

1
2 eje



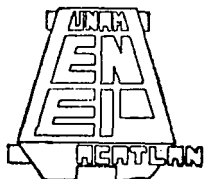
UNIVERSIDAD NACIONAL
AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS
PROFESIONALES ACATLAN

"FORMA DE APRENDIZAJE DE LAS REDES
NEURONALES APLICANDO EL ALGORITMO
REGLA DELTA GENERALIZADA A LA
REPRESENTACION DEL CONOCIMIENTO"

T E S I S
QUE PARA OBTENER EL TITULO DE
LICENCIADO EN MATEMATICAS
APLICADAS Y COMPUTACION
ES PRESENTADA POR:

AGUILAR CASTILLEJOS RAUL ANTONIO
ZUÑIGA JAUREGUI LUIS MANUEL



ACATLAN, MEXICO

A S E S O R:
DR. JESUS FIGUEROA NAZUNO

1994

TESIS CON
FALLA DE ORIGEN



Universidad Nacional
Autónoma de México



UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

DERECHOS RESERVADOS © PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis está protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.



UNIVERSIDAD NACIONAL
AUTONOMA DE MEXICO

INDICE

Introducción.

Capítulo I.

Antecedentes de la Neurocomputación y las Redes Neuronales.

- I.1 Antecedentes.
- I.2 Redes Neuronales.
- I.3 Paradigmas.
- I.4 Perspectivas.

Capítulo II.

Fundamentos de las Redes Neuronales.

- II.1 Topología.
- II.2 Función de Activación, Funcionamiento.
- II.3 Procedimientos de Entrenamiento y Aprendizaje.
- II.4 Valores de entrada.

Capítulo III.

Algoritmos y sus Fundamentos Matemáticos.

- III.1 Taxonomía de los Algoritmos.
- III.2 Redes Neuronales y Clasificadores Tradicionales.
- III.3 Red de Hopfield.
- III.4 Red de Hamming.
- III.5 Clasificador de Carpenter/Grosberg.
- III.6 Perceptrón de Capa Simple.
- III.7 Perceptrón Multicapa.
 - III.7.1 Perceptrón de Retropropagación.
 - III.7.2 Regla Delta Generalizada.
- III.8 Mapa de características de Auto-organización de Kohonen.

Capítulo IV.

El Problema de la Representación de la Información en Neurocomputación.

- IV.1 El problema de la Representación de Información en Neurocomputación.
- IV.2 Hipótesis de Representación.

Capítulo V.

Aplicación del Algoritmo Regla Delta Generalizada para el Aprendizaje Supervisado.

V.1 Implementación del Algoritmo Regla Delta Generalizada

V.2 Aplicación del Algoritmo Regla Delta Generalizada para Aprendizaje Supervisado.

Capítulo VI.

Análisis de Resultados.

VI.1 Análisis e Interpretación de resultados.

VI.2 Técnicas de Análisis de la Representación.

VI.3 Análisis Fractal.

VI.4 Análisis de Convergencia.

Conclusiones.

Anexos.

-Programas (Código fuente en Lenguaje C).

-Archivos de Datos.

-Paquetes de Software Usados.

Glosario.

Bibliografía

INTRODUCCION

Por mucho tiempo se pensó que los desarrollos en el campo de la computación nunca se podrían igualar o compararse con los resultados obtenidos por el conocimiento humano y su forma de pensar. Este problema influyó en el desarrollo de una corriente, con diferentes formas de atacar el problema de igualar el pensamiento humano : **La Inteligencia Artificial.**

El objetivo principal de la Inteligencia Artificial es el que las computadoras imiten la forma en la que los seres humanos asimilan, procesan y comunican información, sin que el humano tenga la necesidad de programar las acciones de la máquina y que además la comunicación con la máquina le sea lo más sencilla posible.

La Inteligencia Artificial desarrolló 2 caminos diferentes:

- **Los Sistemas Expertos.**

Estos imitan el proceso de deducción humana para llegar a un resultado.

- **Las Redes Neuronales Artificiales ó Neurocomputación.**

Partiendo de la idea de que un sistema aprendiera a reconocer patrones adaptativamente, basándose en la experiencia obtenida a partir de sus propias deducciones.

Los Sistemas Expertos mediante una serie de inferencias lógicas, previamente programadas, realizan deducciones y obtienen resultados como lo haría un experto humano en el área de su conocimiento. Todo Sistema Experto esta formado por 3 partes principales:

- **Una Base de Conocimientos.**

En esta base de conocimientos se encuentran todos los hechos que la máquina conoce. Esta base de conocimientos puede ser o no modificable, dependiendo de las necesidades particulares de cada sistema.

- **Una Máquina de Inferencias.**

Esta máquina, mediante una serie de inferencias lógicas y mecanismos altamente recursivos, efectúan un proceso para identificar el hecho al que se hace referencia.

- **Una Interface con el Usuario.**

Mediante esta interface el usuario dá a la máquina los datos que conoce para poder definir un hecho, esta interface, además de servir como enlace con el usuario, permite el control de las acciones de la máquina de inferencias.

Un Sistema Experto funciona de la siguiente forma:

El usuario mediante la interface, dá los datos a la máquina; estos datos son recibidos en la máquina de inferencias, ésta buscará con los datos recibidos en la base de conocimientos, si se está haciendo referencia a algún hecho conocido y lo informará al usuario.

De esta manera las 3 partes, interactuando entre sí, guían las acciones del Sistema Experto y conducen a un resultado.

Los Sistemas Expertos se encuentran escritos, en su mayoría, en lenguajes de programación especializados, no procedurales, basados en reglas y cláusulas como LISP ó PROLOG. Además, estos sistemas tienen el inconveniente de que necesitan reprogramarse, ya sea manual ó automáticamente, para poder utilizar nuevos hechos, y únicamente funcionan en el campo de conocimiento, para el que fueron diseñados, esto significa que nunca se podrá usar el mismo Sistema Experto para dos campos de conocimiento diferentes sin necesidad de reprogramar su máquina de inferencias y su base de conocimientos.

El uso de sistemas expertos, en un campo específico de conocimiento puede igualar o incluso superar en algunos casos el desempeño humano, como es el caso de los sistemas de estrategia, matemáticas simbólicas y algunos procesamientos de imágenes.

Sin embargo no todos los campos del conocimiento son apropiados para el desarrollo de sistemas expertos, como los campos de reconocimiento de patrones y visión, en estos campos existen los problemas "monstruo" de la Inteligencia Artificial, donde la aplicación de los Sistemas Expertos resulta difícil ó imposible. Los problemas de este tipo son los que las personas resuelven fácilmente, por ejemplo, las personas pueden reconocer, visualmente si una cara pertenece a un hombre ó a una mujer, aún con un peinado ó expresión diferente. Las personas hacen esto desde que son bebés, el problema comienza cuando se intenta realizar esto en una computadora. La solución de este tipo de problemas debe de realizarse mediante el uso de Redes Neuronales.

El desarrollo de las Redes Neuronales Artificiales, se basó en el estudio de los sistemas nerviosos de los seres vivos para procesar información y de esta manera implementar una herramienta de análisis en una máquina, para que ésta realizara el proceso de la información de manera similar a la forma en que los sistemas vivos lo hacen. Para hacer estas herramientas se desarrollaron modelos matemáticos que representan una visión enormemente simplificada y modelada, a partir de la estructura del cerebro y del sistema nervioso central.

Se le conoció como Neurocomputación porque al simular en computadoras la forma en que los sistemas nerviosos vivos funcionan, basándose en las teorías de la Neurofisiología, se obtuvo un procesamiento similar al que se da en las redes neuronales de los cerebros y los sistemas nerviosos de los seres vivos.

El interés por el estudio de estos sistemas radica en que éstos poseen una serie de propiedades (Memoria, Cómputo, Aprendizaje, Intencionalidad, Asociación, Fiabilidad) que no son atribuibles a las máquinas o al tipo de cómputo tradicional.

En los campos de aplicación de la Inteligencia Artificial donde el tipo de conocimiento a tratar sigue secuencias o patrones, donde se requiere una respuesta con la mayor exactitud posible y tolerancia a las fallas, y donde se necesitan las respuestas en el menor tiempo posible, los Sistemas Expertos son ineficientes y se debe entonces de aplicar herramientas de Redes Neuronales.

Los campos de aplicación directa de las Redes Neuronales son:

- Reconocimiento de Voz.
- Modelado y Pronóstico Financiero.
- Investigación de Mercados.
- Diagnósticos Médicos.
- Inspección y Control de Calidad.
- Control de Procesos.
- Exploración Petrolera.
- Reconocimiento óptico de caracteres.
- Problemas de optimización, etc.

Las Redes Neuronales han dado un fuerte avance en problemas del tipo de reconocimiento de patrones, en los casos específicos de visión y lectura, reconocimiento y síntesis continua de voz y en vehículos autónomos.

En México el uso y desarrollo de las Redes Neuronales es aún limitado únicamente a instituciones de investigación a nivel superior y de poca difusión, debido principalmente a la falta de conocimiento sobre éstas y al gran desarrollo e impulso que se ha dado en México a los Sistemas Expertos.

Las computadoras son capaces de procesar información de forma lineal o en serie, mientras que el sistema nervioso realiza múltiples acciones simultáneamente o en paralelo. El lograr el procesamiento en paralelo en computadoras conducirá a procesar gran cantidad de información en forma simultánea y casi instantánea.

