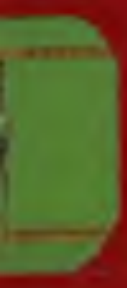


CM

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS

DEL INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION DE COMPUTACION

MODELACION DE REDES NEURONALES

Tesis que presenta el Sr. Martin Figueroa Mireles para obtener el grado de **Maestro en Ciencias** en la especialidad de Ingenieria Eléctrica. Trabajo dirigido por el Dr. Jan Janecek Hyan.

México D. F.
29 de agosto 1990

Becario de
COSNET

INDICE GENERAL

A mis padres:

Arcelia y José.

Por su ejemplo de honradez y de constante esfuerzo para alcanzar el exito.

Y a toda mi **familia** por su incondicional apoyo.

Un profundo agradecimiento al asesor: **Dr. Jan Janecek Hyan**, por sus sugerencias y esmero para alcanzar los objetivos fijados. De igual manera agradezco al **Dr. Guillermo Morales-Luna** y al **M. en C. Sergio Chapa Vergara**, por los comentarios y consejos para llevar a cabo el documento de tesis.

Un reconocimiento muy especial a todos mis compañeros y amigos, quienes formaron un ambiente de trabajo muy confortable.

INTRODUCCION

Conseguir diseñar y construir una máquina que sea capaz de desarrollar procesos inteligentes en el tratamiento de la información de manera semejante al cerebro humano, ha sido desde siempre uno de los principales objetivos de la investigación científica.

En 1943 MacCulloch y Pitts [1] presentaron el primer modelo para simular el funcionamiento del cerebro por medio de redes neuronales artificiales (RNA). Para principios de la década de los sesentas se habían desarrollado modelos de RNA bien definidos. Se expusieron mejores mecanismos de aprendizaje, diferentes estructuras (Cantidad de neuronas de entrada, de salida y su interconexión), etc. La teoría del Perceptron desarrollada por Rosenblatt [2] fue la más exitosa, la estructura contaba con 2 niveles, uno de entrada y el otro de salida, por ello se tenían deficiencias para resolver algunos problemas. Uno de los clásicos es la función XOR, cuya solución se encuentra agregando un nivel intermedio, pero faltaba el algoritmo de aprendizaje que los apoyara. Este problema matemático fue una de las razones principales para abandonar el área de RNA.

Las RNA necesita máquinas masivamente paralelas y que trabajen a velocidades muy altas para realizar las simulaciones, en la década de los sesentas la tecnolgía era insuficiente para construirlas, siendo la segunda razón por lo que se abandonaron las RNA.

Con la moderna tecnología microelectrónica de los últimos años se ha permitido entrever la posibilidad de construir en circuitos de silicio redes cuyo funcionamiento simula a las neuronas del cerebro, con una gran capacidad de trabajar en paralelo y a velocidades altas, por lo que se retoman las teorías matemáticas más importantes de hace más de cuarenta años para diseñar una computadora con arquitectura totalmente diferente a la tradicional, en que de manera semejante al ser humano procese

inteligentemente la información y que tenga sus facultades cognoscitivas (habla, razonamiento, elaboración de conceptos, reconocimiento de imágenes, etc), características que las computadoras actuales no tienen.

El objetivo principal de la técnica de las RNA es crear un cerebro artificial, aunque actualmente se puede considerar como un proyecto irrealizable.

En la tesis se han fijado los siguientes objetivos:

a) Realizar una herramienta que permita desarrollar aplicaciones de redes neuronales a partir de ella.

b) Implementarla en un Transputer T800, con su lenguaje especializado OCCAM2. Se utilizará está máquina por su velocidad para las operaciones de punto flotante. (Ver Capitulo 2).

c) Se realizarán algunos ejemplos con variaciones de los parámetros básicos y por último un reconocedor de caracteres manúscritos.

En cada uno de ellos los resultados son los siguientes:

a) Las características más importantes de la herramienta desarrollada se mencionan a continuación:

i) Nos permite editar la estructura de la red en forma interactiva con el usuario o con el editor de texto de TURBO C, de está última forma se pueden crear archivos para los patrones de aprendizaje.

ii) Se salvan y cargan archivos de forma compacta y de texto.

iii) Para el aprendizaje se da un número de iteraciones. Si el error global es menor que el umbral de convergencia se detiene y despliega en la pantalla la cantidad de barridos ejecutados.

b) La velocidad del transputer T800 para el aprendizaje de la red neuronal es muy alto con respecto a otras máquinas.

c) Con las aplicaciones realizadas se comprobó el correcto funcionamiento del programa y la eficiencia del algoritmo de retro-propagación con una estructura multi-nivel.

Finalmente se realizaron pruebas en diferentes PC, con el lenguaje de programación Turbo C. Las pruebas se hicieron con la función XOR y el programa corrió por un minuto en cada una de las máquinas. Los resultados son los siguientes:

PC/XT PRINTAFORM con 8088 a 8.4 MHz, 230 iteraciones.
PC/AT SPERRY con 80286 a 16 MHz, 660 iteraciones.
PC/AT JAMECO JE2019 con 80386 a 20 MHz, 1600 iteraciones.
PC/AT VECTRA CON 80286 A 16 MHz, con coprocesador aritmético, 5200 iteraciones.
TRANSPUTER T800 a 20 MHz con coprocesador 65000 iteraciones.

En el capítulo 1 de este documento de tesis, se dan los fundamentos teóricos necesarios para la comprensión de la teoría en la que se sustentan las Redes Neuronales Artificiales. Se hace énfasis en las estructuras multi-nivel y en el algoritmo de aprendizaje por retro-propagación.

En el capítulo 2 se describe la arquitectura del Transputer T800, el lenguaje de programación **OCCAM2**, su instalación en la PC/AT SPERRY y la implementación del programa para redes neuronales REDJAN.

En el capítulo 3 se presenta un conjunto de ejemplos de redes neuronales artificiales, cambiando algunos parámetros básicos.

El apéndice A indica al usuario como trabajar con REDJAN. Se ejemplifica con la función XOR.

1. MODELOS DE REDES NEURONALES

1.1 CONCEPTOS BASICOS DE REDES NEURONALES

REDES NEURONALES BIOLOGICAS (RNB)

El ser humano tiene la capacidad de aprender constantemente cosas nuevas, lo que implica procesar de manera inteligente la información, asi como almacenarla, recordarla, recobrarla y con ella aumentar la capacidad de análisis, de intuir, tomar decisiones, comprender el lenguaje, reconocer imágenes, etc.

El único órgano en el cuerpo que realiza estas actividades es el cerebro, el cual está constituído por una enorme red de neuronas conectadas entre si, en grupos. El cerebro está divido por lo menos en 50 grupos, especializados en una actividad específica cada uno de ellos, tales como visión, voz, sistema sensor, auditor, motor, etc. Se considera que se tienen entre 50 y 100 billones de neuronas aproximadamente, una característica muy importante en el hombre es que no pierde el conocimiento adquirido pese a que las neuronas se están muriendo continuamente. Se cree que el cerebro trabaja en **paralelo**, porque varias áreas de neuronas hacen sinapsis con otras, de manera separada pero al mismo tiempo, es decir, se puede ir caminando, ver qué sucede a nuestro alrededor y analizarlo. Estas actividades se hacen con grupos de neuronas especializadas que se encuentran en el cerebro en un lugar determinado por lo que podemos decir que se tienen **distribuidas** las tareas. Las neuronas son dispositivos muy lentos que operan en el rango de 1 a 10 milisegundos para calcular si el estímulo rebasa el umbral, comparándolas con las computadoras actuales que trabajan en el orden de nanosegundos por instrucción; sin embargo el cerebro procesa la información a mayor velocidad por tener áreas especializadas para cada actividad, las que trabajan en paralelo y por contener gran cantidad de neuronas. Si un estímulo de cierta complejidad es presentado responde solamente una determinada área del cerebro y como salida se activará otra totalmente independiente.

De acuerdo con el modelo cerebral más desarrollado, las neuronas tienen tres partes fundamentales para su funcionamiento: el **axón**, las **dendritas** y la **sinapsis**.

El **axón** es la salida de la neurona por donde se envia el impulso a sus receptoras una vez que revisa si rebasa el umbral, la señal puede ser de excitación o deshabilitación.

Las **dendritas** son las ramificaciones del axón para conectar a las neuronas receptoras, asimismo sirven para enviar la información entre ellas.

La **sinapsis** es la fuerza con que manda el estímulo la neurona emisora a las receptoras.

Una neurona tiene varios cientos y a veces hasta miles de entradas y una vez que analiza la información que llega, toma la decisión del tipo de impulso que enviará

Las neuronas típicas de los animales vertebrados pueden llevar un impulso nervioso a una distancia considerable. Los impulsos nerviosos originados en el cuerpo de la célula se propagan a lo largo del axón, algunos son más largos que otros. El axón termina con ramificaciones, éstas en ocasiones llegan a ser miles y sirven de dendritas de las neuronas receptoras. Los componentes principales de la neurona se encuentran ilustrados en la figura 1 y un ejemplo de interconexión en la figura 2.

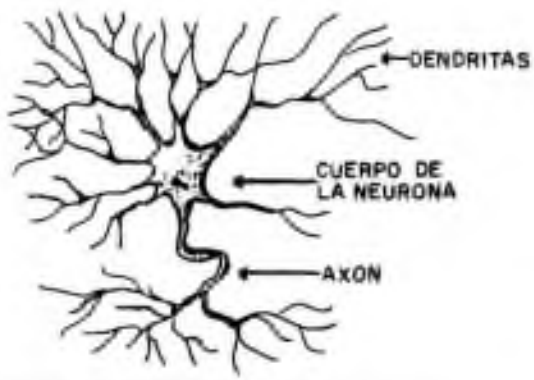


FIG. 1 ESTRUCTURA DE UNA NEURONA BIOLOGICA

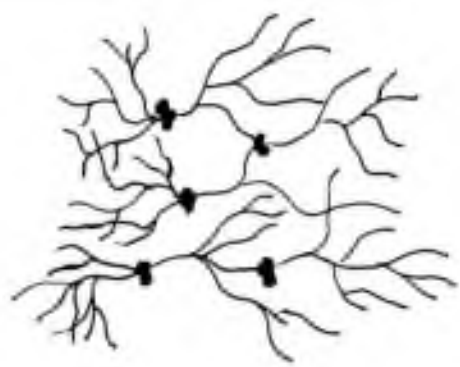


FIG. 2 INTERCONEXION DE NEURONAS

Como ya se mencionó, existen grupos de neuronas especializados para realizar cada actividad, por lo que es lógico pensar que su estructura fisiológica es diferente. Por ejemplo las neuronas del ojo se especializan en captar el color, el movimiento, la posición, intensidad de la luz, etc.

REDES NEURONALES ARTIFICIALES (RNA)

La teoría de RNA tienen una larga historia. El desarrollo de los modelos matemáticos empezó hace más de 45 años con los trabajos de McCulloch y Pitts [1], Hebb [3], Rosenblat [2], Widrow [4], Minsky y Papert [5] y otros. Con los trabajos de Minsky y Papert concluye la primera etapa de RNA, debido a las deficiencias de los algoritmos de aprendizaje y de hardware. Los trabajos más recientes son por Hopfield [6,7,8], Rumeldhart and McClelland [9], Sejnowsky [10], Felman [11], Grossberg [12] y otros que han dado el resurgimiento a este campo. El nuevo interés sobre esta área es debido al desarrollo de nuevas estructuras y algoritmos de aprendizaje [6,7,8,11], la nueva tecnología para la construcción de circuitos integrados [13,14] y obviamente por la creciente fascinación por el cerebro humano.

Las RNA son inspiradas en las biológicas, las cuales están compuestas de igual manera de axones, dendritas y sinapsis. En las RNA el cuerpo de la neurona biológica se simboliza por un **círculo**. La neurona se considera como una compuerta de paso, en cuanto llega la información de las emisoras, la analiza y revisa si es mayor que el **umbral** para enviar una señal inhibitoria o excitatoria a sus vecinas receptoras. Por el axón, la neurona envía el estímulo a las que esta conectada, se simula con un **alambre** o **conexión**. La sinapsis, es la fuerza con que se estimula a las neuronas receptoras y ésta se representa con los pesos de los alambres o conexiones de las neuronas.

Cuando se trabaja con modelos neuronales artificiales nos alejamos del verdadero funcionamiento del cerebro, por lo tanto es

común que a una neurona artificial con sus respectivas señales de entrada, umbral, pesos de las conexiones y las salidas se llame **procesador o elemento de procesamiento o nodo**, ver la figura 3.

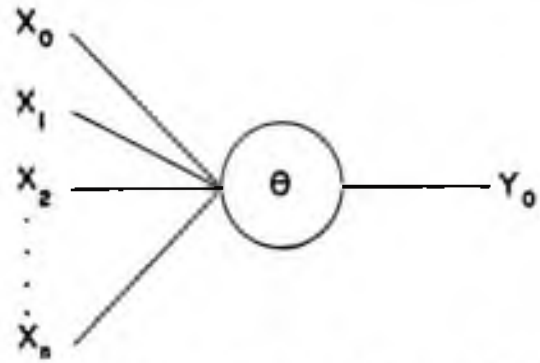


FIG 3. ELEMENTO DE PROCESAMIENTO

Existe circuitería especializada para RNA, la cual trae un procesador electrónico para cada procesador artificial. actualmente hay una máquina con 64 K de procesadores electrónicos que permite interconectarlos como se desee y trabajar con un paralelismo muy alto. Estas máquinas reciben el nombre de **neurocomputadores**.

Se tiene una serie de componentes básicos para RNA, aunque en cada modelo o aplicación difieren de acuerdo a los algoritmos de entrenamiento y estructuras que se utilicen. El modelo **multi-nivel** queda completamente determinado por:

a) Un conjunto de procesadores o neuronas artificiales.
b) Un patrón de conectividad entre los procesadores.
c) Una regla de propagación.
d) Reglas para el aprendizaje de la red.
e) Ambiente dentro del cual el sistema trabaja.

a) Los procesadores son relativamente simples. Cada uno desarrolla un trabajo muy sencillo, que consiste en recibir las entradas y calcular un valor de salida que enviará a sus vecinos receptores. El sistema es inherentemente paralelo, ya que los procesadores de la misma columna pueden ser evaluados al mismo tiempo. Se cuenta

con tres tipos de procesadores en una red neuronal; los del nivel de entrada, del nivel de salida y los niveles ocultos. El número de procesadores que se utilizan depende totalmente del tipo de aplicación que se lleva a cabo. En RNA los procesadores se ponen en forma matricial; la primera columna, se considera como el nivel de entrada; la última, como el de salida y las intermedias como niveles ocultos.

b) El patrón de conectividad de los procesadores depende de la aplicación. Las conexiones que se tengan dentro de la estructura determinan la capacidad para almacenar información, cada una de las interconexiones tiene un peso para poder representar el conocimiento. Se asume que cada procesador influye para la entrada de los vecinos que estén conectados a él. Un peso positivo indica una entrada excitatoria y uno negativo es una salida inhibitoria. Los pesos de las interconexiones se representan por una matríz. La figura 4 muestra cómo se colocan los pesos y las conexiones en una red. Para que se entiendan fácilmente las características se presentará un ejemplo conforme se avance en su explicación.

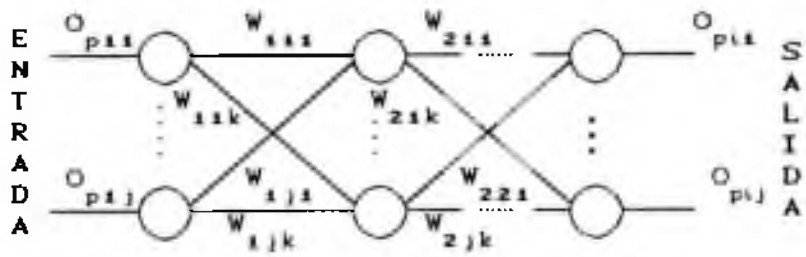


FIG. 4 FORMA DE CONEXION Y COLOCACION DE PESOS EN RNA

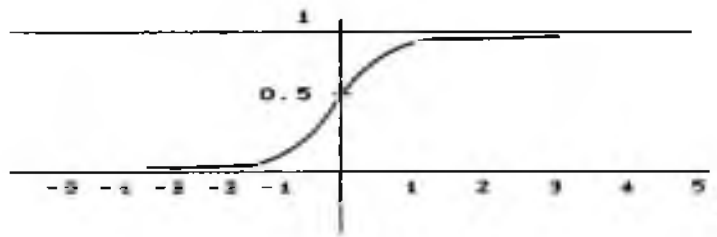
c) Para calcular el estímulo total de un procesador o neurona artificial, se suman los pesos de las conexiones de entrada multiplicados por las salidas de los emisores y finalmente se resta el umbral de la misma. El estímulo de entrada se calcula con la fórmula siguiente:

$$Est_{pij} = \sum W_{(i-1)jk} \, O_{p(i-1)j} - \theta_{(i-1)j}$$

Es necesario ajustar el umbral o nivel de activación (θ_{ij}).

en cada iteración del entrenamiento; el cual, indica con que fuerza dejará pasar el estímulo que llega. Algunas veces el nuevo estado de activación depende del anterior, permitiendo manejarlo como el peso de una conexión, para hacerlo se agrega una neurona en cada columna y los pesos de las conexiones de salida representan los umbrales, se actualizan como cualquier peso. La neurona que simula el umbral no tiene entradas y su salida es siempre uno.

La regla de propagación en RNA se necesita para calcular la salida de los procesadores, ésta depende del estímulo de entrada al mismo. Cuando los patrones de conectividad son muy complejos, se requerirán reglas de propagación de igual manera. Existen tres tipos de salida; la fuertemente limitada, la de umbral lógico y la sigmoidal. En la figura 5, presentamos una ecuación del último tipo con su gráfica; O_{pij} es la salida de un elemento de procesamiento.



$$O_{pij} = \frac{1}{1 + e^{-(Est_{pij})}}$$

FIG. 5 ECUACION DE SALIDA

d) Las reglas de aprendizaje de la red son un punto de gran importancia para RNA, éstas modifican los pesos de las conexiones en función de la experiencia. En principio pueden existir tres clases de modificaciones para el aprendizaje:

a) El poner nuevas conexiones.
b) Quitar conexiones existentes.
c) Modificar los pesos de las conexiones de la red.

En los incisos a) y b) se ha trabajado muy poco, éstos se consideran como casos especiales de c), ya que si el peso de una conexión es cero y éste cambia a un valor positivo o negativo se establece una conexión, en caso contrario si se tiene un valor y se cambia a cero la conexión se pierde.

Existen muchos algoritmos de aprendizaje. Se empezó a trabajar con la regla de Hebbian (1949), la regla de Widrow-Hoff, el teorema de convergencia del Perceptron, la regla delta y memoria asociativa. Dentro de los más actuales tenemos el algoritmo de retro-propagación o propagación de errores, máquinas de Boltzmann, counter-propagación, neocognitron, etc. Las fórmulas que se ponen a continuación son para el algoritmo de aprendizaje por **retro-propagación**.

$$f'(Est_{pij}) = O_{pij}(1 - O_{pij})$$ **DERIVADA DE LA FUNCION PROPAGACION.**

$$\delta_{pij} = (t_{pij} - O_{pij}) f'(Est_{pij})$$ **ERROR DE SALIDA.**

$$\delta_{pij} = f'(Est_{pij}) \sum \delta_{p(i+1)k} W_{ijk}$$ **ERROR OCULTO.**

$$W_{ijk} = W_{ijk} + \eta \delta_{pij} O_{pij}$$ **CAMBIA PESOS.**

O_{pij} = Estímulo de salida de una neurona.

t_{pij} = Patrón de carga de las neuronas de salida.

δ_{pij} = Error de las neuronas ocultas y de salida.

$\delta_{p(i+1)k}$ = Es el error de la neurona a que se conecta.

W_{ijk} = Peso de las conexiones.

η = Rapidez de aprendizaje.

El algoritmo será explicado en **1.3**.

e) La ambientación se caracteriza por el tipo de aplicación y el número de patrones de aprendizaje que se presentan a la red para su entrenamiento.

NIVELES OCULTOS EN RNA

Entre la entrada y salida de la red (característica de los modelos antiguos) puede haber niveles adicionales de procesadores, que se llaman **unidades ocultas** (hidden units). Hay dos puntos de interés en este tipo de modelos:

a) ¿ Cómo influyen los pesos de las conexiones de los niveles ocultos en la representación del conocimiento de la red?

b) ¿ Cómo cambian los patrones de activación o de umbral y qué influencia tienen las unidades ocultas para los patrones que se van a almacenar ?

Las unidades ocultas también pueden ser llamadas **detectores de características aprendidas** o **unidades de re-representación**, porque el patrón de actividad de los niveles ocultos o pesos que representan a este patrón están codificados para que la red los combine como características de entrada. Dicho de otra forma, las unidades ocultas son factibles considerarlas como una entrada más de la red (en la función XOR no se pone porque ya no serían dos entradas solamente y crearía confusión).

En la década de los 80's **Minsky y Papert** demuestran que las redes neuronales no tienen futuro, ya que sólo se puede trabajar con dos niveles (entrada y salida) y éstas no tenían la suficiente potencia para resolver algunos problemas. Su capacidad para clasificar no era muy buena y el almacenamiento de información es muy bajo. Obviamente se pensó poner más niveles a la red, pero no se disponían de algoritmos de aprendizaje que pudieran abarcar estos niveles adicionales. Esta fue una de las razones matemáticas que obligaron a abandonar las RNA por algunas décadas.

En las redes neuronales de dos niveles, los procesadores de entrada están conectados directamente a los de salida. Un problema clásico de estas estructuras es la función **XOR**, la cual no clasifica correctamente en una gráfica los puntos positivos y

negativos como muestra la figura 6 o clasificar clases de patrones en una unidad cuadrada como en la figura 7. Si se agregan niveles de procesadores entre la entrada y la salida se resuelve este problema.

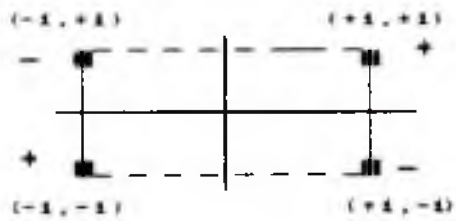
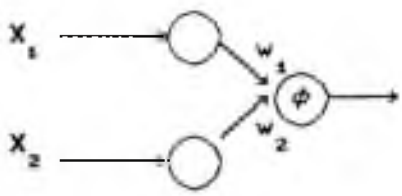


FIG. 6 REPRESENTACION DE LA FUNCION XOR CON SIGNOS

FIG. 7 REPRESENTACION DE LA FUNCION XOR CON PATRONES



La ecuacion que nos entrega la red es la siguiente:

$$v_1 x_1 + v_2 x_2 - \phi = 0, \text{ despej. } x_1$$

$$x_1 = \frac{v_2 x_2}{v_1} + \frac{\phi}{v_1}$$

FIG. 8 REPRESENTACION DE LA FUNCION XOR EN RNA CON ESTRUCTURAS DE DOS NIVELES

Se puede observar que la ecuación de la figura 8 da sólo una línea, la cual no puede separar los puntos positivos y negativos de la figura 6 y las A de las B en la figura 7.

Como ya hemos mencionado el problema anterior se resuelve agregándole un nivel oculto a la estructuras, éste puede tener uno o dos procesadores como indican las figuras 9 y 10.

