1
-1	-1	-1	-1	-1	**1**	**1**	**1**	**1**	-1
-1	-1	-1	-1	-1	-1	-1	**1**	**1**	-1
-1	-1	-1	-1	-1	-1	-1	**1**	**1**	-1
-1	-1	-1	-1	-1	-1	-1	**1**	**1**	-1
-1	-1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	-1
-1	-1	**1**	**1**	**1**	**1**	**1**	**1**	-1	-1

Clase x[3]. *Representación del Tres*

-1	-1	-1	-1	-1	-1	**1**	**1**	-1	-1
-1	**1**	**1**	-1	-1	-1	**1**	**1**	-1	-1
-1	**1**	**1**	-1	-1	-1	**1**	**1**	-1	-1
-1	**1**	**1**	-1	-1	-1	**1**	**1**	-1	-1
-1	**1**	**1**	-1	-1	-1	**1**	**1**	-1	-1
-1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	-1
-1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	-1
-1	-1	-1	-1	-1	-1	**1**	**1**	-1	-1
-1	-1	-1	-1	-1	-1	**1**	**1**	-1	-1
-1	-1	-1	-1	-1	-1	**1**	**1**	-1	-1
-1	-1	-1	-1	-1	-1	**1**	**1**	-1	-1
-1	-1	-1	-1	-1	-1	**1**	**1**	-1	-1

Clase x[4]. *Representación del Cuatro*

-1	-1	-1	1	1	1	1	1	1	1
-1	-1	-1	1	1	1	1	1	1	1
-1	-1	-1	1	1	-1	-1	-1	-1	-1
-1	-1	-1	1	1	-1	-1	-1	-1	-1
-1	-1	-1	1	1	-1	-1	-1	-1	-1
-1	-1	-1	1	1	1	1	1	1	1
-1	-1	-1	1	1	1	1	1	1	1
-1	-1	-1	-1	-1	-1	-1	-1	1	1
-1	-1	-1	-1	-1	-1	-1	-1	1	1
-1	-1	-1	-1	-1	-1	-1	-1	1	1
-1	-1	-1	1	1	1	1	1	1	1
-1	-1	-1	1	1	1	1	1	1	1

Clase x[5]. *Representación del Cinco*

1	1	1	1	1	1	-1	-1	-1	-1
1	1	1	1	1	1	-1	-1	-1	-1
1	1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	1	-1	-1	-1	-1
1	1	1	1	1	1	-1	-1	-1	-1
1	1	-1	-1	1	1	-1	-1	-1	-1
1	1	-1	-1	1	1	-1	-1	-1	-1
1	1	-1	-1	1	1	-1	-1	-1	-1
1	1	1	1	1	1	-1	-1	-1	-1
1	1	1	1	1	1	-1	-1	-1	-1

Clase x[6]. *Representación del Seis*

-1	1	1	1	1	1	1	1	1	1
-1	1	1	1	1	1	1	1	1	1
-1	-1	-1	-1	-1	-1	-1	1	1	1
-1	-1	-1	-1	-1	-1	-1	1	1	1
-1	-1	-1	-1	-1	-1	-1	1	1	1
-1	-1	-1	-1	-1	-1	-1	1	1	1
-1	-1	-1	-1	-1	-1	-1	1	1	1
-1	-1	-1	-1	-1	-1	-1	1	1	1
-1	-1	-1	-1	-1	-1	-1	1	1	1
-1	-1	-1	-1	-1	-1	-1	1	1	1
-1	-1	-1	-1	-1	-1	-1	1	1	1
-1	-1	-1	-1	-1	-1	-1	1	1	1

Clase x[7]. *Representación del Siete*

-1	-1	1	1	1	1	1	1	-1	-1
-1	-1	1	1	1	1	1	1	-1	-1
-1	-1	1	-1	-1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	-1	-1	1	-1	-1
-1	1	1	1	1	1	1	1	1	-1
-1	1	1	1	1	1	1	1	1	-1
-1	1	1	-1	-1	-1	-1	1	1	-1
-1	1	1	-1	-1	-1	-1	1	1	-1
-1	1	1	-1	-1	-1	-1	1	1	-1
-1	1	1	1	1	1	1	1	1	-1
-1	1	1	1	1	1	1	1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Clase x[8]. *Representación del Ocho*

-1	1	1	1	1	1	1	1	1	-1
1	1	1	1	1	1	1	1	1	-1
1	1	-1	-1	-1	-1	-1	1	1	-1
1	1	-1	-1	-1	-1	-1	1	1	-1
1	1	-1	-1	-1	-1	-1	1	1	-1
1	1	1	1	1	1	1	1	1	-1
-1	1	1	1	1	1	1	1	1	-1
-1	-1	-1	-1	-1	-1	-1	1	1	-1
-1	-1	-1	-1	-1	-1	-1	1	1	-1
-1	-1	-1	-1	-1	-1	-1	1	1	-1
-1	-1	1	1	1	1	1	1	1	-1
-1	-1	1	1	1	1	1	1	1	-1

Clase x[9]. *Representación del Nueve*