Aunque en el campo de la electrónica implantar estos sistemas en circuitos electrónicos es relativamente sencillo, incluso actualmente existen tarjetas y chips de computadora, diseñados como Redes Neuronales capaces de acelerar una computadora con un procesador 80386 corriendo a 25 megahertz, de forma que ésta alcanza 1/2 de la velocidad de una computadora Cray-2, el controlar y programar las Redes Neuronales, en términos de software es aún muy difícil.

El desarrollo de las Redes Neuronales permitió que se cambiara el concepto de las Matemáticas Aplicadas en el campo de la Inteligencia Artificial, ya que se consideraba a las Matemáticas Aplicadas, por su proceso de abstracción, alejadas de consideraciones prácticas.

El estudio de las Redes Neuronales como sistemas para la representación y manipulación de la información ha comenzado a progresar de manera cautelosa. Los fundamentos de la Neurocomputación no se encuentran claramente establecidos, ni se tiene aún teorías formales completas. La naturaleza formal y matemática de los procedimientos neurocomputacionales facilitan la búsqueda de sus fundamentos.

Dado que la representación del conocimiento es un punto central para la investigación en neurocomputación se fijan como objetivos principales:

- Encontrar la forma de funcionamiento del aprendizaje de una Red Neuronal usando el Algoritmo Regla Delta Generalizada y así, obtener datos para el estudio de las Redes Neuronales.
- Analizar los criterios de convergencia para Redes Neuronales que, en base a ejemplos, hayan aprendido para determinar, de esta manera su comportamiento.
- Obtener una generalización básica y conceptual sobre cómo y cuándo aprenden las Redes Neuronales para hacer así una generalización teórica.

En este trabajo se tratarán los modelos matemáticos usados para comprender el cerebro y sus analogías computacionales. Para constatar la validez de los modelos neuronales, se expresan matemáticamente y se utilizan herramientas matemáticas, a fin de obtener conclusiones generales y computacionales, para constatarlos. Para este fin adicionalmente se fijan los siguientes objetivos específicos:

- Presentar los antecedentes de las Redes Neuronales.

Dar a conocer la historia del desarrollo y aplicación de las Redes Neuronales desde sus inicios hasta nuestros días. Se dará a conocer el desarrollo histórico del campo de la Neurocomputación y las Redes neuronales, el porque de su desarrollo, y perspectivas futuras de su desarrollo.

- Exponer los fundamentos y bases funcionales de la Neurocomputación y las Redes Neuronales.

Se expondrán y analizarán los conceptos fundamentales de la teoría Neurocomputacional, sobre el funcionamiento y las bases de las Redes Neuronales. Los fundamentos de la neurocomputación por ser de naturaleza formal y matemática, facilitan su estudio, y esto puede servir para el estudio de una teoría general de la representación.

- Exponer y detallar los fundamentos matemáticos para los algoritmos más conocidos y fundamentales en Neurocomputación.

En el estudio neurocomputacional han surgido una serie de algoritmos, cada uno con particularidades propias y propiedades de interés. Se presentaran y explicaran los más conocidos.

- Presentar la Representación de la Información en Neurocomputación y considerar las hipótesis sobre la forma abstracta de codificación de la información externa en una forma comprimida pero conservando todas sus características, para su posterior utilización.

En la teoría neurocomputacional uno de los problemas fundamentales es el de la representación de la información. Se expondrá, explicará y analizará el porque este es uno de los problemas de mayor interés en estudio neurocomputacional y cuales son sus principales hipótesis de solución.

- Aplicación y desarrollo del Algoritmo Regla Delta Generalizada para el Aprendizaje Supervisado como una forma de estudio de Redes Neuronales.

Obtener un sistema para el análisis de Redes Neuronales computacionales, mediante la utilización del algoritmo Regla Delta Generalizada se realizarán estudios computacionales intensivos para analizar el comportamiento interno de las Redes Neuronales en situaciones en donde aprenden y en donde no aprenden. Se implementará el algoritmo Regla Delta Generalizada en un programa en lenguaje C, donde una de sus características importantes será el poder definir los parámetros propios del algoritmo. Usar este algoritmo implementado en una computadora, permite el estudio del comportamiento matemático de una forma muy sencilla, aparte de la solidez matemática de este procedimiento y de ser uno de los más conocidos.

- Analizar los resultados para plantear la forma de funcionamiento de las Redes Neuronales y analizar los criterios de convergencia de las Redes Neuronales que en base a ejemplos hayan aprendido.

Encontrar criterios para determinar el aprendizaje de las redes neuronales computacionales. Los resultados se analizarán, seleccionando algún método para analizar los valores de la Red, y así determinar como se representa la información en la Red Neuronal para lograr el aprendizaje.

- Obtener conclusiones y una generalización básica y conceptual sobre cómo y cuándo aprenden las Redes Neuronales.

Encontrar generalizaciones hechas en base a los criterios encontrados para el aprendizaje de Redes Neuronales.

El título de la tesis "Forma de Aprendizaje de las Redes Neuronales aplicando el Algoritmo Regla Delta Generalizada a la Representación del Conocimiento" indica la aplicación del algoritmo RDG para estudiar la representación del conocimiento, en donde además, se estudiará el comportamiento matemático de una Red Neuronal, en una situación típica de Neurocomputación, en casos donde aprende y donde no aprende.

En este trabajo se explicará el desarrollo y fundamentos de las Redes Neuronales, - histórico, matemático y computacional - (Capítulos I y II), se presentarán y explicarán diferentes modelos de Redes Neuronales y se plantearán la hipótesis para el aprendizaje de la Redes Neuronales (Capítulo III y IV), se utilizará el Algoritmo Regla Delta Generalizada (ARDA) para el estudio de la Representación del Conocimiento dentro de una Red Neuronal (Capítulo V), los resultados del estudio se analizarán para plantear la forma de funcionamiento de las Redes Neuronales (Capítulo VI) y obtener una generalización sobre la forma del aprendizaje de las Redes Neuronales .

Además, este trabajo dará un conocimiento adecuado y accesible sobre las Redes Neuronales, tanto en sus fundamentos matemáticos como en su implementación en computadoras, un acercamiento a la forma de aprendizaje de las Redes Neuronales y una forma de estudio y análisis de las mismas.

CAPITULO I.

ANTECEDENTES DE LAS REDES NEURONALES

Resumen.

Cuando se habla de Redes Neuronales se piensa, por lo general, en fisiología sin imaginar que en campo computacional también este mismo concepto es aplicable.

Se necesita exponer, de una manera clara y concisa qué es la Neurocomputación, como surgieron la Neurocomputación y las Redes Neuronales y que impulsó su desarrollo.

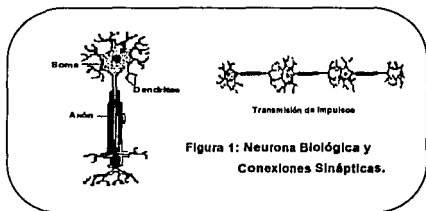
En este capítulo se describirá la historia de la Neurocomputación, qué son las Redes Neuronales, así como los conceptos básicos, no a detalle, usados en Neurocomputación.

La forma en que se expone este capítulo la Neurocomputación es de una manera sencilla y clara ya que los conceptos nuevos no son fácilmente entendidos por los neófitos en el tema.

De no hacer esto el trabajo sería completamente ininteligible.

1.1 Antecedentes.

Entender las formas en las que el cerebro usa el sistema nervioso central o neuronal para procesar patrones, sirve como llave para realizar modelos de redes neuronales artificiales. El funcionamiento del sistema nervioso, en los seres vivos, esta mediatizado únicamente por el paso de impulsos eléctricos a través de células llamadas **neuronas** (Figura 1) descubiertas en 1888 por el histólogo español *Santiago Ramón y Cajal*.



La ciencia que estudia el funcionamiento de tales células se conoce como **Neurofisiología**. Esta se desarrolló a partir de los trabajos realizados por el psicólogo estadounidense Williams James (1842-1910), quién fue el primero en publicar hechos sobre la estructura cerebral y sus funciones.

James, en su libro, publicado en 1890, escribió :

" El monto de actividad en cualquier punto de la corteza cerebral, es la suma de la tendencia de todos los otros puntos a descargar en él, tales tendencias comienzan proporcionalmente al número de veces que la excitación en cada otro punto esté acompañada del punto en cuestión; a la intensidad de tales excitaciones; y a la ausencia de cualquier punto rival funcionalmente desconectado con el primer punto al cual puedan divergir las descargas ".¹

Esto dio la noción tal vez intuitiva, pero correcta, de que la actividad de una neurona comienza con una función de la suma de sus entradas, con la historia de las correlaciones pasadas contribuyendo a las interconexiones entre ellas. Después de los estudios realizados por James, fueron necesarios más de 50 años para que se realizara el siguiente estudio sobre modelos de estructuras biológicas de sistemas neuronales.

A principios de los años 40's *W.C.McCulloch* y *W. Pitts* estudiaron la estructura particular del sistema visual de la rana².

Se investigó este sistema para averiguar las propiedades que ayudan a las ranas a reconocer "presa" y "enemigo", ya que éstas se alimentan de insectos que detectan solo con la vista.

¹ .-"Psicología (Breve Curso)" James, W. Ed. Holt, Nueva York 1890.

² .-"What The Frog Eye Tells To The Frog Brain" 1940.

Las ranas, apresan únicamente objetos en movimiento mientras que los objetos estacionarios nunca llaman su atención. Un objeto grande en movimiento provoca una reacción de huida. Para las ranas una forma que carece de movimiento carece de significado conductual.