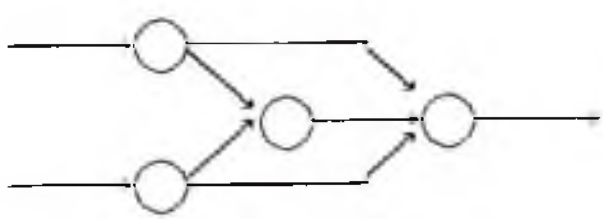


FIG. 9 ESTRUCTURA CON 1 NIVEL OCULTO DE UNA NEURONA

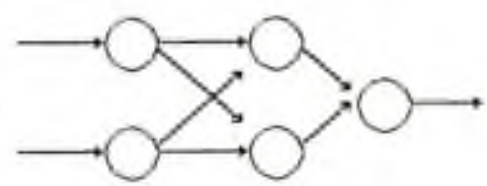


FIG. 10 ESTRUCTURA CON 1 NIVEL OCULTO DE DOS NEURONAS

Si se hace la demostración matematica de estas estructuras, nos entregaran dos lineas rectas mediante las cuales es posible separar los signos positivos y negativos y las letras A y B, como lo manifiestan las figuras 11 y 12 respectivamente.

FIG. 11 CLASIFICACION DE AMBOS SIGNOS CON UN NIVEL OCULTO

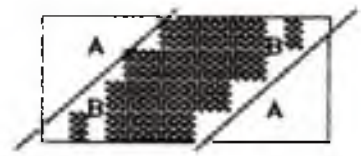


FIG. 12 CLASIFICACION DE LAS LETRAS CON UN NIVEL OCULTO

Si se aumenta a dos los niveles ocultos, se clasificará con mayor precisión, ver la figura 13.

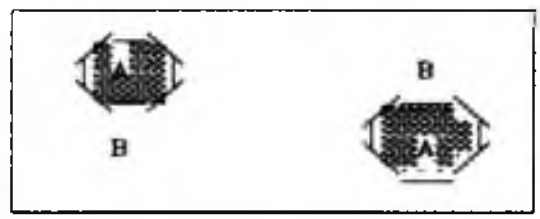


FIG. 13 CLASIFICACION DE PATRONES CON VARIOS NIVELES OCULTOS

La clasificación de la figura 13 se logra con un nivel de entrada, dos niveles ocultos y uno de salida. Con esta estructura es posible separar áreas muy especificas. Si se incrementa el número de procesadores de entrada, la cantidad de niveles ocultos y los de salida se realizan clasificaciones de una gran cantidad de patrones en una sola red neuronal. Actualmente se desconoce aún el potencial de los niveles ocultos, de las interconexiones de los procesadores y de los umbrales.

Es importante hacer la aclaración, que los algoritmos de aprendizaje para estructuras que contienen niveles ocultos y para los que solamente tienen dos niveles (entrada y salida), son muy diferentes.

La figura 14 expone una solución al problema **XOR** encontrada con el algoritmo de aprendizaje por retro-propagación, con una sola unidad oculta. Los números sobre las flechas representan el peso de la conexión entre las unidades de procesamiento, los escritos dentro de los círculos representa el umbral de la unidad. El umbral de +1.5 para la unidad oculta asegura que la salida se activará sólo cuando se presente la entrada (1,1). El valor de umbral de +0.5 de la unidad de salida asegura que se activará cuando llegue una entrada positiva mayor. El peso de -2 de la conexión de la unidad oculta con la unidad de salida asegura que la salida no se active cuando ambas entradas sean 1. Para este ejemplo la salida de la neurona será en números binarios, si rebasa el umbral la salida es un 1, de lo contrario será un 0.

La tabla de verdad de la función **XOR** se encuentra en la figura 15. Note que desde el punto de vista de la unidad de salida, la unidad oculta es tratada como una unidad más entrada, como se aprecia en la tabla 16.

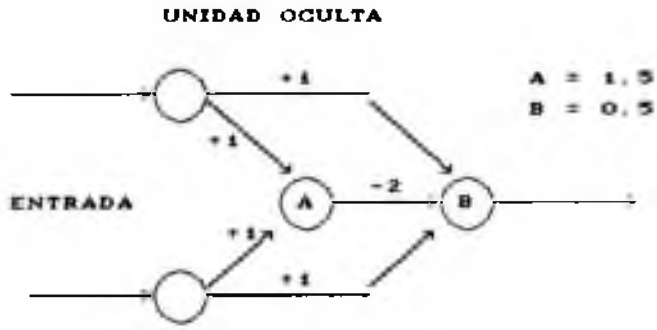


FIG. 14 RNA PARA LA FUNCION XOR CON UNA UNIDAD OCULTA

INPUT	OUTPUT
00	0
01	1
10	1
11	0

FIG. 15 TABLA XOR

INPUT	OUTPUT
00 0	0
01 0	1
10 0	1
11 1	0

FIG. 16 TABLA AUMENTADA

APRENDIZAJE DE LAS RNA

¿ Qué sucede cuando el ser humano adquiere un conocimiento nuevo ? Tal vez lo entienda, almacene, recuerde y se recobre, pero desde el punto de vista fisiológico ¿ Qué cambia en el cerebro ? se pueden crear nuevas conexiones o quitarlas, varía la fuerza con que estimulan a las receptoras o se reentrena el comportamiento de un patrón. También psicológicamente se tienen variaciones en cada persona, así como su estado de ánimo, si es de su interés el tema que se trata en ese momento, etc.

En RNA se consideran sólo algunos puntos fisiológicos para el entrenamiento, el cual no se hacen por programa como en las máquinas de Von Neumann, sino por enseñanza; esto es, se le presentan patrones de entrada como a cualquier humano hasta que la red los aprende.

Los pesos de las interconexiones permiten almacenar el conocimiento en la red y al mismo tiempo reconocer los patrones que se entrenaron.

El entrenamiento o aprendizaje de las RNA consiste básicamente en el constante ajuste de los pesos de las interconexiones, es posible hacerlo manual o automáticamente.

Se tienen dos métodos principales para el entrenamiento: El No-Supervisado y el Auto-supervisado. En el No-supervisado los datos se colocan en la entrada y los pesos se ajustan sin intervención humana, en este proceso se forman agrupamientos de datos internos, cuando se pone un nuevo patrón revisa si existe en algún grupo para no crearlo nuevamente. En el auto-supervisado se da un patrón de entrada y otro de la salida deseada, el aprendizaje ocurre cuando la red se automonitorea y corrige el error en la interpretación de los datos por la retroalimentación.

Inicialmente se asignan pesos aleatoreos a las conexiones, este paso es muy importante pues de él depende que la red aprenda

rápidamente. La red llega a una estabilidad cuando el valor de los pesos que son asociados con las conexiones dejan de cambiar.

La ecuación que modifica todos o algunos de los pesos en las conexiones del procesador responde a las señales de entrada y los valores cambian de acuerdo al estímulo. En efecto; la regla de aprendizaje permite cambiar los pesos conforme pasa el tiempo de entrenamiento, ésto significa que la red adapta los pesos así misma para dar la respuesta deseada a una entrada y organizar la información dentro de ella.

Durante el desarrollo de RNA se han utilizado diferentes algoritmos de aprendizaje, entre los más importantes se encuentran los siguientes:

PERCEPTRON [2,6,15,16,17]. El algoritmo lo desarrolló **Frank Rosenblat** en 1957 se usó para reconocimiento de patrones escritos, su defecto principal es el de no poder reconocer patrones complejos (tales como los chinos), es sensible a las diferencias de escalas, traslación y rotación. Actualmente rara vez se usa, pero fué uno de los mejores de su época.

NEOCOGNITRON [18]. Fue inventado y desarrollado por **Kunihico Fukushima** en 1978-84. Esta enfocado al reconocimiento de patrones escritos, su limitación es la de requerir un gran número de elementos de procesamiento e interconexiones, estas redes son muy complicadas y lo son irrelevantes a la escala, la traslación y la rotación. es muy hábil para reconocer caracteres complejos (tales como los chinos).

RETRO-PROPAGACION [19,20,21,22,23]. Este algoritmo fue inventado por **Paul Werbos, David Parker, David Rumelhart**, se desarrolló durante los años de 1974-85, se ha utilizado en lectura de textos en voz alta, control de robot, reconocimiento de imágenes, etc. Este es uno de los algoritmos más populares que se tienen actualmente, ocupa el primer lugar en la rapidéz de aprendizaje y no es muy complicado.

MAQUINAS DE BOLTZMANN Y DE CAUCHY [24,25,26,27,28,29,30]. La desarrollaron **Joffrey Hinton, Terry Sejnowsky y John Hopkins**, se introdujo en 1985-6, se fundamenta en la teoría termodinámica. Es usado para reconocimiento de imágenes, sonar y radar, este método ocupa el segundo lugar en eficiencia aunque su aprendizaje es más lento que el de retro-propagación.

GROSSBERG [31,32]. Con el apoyo de la teoría de la resonancia adaptativa diseñó una estructura que forma grupos para los patrones que se enseñan y se entrena sin supervisión, puede almacenar gran cantidad de información, pero una pequeña cantidad de ruido causa serios problemas, un conjunto de umbrales puede considerar dos patrones similares como diferentes, el aprendizaje es rápido y puede almacenar patrones hasta que se acaben las neuronas.

HOPFIELD [7,8,9]. Es desarrollado por **John Hopfield** en 1982, se utilizó en la recuperación completa de datos o de fragmentos de imágenes, su implementación puede ser a baja o alta escala. Se han obtenido buenos resultados.

MEMORIA ASOCIATIVA BIDIRECIONAL. Lo estableció **Bark Kosko** en 1985, se aplicó en el contenido de memoria direccionable asociativa, como limitación, tiene una baja densidad de almacenamiento de la información , los datos pueden ser propiamente codificados, éstas redes aprenden con facilidad, es muy buena herramienta educacional, asocia fragmentos de imágenes con una completa que ya se halla aprendido, etc.

COUNTER-PROPAGACION [33]. Este algoritmo fue inventado y desarrollado por **Robert-Hench-Nielsen** en 1986 lo utilizó en comprensión de imágenes, análisis estáticos y otros, su limitación es la de necesitar una gran cantidad de procesadores y de conexiones para dar precisión en las clasificaciones, aunque se basa en retro-propagación no es tan eficiente.

APLICACIONES DE REDES NEURONALES

Las RNA tienen una variedad muy amplia de aplicaciones, como reconocimiento de voz, caracteres manuscritos, texto, sistemas expertos, análisis financieros, manejo de bases de datos, procesamiento de señales, diagnóstico médico, control de procesos, corrección de información confusa, etc. Actualmente se tienen aplicaciones comerciales.

Con la finalidad de entender los conceptos básicos antes mencionados abordaremos una aplicación real.

El **PROYECTO ALVINN** (Manejo Autónomo de un Vehículo Terrestre con RNA) [34] es para manejar el vehículo **NAVLAB** (nombre que recibe) a lo largo de una carretera con cierto espacio de ancho. **ALVINN** (la computadora) tiene dos sensores de entrada del **NAVLAB**. El primero es una imágen de 30X32 pixeles de una cámara de video montada en el techo del vehículo. Cada pixel del video corresponde a una unidad de entrada en la red. El nivel de activación de cada unidad lo indica la brillantes del pixel en la imágen de video. La otra entrada es una imágen de 8X32 pixeles de un láser, con el cual se mide la distancia de los objetos que están en la carretera. El nivel de activación o de umbral de cada unidad en el video de retina representan la distancia correspondiente en la imagen. Las dos retinas (video y láser) se consideran el nivel de entrada de la red neuronal, estás son conectadas a un sólo nivel oculto y finalmente a la unidad de salida.

La respuesta en el nivel de salida es una representación lineal de la dirección, por donde el vehículo viajará . El centro de las unidades de salida representan las condiciones de avance hacia adelante del autómovil, todas las unidades de los extremos del centro de la red representan las formas del lado derecho e izquierdo de la carretera.

El entrenamiento de la red es difícil. Se lleva a cabo con imágenes de carreteras con una ancho variable, como es muy poco

práctico llevar el vehículo por las carreteras físicas para que aprenda, se filmó para entrenarse por medio de la computadora.

El aprendizaje de la red se hizo con 1200 imágenes de carreteras, se presentaron 40 tiempos cada una, los pesos fueron ajustados con el mecanismo de retro-propagación, tomó 30 minutos con la supercomputadora **Carnegie Mellon's systolic-array**. La máquina es diseñada especialmente para ésta aplicación por **Carnegie Mellon's** y construida por **General Electric**, realiza 100 Mflops y ajusta 20 millones de conexiones por segundo.

Ya entrenado, **ALVINN** puede manejar con precisión el vehículo a una velocidad de 3.5 millas por hora. La velocidad tan baja es debido que la computadora no alcanza a procesar toda la información rápidamente.

Cuando se entrena en carreteras de ancho fijo, las unidades ocultas la red actúan como detectores de posiciones y orientaciones variables. Para carreteras de ancho variable, las unidades ocultas actuaron como detectores de los bordes de éstas.

Las unidades ocultas también son excitadas por los obstáculos en la periferia de la imágen e inhibida por los obstáculos en el centro de la misma. Con la fusión de los datos de la cámara y el láser las unidades ocultas determinan la posición y orientación para caminar con precisión en la carretera. La red desarrolló diferentes representaciones internas cuando se entrenó con imágenes de ancho variable.

Si limitamos las unidades ocultas, la red se fuerza a desarrollar detectores de características apropiadas para clasificar correctamente el gran conjunto de patrones de entrada.

Es importante entender que una unidad oculta se ocupa de la actividad colectiva de todas las neuronas de entrada para determinar el comportamiento del NAVLAB.

1.2 MODELOS DE REDES NEURONALES ARTIFICIALES

Durante el desarrollo de las teorías de redes neuronales se han expuesto diversos modelos con diferentes mecanismos de aprendizaje y diferentes estructuras. Las estructuras se clasifican principalmente por el tipo de valores que llegan a su entrada, pueden ser **binarios** o **continuos**. Ambos permiten los métodos de aprendizaje **con y sin supervisión**. La figura 17 muestra detalladamente la clasificación.

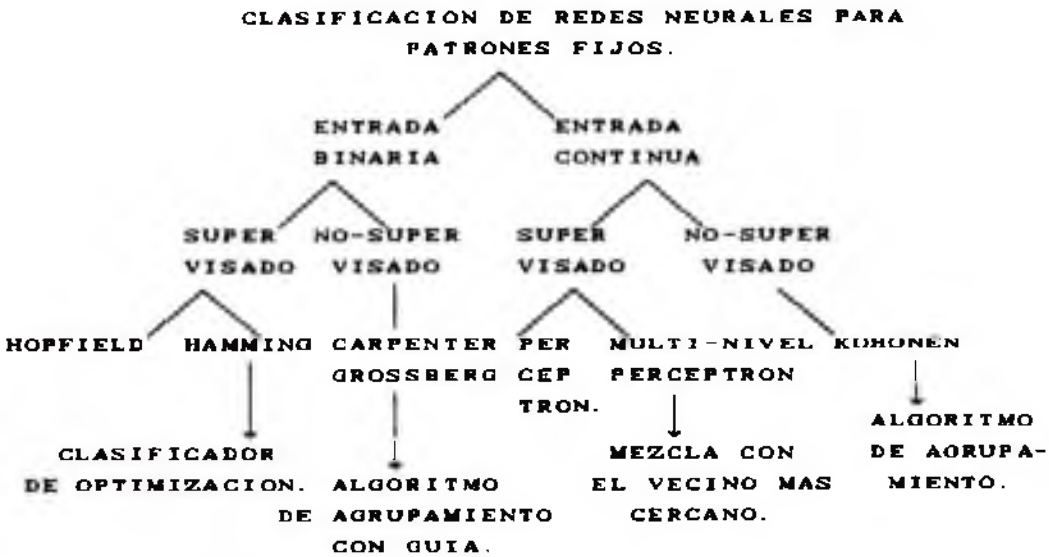


FIG. 17 CLASIFICACION DE SEIS REDES NEURONALES

MODELO DE HOPFIELD

Las redes de Hopfield son normalmente usadas para entradas binarias, tienen un aprendizaje supervisado y su estructura es un sólo nivel de neuronas con retroalimentación . Estas redes son muy apropiadas para representaciones binaria, como en imágenes en blanco y negro, en donde la pantalla tiene una matriz de pixeles y cada uno de ellos sirven de entrada a las neuronas, otro ejemplo es el código ASCCI dado que un caracter se representa con 8 números binarios.

Se tienen diferentes versiones para redes de Hopfield. Estas redes se usan para *memoria asociativa* o para *resolver problemas de optimización*. De la red original [1] es posible hacer una versión que permita ser usada como *memoria direccionable por contenido*. Se aplica principalmente para recobrar información confusa. Actualmente se construyen circuitos integrados con está técnica.

La red mostrada en la figura 18 tiene n nodos, los cuales se consideran como los dispositivos de procesamiento o neuronas. Las neuronas tienen dos estados, como en el método de McCulloch y Pitts [4], $V_i = 1$ si rebasa el umbral el estímulo de entrada y $V_i = -1$ si es menor. Cada neurona tiene un umbral U_i, y es simulado con el peso de una conexión. Cuando la neurona i se conecta a la neurona j se le asigna un peso a la conexión denotado como T_{ij}, la salida de cada nodo retroalimenta al resto de la red (no se conecta a si mismo).

El aprendizaje de la red se divide en cuatro etapas. La primera consiste en asignar un peso inicial a las conexiones usando una fórmula que depende de los estados de las neuronas . En la segunda se presenta un patrón desconocido para forzar a la red que su salida empate con el patrón de entrada. En la tercera etapa se realizan iteraciones hasta que la red lo aprende. Se dice que la red converge cuando el cambio de los pesos de las conexiones es mínimo. Hopfield y otros comprobaron que cuando la red converge la matriz de pesos es simétrica, $T_{ij} = T_{ij}$. El último paso consiste en regresar a la etapa dos para colocar otro patrón a la entrada.

Las redes de Hopfield tienen dos grandes limitación cuando se usan como memoria direccionable por contenido. Primero el número de patrones que pueden ser almacenados y reconocidos con precisión es severamente limitado, por ejemplo 10 patrones pueden requerir más de 70 nodos y aproximadamente 5000 conexiones. La segunda limitación es que será inestable si algunos bits son comunes en los patrones a entrenar.

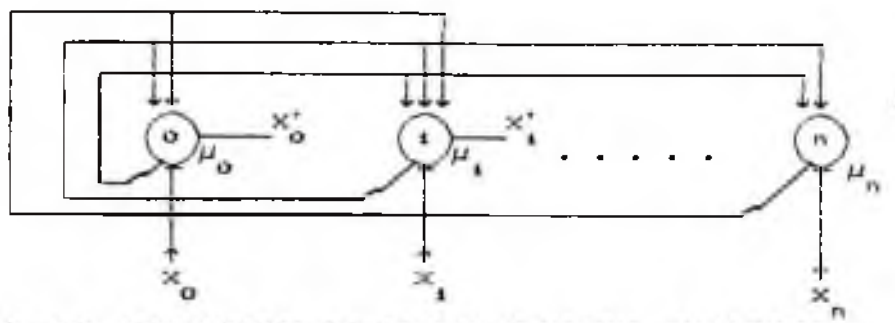


FIG. 18 ESTRUCTURA DE LA RED NEURONAL DE HOPFIELD

MODELO DEL PERCEPTRON

El perceptron [2,5] puede ser usado con entradas binarias o continuas, el aprendizaje es supervisado y la estructura consiste de dos niveles, uno de entrada y el otro de salida. Se utiliza principalmente para reconocer patrones simples, por ejemplo dividir por una linea dos clases de letras. La figura 19 muestra la estructura del Perceptron.

Para el aprendizaje, se calcula el estímulo de entrada a un nodo multiplicando los pesos de las conexiones de entrada por las salidas de los emisores y finalmente restando el umbral. La respuesta de salida será 1 si rebasa el umbral y -1 si no. Los pesos y los umbrales pueden se fijos o ajustados con algún algoritmo. El procedimiento de convergencia para el ajuste de pesos para el Perceptron fue desarrollado por Rosenblatt [2].

El algoritmo de convergencia del perceptron se divide en 5 etapas. En la primera los pesos de las conexiones y de los umbrales son inicializados. En la segunda se presenta un patrón de entrada con N elementos y la salida deseada para el mismo. En la tercer etapa se calcula la salida de los perceptrones. Si las salidas de los perceptrones son diferentes al patrón de carga, los pesos de las conexiones se adaptan o ajustan para obtener la salida deseada. La fórmula con que se hacen los cambios contiene la variable η, la cual nos permite que el aprendizaje sea más rápido, su valor debe estar en el rango de 0.1 y 0.9. En la última etapa se regresa a la dos para entrenar un nuevo patrón.

Un problema con el procedimiento de convergencia del Perceptron es que oscila durante el entrenamiento. Widrow-Hoff [15,16,17] lo modifico utilizando minimos cuadrados, obteniendo buenos resultados. Consiste en minimizar el error entre la salida actual y el patrón de carga del Perceptron.

El Perceptron puede separar perfectamente dos grupos de simbolos por una linea, pero cuando son más de dos no lo puede hacer, tal cosa sucede con la función XOR. Esta deficiencia fue demostrada por Minsky y Papert [5]. Se penso en la posibilidad de poner niveles ocultos de neuronas pero no se tenía un algoritmo de entrenamiento que los soportará.

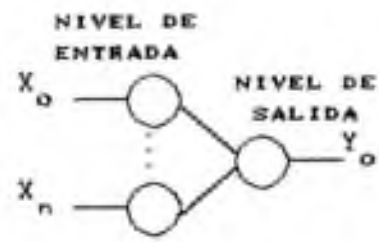


FIG. 19 ESTRUCTURA DEL PERCEPTRON

MODELO DE MAQUINAS DE BOLTZMANN

Las redes neuronales con máquinas de Boltzmann se apoya en la teoría Termodinámica. Si un cuerpo a temperatura ambiente se calienta los electrones fluyen de un nivel a otro y su energía interna se incrementa. Para regresarlo a su estado inicial es necesario enfriarlo lentamente, de lo contrario los electrones quedan en niveles que no le correspoden y no encuentra el Equilibrio Térmico. En las redes neuronales a las conexiones se les asignan un peso y se inteta alcanzar la mínima energía global (se logra cuando la red aprende los patrones) disminuyendo la "Temperatura" T a intervalos pequeños. Si se hace bruscamente es posible encontrar mínimos locales y que la red nunca salga de ese punto. En este caso sólo puede aprender algunos patrones y no se alcanza la energía mínima global del sistema.

La Máquina de Boltzmann [24,25,26,27,28,29,30] es una generalización de las redes de Hopfield [7,8,9], en la cual las unidades actualizan sus estados de acuerdo a una regla de decisión estocástica. Las unidades tienen estados de 1 o 0 y la probabilidad de que la unidad j adopte un estado esta dado por:

$$P_j = \frac{1}{1 + e^{\Delta E_j / T}}$$

En donde $\Delta E_j = X_j$ es la energía de entrada de la j-ésima unidad y T es la "Temperatura", la cual varia en intervalos pequeños para alcanzar el equilibrio térmico. ΔE_j se define por la fórmula siguiente:

$$\Delta E_j = \sum_j W_{kj} S_k S_j$$

A cada configuración de estados se le asigna un número real, llamado "Energía". La energía global de una configuración es interpretado como la capacidad de los pesos de las conexiones para producir una salida adecuada para un patrón de entrada. Si la salida se aleja del patrón deseado se dice que la energía global es alta en el sistema, de lo contrario si la red nos entrega el patrón deseado la energía es mínima. La energía de una configuación está definida por:

$$E = -\sum_{i<j} W_{ij} S_i S_j + \sum_i \theta_i S_i$$

Donde W_{ij} es el peso de la conexión entre las unidades i y j.