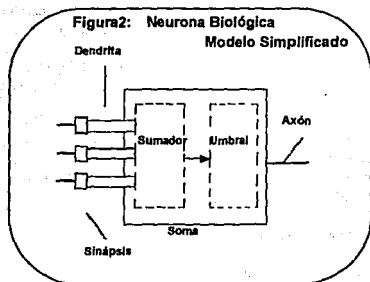
Al parecer, las ranas reconocen su presa y la seleccionan de entre otros muchos objetos de su alrededor, gracias a que ésta posee determinadas características como: movimiento, una cierta forma, algún contraste y quizá también un color determinado. Esta capacidad de las ranas para reconocer a su presa y capturarla, no se ve afectada por cambios en el medio ambiente, como pueden ser cambios de iluminación.

Las investigaciones realizadas por McCulloch y Pitts sobre los procesos neurológicos los condujeron a elaborar teoremas y modelos matemáticos para la representación de los sistemas neuronales biológicos. En 1943, McCulloch y Pitts, publicaron un artículo con sus resultados³. En dicho artículo, basan sus estudios en un modelo simplificado de la actividad neuronal biológica. Ya que la neurona biológica es un complejo aparato de cómputo, para realizar su modelo tomaron como base solamente 5 características físicas:

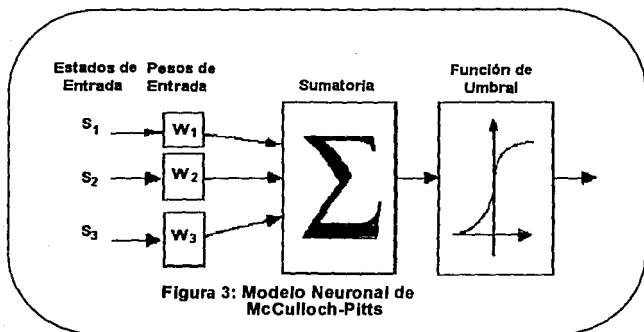
- La neurona biológica consiste básicamente de un cuerpo celular o **soma**, extensas ramificaciones complejas que sirven como entradas o **dendritas** y el canal de salida de la célula o **axón**. El axón lleva una señal eléctrica de una célula a otra, conectándose con el axón de otra célula, que a su vez se conecta con su respectiva dendrita a través de contactos especializados llamados **sinápsis**, que pueden cambiar positivamente o negativamente el potencial del axón. (Figura 2) Se conoce como **periodo de adición latente** al tiempo durante el cual la neurona se encuentra capacitada para detectar los valores en sus entradas, las sinápsis.
- Un cierto número fijo de sinápsis debe ser excitado dentro del periodo de adición latente para excitar una neurona en cualquier momento y este número es independiente de la actividad previa y posición en la neurona. La visión tradicional sostiene que la neurona realiza una función simple de **umbral** sumando valores de entrada. Si el resultado excede un cierto umbral se emite una señal de la neurona. Los valores asignados a cada entrada para incrementar o decrementar su importancia se conocen como pesos y como umbral al punto en el cual se puede detectar un estímulo.
- Cálculo del tiempo del **retardo sináptico**, que es el tiempo entre las sensaciones de entrada y su acción para transmitir un impulso de salida. El retardo sináptico es el único retardo significativo dentro del sistema nervioso.
- La actividad de cualquier sinápsis inhibitoria evita absolutamente la excitación de la neurona en ese momento.
- La forma e interconexión características de las neuronas parecen determinar su función en la actividad cerebral, por lo que la estructura de la red de interconexiones no cambia con el tiempo.

McCulloch y Pitts describieron el tiempo del periodo de adición latente como típicamente menor a 0.25 milisegundos y el del retardo sináptico en el orden de 0.5 milisegundos.

³ "Un cálculo lógico de las ideas que tienen lugar en la actividad nerviosa" Boletín de matemáticas biológicas, No. 3, pp.



La neurona descrita de esta forma, altamente simplificada, se conoce como **Modelo Neuronal de McCulloch-Pitts**. (Figura 3)



En los cerebros de los seres vivos las neuronas se agrupan formando interconexiones altamente complejas conocidas como **Redes de Interconexiones Neuronales** o simplemente **Redes Neuronales**.

McCulloch y Pitts descubrieron estas interconexiones en el sistema visual de la rana y asumieron esto para crear su modelo, basándose en la idea de que, la forma de las interconexiones determina la función de una red neuronal en la actividad cerebral.

Ellos, en su artículo de 1943, probaron que su modelo de redes neuronales podía representar cualquier expresión lógica finita; desarrollaron el uso de la arquitectura masiva en paralelo y dieron las bases para el desarrollo de modelos de redes neuronales y paradigmas⁴ de aprendizaje.

A fines de los años cuarenta se investigó el sistema nervioso del gato y se obtienen las primeras ideas de los modelos de redes neuronales y de algo que no aparecía en el sistema neuronal de la rana: **Aprendizaje.**

En 1949 Donald O. Hebb publica su libro titulado "**La organización del Comportamiento**". En este libro Hebb define el método de actualización de los pesos sinápticos y es el primero en utilizar el termino de **conexionismo**. Hebb presenta su método como un postulado neurofisiológico en el capítulo titulado "**El Perceptrón de primer estado: Crecimiento del ensamble**".

Hebb hace 4 contribuciones primarias a la teoría de las redes neuronales:

- Postula que en una red neuronal la información se almacena en los pesos de las sinápsis (conexiones).
- Sostiene un rango de pesos de aprendizaje que es proporcional al producto de los valores de activación de las neuronas; este postulado asume que los valores de activación son positivos.
- Asume que los pesos son simétricos. Esto es, el peso de una conexión de una neurona A a una neurona B es el mismo que de la neurona B a la neurona A.
- Postula una "**Teoría de ensamblamiento de células**" la cual sostiene que cuando ocurre el aprendizaje, fuerzas y patrones de las conexiones sinápticos (pesos) cambian y el ensamblamiento de células se crea por estos cambios. Puesto de otra manera, si ocurre una activación simultánea de un grupo de conexiones débiles repetidamente, estas células tratan de convertirse en un ensamble mas fuertemente conectado.

Actualmente nos referimos a los esquemas de aprendizaje de una red neuronal y su forma de actualización de pesos, en la mayoría de los modelos, como al "**Hebbiano**". Las cuatro contribuciones de Hebb a las redes neuronales están generalmente implementadas en las herramientas de redes neuronales actuales.

En 1958 un artículo publicado por Frank Rosenblatt definió una estructura de red neuronal artificial: **El Perceptrón**, desarrollada por el grupo del mismo nombre de la Universidad de Cornell. El perceptrón fue probablemente la primera herramienta de redes neuronales artificiales, ya que fue simulada a detalle en una IBM 704 en el Laboratorio Cornell de Aeronáutica.

Este se concibe como un dispositivo de reconocimiento de formas, que no está construido para reconocer un conjunto específico de modelos, sino que sea capaz de aprender a reconocer los modelos de un conjunto después de un número finito de pruebas, el perceptrón, era capaz de aprender a reconocer ciertos conjuntos de patrones como similares ó distintos mediante la modificación de sus conexiones.

⁴.-El término paradigmas se describirá en la sección 1.3.

De esta forma, Rosenblatt y su grupo, suponen que los pesos de cada neurona cambian con el tiempo a fin de conseguir que la red neuronal vaya variando y aprenda. Por lo tanto, esto puede describirse como "**Máquina de aprendizaje**".

Para realizar su modelo, Rosenblatt se basó en la visión biológica, específicamente en el sistema visual del gato y en los conocimientos que ya se tenían sobre éste.

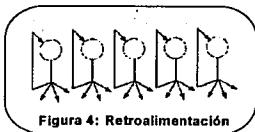
El perceptrón se realizó de la siguiente forma:

- El patrón ó modelo a reconocer, incide en el perceptrón sobre una **retina de unidades sensoriales** (las unidades sensoriales eran fotoceldas). Aunque es posible codificar cualquier entrada sensorial de forma que sirva de entrada a un banco de tales unidades, lo más natural (como sugiere la palabra "retina") es concebir el modelo como una entrada visual de luz y sombra.
- Una fotocelda que reciba una porción relativamente iluminada del modelo es activada, mientras que si recibe una porción relativamente oscura no será activada. Las unidades sensoriales se interconectan formando **redes de grupos de nodos de entrada**.
- Los grupos de nodos de entrada están constituidos de conjuntos aleatorios de unidades sensoriales o **células** en una región de la retina, cada grupo se encuentra conectado a una **unidad de asociación sencilla** o AU en la siguiente capa⁵ superior. Las unidades de asociación son los mecanismos donde se registran los estímulos detectados en las células que tienen conectadas. Las unidades de asociación se conectan bidireccionalmente a **unidades de respuesta** en la tercera capa superior.
- El objetivo del perceptrón era activar las unidades de respuesta correctas para cada clase particular de patrones de entrada. Cada unidad de respuesta típicamente tenía un gran número de conexiones a unidades de asociación, esto las proveía con una retroalimentación hacia las unidades de asociación.

Rosenblatt observó dos formas de implementar la retroalimentación de unidades de respuesta a unidades de asociación:

- En la primera forma la activación de una unidad de respuesta tenderían a excitar las unidades de asociación que manda la excitación de la unidad de respuesta esto se conoce como **retroalimentación positiva**.
- En la segunda conexiones inhibitorias existentes entre la unidad de respuesta y el complemento del conjunto de unidades de asociación que lo excitan, esto se conoce como **retroalimentación negativa**, por lo tanto inhibe la actividad en las unidades de asociación las cuales de esta forma no transmiten (Figura 4).

⁵ -El término "capa" se explica en la sección 1.2.



Con su modelo de perceptrón Rosenblatt pudo responder las siguientes preguntas : ¿ En que forma la información es almacenada o recordada?, ¿Cómo influencia la información almacenada al reconocimiento y comportamiento?. Sus respuestas fueron las siguientes:

*"La información se encuentra contenida en conexiones ó asociaciones más que en representaciones fotográficas. Una vez que la información almacenada toma la forma de nuevas conexiones ó canales de conexión en el sistema nervioso (o la creación de condiciones las cuales son funcionalmente equivalentes a nuevas conexiones), el nuevo estímulo hará uso de ésta nueva que ha sido creada activando automáticamente la respuesta apropiada sin requerir ningún proceso separado para su reconocimiento ó identificación."*⁶

El perceptrón era **auto-organizativo** ó **auto-asociativo** ya que ocurre que la respuesta que llega a ser dominante es esencialmente aleatoria.

Rosenblatt también describió sistemas donde ocurre **entrenamiento** ó **respuestas forzadas** y las bases del **entrenamiento supervisado** y **no supervisado**, que se utilizan actualmente en las Redes de Retropropagación y de Kohonen respectivamente.

Programar una red neuronal, donde ocurre entrenamiento, involucra dos fases:

- Especificar la función de las neuronas y su patrón de interconexiones.
- Entrenar la red con patrones de entrada simples.

Las neuronas, nodos o elementos computacionales están conectados por pesos que son típicamente adaptados mediante el uso para mejorar desempeños, esto se realiza mediante la implementación de algún algoritmo para manejar los pesos, estos algoritmos son conocidos como : **Algoritmos de Aprendizaje**.

En 1960 se implementa el primer algoritmo de aprendizaje.

Bernard Widrow y Marcian Hoff publicaron un artículo titulado "**Circuitaria de Switches Adaptativa**" el cuál se ha convertido en uno de los más importantes artículos en tecnología de redes neuronales.

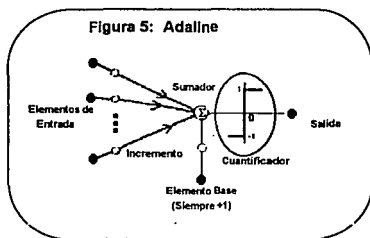
Ellos diseñaron herramientas de redes neuronales, las aplicaron en computadoras y además las implementaron en hardware.

⁶ .-"El perceptrón: Un modelo probabilístico para almacenamiento de información en el cerebro". Revista Psicológica. No. 65 pp.386-408 1958.

La implementación en hardware de las redes neuronales permitió el surgimiento de lo que se conoce como **Neuro-Computadoras** y la rama de la neurocomputación dedicada a su estudio y diseño.

Widrow y Hoff introdujeron un aparato llamado "**Adaline**" - llamado así como una contracción de **Adaptive Linear** -, éste consistía de una neurona sencilla con un número arbitrario de elementos de entrada que podían tomar valores de más o menos uno, y un elemento base que es siempre +1. Antes de ser sumados por el sumador neuronal, cada entrada incluyendo la base se modifica por un peso único que llamaron "**incremento**". A la salida del sumador se encuentra un cuantificador que se pone a +1 si la salida del sumador es mayor que 0 y a -1 si la salida es menor o igual que 0. (Figura 5)

El algoritmo de aprendizaje para el Adaline es particularmente ingenioso.



Uno de los principales problemas con los perceptrones es la cantidad de tiempo que les toma aprender a clasificar patrones correctamente. El algoritmo de Widrow y Hoff da un aprendizaje más rápido y más exacto. Este algoritmo es una forma de aprendizaje supervisado que ajusta los pesos (incrementos) de acuerdo al tamaño del error en la salida del sumador.

Widrow y Hoff demostraron que en la forma en que ellos ajustaban los pesos minimizaba la suma de errores en todos los conjuntos de patrones de entrenamiento.

Por esta razón el método Widrow-Hoff también es conocido como el **algoritmo de los mínimos cuadrados**.

El error es la diferencia entre lo que debería ser la salida del Adaline y la salida del sumador. La suma de errores cuadrados se obtiene midiendo el error de cada patrón presentado al Adaline, elevando al cuadrado cada valor y sumando entonces todos los valores cuadrados.

Minimizar la suma de errores cuadrados involucra un método de reducción de error llamado **gradiente descendiente** o **descenso empinado**. Matemáticamente, esto involucra la derivada parcial del error con respecto a los pesos. Widrow y Hoff demostraron que no se necesita aplicar las derivadas. Estas son proporcionales al error (y su signo) y al signo de la entrada.

Ellos además, demostraron que, para n entradas, reducir la medida del error del sumador en $1/n$ para cada entrada hará un buen trabajo de implementación del gradiente descendiente. De esta forma se ajusta cada peso, hasta que el error se reduce a $1/n$ del error total que se tenía al comenzar.

Este método provee de ajuste de pesos (aprendizaje) uniforme cuando la salida del clasificador es correcta. Esto es una mejora significativa sobre el perceptrón que solamente ajusta pesos cuando la salida del clasificador es incorrecta y es una razón de que el aprendizaje sea más rápido y más exacto.

Actualmente se utilizan algoritmos de aprendizaje con mejoras y extensiones al usado por Widrow y Hoff en las redes neuronales tipo retropropagación. Además sus trabajos de implementación física en hardware de redes neuronales se está empleando en las modernas arquitecturas de microprocesadores tipo VLSI y SVLSI.

1.2 Redes Neuronales Artificiales.

La neurona biológica es un complejo aparato de cómputo análogo, su forma e interconexiones características parecen determinar su función dentro de la actividad cerebral. Los cerebros en los seres vivos son computadoras naturales en paralelo, compuestas de aproximadamente 10^{11} neuronas, con cada neurona conectada a aproximadamente a otras 10^4 neuronas.

Las neuronas se activan mandando impulsos eléctricos que dejan su cuerpo celular y alcanzan la siguiente neurona a través de su unión sináptica. El rango de activación de las neuronas es muy bajo (cerca de 1000 pulsos por segundo), sin embargo el cerebro puede resolver problemas difíciles de visión y lenguaje en menos de 1 segundo.

El modelo funcional de una **neurona artificial** consiste de 3 secciones que corresponden al modelo simplificado de la neurona biológica.

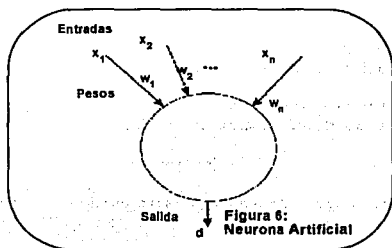
Las secciones de este modelo son:

- Conexiones de pesos de entrada.
- Una función de suma.
- Una función de umbral no lineal que genera la unidad de salida (usualmente encendido o apagado).

Las neuronas son aparatos sencillos que reciben un número de señales de entrada y basados en estas entradas generan una señal de salida.

La neurona artificial opera como un simple aparato de umbral, dependiendo de los estados de sus entradas (x_i) y la fuerza de sus conexiones o pesos (w_i). Este modelo se describe matemáticamente de la siguiente forma:

Una neurona es un elemento de proceso simple con m entradas x_1, x_2, \dots, x_m ($m \geq 1$) y una sola salida d . (Figura 6)



A cada x_i se le asocia un peso w_i , de los m pesos w_1, \dots, w_m , de esta forma la entrada efectiva I al elemento de proceso es el peso total de la entrada, o sea:

$$I = \sum w_i x_i \quad i = 1..m.$$

La neurona opera dentro de una escala discreta de tiempo $t=1,2,3,\dots,n$ y la activación de su salida en el instante $n+1$, queda determinada por la activación de sus entradas en el instante n , de acuerdo con la siguiente regla:

La neurona **dispara** es decir, envía un impulso a lo largo de su axón, en el instante $n+1$, si y solo si el peso total de sus entradas estimuladas en el instante n supera el valor del umbral θ de la neurona. Este valor de umbral es con frecuencia cero (0).

Si introducimos la notación :

$$\begin{aligned} m(t)=0 & \quad \text{para "m no dispara en el instante t".} \\ m(t)=1 & \quad \text{para "m dispara en el instante t".} \end{aligned}$$

donde m puede ser una salida axonal o una entrada sináptica de una neurona.

Podemos expresar la regla en forma simbólica:

$$d(n+1)=1 \quad \text{si y solo si } \sum w_i x_i(n) \geq \theta.$$

Si la entrada es mayor ó igual que el valor del umbral, en el momento $n+1$, el elemento de proceso dispara. Si la entrada es menor que el umbral, el elemento de proceso no dispara.

Una **entrada excitadora** tiende a causar que el elemento de proceso dispare. Una **entrada inhibidora** significa que la señal tiende a mantener el elemento sin disparar. Obsérvese que los pesos positivos $w_i > 0$ corresponden a sinápsis excitadoras (es decir, entradas a la neurona) mientras que un peso negativo $w_i < 0$ indica que x_i es una sinápsis inhibidora.

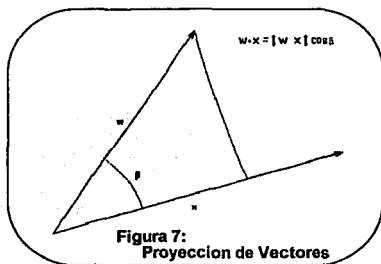
Podemos tomar a las entradas y a los pesos como vectores $x = (x_1 \dots x_m)$, $w = (w_1 \dots w_m)$. Si hacemos esto la señal de entrada total es solo el producto interno del vector de pesos y el vector de entradas.