S_i es 1, si la unidad i está activa y 0 desactiva.

θ_i Umbral de la neurona i.

Durante el "simulated annealing se puede calcular la energía mínima global para ver si va disminuyendo conforme se entrena la red neuronal.

Para el **aprendizaje** de la red, se inicializan los pesos de las conexiones con un valor de cero. El siguiente paso es realizar un ciclo de aprendizaje, el cual se divide en dos fases. En la primera se coloca el patrón de entrada y el de salida en la red y se busca el equilibrio térmico para dicho patrón (definido como la

situacion en la cual la distribución de probabilidad es estable) utilizando la técnica de "Simulated Annealing". La cual consiste en ir disminuyendo la "Temperatura" T para el patrón y llevar un conteo de las unidades conectadas entre si que tienen un estado activo. Este procedimiento se aplica a todos los patrones de aprendizaje y se calcula un promedio (P_{ij}) de las unidades interconectadas . En la segunda fase se coloca sólo el patrón de entrada y se busca el equilibrio térmico de la red, llevando igual que en la primer fase un conteo de todas las unidades interconectadas entre si que tengan un estado activo, este proceso se aplica a todos los patrones y se calcula un promedio para obtener (P'_{ij}). Una vez que se tienen las dos probabilidades se ajustan los pesos con la siguiente fórmula.

$$\Delta W = \eta \, (\langle P_{ij} \rangle - \langle P'_{ij} \rangle) \qquad \text{En donde}$$

η Es la constante de **rapidez de aprendizaje**.

$\langle P_{ij} \rangle$ Es la probabilidad de que las unidades i y j conectadas entre si estén activas con los patrones de entrada y salida.

$\langle P'_{ij} \rangle$ Es la probabilidad de que las i y j conectadas entre si estén activas, solamente con el patrón de entrada.

Para saber si la red aprende durante el entrenamiento se aplica la fórmula siguiente:

$$G = \sum P^{+}(V_{\alpha}) \ln \frac{P^{+}(V_{\alpha})}{P^{-}(V_{\alpha})} \qquad \text{En donde:}$$

$P^{+}(V_{\alpha})$ Probabilidad con los patrones de entrada y salida (fase 1).

$P^{-}(V_{\alpha})$ Probabilidad sólo con el patrón de entrada (fase 2).

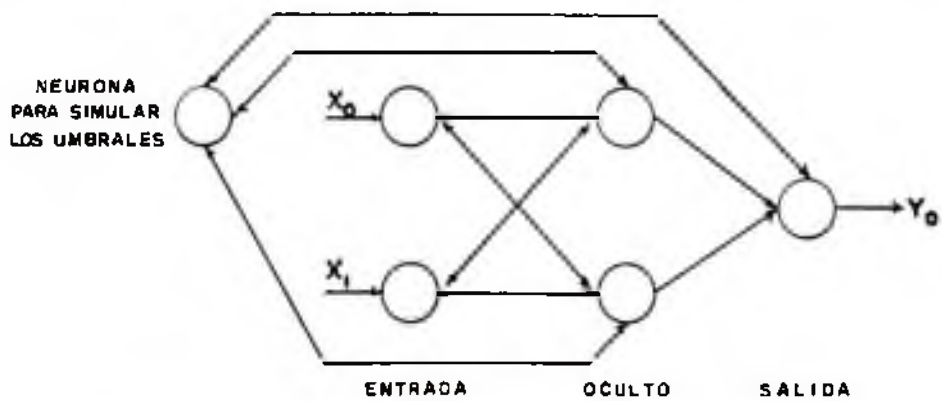


FIG. 20 REPRESENTACION DE LA FUNCION XOR CON MAQUINAS DE BOLTZMANN

MODELO MULTI-NIVEL

La estructura para redes Multi-nivel consiste en varios niveles de neuronas como indica la figura 21. Cada uno puede tener un número finito de neuronas o elementos de procesamiento. El primer nivel, recibe las señales de entrada y las distribuye en la siguiente capa de la red. El último, entrega la respuesta de las señales de entrada. Los niveles ocultos que quedan entre la entrada y la salida realizan un procesamiento de información que ayuda a retener las características relevantes de los patrones usados durante el entrenamiento. La red es unidireccional, ya que la información se propaga en un sólo sentido. Es importante hacer notar que la red no se retroalimenta como en el modelo de Hopfield.

La entrada de la red pueden ser números binarios (0,1) o números reales entre el rango de (0.1, 0.9). Las salidas siempre serán reales porque los pesos también lo son.

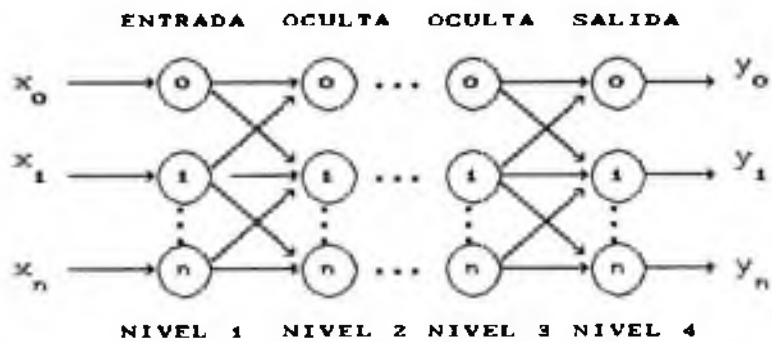


FIG. 21 ESTRUCTURA PARA REDES MULTI-NIVELES

La estructura de la figura 21 se implementó en el Transputer T800, pero con la posibilidad de tener una cantidad variable de niveles y de neuronas en cada uno de ellos.

El algoritmo que se usa para estas redes es el de retro-propagación, el cual se explica detenidamente en el punto 1.3.

1.3 ALGORITMO DE APRENDIZAJE POR RETRO-PROPAGACION

El algoritmo de retro-propagación es uno de los más populares para redes neuronales con estructuras multi-nivel. Se considera como una generalización de la regla de corrección de errores de Widrow-Hoff [4], pero con la capacidad de calcular los errores en las neuronas ocultas. El aprendizaje con retro-propagación es más rápido que con máquinas de Boltzmann, aunque para redes grandes es exponencialmente más lento conforme se incrementan las neuronas ocultas, especialmente cuando el sistema tiene muchos niveles ocultos. Estos dos algoritmos son los más utilizados actualmente. Un punto importante en retro-propagación es que las neuronas no se retroalimentan como en el cerebro.

Algunos de los trabajos más relevantes con retro-propagación son un lector de inglés en voz alta [35] y el manejo automático de un automóvil [34].

Para el entrenamiento de la red es necesario presentar un par de patrones. El primero es usado como el vector de entrada, el cual debe aprender la red. El segundo es el patrón de carga para las neuronas de salida, se considera como la respuesta que deseamos que nos entregue la red cuando haya aprendido el patrón de entrada. Los patrones se presentan aleatoriamente o de manera secuencial una vez cada uno.

El algoritmo consiste fundamentalmente en dos fases:

a) **Hacia Adelante.** Una vez colocados los patrones en la red, el sistema usa el vector de entrada para propagarlo a través de la red y producir una salida, los valores obtenidos son comparados con el patrón de carga para poder calcular el error de las neuronas de salida.

b) **Retroceso.** Se calculan los errores de las neuronas ocultas y se ajustan los pesos de las conexiones. Se pueden calcular todos los errores ocultos en una fase de retroceso y en otra ajustar los

pesos o en una sola fase hacer las dos cosas. Se empieza con la neurona del último nivel oculto y se recorre hacia la entrada.

A continuación se explicarán datalladamente las dos fases del algoritmo, dividiendolas en 6 pasos.

FASE HACIA ADELANTE

Paso 1. Inicialización de los pesos.

De manera aleatoria se asignan pesos a las conexiones. Los valores deben de estar en el rango de 0.99 y -0.99. De los pesos iniciales depende de que la red aprenda con lentitud o rapidez.

Paso 2. Presentar el patrón de entrada y de salida.

El número de elementos que representan al patrón a entrenar debe ser igual al número de neuronas del nivel de entrada, de igual manera para el patrón de carga. Los valores de entrada y salida pueden ser continuos o enteros. El patrón de entrada será almacenado en el vector llamado O_{pij}, en donde el subíndice "p" indica el patrón que se trabaja en ese momento, "i" el nivel de la estructura y "j" indica el número de neurona del nivel entrada. El patrón de carga o de salida deseada se almacena en el vector t_{pij}, en donde el subíndice "i" es igual al número de niveles ocultos más dos y "j" la cantidad de neuronas del nivel de salida. En la figura 22 se tiene un patrón de entrada de 4 elementos y en la salida un patrón de 3 elementos.

Paso 3. Propagar el patrón de entrada en la red.

El patrón de entrada debe propagarse a través de la red neuronal hasta llegar al nivel de salida, éste procedimiento se divide en dos partes:

a) Calcular el estímulo de entrada de la neurona.
b) Calcular la respuesta de la neurona.

Se utilizará la figura 22 para explicar con precisión las dos fórmulas.

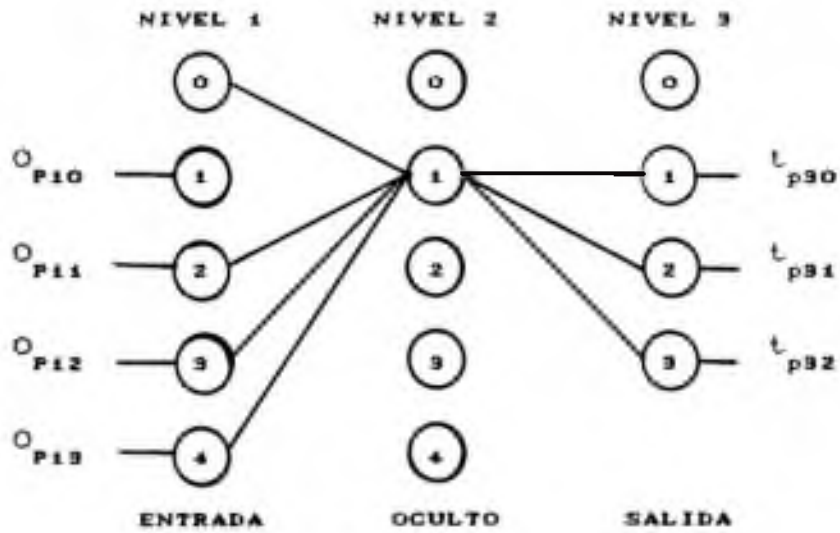


FIG. 22 ESTRUCTURA PARA CALCULAR ENTRADA Y SALIDA DE UNA NEURONA

a) El estímulo de entrada para una neurona se cálcula con la siguiente fórmula.

$$Est_{pij} = \Sigma\ W_{(i-1)jk}\ O_{p(i-1)j} - \theta_{(i-1)ok}$$ En donde:

$Est_{p(j)}$ = Representa el estímulo de entrada del patrón "p" para la neurona "j" que se encuentra en el nivel "i".

$W_{(i-1)jk}$ = Representa el peso de la conexión de la neurona "j" del nivel "i" con el nivel "i-1", "k" significa el número de conexiones de salida de la neurona "j".

$O_{p(i-1)j}$ = Salida de la neurona "j" del nivel "i-1" para el patrón "p".

$\theta_{(i-1)ok}$ = Umbral de la neurona en la que se calcula el estímulo de entrada.

Como ya hemos mencionado el umbral se simula con el peso de una conexión, se toma siempre de la neurona "o" del nivel "(i-1)" y la conexión de salida "k". Por lo que:

$$\theta_{(i-1)ok} = W_{(i-1)ok} * 1$$

La fórmula finalmente queda:

$$Est_{pij} = \Sigma\ W_{(i-1)jk}\ O_{p(i-1)j}$$

Los pesos de las conexiones se almacenan en la neurona que envía el estímulo, por esta razón la variable de los pesos y estímulos de salida tiene el subíndice "i-1".

La fórmula indica que realicemos la sumatoria de las multiplicaciones de los pesos de las conexiones de entrada de la neurona ($W_{(i-1)jk}$) por el estímulo ($O_{p(i-1)j}$) que nos envían las emisoras.

Se calculará el estímulo para la neurona 1 del nivel 2 de la figura 22. Observemos que la neurona 1 del nivel 1 no se conecta a la neurona 1 del nivel 2. Desarrollando la fórmula para calcular el estímulo nos queda de la siguiente manera.

$$Est_{p21} = W_{100} * 1 + W_{120} * O_{p12} + W_{130} * O_{p13} + W_{140} * O_{p14}$$

b) La salida de la misma neurona se calcula con la fórmula que aparece a continuación:

$$O_{p21} = \frac{1}{1 + e^{-(Est_{p21})}}$$

El resultado de la función de propagación siempre se encontrará en el rango de 0 y 1.

Terminando de calcular la salida de la neurona 1 se continua con la 2, hasta terminar con el nivel. De igual manera se realiza para todos los niveles, finalizando con el de salida.

Paso 4. Calcular el error de las neuronas de salida.

En cuanto se obtienen los valores en las neuronas de salida del patrón propagado, se calculan los errores de salida para poder ajustar posteriormente los pesos de las conexiones. La fórmula que se utiliza es la siguiente:

$$\delta_{pij} = (t_{pij} - O_{pij})\, f'(Est_{pij}) \qquad \text{En donde}$$

$$f'(Est_{pij}) = O_{pij}(1 - O_{pij})$$

δ_{pij} = Representa el error de la neurona "j" del nivel "i" para el patrón "p"

$f'(Est_{pij})$ es la derivada de la función de propagación, calcula la variación de la salida con respecto a la entrada de la neurona.

Se resta la salida actual de la neurona (O_{pij}) a el patrón de carga (t_{pij}) y se multiplica por la derivada.

FASE DE RETROCESO

Paso 5. Calcular los errores ocultos y ajustar los pesos.

La capacidad de calcular los errores ocultos en los niveles intermedios de la red, es lo que hace poderoso al algoritmo de retro-propagación. En las neuronas ocultas no tenemos un patrón de carga determinado, por lo que se sigue otro procedimiento para calcular el error con que contribuye a las neuronas de salida.

La fórmula para calcular los niveles ocultos es la siguiente:

$$\delta_{pij} = f'(Est_{pij}) \sum \delta_{p(i+1)jk} * W_{ijk}$$

δ_{pijk} = Variable para almacenar el resultado del cálculo de error oculto de la neurona "j" del nivel "i".

$\delta_{p(i+1)jk}$ = Error de la neurona del nivel "i+1", la cual recibe el estímulo de la neurona del nivel "i", en donde se calcula el error oculto.

Se calculará el error oculto de la neurona 1 del nivel 2 de la figura 22 para ejemplificar.

Para calcular el error de una neurona oculta es necesario saber con que porcentaje de error contribuye en las neuronas a que

envía información. Para ésto, se suman los productos de los pesos de las conexiones con los errores de las neuronas a que se conecta y finalmente se multiplica por la derivada. Desarrollando la fórmula para el error oculto nos queda de la siguiente manera:

$$\delta_{p20} = f'(Est_{pij})(\delta_{p30} * W_{200} + \delta_{p31} * W_{201} + \delta_{p32} * W_{202})$$

Terminando con el cálculo del error oculto de la neurona se ajustan los pesos con la formula siguiente:

$$W_{ijk} = W_{ijk} + \eta\delta_{p(i+1)k}O_{pij}$$ En donde:

$\delta_{p(i+1)k}$ = Error de la neurona que recibe el estímulo O_{pij}

O_{pij} = Estímulo que envía la neurona en donde se calcula el error oculto.

η = Rapidez de aprendizaje.

En el procedimiento de aprendizaje se requiere que los cambios sean proporcionales a los errores con respecto a los pesos. El algoritmo de gradiente descendiente necesita una serie infinita de pasos para llegar a un mínimo. La rapidez de aprendizaje nos permite disminuir la cantidad de pasos. Para valores grandes para la rapidez de aprendizaje la red en algunas ocasiones oscila, por lo que es necesario disminuir la constante en caso de que ocurra. Con el concepto de *momentom* se puede aumentar el valor de la rapidez de aprendizaje e incrementar la velocidad de aprendizaje en la red.

Para ajustar los pesos de la neurona 1 del nivel 2, de la figura 22, las fórmulas quedan como se presentan a continuación:

$$W_{210} = W_{210} + \eta\delta_{p30}O_{p20}$$

$$W_{211} = W_{211} + \eta\delta_{p31}O_{p20}$$

$$W_{212} = W_{212} + \eta\delta_{p32}O_{p20}$$

Terminado de ajustar los pesos de las conexiones de las neuronas de un nivel, se retrocede para tomar otro, hasta llegar al de salida.

Paso 6. Regresa al paso 2 para presentar un nuevo patrón.

Cuando el patrón ha sido propagado y ajustado los pesos, se presenta el siguiente para que la red lo aprenda. Se pueden entrenar todos los patrones secuencialmente o de manera aleatoria.

ERROR GLOBAL

Con la finalidad de saber si la red esta aprendiendo durante el proceso de entrenamiento se calcula el error global. El procedimiento consiste en dos pasos:

a) Calcular el Error por Patrón.
b) Calcular el Error Global

a) Se propaga el patrón de entrada para poder calcular el error en las neuronas de salida. Con la fórmula que se presenta a continuación se calcula el promedio de los errores de las neuronas de salida, para obtener el error del patrón entrenado en ese instante.

$$E_p = \frac{1}{2} \sum_{j=0}^{n} (t_{pij} - O_{pij})^2$$

b) Generalmente en una red neuronal se entrenan cierta cantidad de patrones, por lo que es necesario calcular el promedio de los errores por patrón, para obtener un error global, utilizando la fórmula que aparece a continuación:

$$E_g = \sum_{n=1} E_p$$

En el proceso de entrenamiento, el error global debe ir disminuyendo en caso de que la red aprenda, de lo contrario se mantiene constante en un mínimo local u oscila.

MINIMOS LOCALES

Durante el entrenamiento de la red, el error global sigue una trayectoria que depende de la configuración inicial de los pesos de las conexiones. Esta trayectoria, algunas ocasiones encuentra puntos (mínimos locales) en donde no puede continuar minimizando el error global y necesita un número muy grande de iteraciones para abandonarlos. En la figura 23 se muestra una posible trayectoria en la minimización del error global, los puntos marcados con una cruz son probables mínimos locales.

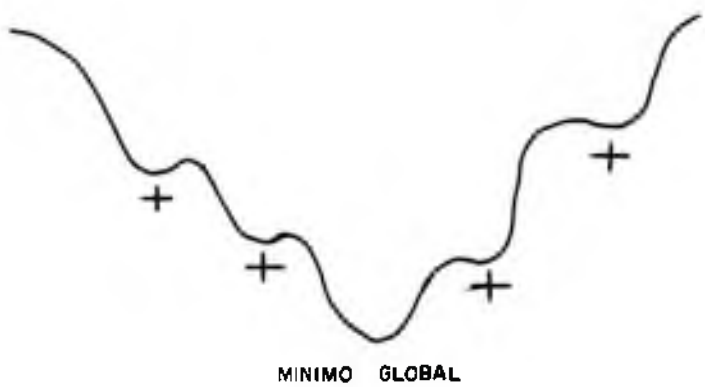


FIG. 23 TRAYECTORIA DEL ERROR GLOBAL CON ALGUNOS MINIMOS LOCALES

UMBRAL DE CONVERGENCIA

Para que el reconocimiento de los patrones entrenados sea preciso, es necesario minimizar el error global hasta cierto valor, el cual, recibe el nombre de *umbral de convergencia*. Para redes con un número alto de neuronas de salida o con un gran número de patrones a entrenar se debe tener un umbral de convergencia (0.05), ya que el error global depende directamente de los dos factores antes mencionados.

2. CARACTERISTICAS DEL TRANSPUTER E IMPLEMENTACION DE REDJAN

2.1 MAQUINAS PARALELAS

Para simular el comportamiento de un avión en vuelo, el de un vehículo en colisión, para interpretar imágenes complejas, diseñar elaborados circuitos integrados o estudiar el comportamiento de una molécula o el de una red neuronal, hacen falta computadoras que tengan una gran potencia de cálculo. La arquitectura de las supercomputadoras de hoy, ejecutan secuencialmente las instrucciones, sin embargo muchos modelos son capaces ya de efectuar algunas operaciones simultáneamente a gran velocidad debido a sus unidades especializadas. La potencia de estas máquinas, que suelen ser muy caras, aumentará todavía, pero no indefinidamente. Por ello se desea construir computadoras capaces de efectuar un gran número de operaciones a la vez, en paralelo, en base a una gran cantidad de datos. Este nuevo tipo de máquinas paralelas o masivamente paralelas, con un número de procesadores fácilmente extendible y con unidades de cálculo homógeneas, han de permitir la obtención de prestaciones equivalentes e incluso superiores a las computadoras clásicas y a precios más bajos.

Los procesadores de las máquinas paralelas pueden ser más lentos que en las secuenciales, ya que la velocidad se suple usando varios procesadores. Esta característica permite que sean baratos por ser menos complejos y que se produzcan en grandes series. Además las arquitecturas paralelas presentan la ventaja de estar diseñadas para responder mejor a una amplia gama de necesidades.

Todavía es necesario hacer un gran esfuerzo de investigación y de desarrollo para adquirir un perfecto dominio de estas máquinas y aumentar el conocimiento práctico necesario para su empleo. Porque ponerlas en funcionamiento es complejo. No se trata de conectar entre si muchos procesadores, ya que con mil se necesitan un millón de cables, lo cual no es realizable aún; es indispensable hacer un estudio para definir la topología,

sincronización y que intercambien información eficazmente. Muchos laboratorios buscan la topología que minimizaría el número de interconexiones. Sin embargo el problema no es sólo de arquitectura, lo es también de "software", ya que los lenguajes de programación que habitualmente se emplean se han creado para máquinas secuenciales.

Los microprocesadores, unidades de cálculo en un solo chip, pesentan la ventaja de poder fabricarse en grandes cantidades y a bajo precio. Desde hace mucho tiempo se desea encontrar un componente que sería a las máquinas masivamente paralelas, lo que es el transistor a los componentes electrónicos, una pieza fundamental, que reuniera las condiciones esenciales para integrarse a una arquitectura paralela. La empresa británica **INMOS** en 1979, con un pequeño grupo de investigadores inicia el estudio de un componente perteneciente a un nuevo género, destinado a trabajar en grupo, es decir, en red. Este programa dio lugar en 1985 al primer **Transputer** (La primer versión fue un chip de 45 mm^2, se presentó a finales de 1983). El Transputer se ha convertido en un nombre genérico que abarca una familia completa de componentes integrados programables, utilizados como procesadores independientes, pero sobre todo ofrecen la posibilidad de interconectarse en un número importante, permitiendo una realización eficaz de sistemas paralelos y a bajo costo. Por lo anterior se considera al Transputer como una pieza fundamental de las máquinas paralelas. Cada Transputer puede reunir en un solo chip varias unidades de cálculo, memoria y varios enlaces de interconexión rápidos, que permiten intercabiar datos con otros Transputer o con otros componentes a su alrededor.