Por geometría analítica el producto interno de 2 vectores es equivalente a la proyección de un vector sobre el otro. Matemáticamente el producto interno es equivalente a:

$$|w| x \cos(\beta)$$

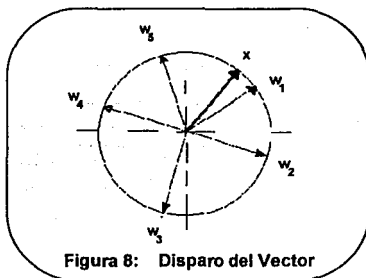
donde w es la norma o magnitud del vector w , x es la norma del vector x y β es el ángulo entre los 2 vectores. (Figura 7)

Esto es muy importante porque nos da una imagen visual de lo que estos vectores significan. La proyección del vector de pesos sobre el vector de entradas será la mayor cuando la dirección de los 2 vectores sea casi la misma y será la menor cuando la dirección de los 2 vectores sea muy diferente.



Esencialmente, esto es una medida de la cercanía de los 2 vectores. Claramente podemos observar que la entrada será mayor cuando el vector de pesos y el vector de entradas estén muy cerca.

Supóngase que se tiene un conjunto de elementos de proceso, cada uno con el mismo conjunto de señales de entrada, pero diferentes valores de pesos en las señales de entrada. Fácilmente se puede ver que el elemento de proceso que dispara es aquel con el vector de pesos más cercano al vector de entradas.(Figura 8)



Supóngase ahora, que queremos un sistema que reconozca entradas y las clasifique en uno de varios patrones.

Todo lo que tenemos que hacer es poner en los elementos de proceso, vectores de peso que apunten a las versiones clásicas de cada uno de los patrones que queremos reconocer. Entonces presentamos la señal de entrada de un ejemplo desconocido a la entrada de cada uno de los elementos de proceso simultáneamente, el elemento de proceso que concuerde mejor disparará con la mayor fuerza y esto nos dirá con cual patrón de entrada concuerda mas exactamente.

El manejo de los vectores involucrados en la red, resulta ser más conveniente si se fuerza a todos los vectores a tener la misma longitud; esto indica que es mas conveniente trabajar con sistemas de vectores normalizados donde todos los vectores han sido forzados a tener una magnitud de 1. Esto es usualmente fácil de hacer; dividiendo cada componente del vector entre la magnitud total del vector, funciona bien y elimina los problemas con factores de escala. Una de las mejores características de las redes neuronales es que si se comienza con pesos y entradas normalizadas, la dinámica de la red tiende a mantenerlos normalizados (o casi normalizados).

La salida de cada elemento de proceso se puede llevar como entrada a otros elementos de proceso. Esta también puede actuar como entrada al mismo elemento de proceso, si la red requiere de retroalimentación directa.

Las neuronas biológicas son mucho mas complicadas que las neuronas artificiales. Durante el funcionamiento normal de las neuronas biológicas, el proceso de la información involucra muchos procesos complejos. Las neuronas artificiales comienzan con un modelo de umbral simple y añaden complejidad solamente cuando se obtiene un nuevo conocimiento o el modelo resulta insuficiente.

Este punto ha sido central en el progreso hecho en el campo de las redes neuronales.

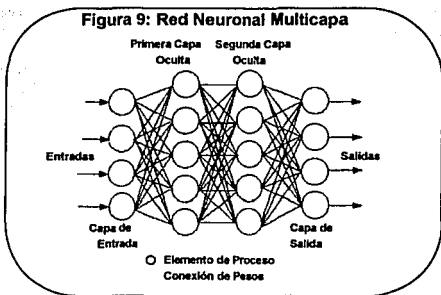
Una **Red Neuronal Artificial** se compone de 2 elementos primarios: **Elementos de proceso** ó neuronas e **Interconexiones**. La estructura de la red es determinada mediante la arquitectura de interconexión entre los elementos de proceso, las reglas que determinan cuando dispararán o no los elementos de proceso (**función de transferencia**) y las reglas que gobiernan los cambios, en la importancia relativa de las interconexiones individuales, de las entradas a los elementos de proceso (**leyes de entrenamiento**).

Una Red Neuronal Artificial se define como un sistema hecho de algunos **Elementos de Proceso Simple** (Neuronas) o EP's, todos con la misma escala de tiempo, altamente interconectados a base de ramificar la salida de cada neurona en un cierto número de líneas y conectar algunas de ellas, o todas, a las entradas de otras neuronas. Así una salida puede conducir a varias entradas neuronales, pero una entrada solo puede proceder de una salida neuronal como máximo.

Las interconexiones entre las neuronas o EP's, de un modelo de red neuronal, se pueden representar como un grafo dirigido. Usualmente los EP's se encuentran colocados en estructuras disjuntas llamadas **capas**.(Figura 9)

Las **líneas de entrada** de una red son aquellas entradas i_0, i_1, \dots, i_{m-1} de neuronas de la red que no están conectadas a salidas neuronales.

Las **líneas de salida** de una red son aquellas salidas p_0, p_1, \dots, p_{r-1} de neuronas de la red que no están conectadas a entradas neuronales.



Las neuronas procesan información por su estado dinámico de respuestas a entradas externas. Esto significa que, en su forma más básica, una computadora serial es un procesador central sencillo que puede direccionar un arreglo de localidades de memoria. Un sistema serial (aún un sistema standard en paralelo) es esencialmente secuencial: Todo ocurre en una secuencia determinística de operaciones.

En contraste una red neuronal no es ni secuencial ni necesariamente determinística. No tiene arreglos de memoria separada para almacenar datos. Los elementos de proceso que componen una red neuronal no son unidades de proceso altamente complejas. De hecho, una red neuronal está compuesta de muchos elementos de proceso simples que hacen un poco más que la simple suma de sus entradas. Una red neuronal no ejecuta una serie de instrucciones; responde, en paralelo, a las entradas que se le presentan.

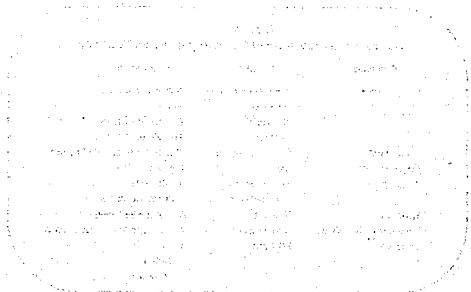
La Tabla 1 resume, la correspondencia entre el cerebro y las redes neuronales artificiales.

Tabla 1:
Correspondencia entre el Cerebro y una Red Neuronal

Elemento	Cerebro	Red Neuronal
Componentes	Dendritas y Axones	Entradas y Salidas
	Sinapsis	Pesos
	Sumador	Función Sumatoria
	Umbral	Función de Umbral
Organización	Red de Neuronas	Red de Elementos de Proceso
Procesamiento	Análogo	Digital o Análogo
Arquitectura	10-100 billones de Neuronas	1-1,000,000 de Procesadores
Hardware	Neurona	Aparato de Switcheo
Velocidad de Switcheo	1 milisegundo	1 nanosegundo a 1 nanosegundo
Tecnología	Biológica	Silicon Óptica Molecular

La principal diferencia entre los 2 es la velocidad de switcheo entre las neuronas biológicas y los aparatos de switcheo artificiales.

Una neurona switchea en el orden de 1 milisegundo, mientras que un transistor puede switchear en 1 nanosegundo. Esta disparidad demuestra el hecho, de que el cerebro usa paralelismo extensivo, ya que le toma menos de 100 pasos, el procesar una tarea compleja -de tipo estimulo-reacción como visión, coordinación, comunicación, etc.- en menos de medio segundo.



1.3 Paradigmas.

Los patrones de interconexión de neuronas es quizá la característica más distinguible de un modelo de red neuronal.

Cada modelo lleva asociado un algoritmo, a estas 2 características se les conoce como **paradigma**.

Existen muchas clases de modelos de redes neuronales artificiales, se han diseñado más de 50 tipos diferentes de redes neuronales de los cuales solo 15 modelos están en uso común.

Algunas de las más conocidas son:

- Red de Hopfield/Kohonen.
- Perceptrón.
- Perceptrón de capa múltiple/Regla Delta.
- Retropropagación (Back propagation BP).
- Máquina de Boltzman.
- Contrapropagación (Counterpropagation).
- Mapa de auto-organización.
- Neocognitrón.

1.4 Perspectivas.

En 1969 Marvin Minsky, considerado en aquella época uno de los más importantes investigadores en lo que se refería a redes neuronales e inteligencia artificial, en su libro llamado "Perceptrones" concluyó que toda investigación en redes neuronales resultaba estéril.

Esta declaración evitó el desarrollo de las redes neuronales desde 1969 hasta 1987.

Las redes neuronales artificiales han experimentado un crecimiento espectacular en los últimos años, con la aparición del procesamiento distribuido en paralelo y la demostración del error de Minsky en 1987 por Terry Sejnowski, que en la Universidad Johns Hopkins desarrolló NETTALK, una computadora neuronal que aprendió a leer. A partir de ese momento las redes neuronales reciben un nuevo y sorprendente desarrollo; se desarrollan para computadoras personales y se ligan de nuevo las redes neuronales con la inteligencia artificial.

Otros factores para este desarrollo han sido las nuevas tipologías y algoritmos de red, las técnicas de implementación analógica VLSI (Circuitos de muy alta escala de integración) ya que solo éstos pueden realizar el verdadero cómputo potencial del paralelismo masivo, teniendo rangos de integración de 1Mbit/100 mm., la comercialización de las nuevas arquitecturas conectivas donde se interconectan de 50,000 a 100,000 procesadores elementales, a esto se le conoce como TRANSPUTER y la teoría de que el paralelismo masivo es esencial para grandes logros en procesos tipo humano.

Los modelos de redes neuronales artificiales han sido estudiados con la esperanza de lograr funciones tipo humano en los campos de lenguaje, reconocimiento de imágenes y patrones. Las redes neuronales se usan a una amplia clase de aplicaciones que comprenden una variedad de problemas reales que incluyen:

- Procesamiento de imágenes.
- Procesamiento de habla.
- Procesamiento de conocimiento inexacto.
- Procesamiento de lenguaje natural.
- Procesamiento de sentidos.
- Planeación.
- Pronóstico.
- Optimización.

Para estos problemas las redes neuronales artificiales han demostrado un número de propiedades análogas al cerebro:

- Asociación.
- Generalización.
- Búsqueda en paralelo.
- Aprendizaje.
- Flexibilidad.

Las redes neuronales han demostrado habilidad para dar soluciones simples y poderosas en áreas que por muchos años han retado la realización de cómputo convencional.

Muchos de los trabajos en esta área han estado enfocados a problemas en pequeña escala, las redes neuronales artificiales prometen ser extensibles a problemas de tamaño y tiempo real en reconocimiento de patrones.

En los últimos meses ha aparecido un enorme crecimiento en el interés sobre los sistemas de redes neuronales artificiales (Redes Neuronales y Neuro-Computadoras), debido al anuncio en el Japón del desarrollo de su proyecto de **Sexta Generación** de computadoras.

El proyecto consiste en mucho más que esfuerzos en ciencias computacionales. De hecho, el vocero japonés acuñó el término "**Inteligencia Natural**", para explicar el concepto de que las computadoras podrán desplegar **comportamientos** basados en modelos biológicos más que en modelos de silicio.

Esto hace pensar que las redes neuronales tendrán un gran impacto en la industria de las computadoras.

CAPITULO II

FUNDAMENTOS DE LAS REDES NEURONALES.

Resumen.

En el capitulo anterior se dieron los antecedentes de la Neurocomputación y las Redes Neuronales.

En este capitulo se explicaran detalles técnicos y matemáticos sobre los principios y fundamentos básicos de las Redes Neuronales. Se describirá de manera mas extensa y técnica cuales son los conceptos básicos de las Redes Neuronales.

Como ya se vio las redes neuronales se representan mediante grafos dirigidos donde las neuronas ó EP's son representadas por los nodos y las conexiones representan los pesos ó valores de las entradas. Estos pesos y el estado de los EP's representan la información de la red. Una red neuronal que aprende a reconocer patrones, lo hace ajustando los pesos de las conexiones de los EP's.

Una red recuerda patrones basándose en la información obtenida de asociaciones ya establecidas entre patrones de entrada y salida. Así, las redes neuronales son inherentemente adaptativas dado que se ajustan a los datos imprecisos, ambiguos y de naturaleza falible, del mundo real.

Podemos caracterizar a las redes neuronales por algunas propiedades claves:

- Topología de la red.
- Procedimiento de recuerdo (Función de activación).
- Procedimientos de aprendizaje y entrenamiento.
- Valores de entrada.

El entendimiento de este capitulo es fundamental para comprender lo que mas adelante se diga sobre el tema.

II.1 Topología.

La **topología** de una red se encuentra determinada por el modelo característico del patrón de interconexiones de las neuronas. Usualmente los arreglos de EP's se encuentran en estructuras disjuntas llamadas **capas**.

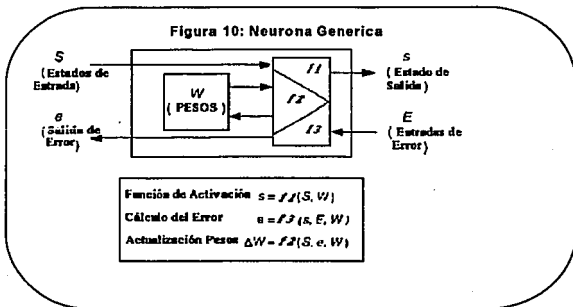
Los primeros modelos de redes neuronales, tales como el perceptrón, estaban constituidos por una **capa simple** donde cada entrada de la red conectaba a todos los EP's. La información fluía a través de la red, de entrada a salida, sin retroalimentación de la salida. Los modelos de este tipo son llamados **redes de alimentación hacia adelante** (feed-forward).

Los modelos más recientes han extendido esta idea a estructura de **capas múltiples** hacia adelante. En las redes de capas múltiples o **multicapa** las capas centrales no son visibles para las capas externas, que son la **capa de entrada**, donde se reciben los valores de los patrones y la **capa de salida**, donde se obtienen los resultados, por esta razón las capas centrales se conocen como **capas ocultas**. Estas capas ocultas también reciben y procesan la información recibida de las otras capas. (Figura 9). Otros modelos han introducido el concepto de **conexiones retroalimentadas**. Las redes retroalimentadas realizan el procedimiento de aprendizaje y entrenamiento, retroalimentando sus resultados (errores) de una capa exitosa a la anterior.

En este punto es necesario introducir el concepto de **neurona artificial genérica** o simplemente **neurona genérica**, que incorpora las principales variaciones de los procedimientos de aprendizaje y recuerdo encontrados en las diferentes topologías de redes neuronales.

La neurona genérica comprende tres funciones específicas (f_1, f_2, f_3), una tabla de pesos (W) y un conjunto de señales de entrada (S, E) y de salida (s, e).

Una neurona recibe los estados de entrada S de las neuronas de la capa previa y avanza su estado de salida s . De la misma forma un conjunto de errores de entrada E y un error de salida e dan alimentación en la red. f_1 funciona como la función de activación, f_2 como la función de actualización de pesos y f_3 como la función del cálculo de error. (Figura 10)



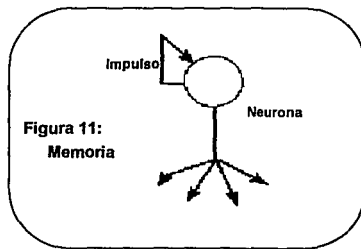
II.2 Función de activación, Funcionamiento (Procedimiento de Recuerdo).

Una red neuronal tiene propiedades adicionales como memoria.

Parece evidente que los seres humanos tenemos 2 tipos de memoria: a **corto plazo** y a **largo plazo**, además parece ser que tenemos que retener una idea en la memoria de corto plazo durante un buen rato, antes de ser transferida a la de largo plazo.

El tiempo necesario para esa transferencia varía, según estimaciones, en un promedio de 20 minutos. Al entrar al estado de coma, los recuerdos de los 20 minutos anteriores se pierden para siempre, es decir, son transferidos a la memoria de largo plazo. Hoy en día se acepta generalmente que esta memoria de corto plazo es precisamente la que proporcionamos a nuestra red neuronal (el paso de complicados impulsos eléctricos a través de la red). Y parece ser que si tal actividad transitoria persiste el tiempo suficiente, acaba por modificar la red.

En la Figura 11, la neurona posee una memoria de corto plazo que le permite recordar, si ha activado o no la entrada; tal memoria esta almacenada en la recurrencia del impulso en el bucle.



Se tendría una memoria de largo plazo, si su umbral pasara de 1 a 0, pues entonces la memoria persistiría aún en caso de que la recurrencia se extinguiera.

Biológica, teórica y experimentalmente se han postulado diversas teorías sobre la memoria a largo plazo para exhibir ciertos aspectos de aprendizaje.

El mecanismo exacto, para la memoria a largo plazo nos es aún desconocido, sin embargo se han propuesto varios mecanismos. Uno de los mecanismos propuestos consiste en que las conexiones neuronales, aumenten con el uso repetido, incrementando así el peso de la correspondiente entrada sináptica, lo cual hace restablecer los patrones de impulso, usando esa sinápsis y como consecuencia acceder a la correspondiente memoria.

Cambiando la magnitud del impulso transmitido por una conexión (cambiando los pesos de las entradas de las neuronas) se obtiene una memoria de largo plazo. Estos cambios dependen de la actividad pretérita de las conexiones neuronales. La regla que rige esta dependencia se llama **regla de refuerzo**, porque está pensada para reforzar las respuestas correctas de la red a los estímulos que le son presentados.

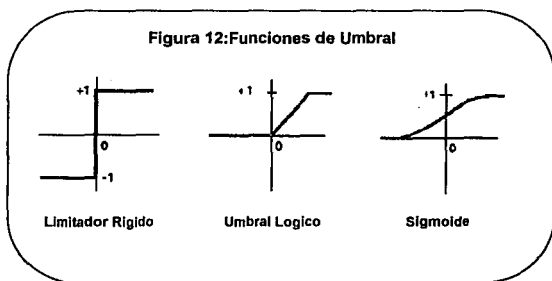
El procedimiento de recuerdo se encuentra especificado por la **función de activación f** , la cual comprende la **regla de propagación net** y la **función de umbral T** :

$$net = f(S, W) \text{ y } s = T(net)$$

En su forma más simple la regla de propagación calcula la suma de los pesos de las entradas, modificados por un valor θ que define el ajuste de la neurona:

$$net = \sum S \cdot W - \theta$$

La función de umbral (no lineal) de la regla de propagación calcula el estado de la neurona. Los tres tipos más comunes de no linealidades son el **limitador rígido**, el **umbral lógico** y la **sigmoide**. (Figura 12)



Otros tipos de umbral, menos comunes, pueden incluir integración temporal u otros tipos de dependencia de tiempo y operaciones matemáticas más complejas.