2.2 HARDWARE DEL TRANSPUTER T800

El Transputer [36,37,38,39] es descrito como un procesador **RISC** (Reduced Instruction Set Computer) [40], los cuales se caracterizan por tener un número pequeño de instrucciones y trabajar con "pipeline". Aunque, también cuentan con un número de

instrucciones concernientes al "scheduling" y al intercambio de mensajes. Un programa concurrente puede correr en un sólo Transputer, el cual maneja la concurrencia por medio del "hardware", sin intervención de "software" (un kernel), la comunicación entre procesos no es complicada. El mismo programa puede correr en varios procesadores y el intercambio de mensajes y la sincronización a través del "hardware". La comunicación entre Transputer es por medio de conexiones o enlaces, cada Transputer tiene 4 enlaces, los cuales se conectan entre ellos o algún otro dispositivo.

Los Transputer más conocidos son los **T212, T414** y **T800**. El T212 es un procesador de 16 bits, los otros dos son de 32 bits, para el T800 se tienen dos versiones, el T800-20 que calcula 1.5 **Mflos** (Millones de Operaciones de Punto Flotante por Segundo) y el T800-30 2.5 Mflops. Aunque, el T800 presenta una potencia de cálculo equivalente a la unidad central de una minicomputadora **VAX 86000** de Digital Equitment.

Por ser el Transputer una máquina que trabaja en paralelo y realizar cálculos de punto flotante a una velocidad muy alta, es posible implementar en él un programa para la simulación de redes neuronales. Se cuenta con un Transputer T800, por eso se explicará su hardware a mayor detalle.

Los Transputer se utilizan en aplicaciones, tales como, la simulación, control de robot, síntesis de imagen y procesamineto de señales. El IMS T800 puede incrementar la eficiencia para tales sistemas por tener internamente un coprocesador de punto flotante, permitiendo realizar operaciones a grandes velocidades. La unidad diseñada para punto flotante hace extensivo el uso de técnicas formales que asegura que cada operación produzca un resultado correcto como se especifica por la ANSI.IEEE Standard 754-1985 para aritmética binaria de punto flotante.

El IMS T800 contiene internamente la unidad de punto flotante y es tan sólo un 20% más grande que T414. El procesador de punto

flotante WTL1167 para el 80386 de Intel requiere de 3 chips.

El T800 direcciona 4 Gigabytes de memoria, la frecuencia de reloj es de 20 MHz, ejecuta 10 MIPS (Millones de Instrucciones Por Segundo), tiene 4 Mbytes de memoria externa y 4 KBytes de Memoria Interna. Para transferir datos utiliza el DMA con un costo de software mínimo, los enlaces pueden transmitir en ambas direcciones y todos lo hacen simultáneamente. La figura 24 muestra un diagrama del Transputer con 4 Enlaces.

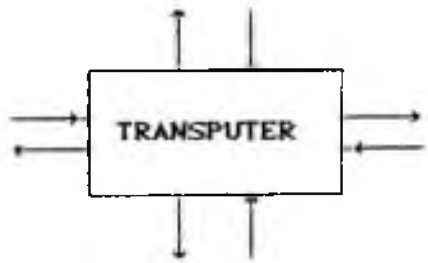


FIG. 24 ENLACES DE UN TRANSPUTER

Cada Transputer tiene 4 enlaces que le permiten dialogar fácilmente con 4 de sus homólogos. Resulta sencillo conectar una red de 4 Transputers, parcial o totalmente interconectados. Esta red se considera como una supercomputadora, que podría conectarse con otros transputer sucesivamente. La información circula a una frecuencia de 1.7 MegaBytes por segundo en el T800. La figura 25 muestra la red.

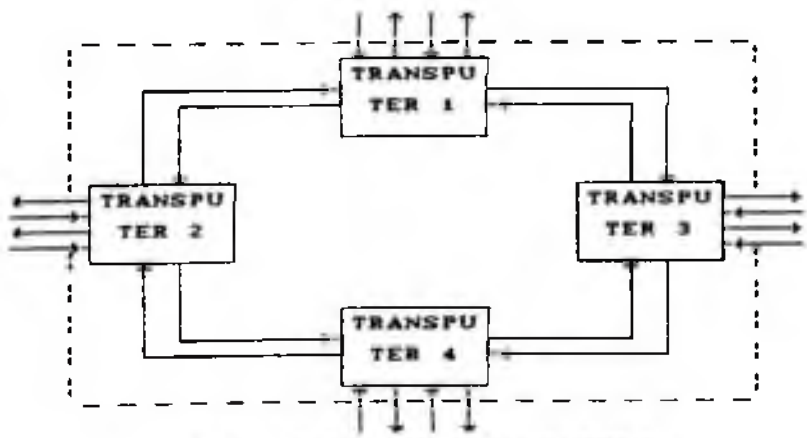


FIG. 25 RED DE TRANSPUTERS

La serie de transputer desarrollada por la firma británica INMOS esta elaborada sobre una misma arquitectura de base que se esquematiza en la figura 26. Se caracterizan por la presencia de una o dos unidades de cálculo (dos para el T800 como se observa en al figura C, la CPU y la FPU), de una memoria interna, de una unidad de gestión de señales externas (reloj, análisis, error, reset, etc.), de una interfaz de memoria externa y de cuatro enlaces que permiten a estos componentes dialogar simultáneamente con sus homólogos. Se ha diseñado para que pueda ejecutar sin dificultades programas escritos especialmente para arquitecturas paralelas. Estas arquitecturas consiguen reunir un número considerable de procesadores, los cuales colaboran en una misma tarea.

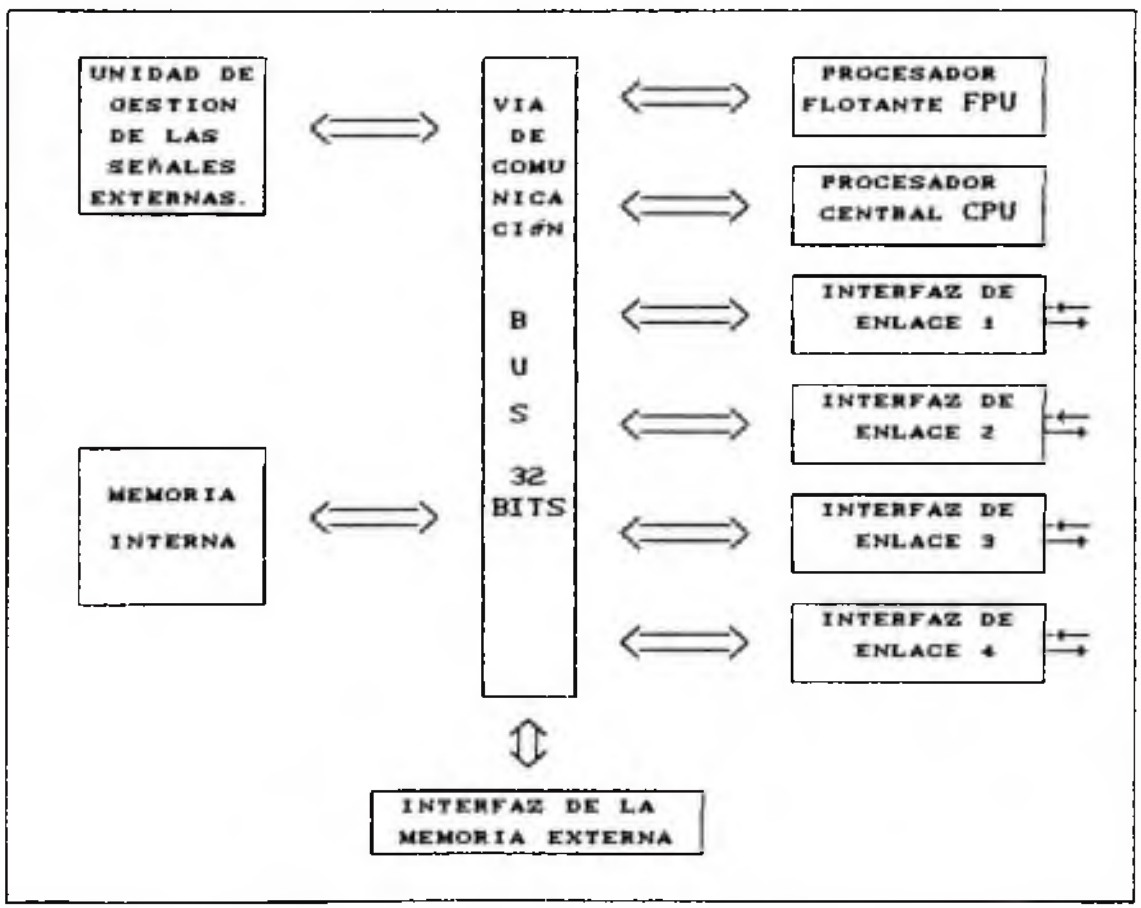


FIG. 26 ARQUITECTURA BASICA DE LA SERIE DE TRANSPUTERS

El Transputer es original porque permite que el procesador

central (CPU), el procesador de punto flotante (FPU) y las cuatro vías de comunicación funcionan simultáneamente. Esta característica permite que la CPU y FPU trabajen miéntras el Transputer envía o recibe información de sus homólogos. Evitando que la CPU pierda tiempo para controlar el resto de los componentes (FPU, enlaces, memoria externa, etc.). puesto que todos son autónomos. El Transputer también incrementa su velocidad de procesamiento de información por la memoria interna, que se encuentra dentro del chip, en donde la velocidad de acceso a datos e instrucciones es mayor, se puede considerar como una memoria cache, su función principal es mantener bloques de instrucciones cerca de la CPU para evitar el acceso a la memoria externa por cada instrucción a ejecutar, en los bloques contienen instrucciones que se usan con mayor frecuencia. En procesos concurrentes los cambios de contexto son muy rapidos, incrementando la velocidad de procesamiento.

2.3 LENGUAJE DE PROGRAMACION OCCAM2

El diseño de un componente destinado a integrarse a una arquitectura paralela sería incompleto sin un lenguaje adecuado de programación. Occam2 [41,42,43,44] se trata de un lenguaje formalmente definido antes de diseñar el hardware del Transputer. El Transputer se emplea como procesador independiente, pero es interesante en el momento en que se utiliza como red.

Occam2 es un lenguaje de programación paralela de alto nivel, surgido del modelo de programación paralela CSP (Communicating Secuential Processes), desarrollano inicialmente en la Universidad de Oxford por el equipo de C.A.R. Hoare. Presenta una notable simplicidad sintáctica y un formalismo para la construcción correcta de sistemas paralelos. Occam2 deja expresar un programa como una colección de procesos concurrentes que es posible ejecutarlos en un transputer o en varios en paralelos. Un proceso Occam2 es una entidad que intercambia informaciones con su entorno, formado por otras entidades del mismo tipo, mediante

mensajes enviados a través de los canales, los cuales constituyen verdaderos contactos unidirecionales que enlazan los procesos. Este protocolo básico de comunicación se establece a nivel de hardware con los enlaces de los Transputer.

Las primitivas o procesos del lenguaje de programación Occam2 son de alto nivel. Para la comunicación entre pocesos se utiliza el simbolo ? para entrada y ! para la salida, por ejemplo:

```
ch ! e      salida
ch ? v      entrada
```

El constructor **SEQ** permite ejecutar una serie de procesos secuencialmente, por ejemplo:

```
SEQ
  A := 4
  B := A + 42
  out ! B
```

El constructor **PAR** ejecuta una serie de procesos en paralelo, por ejemplo:

```
PAR
  A := 4
  B := A + 42
  out ! B
```

Para ejecutar una secuencia hasta que se cumpla una condición se usa el constructor **WHILE**, por ejemplo:

```
WHILE A < 1024
  in ? A
  A := A * A
  Out ! A
```

Para realizar un número especifico de iteraciones se usa el constructor **Var i = 0 FOR N**, en donde **Var** puede ser sustituido por los constructores **SEQ, PAR, IF, ALT**. Este constructor evalua en paralelo las neuronas de la red, o hacer funcionar varios procesos a la vez.

```
PAR
  PAR P = 0 FOR  N
    WHILE TRUE
      SEQ
        C[P] ? X
        C[P+1] ! X
```

```
PAR I = 1  FOR  3
  PAR
    P(I)
    Q(I)
```

Para asignar procesos a diferentes Transputer se emplea el constructor **PLACED PAR**, por ejemplo:

```
PLACED PAR
  PROCESSOR 1
    P1
  PROCESOR 2
    P2
```

2.4 INSTALACION DEL TRANSPUTER

El Transputer se encuentra instalado dentro de una computadora PC/AT SPERRY, la cual sirve como nodriza. La comunicación hacia la nodriza se lleva a cabo a través de un conjunto de puertos y hacia el transputer por dos canales, uno para enviar y el otro para recibir información. La figura 27 muestra lo mencionado anteriormente.

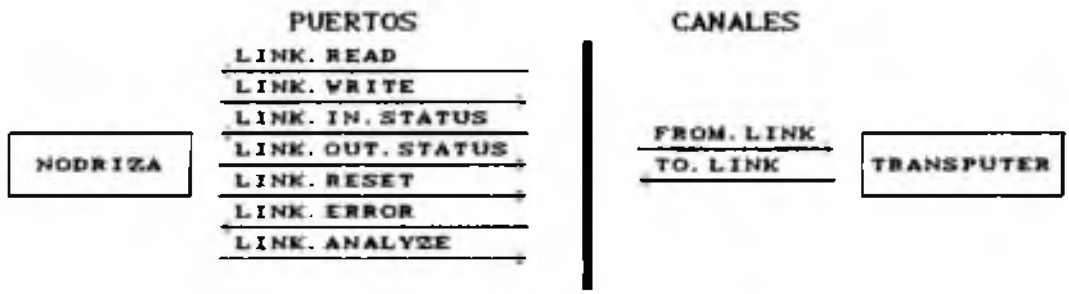


FIG. 27 ENLACE ENTRE EL TRANSPUTER Y LA NODRIZA

PUERTOS DE COMUNICACION

A continuación se describirá cada una de las funciones de los puertos empleados para la comunicación del transputer hacia la nodriza.

LINK.READ. Por este puerto la nodriza recibe los datos que envía el Transputer.

LINK.WRITE. Por medio de este puerto manda datos la nodriza al Transputer.

LINK.IN.STATUS. Es empleado por el Transputer para indicar a la nodriza que espera un mensaje. Además es usado para habilitar

operaciones de Acceso Directo a Memoria.

LINK.OUT.STATUS. A través de él indica el Transputer a la nodriza que tiene un dato para ella. También es usado para Acceso Directo a Memoria.
LINK.RESET. Es empleado por la nodriza para inicializar al Transputer.
LINK.ERROR. En este puerto se señala si ha sucedido un error en el sistema del Transputer.
LINK.ANALYZE. Es usado junto con LINK.RESET para inicializar el Transputer, pero preservando el estado del procesador del mismo.

CANALES DE COMUNICACION

Son canales lógicos asociados con canales físicos del Transputer, los cuales son empleados para comunicarse con la máquina nodriza, a continuación se describen.

FROM.LINK. Este canal se emplea por el Transputer para recibir todos los datos que envíe la máquina nodriza.
TO.LINK. Se usa para que el Transputer mande información a la nodriza.

Para la comunicación de las dos máquinas se cuenta con varios programas para definir y manejar los protocolos necesarios. Existe un programa servidor que corre en la nodriza, el cual se encarga de recibir e interpretar la información que transmite el Transputer, el programa se llama AFSERVER. Mientras que en el Transputer corre el programa llamado HARNESS, el cual se encarga de llamar a la rutina del usuario para crear un nuevo proceso. La función del programa FILTER es entablar la comunicación con la nodriza, como se puede apreciar en la figura 28. El programa HARNESS nos permite correr en paralelo el programa de usuario y el FILTER.

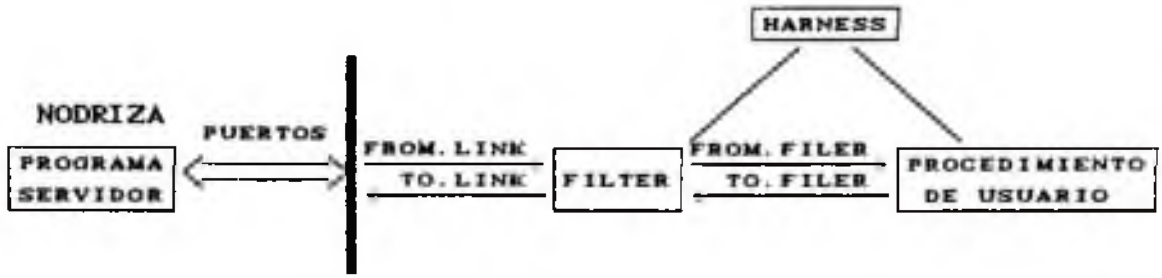


FIG. 28 PROCESOS DE ENLACE ENTRE TRANSPUTER Y NODRIZA.

El sistema de comunicación esta divido en tres niveles. El primero, el de más bajo nivel, se basa en un protocolo básico; por él es manejado directamente los procesos FILTER Y AFserver. El segundo, consiste en un conjunto de primitivas que sirven para definir el protocolo del tercer nivel.

2.5 IMPLEMENTACION DE REDJAN

El lenguaje de programación Occam2 no trabaja con apuntadores y estructuras, como lo hacen los lenguajes Turbo C Y Pascal. Por lo que es necesario usar un conjunto de arreglos para la implementación de REDJAN.

Se explicarán las variables estáticas y dinámicas, arreglos y funciones más importantes para el entendimiento del programa.

A las variables estáticas se les asigna un valor constante, permiten hacer cambios con sencillez dentro del programa, se usan para dar las dimensiones en los arreglos. Las variables de este tipo y su máxima capacidad se presentan a continuación:

Max. Neu. Ent : = 100. Neuronas del nivel de entrada.
Max. Neu. Sal : = 100. Neuronas del nivel de salida.
Max. Niv. Ocu : = 5. Entre el nivel de entrada y de salida tiene a lo más 5 niveles.
Max. Conex. Neu : = 100. Conexiones de entrada o de salida.
Max. Neu. Ocu : = 100. Cantidad de neuronas de los niveles ocultos.

Max.Num.Pat : = 100. Para el entrenamiento de la red se pueden tener 100 pares de patrones (entrada y salida deseada).

Una vez que el programa es ejecutado debemos definir el número de neuronas de entrada, de niveles ocultos, la cantidad de neuronas de los niveles ocultos, cuantas neuronas de salida y el porcentaje de aprendizaje, para formar la estructura de la red.

Para lo anterior se ocupan las siguientes variables dinámicas.

No.Neu.Ent. Número de neuronas de entrada.
No.Neu.Sal. Número de neuronas de salida.
No.Niv.Ocu. Número de niveles ocultos.
No.Conex.Sal. Número de conexiones de entrada en una neurona.
Porc. Aprend. Rapidez de aprendizaje.
No.Pat. Número de pares de patrones a entrenar.
Er.Pat. Error por patrón.
Er.Global. Error promedio de todos los patrones entrenados.

A las siguientes variables se les asignará un valor al ejecutarse el programa, pueden ser cambiados por medio de una de las opciones del menú.

Umbral.Converg : = 0.0005. La variable nos indica con que precisión necesitamos que se ajusten los pesos de la red, el entrenamiento se detiene en el momento que se alcanza un valor menor al de umbral de convergencia.

UNO : = 0.9
CERO : = 0.1

Las variables UNO y CERO permiten experimentar con valores simétricos entre 0 y 1, por ejemplo 0.8 para UNO y 0.2 para CERO. Si graficamos la función de propagación sus salidas se encuentran entre 0 y 1, pero alcanzar los extremos es difícil, razón por lo que se recomienda asignar valores entre rango de 0.9 y 0.1. Si usamos valores de 1 para UNO y 0 para CERO el aprendizaje de la

red es más lento por tratar de llegar al 1 y 0 en las salidas.

Los arreglos y funciones para formar la estructura de la red neuronal se explican a continuación, el valor de las variables de las dimensiones se proporcionaron anteriormente.

[Max. Niv. Ocu+1][Max. Neu. Ent][Max. Conex. Neu] REAL32 PES. RED. NEU. En este arreglo se almacenan los pesos de las conexiones de las neuronas. Los pesos se guardan en la neurona que envía los estímulos, para ejemplificar utilizaremos la neurona 0 (cero) del nivel 1 de la figura 29, la cual tiene 2 conexiones de salida y por lo tanto 2 pesos. Los elementos del arreglo quedan de la siguiente manera:

a) PES. RED. NEU[1][0][1] := 0.028
b) PES. RED. NEU[1][0][2] := -0.73

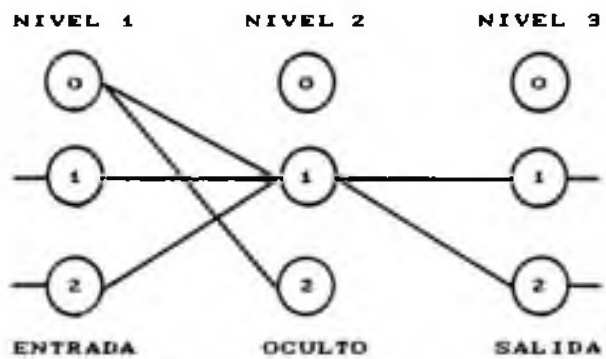


FIG. 29 ESTRUCTURA PARA MOSTRAR LA CANTIDAD DE CONEXIONES DE SALIDA, DE ENTRADA Y SUS RESPECTIVOS PESOS

[Max. Neu. Ent][Max. Conex. Neu] INT No. Conex. Sal. Se usa para controlar la cantidad de conexiones de salida de cada neurona. Para la neurona 0 (cero) nivel 1 de la figura 29 el arreglo queda como sigue:
No. Conex. Sal[1][0] := 2

[Max. Neu. Ent][Max. Conex. Neu] INT No. Conex. Ent. Almacena la cantidad de conexiones de entrada de cada neurona. Para la neurona 1 del nivel 2 de la figura 29, al arreglo contiene:

No.Conex.Ent[2][1] := 3

[Max.Niv.Ocu+1][Max.Neu.Ent][Max.Conex.Neu] INT Dir.Pes.Conex. En este arreglo se almacena la dirección de los pesos de las conexiones de entrada a una neurona. La dirección se compone por el nivel (un digito), la neurona (dos digitos) y la conexión (dos digitos). La neurona 1 del nivel 2 de la figura 29, tiene 3 conexiones de entrada, que vienen del nivel 1 y de la neurona 0, 1 y 2. La dirección para la primer conexión es **10000**, en donde **1** es para el nivel, los primeros **00** para la neurona y los últimos **00** para el número de la conexión. El contenido de los tres elementos del arreglo quedan como sigue:

a) Dir.Pes.Conex[2][0][0] := **10000**
b) Dir.Pes.Conex[2][0][1] := **10100**
c) Dir Pes.Conex[2][0][2] := **10200**

En el inciso a), los subíndices del arreglo Dir.Pes.Conex nos dicen que se trata del nivel 2, neurona 1 y la conexión 0 (cero) de entrada. La dirección del peso de la conexión es **10000**, el cual se convierte en los subíndices del arreglo PES.RED.NEU[1][0][0] y su contenido es 0.28.