II.3 Procedimientos de Entrenamiento y Aprendizaje.

Este proceso, típicamente iterativo presenta muchos conjuntos de patrones de entrenamiento. Este es, por lo tanto, computacionalmente más intensivo que el procedimiento de recuerdo.

Los procedimientos de aprendizaje pueden realizarse de dos formas:

1) El sistema somete una pareja de entrenamiento, consistente de un patrón de entrada y un objetivo de salida, a la red. La red ajusta los pesos basada en el valor e del error de los EP's (usualmente la diferencia entre la salida esperada y la salida calculada para cada EP) así que la diferencia disminuye con cada ciclo. A esta forma de aprendizaje se le conoce como **aprendizaje supervisado**.

2) El procedimiento de aprendizaje clasifica los patrones de entrada sin requerir información de los objetivos de salida. En cada procedimiento, la red debe detectar el patrón de regularidades y el agrupamiento para cada entrada aplicada para producir una salida consistente. Este tipo de procedimiento de aprendizaje se conoce como **aprendizaje no supervisado**. Se creé que este tipo de entrenamiento corresponde con los modelos más plausibles de entrenamiento biológico.

Los algoritmos de aprendizaje generalmente involucran 2 funciones (en adición a la función de activación). La función de cálculo de error $e = f_3(s, E, W)$ controla la actualización de los pesos, mientras la función $\Delta W = f_2(S, s, W)$ actualiza los pesos.

Muchos modelos emplean como función de actualización de pesos alguna variación de la regla de aprendizaje del Hebbiano. Esta regla sostiene que los pesos entre neuronas deben ser reforzados proporcionalmente a sus actividades:

$$\Delta w_{ij} = S_i s_j$$

donde W_{ij} es el peso asociado a la conexión entre EP i y EP j , S_i es el valor del estado de EP i , y s_j es el valor del estado de EP j .

Otras reglas de aprendizaje incluyen la Regla Delta, Algoritmo de Competencia, la Regla de Energía Mínima de Hopfield, el Algoritmo de Aprendizaje de Boltzmann, la Regla Delta Generalizada y las unidades Sigma-Pi.

II.4 Valores de entrada.

Podemos caracterizar los modelos de redes neuronales por los valores de entrada que acepta y el rango que estos adoptan; éstos pueden ser entradas evaluadas **binariamente** ó **continuamente**.

Esto significa que los valores de entrada pueden ser como un valor sencillo continuo ó como un conjunto de rangos a los que les son asignados valores binarios.

Algunas veces para que los datos tengan sentido para una red neuronal, éstos deben ser convertidos de un formato a otro.

CAPITULO III

ALGORITMOS Y SUS FUNDAMENTOS MATEMATICOS

Resumen

Los modelos de **redes neuronales artificiales** o simplemente redes neuronales adoptan muchos nombres tales como modelos coleccionistas, modelos de proceso distribuido en paralelo y sistemas neuromórficos. Los modelos de redes neuronales tienen gran potencial en áreas como el habla y el reconocimiento de imágenes, donde se persiguen muchas hipótesis en paralelo, se requieren altos rangos de computación y los mejores sistemas actuales están lejos de alcanzar desempeños humanos. En vez de realizar un programa de instrucciones secuenciales como en una máquina Von Neumann, los modelos de redes neuronales exploran muchas hipótesis participantes simultáneamente, usando masivamente las redes paralelas compuestas de muchos elementos computacionales conectados por uniones con pesos variables.

Los beneficios potenciales de las redes neuronales se extienden más allá de los altos rangos de computación dados por paralelismo masivo. Las redes neuronales dan típicamente un mayor grado de **robustez** es decir mayor tolerancia a las fallas que las computadoras secuenciales Von Neumann, porque hay muchos mas nodos procesando con cada conexión local primaria, de esta forma, un daño a pocos nodos o uniones no necesariamente perjudica el desempeño total significativamente. Muchos algoritmos de redes neuronales también adaptan conexiones de peso en tiempo para mejorar el desempeño basado en los resultados actuales.

La **adaptación ó aprendizaje** es el mayor enfoque de las investigaciones en redes neuronales. La habilidad para adaptarse y continuar aprendiendo es esencial en áreas como el reconocimiento del habla donde el entrenamiento de los datos está limitado y nuevos hablantes, nuevas palabras, nuevos dialectos, nuevas frases y nuevos medioambientes se encuentran continuamente. La adaptación también da un grado de robustez por compensación para variaciones menores en características de elementos de proceso.

Las técnicas estadísticas tradicionales son no adaptativas pero típicamente procesan todos los datos entrenados simultáneamente antes de ser usados con nuevos datos.

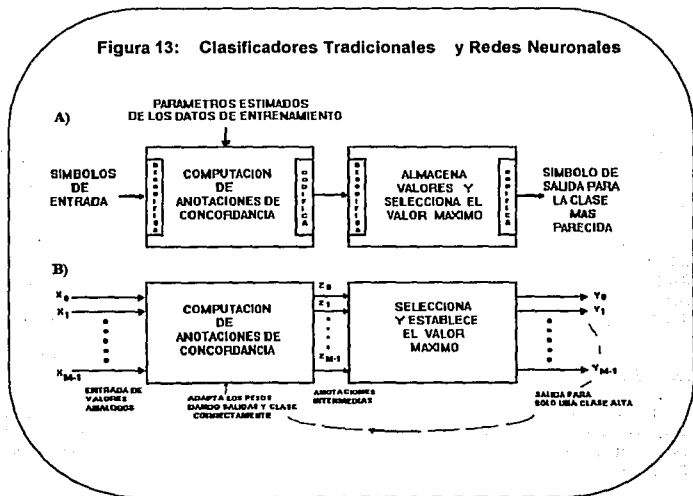
Los clasificadores de redes neuronales son también no paramétricos y hacen suposiciones más débiles acerca de la forma bajo las distribuciones, que los clasificadores estadísticos tradicionales. Estos pueden así ser más robustos cuando las distribuciones son generadas por procesos no lineales y son fuertemente no Gaussianas.

III. 1 Redes Neuronales y Clasificadores Tradicionales.

Un concepto central en el reconocimiento de patrones es el de **discriminantes**. La idea es que un sistema de reconocimiento de patrones aprenda adaptativamente de su experiencia y obtenga varios discriminantes, cada uno apropiado para su propósito. Por ejemplo, si solamente es de interés los miembros de una clase, el sistema aprende de observaciones de patrones que son identificados por clases e infiere un discriminante para clasificación.

Se conoce como **clasificadores tradicionales** a los algoritmos y técnicas usadas tradicionalmente ó más comúnmente para el reconocimiento de patrones. Estos se desarrollaron en los años 70's principalmente para los campos de visión, reconocimiento de imágenes, sistemas sensores para robótica y reconocimiento de voz.

Los diagramas de bloques de **clasificadores tradicionales (A)** y de **redes neuronales (B)** se presentan en la Figura 13.



Ambos tipos de clasificadores determinan cuál de las M clases es más representativa de un patrón de entrada estático y desconocido, conteniendo N elementos de entrada.

En un reconocedor de voz las entradas pueden ser valores envueltos en la salida de un banco de filtros de la prueba de un analizador espectral en un instante de tiempo y las clases pueden representar diferentes vocales. En un clasificador de imágenes la entrada puede ser la escala de niveles de gris de cada pixel de una foto y las clases pueden representar diferentes objetos.

Los clasificadores tradicionales, en la parte alta de la Figura 13 contienen 2 etapas.

- La primera etapa anota la concordancia computada para cada clase. La entrada a la primera etapa son símbolos que representan valores de los N elementos de entrada.
- En la segunda etapa, se selecciona la clase con la anotación máxima. Los datos de la etapa anterior son accedidos secuencialmente y decodificados de la forma simbólica externa, en una representación interna usada para realizar operaciones aritméticas y simbólicas. Un algoritmo computa una anotación de concordancia para cada una de las M clases la cuál indica que tan cercanamente la clase concuerda con el patrón de ejemplo para cada clase. Este patrón de ejemplo es el patrón que es más representativo para cada clase.

En muchas situaciones, un modelo probabilístico es usado para modelar la generación de patrones de entrada para ejemplos y las anotaciones de concordancia, representan la semejanza o probabilidad que el patrón de entrada generó para cada uno de los M posibles ejemplos. En estos casos, típicamente, se realizan fuertes suposiciones, bajo las distribuciones de los elementos de entrada. Los parámetros de las distribuciones pueden entonces ser estimados usando entrenamiento de datos, como se mostró en la Figura 13. Distribuciones Gaussianas multivariadas son usadas con frecuencia conduciendo a algoritmos relativamente simples para calcular **anotaciones de concordancia**. Las anotaciones de concordancia son codificadas en representaciones simbólicas y enviadas secuencialmente a la segunda fase del clasificador. Aquí son descodificadas y la clase con la máxima anotación se selecciona. Un símbolo representando esa clase se manda para completar la tarea de clasificación.

En su forma quintaesencial, en el reconocimiento de patrones, ambos, el aprendizaje y la fase de reconocimiento deben de ser realizados con procesos concurrentes distribuidos y el proceso completo debe ser poderoso y rápido.