[Max.Niv.Ocu+2] INT No.Neu.Ocu. Sirve para almacenar la cantidad de neuronas de cada nivel.

Crea.Neuronas(). Al inicio llama a la función **Limp.Red()** para limpiar todos los arreglos que se usarán. Después nos pregunta la cantidad de neuronas de entrada, número de niveles ocultos, cantidad de neuronas de cada uno, número de neuronas de salida y finalmente la rapidez de aprendizaje.

Crea.Conex(). Una vez que se asigna la cantidad de neuronas en cada nivel, con esta función se crean las conexiones entre las neuronas. Al terminar las conexiones llama la función **Inic.Pes.Red()** para asignarle un peso a las conexiones aleatoriamente entre -0.99 y 0.99.

Salva.Arch(). La función permite salvar la estructura en disco magnético. Lo puede hacer de manera compacta con la función **Salva.Comp(INT file.id)** y en forma de texto con la función **Salva.Txt(INT file.id)**, estás funciones son llamadas por medio de un submenú.

Carga.Arch(). Con esta función podemos cargar la estructura de disco magnético. Es posible hacerlo de manera compacta con la función **Carga.Comp(INT file.id)** y en forma de texto con la función **Carga.Txt(INT file.id)**, estás funciones son llamadas por medio de un submenú.

Para el algoritmo de aprendizaje los arreglos y funciones son las siguientes:

[Max.Niv.Ocu+2][Max.Neu.Ent] REAL32 Deltas.Red. En este arreglo se almacenan los errores de las neuronas Ocultas y de las de salida.

[Max.Niv.Ocu+2][Max.Neu.Ent] REAL32 Pat.Est. Contiene los estímulos de salida de cada neurona.

[Max.Num.Pat][Max.Neu.Ent] REAL32 Pat.Apren.Ent. Guarda los patrones de entrda de la red.

[Max.Num.Pat][Max.Neu.Ent] REAL32 Pat.Apren.Sal. Almacena los patrones de las salidas deseadas.

Aprend.Red(). Tiene todas las funciones para el aprendizaje de la red neuronal. Al entrar a la opción de aprendizaje en el menú, pregunta cuantas veces se repetirá su llamado para el entrenamiento de la red. Para la primer fase, se llama la función **Cambia.Pat()** para cambiar los patrones de entrada y de salida deseada. Propagamos el patrón de entrada por la red con la función

Propag.Red(Est.Neu.Niv) y con la función **Error.Sal()** calculamos los errores de salida. La fase de retroceso se divide en dos partes, en la primera con la función **Errores.Ocu(Est.Neu.Niv)**

calculamos todos los errores de las neuronas ocultas y en la segunda con la función **Cambia.Pes(Est.Neu.Niv)** ajustamos los pesos de las conexiones. El parametro **Est.Neu.Niv[Max.Niv.Ocu+2]** contiene la cantidad de neuronas de cada uno de los niveles.

Error.Pat(). Durante el proceso de entrenamiento se van calculando los errores globales para checar si la red aprende o no. En caso de haber pedido un número muy grande iteraciones y que la red alcance el umbral de convergencia, el entrenamiento se detiene y nos dice cuantas iteraciones se llevaron acabo. Con esta función calculamos el error por patrón y se calcula un promedio para obtener el error global.

Rec.Pat(). Una vez que termina el entrenamiento, es necesario checar si los patrones han sido aprendidos, la función nos pide el patrón de entrada para propagarlo y entregarnos una salida. Se considera UNO, si es mayor de 0.8 y CERO si es menor de 0.2.

Carga.Pat(). Si la red tiene una cantidad grande de patrones a entrenar y el número de neuronas de entrada y de salida es alto, será demasiado tedioso teclear todos los patrones de entrenamiento. Con esta función se lee de un archivo con todos los patrones a entrenar

Todos.Pat.Aprend(). Esta función carga todos los patrones por medio del teclado.

Un.Pat.Aprend() En algunas ocasiones la red no aprende un patrón con la misma rapidez que los demás. La función permite cargarlo y decirle cuantas veces repita su entrenamiento.

Desp.Pes(). Sirve para desplegar los pesos de las conexiones y poder apreciar los cambios durante el entrenamiento, cuenta con dos opciones. La primera, despliega en forma numérica los pesos de las conexiones con la función **Desp.Num()** y la segunda los despliega por medio de colores con la función **Desp.Graf()**. en donde cada color representa un valor.

Negro. Para valores entre el rango de (-1, -10) y (1,10).
Rojo. Para valores entre negativos.
Verde. Para valores entre (0.0,0.2).
Azul. Para valores entre (0.21,0.4).
Cyan. Para valores entre (0.41,06).
Cafe. Para valores entre (0.61.0.8).
Gris. Para valores entre (0.81,1.0).

Si el número es mayor de 1 o menor de -1 se antepone el color negro para indicar que se debe multiplicar por diez cualquier valor que se presente a continuación de él; por ejemplo, para el 4 se mostrará un color negro y después un Cyan. En cualquier caso negativo se utiliza el rojo para denotarlo; por ejemplo; -0.25 se mostrarán los colores rojo y azul, para -6.2 se despliega el negro indicando que se tiene que multiplicar por 10, el rojo pora decirnos que es negativo y finalmente el cafe. Para redes chicas no se aprecia la ayuda que nos puede brindar, pero en redes grandes es muy útil, ya que es difícil analizar todos los pesos y es mejor tener una visión global de los pesos por medio de los colores.

Para simular los umbrales se usan los pesos de las conexiones de salida de las neuronas de la fila 0 (cero). Estas como ya se mencionó no tiene estimulo de entrada y su salida siempre será 1. Por lo que es necesario al editar la red aumentar en una el número de neuronas de cada nivel como lo muestra la figura 30.

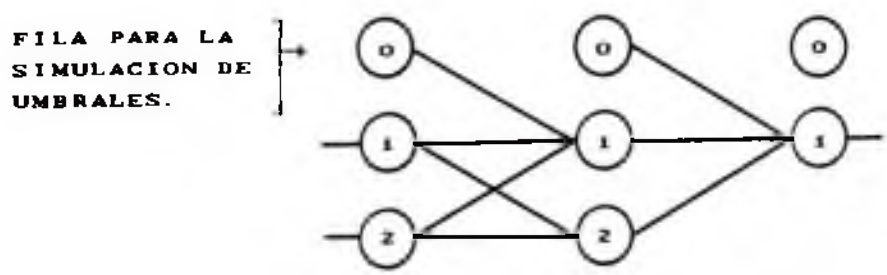


FIG. 30 ESTRUCTURA PARA LA FUNCION XOR SIMULANDO LOS UMBRALES

3. EXPERIMENTOS REALIZADOS

Uno de los propósitos de este trabajo de tesis es realizar aplicaciones una vez que se ha implementado el programa. Se presentará un conjunto de ejemplos para demostrar que funciona correctamente la implementación de la estructura y el algoritmo de aprendizaje de la red. Se experimentará cambiando algunos parametros básicos.

3.1 FUNCION XOR

La función XOR es una aplicación sencilla para redes neuronales, se experimentó con ella quitando los umbrales a las neuronas, con umbrales, con diferentes configuraciones de pesos iniciales en las conexiones y tomando los extremos de las variables UNO = 1.0 y CERO = 0.0.

FUNCION XOR SIN UMBRALES

Si una neurona tiene tres conexiones de entrada y un umbral con valores determinados, el peso de las conexiones pueden modificarse dividiendolos entre el valor del umbral y reconocer los patrones ya entrenados. Lo anterior se probó manualmente en una red ya entrenada, reconociendo los patrones con una precisión menor que con umbrales. Por lo que aparentemente es igual trabajar con o sin umbrales. Para calcular el estímulo de entrada de una neurona se considera $\theta_{pij} = 0$. Durante el proceso de entrenamiento la red aprende algunos patrones en el mejor de los casos, si se continua entrenando aprende los que hacian falta y pierde el conocimiento de los ya aprendidos.

Utilizando 4 configuraciones iniciales diferentes para los pesos de las conexiones se entrenó la red, obteniendose malos resultados. Las subsecuente gráficas son de la red que más patrones aprendio. La figura 31, muestra los pesos iniciales; la 32, los finales; la 33, la velocidad de convergencia de la red y

la 34, el número de patrones aprendidos después de cierta cantidad de iteraciones.

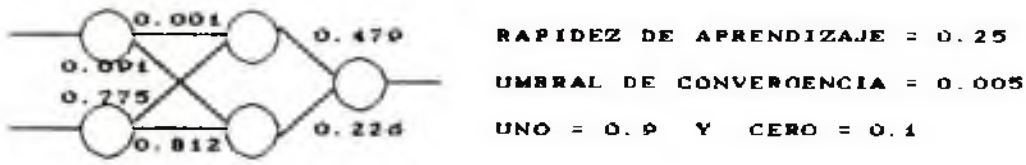


FIG. 31 XOR CON PESOS INICIALES

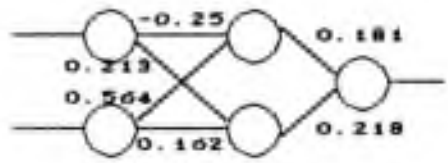


FIG. 32 XOR CON LOS PESOS FINALES

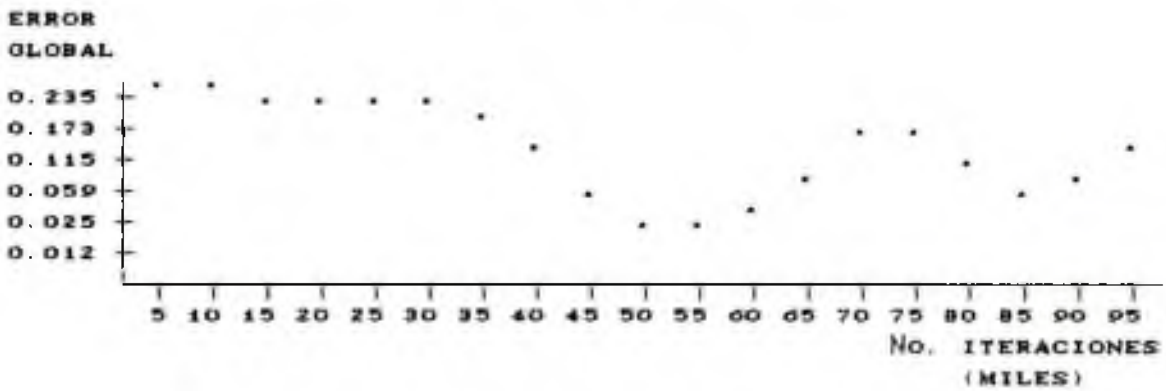


FIG. 33 GRAFICA DE RAPIDEZ DE CONVERGENCIA

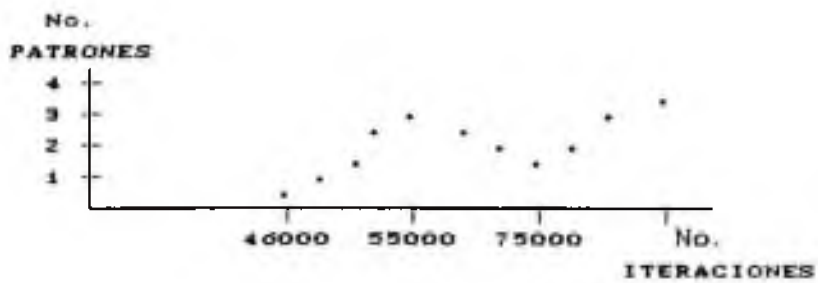


FIG. 34 GRAFICA DE RECONOCIMIENTO DE PATRONES

Los pesos de las conexiones en figura 32 es la mejor

configuración alcanzada, la cual reconoce sólo 3 casos. Para el patrón de entrada 0 0 y salida 0 da una respuesta errónea. Se llegó a estos pesos después de 50000 iteraciones, en 7.5 minutos. Parece obvio que si se continua el entrenamiento se encontrarian los pesos adecuados, pero pierde conocimiento de patrones que ya reconoce y aprende los que faltaban.

Para los otros 3 casos que se experimentó, sólo aprendieron un patrón con un número muy grande de barridos. Por lo que concluirmos que los umbrales en las neuronas son necesarios para el proceso de esnseñanza de la red.

FUNCION XOR CON UMBRAL

Se experimentó en redes con umbrales obteniéndose excelentes resultados. La figura 35, muestra la estructura para la función XOR con umbrales; la 36, la gráfica de la velocidad de convergencia; la 37, el reconocimiento de patrones y en la 38 los pesos finales.

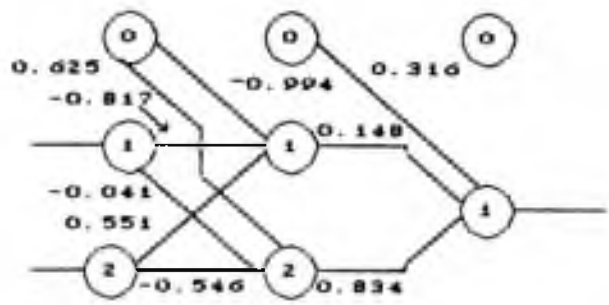


FIG. 35 ESTRUCTURA PARA LA FUNCION XOR CON LOS PESOS INICIALES

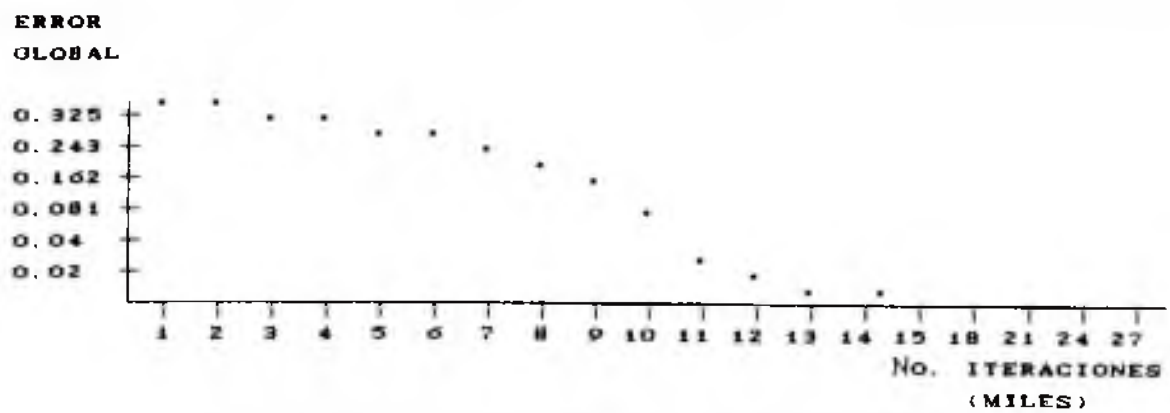


FIG. 36 GRAFICA DE RAPIDEZ DE CONVERGENCIA

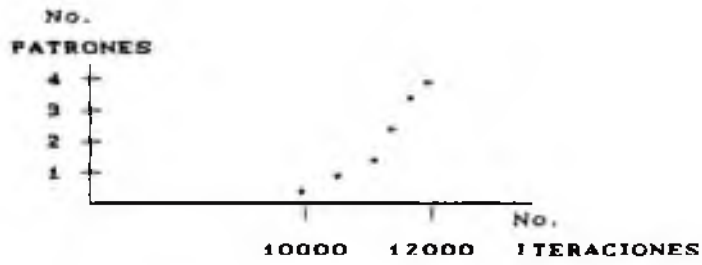


FIG. 37 GRAFICA DE RECONOCIMIENTO DE PATRONES

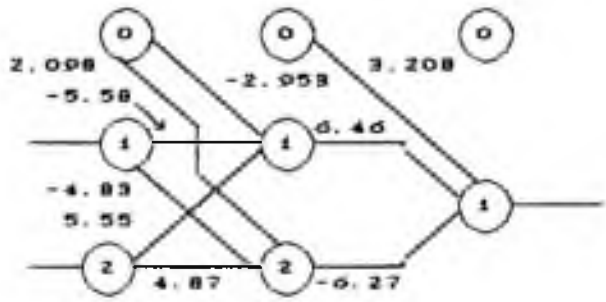


FIG. 38 ESTRUCTURA PARA LA FUNCION XOR CON LOS PESOS FINALES

El entrenamiento se llevo a cabo con los valores siguientes:

UNO = 0.9
CERO = 0.1
Umbral de Convergencia = 0.0005
Rapidez de Aprendizaje = 0.25
Convergió a las 19000 iteraciones.
Tiempo de aprendizaje: 17 Seg.

Con los umbrales la red aprende correctamente los patrones.

La *red converge* cuando el *error global* es menor que el *umbral de convergencia*. Esto implica que la red ya aprendio los patrones que se entrenan. Con el umbral de convergencia con un valor de 0.0005 el estimulo de salida de las neuronas del nivel de salida no se aproximan al valor de 0.9 para UNO y 0.1 para CERO. Por lo que es necesario disminuirlo a 0.00005, para poder seguir entrenando la red hasta que alcance un error global menor que 0.00005. Con esta condición en las neuronas de salida

prácticamente se alcanza el UNO y EL CERO. En caso que la red tenga un número mayor de neuronas, el umbral de convergencia debe disminuir para alcanzar mayor precisión en las salidas de la red.

Si se observa las gráficas 38 y 37, nos percatamos que llega un momento en que la red converge rápidamente y que el reconocimiento de los patrones se logra en un intervalo muy pequeño. En caso que no se aprendan algunos patrones y el error global no se decrememte es probable que nos encontremos en un mínimo local, no obstante, es difícil que se presente. Los pesos de la figura 38 se alcanzaron a las 26000 iteraciones.

FUNCION XOR CON DIFERENTES PESOS INICIALES

Para estimar la importancia de escoger una configuración de pesos iniciales adecuada para que la red aprenda con mayor rapidéz, se mostrará otro caso de la función XOR (Tomar como referencia las figuras 35, 36, 37, y 38). La red de la figura 39 tiene los pesos iniciales diferentes que en la 35, en la gráfica de la figura 40, se observa que la velocidad de convergencia es más lenta que en la 36; la figura 41, indica en que número de iteraciones reconoce los patrones y la 42, los pesos finales.

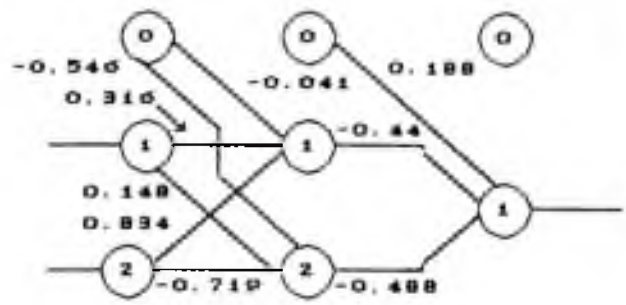


FIG. 39 ESTRUCTURA PARA LA FUNCION XOR CON LOS PESOS INICIALES

Con los umbrales la red aprende correctamente los 4 patrones.

UNO = 0.9

CERO = 0.1

Umbral de Convergencia = 0.0005

Rapidez de Aprendizaje = 0.25

Convergió a las 23000 iteraciones.

Tiempo de aprendizaje: 20 Seg.

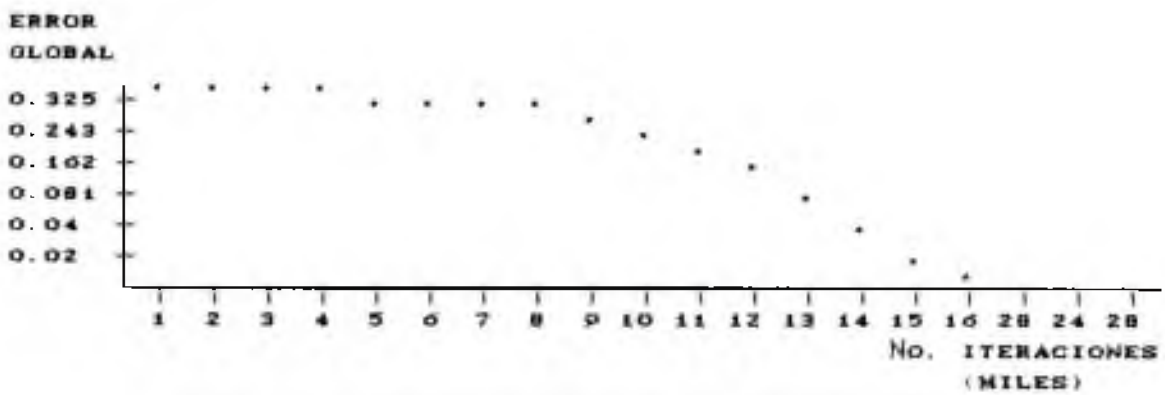


FIG. 40 GRAFICA DE RAPIDEZ DE CONVERGENCIA

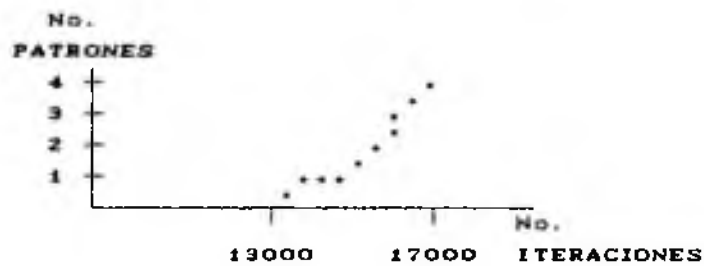


FIG. 41 GRAFICA DE RECONOCIMIENTO DE PATRONES

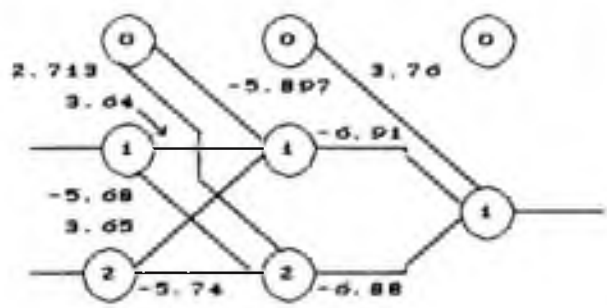


FIG. 42 ESTRUCTURA PARA LA FUNCION XOR CON LOS PESOS FINALES

En el primer ejemplo, la red converge más rápido debido a una mejor configuración de pesos. El principal problema en redes neuronales, es que el aprendizaje en redes grandes es muy lento,

por lo que actualmente se pretende por medio de algunos mecanismos buscar los pesos y asignarlos directamente a las conexiones.

FUNCION XOR CON UNO = 1.0 Y CERO = 0.0

Si entrenamos la red con los valores de **UNO**=1.0 y **CERO**=0.0 el aprendizaje es muy lento después de un tiempo, por lo que se necesita un número muy grande de iteraciones para poder aproximar las salidas de la red a los valores de UNO y CERO, además los pesos de las conexiones crecen demasiado al intetar llegar a los valores extremos de 1.0 y 0.0.

Para demostrar lo anterior se parte de los pesos iniciales de la red de la figura 35 para entrenar la red. Si analizamos la figura 43, nos damos cuenta que la red converge rápidamente en las primeras 9 mil iteraciones y después lo hace lentamente; la figura 44, exhibe en que número de iteraciones reconoce los 4 patrones entrenados y la 45, contiene los pesos finales de la red.

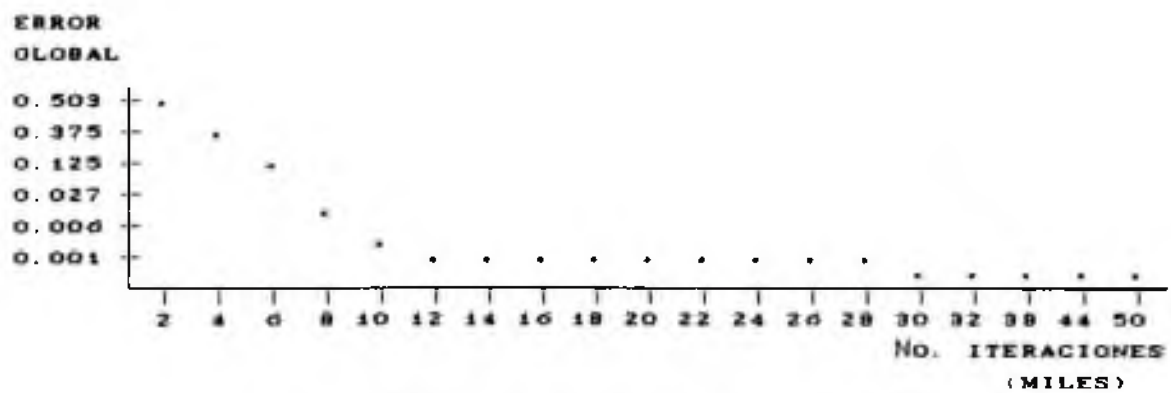


FIG. 43 GRAFICA DE RAPIDEZ DE CONVERGENCIA

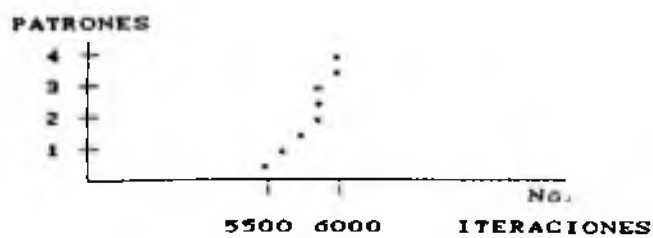


FIG. 44 GRAFICA DE RECONOCIMIENTO DE PATRONES

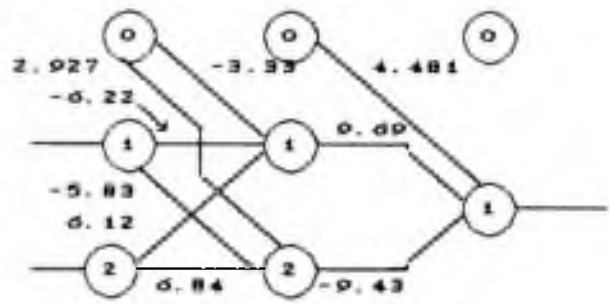


FIG. 45 ESTRUCTURA PARA LA FUNCION XOR CON LOS PESOS FINALES

Comparando los pesos de la figura 38 con la 45 se aprecian diferencias fuertes en los pesos las conexiones de las neuronas ocultas a las de salida, el crecimiento se debe al inteto de alcanzar el 0.0 y el 1.0 respectivamente.

Los valores con que se entrenó la red son los siguientes:

UNO = 1.0 y CERO = 0.0
Umbral de Convergencia = 0.0005
Rapidez de Aprendizaje = 0.25
Convergió a las 90000 iteraciones.
Tiempo de aprendizaje: 75 Seg.

3.2 RECONOCEDOR DE LINEAS CON DIFERENTE RAPIDEZ DE APRENDIZAJE

Para probar el efecto de la constante de la rapidez de aprendizaje se presenta un reconocedor de lineas verticales, horizontales, inclinadas a la derecha e inclinadas a la izquierda. Las lineas se representan por una matriz de 5x5 puntos, en donde una linea inclinada hacia cualquier sentido se forma por al menos de dos puntos, las verticales y horizontales por los 5 puntos de la matriz. La entrada de la red serán los 25 puntos de la matriz, por lo que se utilizan 25 neuronas de entrada, en la salida 2 neuronas para poder clasificar los 4 tipos de lineas y con un nivel oculto de 10 neuronas. La figura 46, presenta 4 matrices con una linea de cada tipo y sus respectivas salidas y entradas.

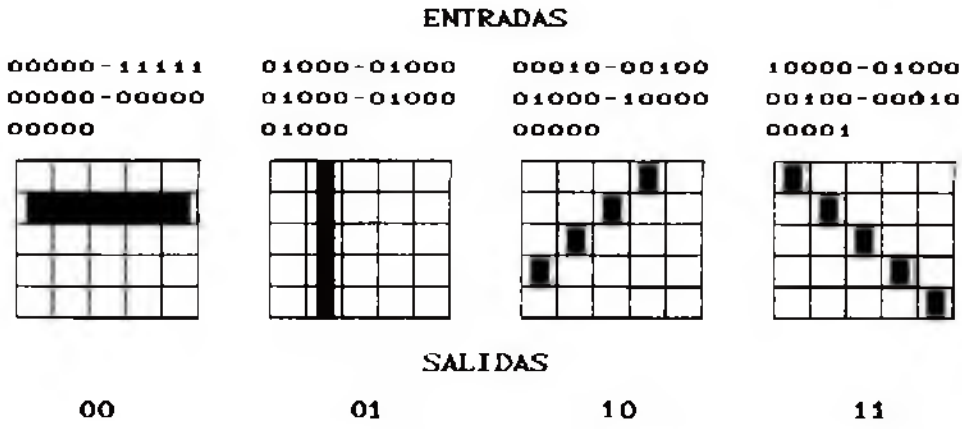


FIG. 46 MATRIZ DE PUNTOS PARA LAS LINEAS

En el entrenamiento de la red las lineas se presentan una vez cada una, en el orden que aparece en la figura 46, hasta terminar con los 24 patrones (5 horizontales, 5 verticales, 7 inclinadas a la izquierda y 7 inclinadas a la derecha).

El primer ejemplo, tiene una rapidez de aprendizaje (η) de 0.9. Aunque el valor es muy alto no se encontraron minimos locales ni oscilaciones. La figura 47, indica la rapidez de convergencia de la red y la 48, el número de patrones reconocidos después de cierta cantidad de iteraciones.

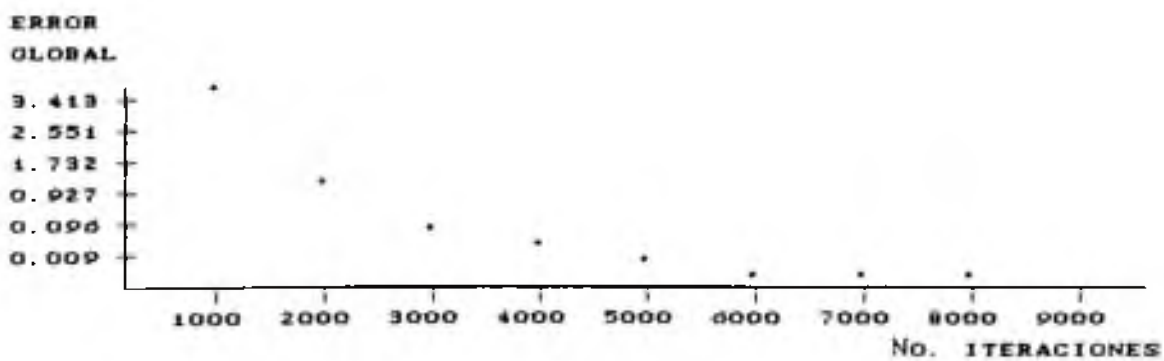


FIG. 47 GRAFICA DE RAPIDEZ DE CONVERGENCIA

La velocidad de ajuste de los pesos de las conexiones es muy alto gracias a que la constante de rapidez de aprendizaje es muy grande. (ver explicación en I.3)

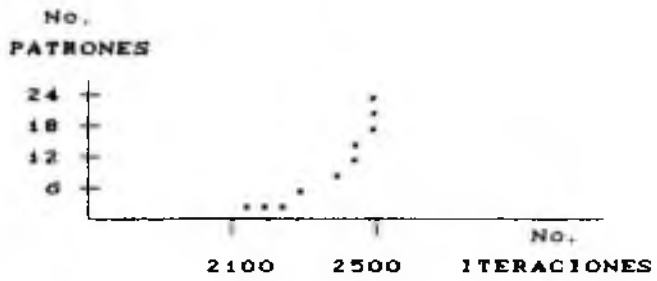


FIG. 48 GRAFICA DE RECONOCIMIENTO DE PATRONES

El entrenamiento se efectúo con los valores siguientes:

UNO = 1.0 y CERO = 0.0
Umbral de Convergencia = 0.0005
Rapidez de Aprendizaje = 0.9
Convergió a las 10600 iteraciones.
Tiempo de aprendizaje: 2 Minutos 38 Seg.

Para comprobar con claridad que la constante de rapidez de aprendizaje influye para que la red aprenda más rápido se presentará el siguiente ejemplo con un valor muy bajo, $\eta = 0.1$.

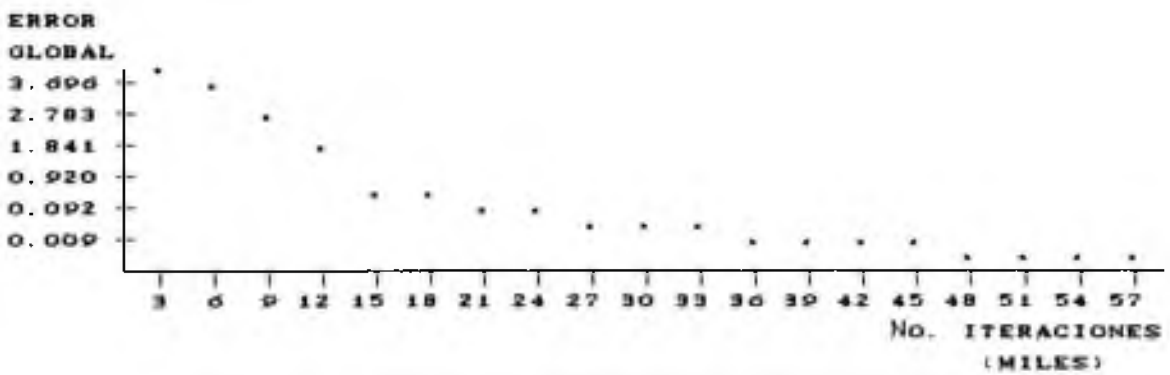


FIG. 49 GRAFICA DE RAPIDEZ DE CONVERGENCIA

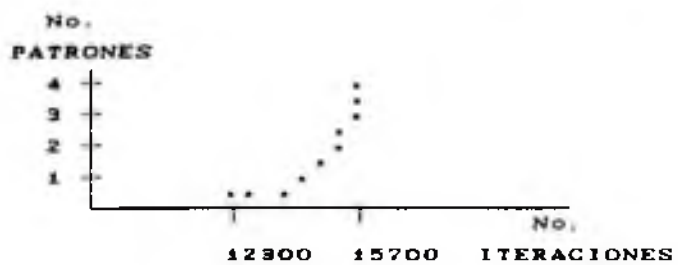


FIG. 50 GRAFICA DE RECONOCIMIENTO DE PATRONES

Los valores con que se entrenó la red son los siguientes:

UNO = 1.0 y CERO = 0.0
Umbral de Convergencia = 0.0005
Rapidez de Aprendizaje = 0.1
Convergió a las 71500 iteraciones.
Tiempo de aprendizaje: 17 Minutos 38 Seg.

En la gráfica de la figura 49 se puede apreciar que la velocidad de convergencia es muy lenta con respecto a la figura 47. Esto se debe a la diferencia del valor que se asigna a la constante de rapidez de aprendizaje en cada uno de los ejemplos.

3.3 RECONOCEDOR DE LA T EN SUS 4 POSICIONES VARIANDO LAS NEURONAS EN EL NIVEL OCULTO

Para comprobar la necesidad de las neuronas ocultas en la clasificación de patrones se entrenó una red para que reconoce la letra T en sus cuatro posiciones. Para el primer ejemplo la estructura tendra 9 neuronas de entrada (la matriz consiste de 9 elementos), 9 ocultas y 2 de salida, para clasificar las 4 posiciones. Para el segundo, se usarán sólo 2 neuronas ocultas y el resto de la estructura igual al primer ejemplo. La figura 51, muestra la T en sus 4 posiciones en una matriz de puntos de 3x3. Para el primer ejemplo, la gráfica de la figura 52, expone la velocidad de convergencia de la red y la 53, el número de patrones reconocidos a una determinada cantidad de iteraciones.

FIG. 51 MATRICES DE PUNTOS PARA LAS T EN SUS 4 POSICIONES

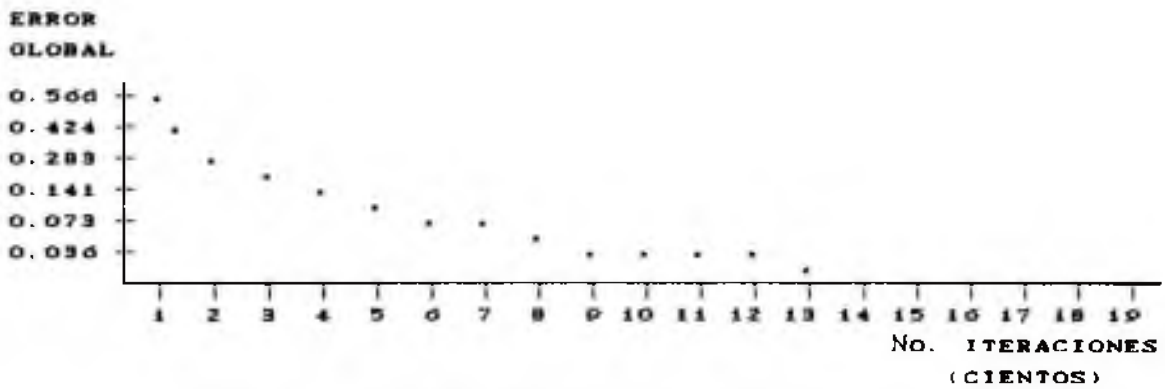


FIG. 52 GRAFICA DE RAPIDEZ DE CONVERGENCIA

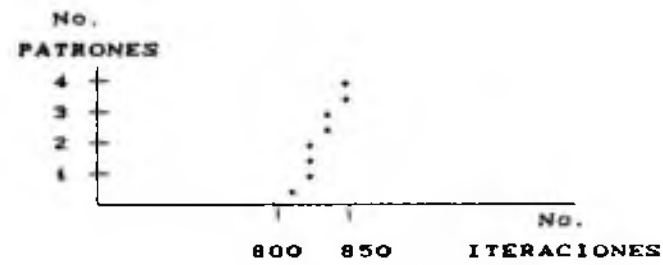


FIG. 53 GRAFICA DE RECONOCIMIENTO DE PATRONES

El entrenamiento se efectúo con los valores siguientes:

UNO = 0.9
CERO = 0.1
Umbral de Convergencia = 0.0005
Rapidez de Aprendizaje = 0.25
Convergió a las 2700 iteraciones.
Tiempo de aprendizaje: 18 Seg.

El número de neuronas ocultas es muy grande para que la red almacene solamente cuatro patrones; por esta razón, la velocidad de convergencia es muy alta (ver figura 52).

Para el segundo ejemplo, la gráfica de la figura 54, indica la velocidad de convergencia de la red; la 55, muestra el número de patrones aprendidos después de cierta cantidad de iteraciones. Recordemos que en este caso se tienen sólo dos neuronas en el nivel oculto.

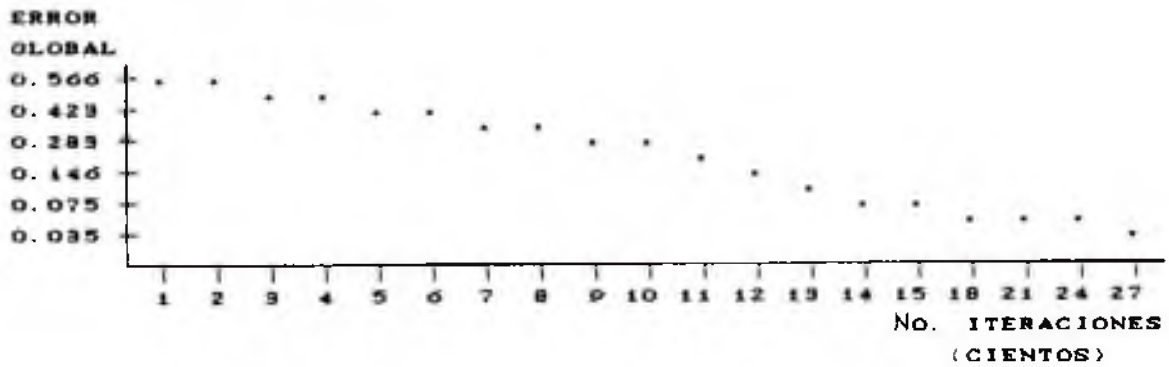


FIG. 54 GRAFICA DE RAPIDEZ DE CONVERGENCIA

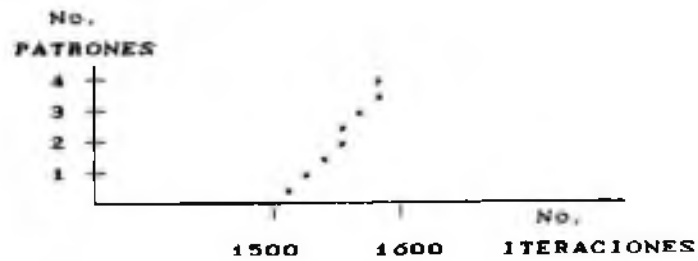


FIG. 55 GRAFICA DE RECONOCIMIENTO DE PATRONES

En la figura 54, se observa que la velocidad de convergencia es muy baja; esto se debe a que solo se tienen dos neuronas ocultas (en el caso anterior 9).

La red converge a las 9000 iteraciones en 1 Minuto.

3.4 RECONOCEDOR DE LOS NUMEROS DEL 0-9 VARIANDO LA CANTIDAD DE CONEXIONES EN LAS NEURONAS

El conocimiento de la red se representa por los pesos de las conexiones de la red. En el siguiente ejemplo, se pretende demostrar que con un número pequeño de conexiones la red no aprende los patrones. Una característica de las redes con una gran cantidad de neuronas es que reconcen información confusa. Para verificar los dos puntos anteriores se realizó una aplicación para el reconcimiento de números enteros manuscritos del 0 al 9, se representan por una matriz gráfica de 6x6 puntos, por lo que se

necesitan 36 neuronas de entrada, 4 de salida para formar 4 bits y 5 en el nivel oculto. En el primer experimento se conectaron todas las neuronas de entrada a las cinco ocultas y estas a las de salida. En la figura 56, se ilustra como se representan los números con sus respectivas entradas y salidas.

FIG. 56 MATRIZ PARA LOS NUMEROS DE ENTRADA

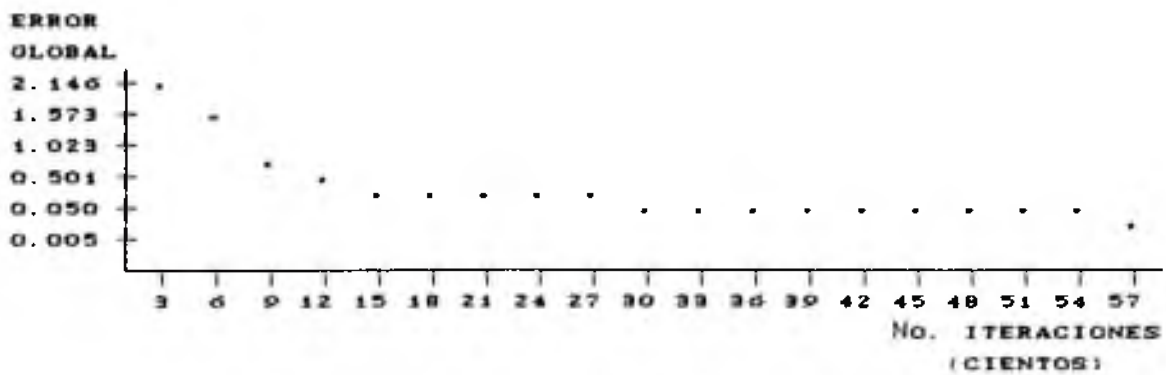


FIG. 57 GRAFICA DE RAPIDEZ DE CONVERGENCIA

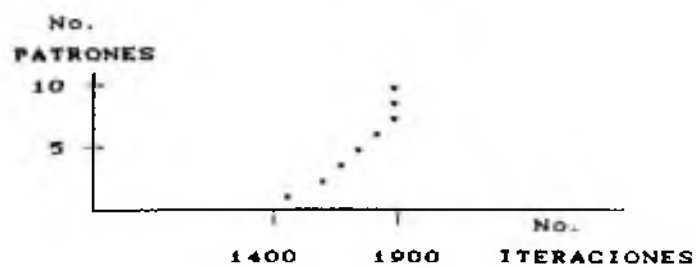


FIG. 58 GRAFICA DE RECONOCIMIENTO DE PATRONES

Durante el entrenamiento de la red la velocidad de convergencia es muy alta en las primeras 3000 iteraciones y el

resto es muy lento debido a que es alto el número de neuronas de la red. Esto se puede apreciar en la gráfica de la figura 57, la 58, muestra el número de iteraciones necesarias para empezar reconocer patrones.

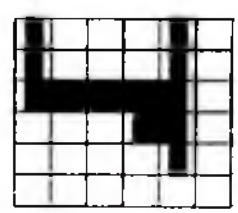
El entrenamiento se realizó con los valores siguientes:

UNO = 0.9
CERO = 0.1
Umbral de Convergencia = 0.0005
Rapidez de Aprendizaje = 0.6
Convergió a las 48000 iteraciones.
Tiempo de aprendizaje: 9 Minutos 10 Segundos.

Se disminuyó la cantidad de conexiones en aproximadamente un 45%. Las neuronas de entrada se conectaron en forma alterna a las ocultas y éstas a todas las de salida. En este caso como la reducción de las conexiones es muy fuerte, la red oscila sin apreder todos los patrones; en algunos casos aprende 3 y si continuamos el entrenamiento los olvida para aprender otros. Se inteto inicialmente con una rapidéz de aprendizaje de 0.6, con la cual la red oscila fuertemente y con 0.1 oscila ligeramente, aprendiendo 5 patrones.

Para demostrar que la red reconoce información confusa se utilizará el número 4 con las variaciones que se aprecian en la figura 59.

ENTRADA
100010,100010
111110,000110
000010,000000



SALIDA
0100

FIG. 59 NUMERO 4 DEFORMADO

La red reconoce perfectamente el número 4. Para poder percibir las variciones comparar la figura 56 con la 59.

Concluimos que las conexiones de las neuronas son un factor importante para almacenar mayor cantidad de conocimiento y que las redes neuronales artificiales permiten reconocer información confusa.

3.5 RECONOCEDOR DE PARIDAD VARIANDO EL NUMERO DE NIVELES OCULTOS

Para demostrar la utilidad de los niveles ocultos de la red, se expone un reconocedor de paridad para 5 bits. Este problema tiene cierta complicación en el momento de entrenar la red, ya que son 32 patrones con una diferencia mínima entre uno y otro, además, la salida de la red es tan sólo una neurona. Para el primer ejemplo, se utiliza una estructura con 5 neuronas de entrada, un nivel oculto con 18 y 1 en la salida. Los patrones para el entrenamiento se forman de la manera siguiente:

```
00000     0
00001     0
00010     0
00011     1
00100     0
00101     1
  .
  .
  .
11111     0
```

La gráfica de la figura 60, muestra la velocidad de convergencia de la red y la 61, la cantidad de patrones reconocidos después de cierta cantidad de iterciones.

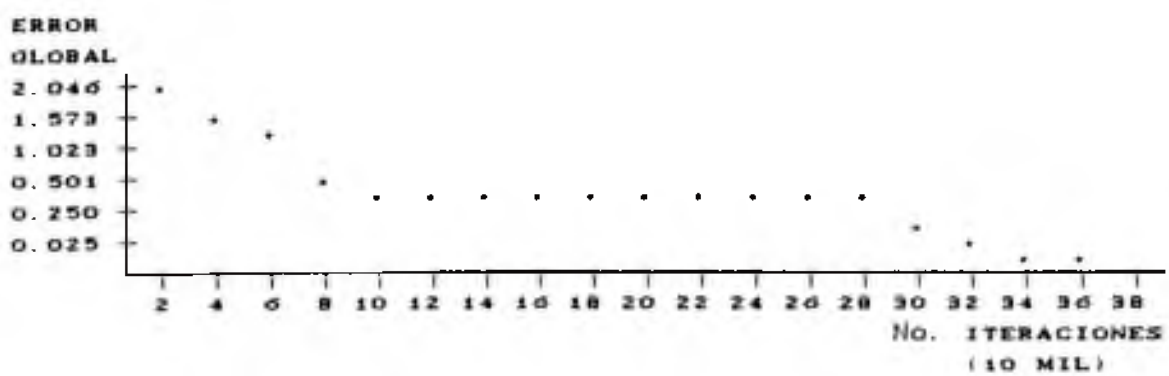


FIG. 60 GRAFICA DE RAPIDEZ DE CONVERGENCIA

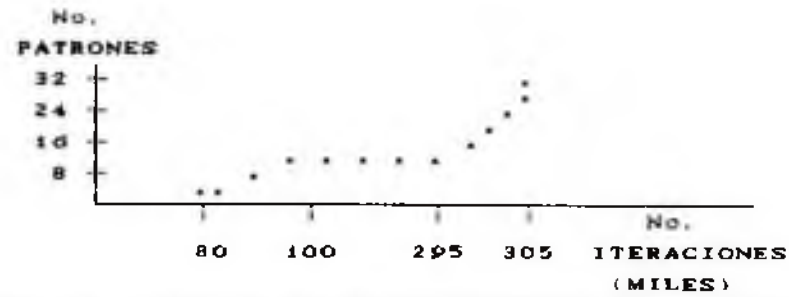


FIG. 61 GRAFICA DE RECONOCIMIENTO DE PATRONES

En la gráfica de convergencia de la figura 60, se encuentra un mínimo local entre la iteración 100000 a la 280000. Por esta razón el aprendizaje de la red es muy lento. Lo anterior se refleja en la gráfica de la figura 61, en donde el reconocimiento de patrones se detiene hasta abandonar el mínimo local.

El entrenamiento se realizó con los valores siguientes:

UNO = 0.9
CERO = 0.1
Umbral de Convergencia = 0.0005
Rapidez de Aprendizaje = 0.6
Convergió a las 352000 iteraciones.
Tiempo de aprendizaje: 52 Minutos 40 Segundos.

En el segundo ejemplo, se tiene una estructura con 5 neuronas de entrada; 2 niveles ocultos de 8 y 7 respectivamente y 1 de salida. Es importante hacer notar que se tienen 15 neuronas en total con los niveles ocultos, mientras que en el primer ejemplo son 18. Esta reducción se debe a que se generan mayor cantidad de conexiones, si lo dos niveles ocultos se construyen de 9 cada uno, con la finalidad de tener la misma cantidad de neuronas. Para demostrar el efecto de los niveles ocultos, es necesario tener la mismo número de conexiones en lo dos ejemplos, para evitar que uno tenga mayor capacidad de almacenamiento que el otro. En las dos redes las conexiones tienen los mismos pesos. Los resultados para el segundo ejemplo, se tienen en la figura 62, para la velocidad de convergencia y 63, para los patrones reconocidos después de cierta cantidad de iteraciones.

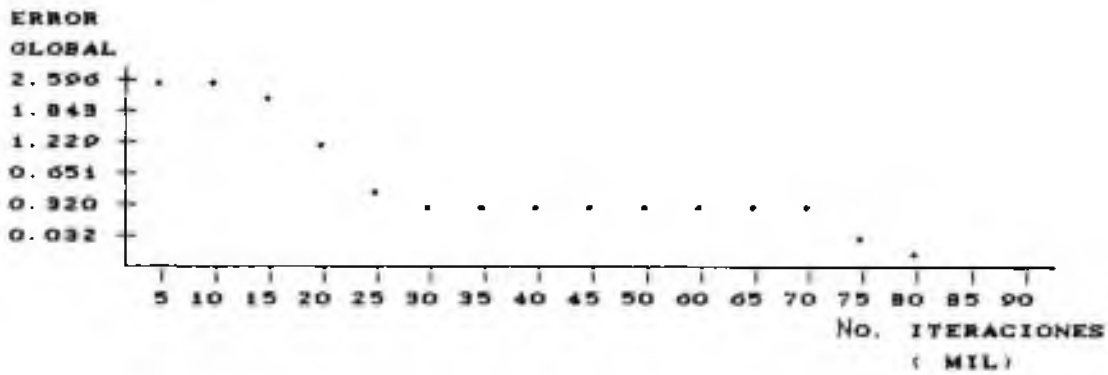


FIG. 62 GRAFICA DE RAPIDEZ DE CONVERGENCIA

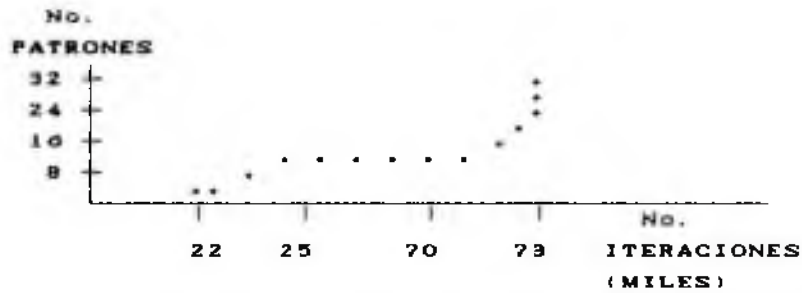


FIG. 63 GRAFICA DE RECONOCIMIENTO DE PATRONES

Analizando las gráficas de las figuras 60 y 63, podemos concluir que el ejemplo con los dos niveles ocultos convergen a una velocidad más alta debido a que tiene mayor capacidad de combinar las características de los patrones de entrada.

3.6 RECONOCEDOR DE FIGURAS GEOMETRICAS

A continuación se presenta un reconocedor de figuras geométricas, tales como un cuadrado, un triángulo, un rombo y un circulo. En una matriz de puntos de 5x5 se representan los 4 tipos de figuras. Para el cuadrado se obtuvieron 11 patrones, para el triángulo 4, para el circulo 4 y para el rombo solamente 1. La estructura de la red contiene 25 neuronas de entrada, 10 ocultas y 2 de salida. La figura 64, muestra algunos ejemplos de los patrones; la 65, la velocidad de convergencia y la 66, el número de patrones reconocidos después de una cierta cantidad de iteraciones.

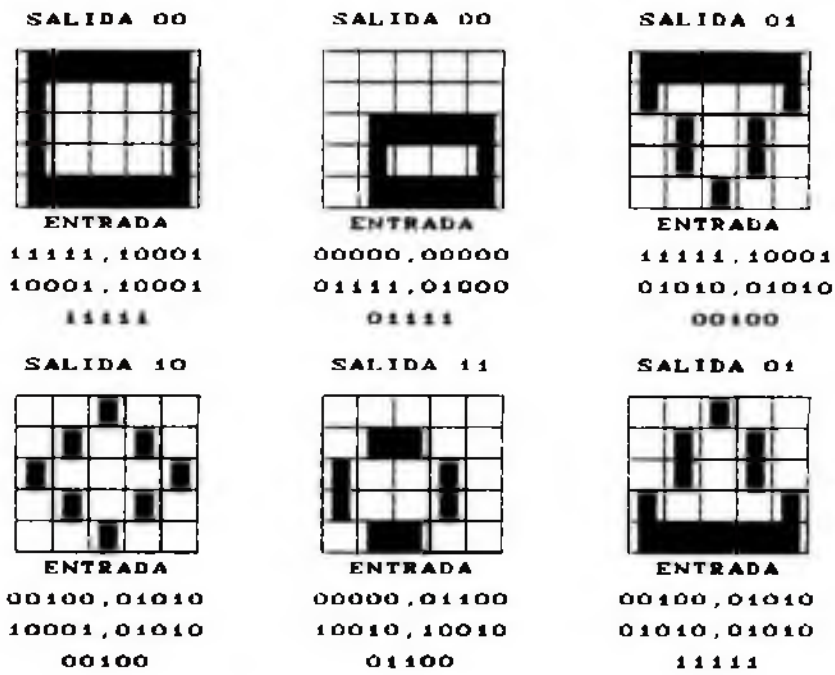


FIG. 64 FIGURAS GEOMETRICAS ENTRENADAS

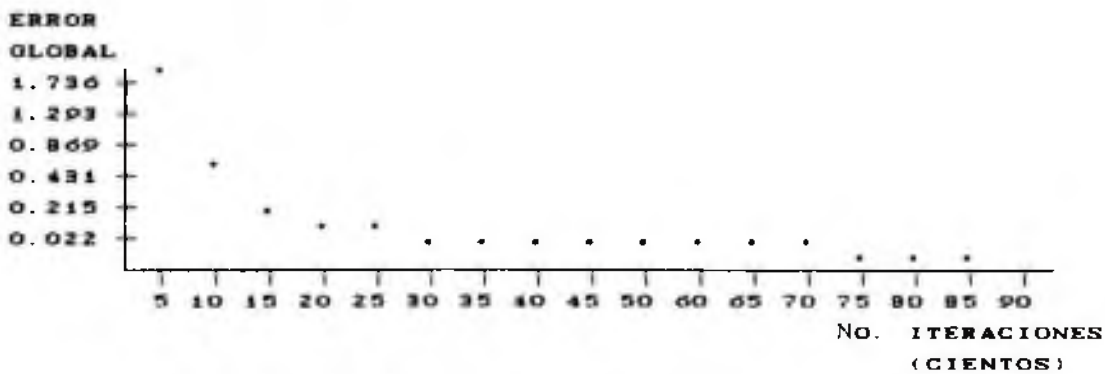


FIG. 65 GRAFICA DE RAPIDEZ DE CONVERGENCIA

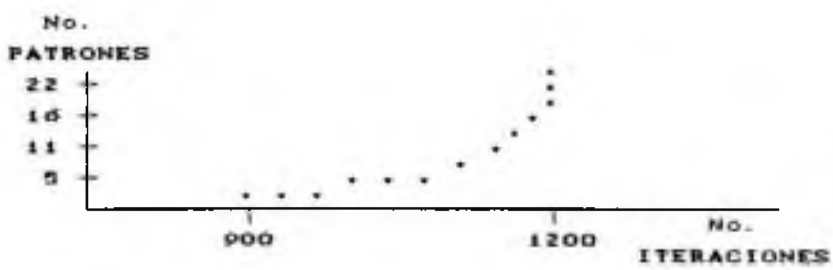


FIG. 66 GRAFICA DE RECONOCIMIENTO DE PATRONES

En la figura 66, se observa que la velocidad de convergencia es muy alta, debido a que son pocos patrones para una red tan

grande. Para formar los patrones se tuvo cuidado en que no ocuparan dos filas de la matriz de puntos y en caso del circulo que sólo tocara las esquinas de los puntos de matriz.

El entrenamiento se realizó con los valores siguientes:

UNO = 0.9
CERO = 0.1
Umbral de Convergencia = 0.005
Rapidez de Aprendizaje = 0.6
Convergió a las 38000 iteraciones.
Tiempo de aprendizaje: 10 Minutos 30 Segundos.

3.7 RECONOCIMIENTOS DE CARACTERES MANUSCRITOS

El reconocimiento de caracteres ha sido definido como la conversión de caracteres de un texto a código entendible para una máquina. El rango de aplicaciones es muy amplio, tales como, lectura de códigos postales, entrada automática de datos para sistemas administrativos muy grandes, transacciones bancarias, cartografía atomátizada y reconocimiento de caracteres manuscritos o a máquina. Con RNA se automatizaría los procesadores de texto.

Un reconocedor de caracteres debe reunir las características de precisión, velocidad y flexibilidad.

Un problema crítico para el reconcimiento es que algunas letras son muy semejantes; algunas de las características más comunes de los caracteres manuscritos son las curvas, huecos, lineas horizontales, verticales e inclinadas, etc.

Como aplicación final se realizará un reconocedor de caracteres manuscritos (algunas letras del abecedario). Las redes neuronales artificiales pueden reconocer imágenes confusas y para incrementar esta flexibilidad se coleccionaron con diferentes estudiantes una misma letra, ya que cada persona tiene una manera

muy particular de escribir, esto permite que reconozca una variedad más amplia del mismo caracter.

Para la implementación se utilizó una matriz de 7x7 para graficar las letras manuscritas y la salida será el código ASCII del caracter. Por lo que se necesitan 49 neuronas de entrada, 8 de salida y un nivel oculto de 12. La red se entrenó con las letras **A, B, G, N, P, R, S, T, V, X, Y,** y **Z**; con 8 tipos diferentes de cada una de ellas. La estructura contiene 69 neuronas para la aplicación, más 3 para simular los umbrales y 704 interconexiones. Se entreno un total de 96 patrones (12 letras con 8 tipos cada una). La figura 67 muestra algunos ejemplos de una letra escrita de diferente manera y los patrones de entrada y salida.

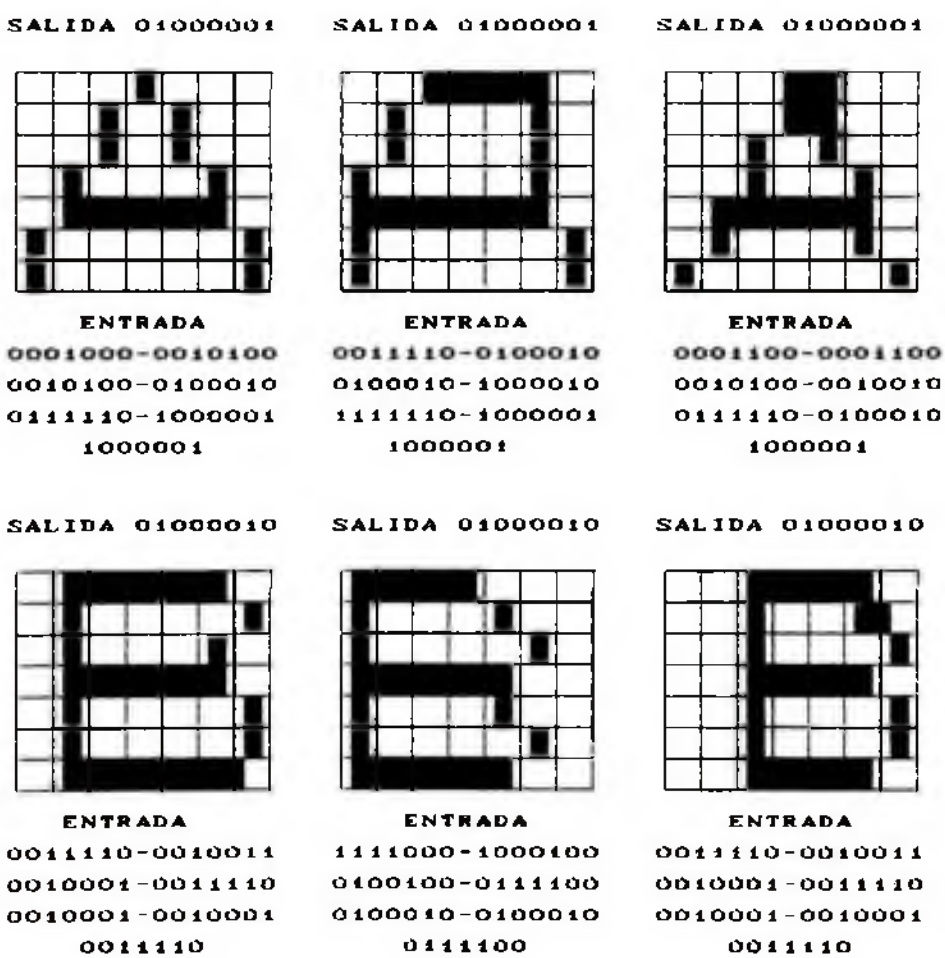


FIG. 67 TIPOS DE "A" Y "B" RESPECTIVAMENTE

La figura 68, indica la velocidad de convergencia y la 69, el número de patrones reconocidos después de cierta cantidad de iteraciones.

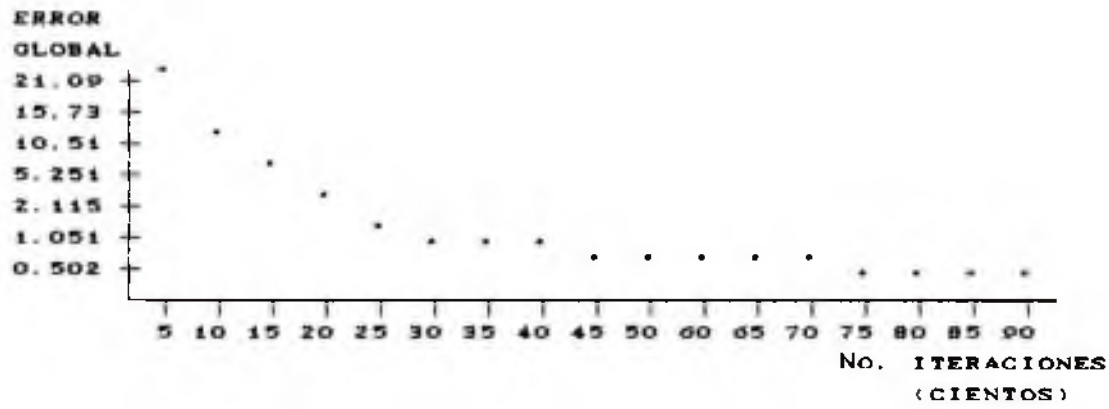


FIG. 68 GRAFICA DE RAPIDEZ DE CONVERGENCIA

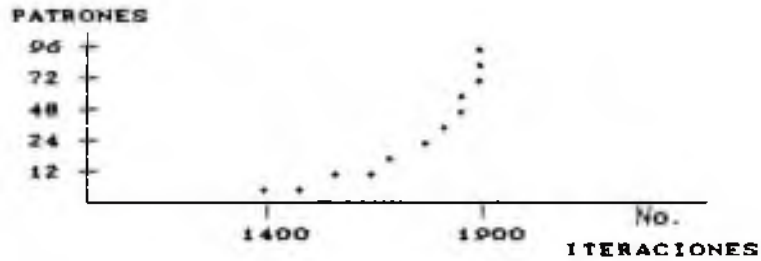


FIG. 69 GRAFICA DE RECONOCIMIENTO DE PATRONES

Analizando la figura 68, apreciamos que el error global a las primeras 1000 iteraciones es muy alto. Esto se debe a que el número de neuronas de la estructura y el número de patrones a entrenar es grande. La red converge rápidamente hasta las 3000, en este momento la red aprende prácticamente los 96 patrones, aunque las salidas no son aproximadas a UNO y CERO, a partir de aqui la convergencia es muy lenta. Una vez más se comprueba que para redes grandes el umbral de convergencia no debe ser grande.

El entrenamiento se realizó con los valores siguientes:

UNO = 0.9 y CERO = 0.1
Umbral de Convergencia = 0.2
Rapidez de Aprendizaje = 0.6
Convergió a las 25500 iteraciones.
Tiempo de aprendizaje: 20 Minutos 40 Segundos.

Se prepararón 12 tipos de cada letra, con la finalidad de usar 8 para el entrenamiento y 4 para realizar pruebas una vez enseñados a la red. Los 96 patrones entrenados son reconocidos en un 100% y los 4 restantes, los cuales no fueron entrenados, en tan sólo un 20%.. Lo anterior se debe a que los 12 patrones mantienen diferencias muy fuertes; por ejemplo, las B de la figura 67, están desplazadas en la primera, segunda y tercera columna, otra diferencia es el tamaño de la letra. Para incrementar el porcentaje de las letras reconocidas es necesario tratar de que la letra ocupe totalmente la matriz de puntos. La salida de la red es de 8 neuronas y en la mayor parte de los patrones no recocidos, era por 1 o 2 neuronas de salida (1 o 2 bits).

Existen algunas técnicas para reconocer patrones que no dependen de los tamaños y posiciones.

CONCLUSIONES

Como resultado principal de este trabajo de tesis, se obtuvo un programa (REDJAN) que simula el comportamiento de un conjunto de neuronas. El simulador REDJAN nos permite crear estructuras con un número variable de neuronas de entrada, de salida y de los niveles ocultos, a la vez de entrenar una red y reconocer los patrones aprendidos

El algoritmo de aprendizaje por **retro-propagación** no presentó un grado de complejidad muy alto para su implementación. Durante el entrenamiento de la red se encontraron pocos mínimos locales y el aprendizaje fue muy rápido. Lo anterior comprueba que es un algoritmo de aprendizaje más sencillo y efectivo que Máquinas de Boltzmann.

Analizando los ejemplos realizados se llegó a las siguientes conclusiones:

Los umbrales en las neuronas son muy importantes en el momento del entrenamiento de la red, en caso de no manejarlos se presentan oscilaciones y la red no aprende todos los patrones.

La configuración inicial de los pesos de las conexiones es un factor determinante para que la red aprenda con rapidez, con lentitud u oscile constantemente.

Es importante no tomar los números binarios de 0 y 1 para la representación de los patrones, sino una aproximación para evitar que la red tarde en alcanzar el umbral de convergencia.

La constante de aprendizaje (η), nos permite acelerar la velocidad de convergencia en la red, pero si es muy grande es posible encontrar mínimos locales.

La cantidad de niveles ocultos en la estructura de la red nos permite tener una mayor combinación de las características de los patrones a entrenar. Además el aprendizaje es más rápido.

La arquitectura del transputer T800 permitió llevar a cabo el entrenamiento de la red a velocidades muy altas (con respecto a las PC), fundamentalmente gracias al coprocesador de punto flotante y la capacidad de trabajar simultaneamente todas sus unidades (CPU, FPU y entrada\salida).

El lenguaje de programación OCCAM2 (diseñado especialmente para el Transputer) nos permitió calcular operaciones en paralelo de una manera muy sencilla, aumentando la velocidad y eficiencia del Transputer. Las limitaciones principales al utilizar esta computadora fue la falta de información técnica y que el Compilador presenta algunas deficiencias (por ejemplo en algunas ocasiones no incrementar variables).

Este trabajo de tesis abre una nueva linea de experimentación e investigación, ya sea para desarrollar aplicaciones y analizar su comportamiento, incrementar la eficiencia del programa para que reconozca caracteres sin depender del tamaño y la orientación (existen técnicas adecuadas para resolver estos problemas). Otra área de interés sería desarrollar un editor de texto por medio de un lector optico, ya que inicialmente se puede entrenar la red para que reconozca los caracteres y la entrada de la red sería la salida del lector optico.

El simulador REDJAN puede ser aplicado con un grado muy alto de eficiencia para reconocimiento de patrones.

Finalmente concluimos que las Redes Neuronales Artificiales abren un camino más para el reconocimiento de patrones, voz, sistemas expertos, etc. Además que el desarrollo tecnológico del "hardware" permite la construcción de máquinas que simulan una red de neuronas.

El transputer y el lenguaje OCCAM2 son ejemplos de una nueva generación de computadoras, los cuales nos permiten evaluaciones a grades velocidades.

REFERENCIAS

[1] McCulloch W. S. & Pitts, "A Logical Calculus of the Ideas Immanent in Nervus Activity", Bulletin of Mathematical Biophysics, Vol. 5, 115-133, 1943.

[2] R.Rusenblatt, "Principles of Neurodynamics", New York, Spartan Book, 1959.

[3] D. O. Hebb, "The Organization of Behavior", John Wiley & Sons, New York, 1949.

[4] B. Widrow and M. E. Hoff, "Adaptive Switching Circuits", 1960 IRE WESCON Conv. Record, Part 4, 96-104, Agosto de 1960.

[5] Minsky M. & Papert S. "Perceptrons: An Introduction to Computational Geometry", MIT Pres, Cambridge MA, 1969.

[6] J. J. Hopfield, "Neural Network and Physical Systems With Emergent Collective Computational Abilities", Proc. Natl. Acad. Sci. USA, Vol. 79 2554-2558, Abril 1982.

[7] J. J. Hopfield, "Neuron With Graded Response Have Collective Computational Properties Like Those of Two-State Neurons", Proc. Natl. Acad. Sci. USA, Vol. 81 3088-3092, mAYO 1984.

[8] J. J. Hopfield and D. W. Tank, "Computing With Neural Circuits: A Model" Scinse, Vol 233, 625-633, Agosto 1986.

[9] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning Internal Representation By Error Propagation", in D. E. Rumelhart & J. L. McClelland (Eds). Parallel Distributed Processing: Exploration in the Microstructure of Cognition", Vol. 1, Foundations MIT Pres, 1986.

[10] T. Sejnowsky and C.R. Rosberg, "NETtalk: A Parallel Network That Learn to Read Aloud", Johns Hopkins, Uni Technical Report JHU/EECS-86/01, 1986.

[11] J. A. Feldman And D. H. Ballard, "Connectionist Models and Their Properties", Cognitive Sciense, Vol. 6, 205-254, 1982.

[12] S. Grossberg, " The Adaptive Brain I: Cognition, Learning, Reinforcement and Rhythm", and "The Adaptive Brain II: Vision, Speech, Language and Motor Control", Elseiver/North-Holland, John Wiley & Sons, New York, 1975.

[13] C. A. Mead, "Analog VLSI and Neural Systems" Course Notes, Computer Sciense Dept, California Institute of Technology, 1986.

[14] H. P. Graf, L. D. Jackel, R. E. Howard, B. Straughn, J. J. Denker, W. Hubbard, D. M. Tennant and D. Schwartz, "VLSI implementation of a Neural Network Memory With Several Hundreds of Neurons", in J. S. Denker (Ed) AIP Conference Procedings 151, Neural Network For Computing, Snowbird Utah, AIP, 1986.

[15] B. Widrow and M. E. Hoff, "Adaptive Switching Circuits" IRE WESCON Conv. Record, Part 4, pp 96-104, Agosto 1960.

[16] B. Widrow and S. D. Stearns, "Adaptive Signal Processing" Pretice-Hall, New jersey 1985.

[17] R.O. Duda and P E. Hart, "Pattern Classification and Scence Analisys", John Wiley & Sons, New Yersey 1973.

[18] Kunihiko Fukushima, Sei Miyake and Takayuki Ito "Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition" IEEE Transation on Systems Man and Cybernectics 13(5) pp 826-834 Septiembre/Octubre 1983. In "Neurocomputing: Foundations of Research", Edited by James A. Anderson and Edward Rosenfeld. Capitulo 31, pp 523-534, 1986. In "Cpmputer Society Press Technology Series Neural Network, Artificial Neural Networks: Theorical concepts. V. Vemury pp 136-144.

[19] D. E. Rumelhart, G. E. Hinton and R. J.Williams, "Learning Internal Representation by Error Propagation" Parrallel Distributed Processing: Explorations in the Microstructures of Cognition. Vol 1, Cambridge, MA, MIT Press, pp 318-362, o en Neurocomputing, Foundations of Research. Edited by James A. Anderson And Edward Rosenfeld. Capitulo 41, pp 673-695, 1986.

[20] D. E. Rumelhart, Geoffrey E. Hinton and R. J. Williams, "Learning Representations by Back-propagating Errors", Nature 323:533-536, o en Neurocomputing, Foundations of Research, Edited by James A.Anderson and Edward Rosenfeld, Capitulo 42 pp 696-699, 1986.

[21] Williams P. Jones and Josiah Hoskins, "Back-propagation, A Generalized Delta Learning Rule", BYTE Octubre 1987, pp 155-162.

[22] Robert M. Kuczewski, Michael H. Myers, William J. Crawford, "Exploration of backward Error Propagation as a Self-Organized Structure", IEEE First International Conference on Neural Networks, San Diego California, Junio 21-24 1987, Vol II, pp 89-96.

[23] Adrew. G. Barto and Michael I. Jordan, "Gradient Following without Back-propagation in Layered Networks", IEEE First International Conference on Neural Networks, San Diego California, Junio 21-24 1987, Vol II, pp 629-636.

[24] David H. Ackley, Geoffrey E. Hinton and Terrence J. Sejnowski, "A Learning Algorithms for Boltzmann Machines", Cognitive Science 9, pp 147-169 1985.

[25] G. E. Hinton and T. J. Sejnowsky, "Learning and Relearnig in Boltzmann Machines" Parallel Distributed Prossesing, Cambridge, MIT Press, Vol 1, Capitulo 7.

[26] H. J. Sussmann, "Learning Algorithms for Boltzmann Machines" Proceding of the 27th Conference on Decision and Control, Austin Texas, Diciembre 1988.

[27] Ian Pauberry and George Selnitger, "Relating Boltzmann Machine to Conventional Models of Computation", Neural Network, Vol II, pp 59-67, 1989.

[28] Emile H. L. Aarts and Jan H. M. Kurst, " Computations in massively Parallel Networks based on Boltzmann Machine: A Review", Parallel Computing 9 (1988/1989) pp 129-145.

[29] Emile H. L. Aarts and Jan H. M. Kurst, "Boltzmann Machines and Their Applications", Phillips Research Laboratories, Lecture Notes in Computer Science 258.

[30] Kurt M. Gutzmann, "Combinational Optimization Using a Continous State Boltzmann Machine" IEEE First International Conference on Neural Networks, San Diego California, Junio 21-24 1987, Vol III, pp 721-734.

[31] G. A. Carpenter and Grossberg, "Neural Dynamics of Category Learning and Recognition: Attention, Memory Consolidation and Amnesia", in J Davis R. Newburgh and E Wegman (Eds.), Brain Structure Learning and Memory, AAAS Symposium Series, 1986, in Computer Society Press Technology Series. Neural Network, Artificial Neural Networks: Theorical Concepts. V. Vemuri.

[32] J. A. Hartigan, Clustering Algoritms, John Wiley & Sons, New York 1975

[33] Robert Hecht-Nielsen, "Counterpropagation Networks", IEEE First International Conference on Neural Networks, San Diego California, Junio 21-24 1987, Vol II, pp 19-32.

[34] David S. Touretzky and Dean A. Pomerleau, "What's Hidden in hidden Layers?", BYTE Agosto 1989, pp 227-233.

[35] Terrence J Sejnowski and Charles R Rosenberg, "NETtalk: a parallel network that learns to read aloud", Neurocomputing Foundations of Research, Edited by James A.Anderson and Edward Rosenfeld, Capitulo 40, pp 661-672.

[36] Traian Muntean, "Los Superordenadores de Transputer" Mundo Cientifico No. 87, Volumen 9, 1989.

[37] John Wexler and Domic Prior, "Solving Problems with Transputers: Backgroud and experience", Butterworth & Co LTD pp 67-77, Vol 13 No. 2, marzo 1989.

[38] M. Elizabeth C. Hull and Adib Zarea-Aliabadi, "Real-Time System Implementation the Transputer and Occam Alternative" North-Holland Microprocessing and Microprogramming, pp 77-84 26(1989).

[39] Jean-Daniel Micound and Andrew Martin Tyrrell, "The Transputer T414 Instruction Set", IEEE MICRO, pp 60-74, Junio 1989.

[40] Tabab D. "RISC Architecture Research Studies Press/ John Wiley, Chichester UK, 1987.

[41] INMOS, Occam2 Standlone Compiler for the Transputer USER'S MANUAL, 1989.

[42] Alan Burns, "Programming in Occam2" Addison-Wesley Publishing Company, 1987.

[43] INMOS Limented, Occam2 REFERENCE MANUAL, Prentice Hall, 1988.

[44] Geranit Jones, Michael Goldsmith, "Programming in Occam2", Prentice Hall, 1988.

APENDICE A

MANEJO DEL SIMULADOR REDJAN

A continuación se explica como manejar las opciones del simulador de redes neuronales artificiales REDJAN. Nos apoyaremos en la función XOR para ejemplificar cada una de ellas. La figura H, muestra la estructura para la función XOR. (note que la fila 0 (cero) sirve para simular los umbrales de las neuronas).

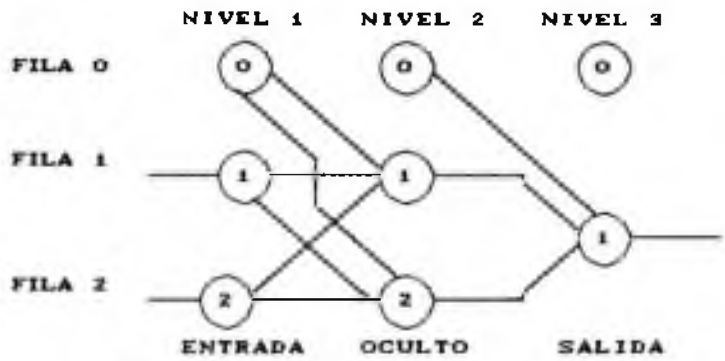


FIG H ESTRUCTURA PARA LA FUNCION XOR

Al ejecutar el programa en el Transputer se despliega un menú principal que presenta las siguientes opciones:

NENU PRINCIPAL

1: Crear Neuronas.
2: Crear conexiones entre las neuronas.
3: SALVAR la Estructura.
4: CARGAR la Estructura.
5: FUNCIONES DE LA RED.
6: Abandonar el programa.
Escribe OPCION:

CREAR ESTRUCTURAS

Para crear la estructura de la red, se cuenta con dos formas:
a) La interactiva con el usuario.
b) Con el editor de texto de TURBO C.

Para el inciso a) se necesita la opcion 1 y 2 del Menú Principal.

1: Crear Neuronas.

- *Número de neuronas de Entrada:* 3
- *Número de niveles Ocultos:* 1
- *Número de neuronas del nivel Oculto No. 1* 3
- *Número de Neuronas de salida:* 2
- *Valor de la Rapidez de Aprendizaje:* 25.0E-02

Es importante hacer notar que en cada columna se agrega una neurona para manejar los umbrales, aunque en la última no es necesaria, pero simplifica la programación para incrementar la velocidad del aprendizaje.

El valor de la rapidéz de aprendizaje debe ser dado en forma exponencial (25.0E-02). Si escribimos .25 ó 0.25 el valor no puede ser reconocido.

Para crear las conexiones se utiliza la opción 2.

2: Crear conexiones entre neuronas.

- *Número de conexiones para la neurona Oculta 2 1* 3
- *Conexión 1* 1 0 0
- *Conexión 2* 1 1 0
- *Conexión 3* 1 2 0
- *Número de conexiones para la neurona Oculta 2 2* 3
- *Conexión 1* 1 0 1
- *Conexión 2* 1 1 1
- *Conexión 3* 1 2 1
- *Número de conexiones para la neurona de Salida 3 1* 3
- *Conexión 1* 2 0 0
- *Conexión 2* 2 1 0
- *Conexión 3* 2 2 0

Para contruir la conexiones nos colocamos en la neurona a la que llegan los estímulos. Por ejemplo en la figura H la neurona 0 (cero) del nivel 2 tiene tres conexiones de entrada, de las neuronas 0, 1 y 2 del nivel 1.

Al terminar de hacer la última conexión, automáticamente se les asigna aleatoriamente un peso.

El procedimiento interactivo es eficiente para redes pequeñas, para redes grandes es muy cansado y si se cometen errores es necesario empezar a editar toda la red.

Para crear la estructura con el editor de texto de Turbo C, el archivo queda de la siguiente manera:

```
CONNECT  2/1  TO  1/0  WEIGTH  0.7432784     /* UMBRAL */
CONNECT  2/1  TO  1/1  WEIGTH  -0.076832
CONNECT  2/1  TO  1/2  WEIGTH  0.183465
CONNECT  2/2  TO  1/0  WEIGTH  0.9783565     /* UMBRAL */
CONNECT  2/2  TO  1/1  WEIGTH  0.004324
CONNECT  2/2  TO  1/2  WEIGTH  -0.39767
CONNECT  3/1  TO  2/0  WEIGTH  0.0242576     /* UMBRAL */
CONNECT  3/1  TO  2/1  WEIGTH  0.528534
CONNECT  3/1  TO  2/2  WEIGTH  0.2394371
END
RAPIDEZ DE APRENDIZAJE 0.25
```

Para formar las conexiones nos colocamos en la neurona que llegan los estímulos, igual que en la forma interactiva.

Para marcar el final de las conexiones se usa la palabra **END** y como dato final se dará la rapidez de aprendizaje. Es necesario que las palabras claves sean mayúsculas y que las líneas no sean mayores de 80 columnas.

Cuando la red es grande es conveniente utilizarel editor de texto, los pesos y umbrales se asignan manualmente.

SALVAR Y CARGAR ARCHIVOS

Una vez que se terminó de crear la estructura se puede salvar en disco magnético con las siguientes opciones:

3: Salvar la Estructura.

La opción desplegará el siguiente submenú:

1.- *SALVAR en forma de TEXTO.*

2.- *SALVAR en forma COMPACTA.*

Escribe la OPCION:

En el caso 1, la red queda en forma de texto, la extensión debe ser **.TXT** para distinguir de los archivos compactos. Con estos archivos es muy sencillo analizar si hay conexiones erróneas, aunque es bastante grande el archivo.

En el caso 2, el tamaño de los archivos es en un 45% menor que de los tipo texto, la extensión es **.RED** para diferenciar. En el siguiente archivo se representa como se almacenaría la función XOR en forma compacta.

```
1   3   3   2   0   3   3   0   3   0.25   10000   10100   10200
10001   10101   10201   20000   20100   20200   2   2   2   1   1
1   -0.994543493   0.62565434   -0.817625344   -0.041423671
0.551534295   -0.546560109   0.318549301   0.14854399   -0.8345438
```

El primer Número (1), indica la cantidad de niveles ocultos; el segundo (3), la cantidad de neuronas de entrada; el tercero (3), la cantidad de neuronas ocultas para las columnas que nos señale el primer número, en éste caso 1; el cuarto (2) las neuronas de salida; los siguientes cinco números (0,3,3,0,3) dicen cuantas entradas tiene cada neurona, empezando por las neuronas de la columna oculta hasta las neuronas de salida, obviamente sin considerar las de entrada. El valor de 0.25 es la rapidéz de aprendizaje; los 9 valores que se encuentran después 0.25 son las conexiones, los tres 2 y 1 son las salidas de las neuronas empezando por las de entrada y finalmente los pesos de la red.

SALVAR Y CARGAR ARCHIVOS

Una vez que se terminó de crear la estructura se puede salvar en disco magnético con las siguientes opciones:

3: Salvar la Estructura.

La opción desplegará el siguiente submenú:

1.- *SALVAR en forma de TEXTO.*

2.- *SALVAR en forma COMPACTA.*

Escribe la OPCION:

En el caso 1, la red queda en forma de texto, la extensión debe ser .TXT para distinguir de los archivos compactos. Con estos archivos es muy sencillo analizar si hay conexiones erróneas, aunque es bastante grande el archivo.

En el caso 2, el tamaño de los archivos es en un 45% menor que de los tipo texto, la extensión es .RED para diferenciar. En el siguiente archivo se representa como se almacenaría la función XOR en forma compacta.

```
1   3   3   2   0   3   3   0   3    0.25    10000    10100    10200
10001    10101    10201    20000    20100    20200    2   2   2   1   1
1      -0.994543493      0.62565434      -0.817625344      -0.041423671
0.551534295    -0.546560109    0.316549301    0.14854399    -0.8345438
```

El primer Número (1), indica la cantidad de niveles ocultos; el segundo (3), la cantidad de neuronas de entrada; el tercero (3), la cantidad de neuronas ocultas para las columnas que nos señale el primer número, en éste caso 1; el cuarto (2) las neuronas de salida; los siguientes cinco números (0,3,3,0,3) dicen cuantas entradas tiene cada neurona, empezando por las neuronas de la columna oculta hasta las neuronas de salida, obviamente sin considerar las de entrada. El valor de 0.25 es la rapidéz de aprendizaje; los 9 valores que se encuentran después 0.25 son las conexiones, los tres 2 y 1 son las salidas de las neuronas empezando por las de entrada y finalmente los pesos de la red.

Para cargar a memoria los archivos se utiliza la siguiente opción:

4: Cargar la Estructura.

La opción despliega el submenú:

1.- *CARGAR en forma de TEXTO.*

2.- *CARGAR en forma COMPACTA.*

Escribe OPCION:

La opción 1, sirve para cargar los archivos que fueron generados con el editor de Turbo C y su extensión es **.TXT**; la dos es para los archivos en forma compacta con extensión **.RED**.

5.- FUNCIONES DE LA RED.

Con la opción 5 se pasa a un nuevo menú, en donde se puede realizar el entrenamiento, dar los patrones de aprendizaje, hacer los reconocimientos de los patrones entrenados, etc. La opción despliega el siguiente menú:

FUNCIONES DE LA RED.

1: Cargar Patrones de APRENDIZAJE.

2: Cambiar Valores de la red.

3: APRENDIZAJE de la RED.

4: Desplegar pesos de la red.

5: RECONOCER patrones.

6: Regresar al menú principal.

Escribe la OPCION:

1.- Cargar Patrones de APRENDIZAJE

La opción muestra el siguiente submenú:

1.- *Meter un patrón.*

2.- *Meter todos los patrones.*

3.- *Cargar un Archivo con los Patrones.*

Escribe la OPCION:

En el caso 1, se proporciona un patrón que no ha sido

aprendido con la misma velocidad que los demás, a la vez pregunta cuantas veces va repetir el entrenamiento. Esta posibilidad se tiene porque en algunas ocasiones un patrón no es aprendido y necesita ser entrenado mayor cantidad de veces. Una cantidad razonable sería presentar 50 veces el patrón y 20 el resto. De lo contrario se perdería lo aprendido en la red.

Para el caso 2, de entrada pregunta cuantos patrones de entrenamiento se van a editar, después va pidiendo la entrada y la salida deseada para cada uno de ellos.

El caso 3, permite cargar archivos que contienen los patrones de entrenamiento. Es muy útil para redes con un número alto de neuronas de entrada y de salida y una catidad grande de patrones a entrenar. El archivo se edita en Turbo C y tiene una extensión **.pat**. El archivo que continuación se preseta contiene los patrones para la función XOR.

```
IN  11
OUT 0
IN  10
OUT 1
IN  01
OUT 1
IN  00
OUT 0
END
```

2.- Cambiar Valores de la Red.

La opción 2 despliega el siguiente submenú.

CAMBIAR VALORES.

1.- *Cambiar RAPIDEZ DE APRENDIZAJE.*

2.- *Cambiar UMBRAL DE CONVERGENCIA.*

3.- *Cambiar frecuencia de muestreo del ERROR GLOBAL.*

Escribe la OPCION:

Estas opciones presentan el valor que actualmente tienen y piden su nuevo valor.

3.- APRENDIZAJE de la Red.

Para el aprendizaje de la red se utilizan las variables del umbral de convergencia y la frecuencia de muestreo del error global, los cuales tienen valores por defecto de 0.0005 y de 100 respectivamente. Estos pueden ser cambiados en cualquier momento del entrenamiento. La frecuencia de muestreo del error global se tiene variable porque una de las finalidades de este trabajo es mostrar la velocidad de convergencia de la red. Antes de empezar el entrenamiento, es necesario proporcionar los patrones de aprendizaje por medio de la opción 1 del menú.

Con la opción 3, se empieza el entrenamiento de la red, pregunta cuantos barridos o presentaciones se ejecutarán. El número de barridos se divide entre la cantidad de patrones existentes. Si tenemos 4 patrones para la XOR y pedimos 20 barridos, se entrenarán 5 veces cada patrón.

En caso que la frecuencia del error global sea mayor que el número de barridos no desplegará nada en pantalla.

4.- Desplegar los Pesos de la Red.

Con la opción 4 aparece el siguiente submenú.

1.- *En forma NUMERICA.*
2.- *En forma GRAFICA.*
Escribe OPCION:

El caso 1, despliega los pesos de cada una de las conexiones de una neurona, empieza con la neurona 0 del nivel 1, hasta llegar a la neurona final de la última columna oculta. Al terminar con una neurona dá un RETURN para que se aprecie mejor los pesos de las conexiones y continua con la neurona subsecuente.

El caso 2, despliega los pesos con colores empezando con la neurona 0 del nivel 1 hasta la última de los niveles ocultos. (Ver II.4).

5.- RECONOCIMIENTO DE PATRONES.

Con la opción 5, se hacen los reconocimientos de patrones. Al inicio pide el patrón de entrada y como respuesta nos entrega la salida deseada, en caso que la red lo haya aprendido, de lo contrario enviará un **-1** en el lugar del **0** y **1**.

Con la opción 6, se regresa al Menú Principal.

RECOMENDACIONES PARA EL USO DE REDJAN

Para comprobar que las conexiones de la red quedaron correctamente cuando se crean por medio del editor de Turbo C, se salva la red en un archivo compacto y éste nos permite analizar fácilmente las conexiones de salida y entrada, número de neuronas y los pesos de cada conexión.

Utilizar al frecuencia de muestreo del error global para ver si la red va aprendiendo durante el entrenamiento.

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL I.P.N.
APARTADO POSTAL 14 740
MEXICO 07000, D.F.

EL JURADO DESIGNADO POR LA SECCION DE COMPUTACION DEL DEPARTAMENTO DE INGENIERIA ELECTRICA, ACEPTO EL DIA 11 DEL MES DE JUNIO DEL AÑO 1990 EL TRABAJO DE TESIS " MODELACION DE REDES NEURONALES

DESARROLLADO POR EL ALUMNO: MARTIN FIGUEROA MIRELES

; PARA SU IMPRESION Y TRAMITES CORRESPONDIENTES DEL EXAMEN DE GRADO.

DR. JAN JANECEK HYAN

DR. GUILLERMO MORALES LUNA

M. EN C. SERGIO CHAPA VERGARA

M. EN C. OSCAR OLMEDO AGUIRRE

'lda.