Este parece ser el caso en los sistemas neuronales biológicos y también debe ser el caso para los sistemas computacionales de reconocimiento de patrones adaptativos. Mostremos ahora como es posible implementar el reconocimiento adaptativo de patrones con los algoritmos y arquitecturas de sistemas basadas en proceso distribuido en paralelo. Un clasificador de red neuronal adaptativo se muestra en la parte baja de la Figura 13.

Aquí los valores de entrada son puestos en paralelo a la primera fase vía N conexiones de entrada. Cada conexión lleva un valor análogo, el cuál puede tomar 2 niveles para entradas binarias o puede variar sobre un gran rango de valores continuos de entrada.

- En la primera fase se computan las anotaciones de concordancia y se sacan estas anotaciones en paralelo a la siguiente fase sobre M líneas análogas de salida. Aquí el máximo de estos valores es seleccionado y realzado.
- La segunda fase tiene una salida para cada una de las M clases, después de que se completa la clasificación, solamente aquella salida correspondiente a la clase mas común será reforzada o **alta**; las otras salidas serán **bajas**.

Note que en este diseño, existen salidas para cada clase y que esta multiplicidad de salidas debe preservarse en posteriores fases de procesamiento tan grandes como sean las clases consideradas distintas.

En el sistema de clasificación mas sencillo estas líneas de salida pueden ir directamente a luces con etiquetas que especifican identidades de clase. En casos más complicados estas pueden ir a fases posteriores de procesamiento donde las entradas de otras modalidades o dependencias temporales se toman en consideración.

Quando se da la clase correcta entonces esta información y las salidas del clasificador pueden ser retroalimentados a la primera fase del clasificador para adaptar pesos usando un algoritmo de aprendizaje como se muestra en la Figura 13. La adaptación dará una respuesta correcta más semejantemente para la entrada exitosa de patrones que son similares al patrón actual.

Las entradas en paralelo requeridas por los clasificadores de redes neuronales sugieren que las implementaciones de hardware en tiempo real deberán incluir **preprocesadores** de propósito especial. Una estrategia para diseñar tales procesadores es construir preprocesadores basados biológicamente, modelados con los sistemas sensores humanos, como ejemplo, un preprocesador para clasificación de imágenes se modela usando para ello el modelo de la retina humana y se diseña usando circuitería análoga VLSI. También han sido construidos los preprocesadores de banco de filtros para reconocimiento del lenguaje estos son analogías crudas del caracol del oído humano. Los algoritmos mas recientes de preprocesadores basados fisiológicamente para el reconocimiento del habla intentan dar información similar a la que se dispone en el nervio auditivo. Muchos de estos algoritmos incluyen bancos de filtros con análisis espectral, control automático de ganancia y procesamientos que usan periodicidad ó información sincrónica, en adición a las salidas envueltas del filtro suavizado.

Los clasificadores en la Figura 13 pueden realizar 3 diferentes tareas :

- **Primero**, como se describe anteriormente, pueden identificar cuales clases representan mejor un patrón de entrada donde se asume que las entradas han sido alteradas por ruido ó algún otro proceso. Este es un problema clásico de teoría de decisiones.
- **Segundo**, los clasificadores pueden ser usados como una **memoria de contenido direccionable ó memoria asociativa**, donde se decide la clase ejemplar y el patrón de entrada se usa para determinar cual ejemplo se producirá. Una memoria de contenido direccionable es útil cuando solamente una parte del patrón de entrada esta disponible y se requiere el patrón completo, como en recortes bibliográficos de revistas de referencia para información parcial.

Esto normalmente requiere una tercera fase para regenerar el ejemplo de la clase más semejante. Una fase adicional es innecesaria para algunas redes neuronales como la red Hopfield, las cuales están diseñadas específicamente como memorias de contenido direccionable.

- **Tercero**, otra tarea que pueden realizar estos clasificadores es la **cuantificación del vector**⁸ ó **agrupar**⁹ las N clases en M grupos. Las cuantificaciones de vector se realizan en sistemas de transmisión de imagen y habla para reducir el número de bits necesarios para transmitir datos análogos. En aplicaciones de reconocimiento de habla e imagen son usados para comprimir la cantidad de datos que deben ser procesados sin perder información importante. En estas aplicaciones el número de grupos puede ser pre-especificado o puede permitirsele crecer hasta un límite determinado por el número de nodos disponibles en la primera fase.

Diseñar redes neuronales para resolver problemas y estudiar también redes biológicas reales puede también cambiar nuestra forma de pensar y llevarnos a nuevas vistas y perfeccionamientos algorítmicos.

Trabajar en modelos de redes neuronales artificiales tiene una larga historia. El desarrollo de detallados modelos matemáticos comenzó hace más de 40 años con los trabajos de McCulloch y Pitts, Hebb, Rosenblatt, Widrow, y otros. Trabajos más recientes por Hopfield, Rumelhart y McClelland, Sejnowski, Feldman, Grossberg, y otros han dado un nuevo resurgimiento al campo.

Dada la gran diversidad de modelos de redes neuronales, es necesario clasificarlos de alguna manera.

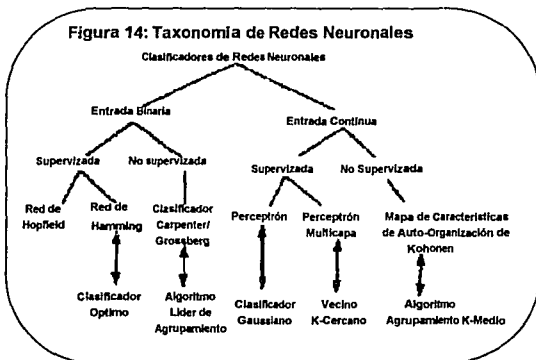
⁸ "Vector Quantization in speech coding" IEEE Proceedings, 1551-1588

⁹ "Clustering Algorithms" Hartigan J. Ed. John Wiley, New York.

III. 2 Taxonomía de las redes neuronales.

A la forma de clasificar los modelos de redes neuronales se le conoce como **taxonomía**, dado que la mayoría de estos modelos tienen un remoto origen biológico. Para realizar la taxonomía de los modelos de redes neuronales se clasifican, igual que en una taxonomía biológica, de acuerdo con sus propiedades características.

Primero se toma el tipo de valores de entrada que la red recibe: binarios o continuos. Después las redes se dividen de acuerdo a la forma de su procedimiento de aprendizaje/entrenamiento: supervisado ó no supervisado. (Figura 14)



Para este análisis se tomaron únicamente los seis modelos de redes neuronales más importantes, (Red de Hopfield, Red de Hamming, Clasificador de Carpenter/Grossberg, Perceptrón, Perceptrón de Multicapa, Mapa de Características de Auto-organización de Kohonen), ya que muchos de los modelos que actualmente se conocen son extensiones, modificaciones, hibridaciones ó mejoras de estos modelos.

Las redes entrenadas con supervisión tales como la red de Hopfield y los perceptrones son usadas como memorias asociativas ó como clasificadores. Estas redes están provistas de información lateral o etiquetas que especifican la clase correcta para nuevos patrones de entrada durante el entrenamiento. Muchos clasificadores estadísticos tradicionales, tales como los clasificadores Gaussianos son entrenados con supervisión, información etiquetada de entrenamiento.

Las redes entrenadas sin supervisión, tales como las redes que forman Mapas de Características de Kohonen, son usadas como cuantificadores de vectores o para formar grupos. En estas redes, durante el entrenamiento, no se provee información concerniente a la clase correcta. Los algoritmos clásicos de K-Cercano y del Líder de Agrupamiento son entrenados sin supervisión.

Otra diferencia entre las redes que pueden ser usada para su clasificación es si éstas soportan el entrenamiento adaptativo. Todas estas redes pueden ser entrenadas adaptativamente, pero la red de Hopfield y la red de Hamming son usadas generalmente con pesos fijos.

En la parte baja de la Figura 14 se encuentran los algoritmos clásicos que son muy similares o que realizan la misma función a su correspondiente red neuronal. En algunos casos una red implementa un algoritmo clásico exactamente.

Por ejemplo, la red de Hamming es una implementación de Red Neuronal del clasificador óptimo para patrones binarios corruptos por ruido aleatorio. También se puede mostrar que la estructura del perceptrón realiza los cálculos requeridos por un clasificador Gaussiano cuando los pesos y umbrales son seleccionados apropiadamente.

En otros casos los algoritmos de redes neuronales son completamente diferentes de los algoritmos clásicos.

Por ejemplo, los perceptrones entrenados utilizando el procedimiento de convergencia Perceptrón se comportan completamente diferente a los clasificadores Gaussianos. De la misma forma, la Red de Kohonen no realiza el entrenamiento iterativo del algoritmo K-Cercano. De hecho cada nuevo patrón se presenta solamente una vez y los pesos son modificados después de cada presentación; sin embargo, la red de Kohonen forma un número pre-especificado de grupos como en el algoritmo K-Cercano, donde la K se refiere al número de grupos formados.

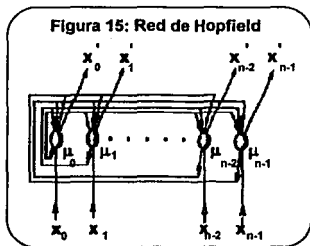
A continuación se describen los seis modelos usados para representar la taxonomía de la redes neuronales. Se realiza esta revisión para describir el propósito y diseño de cada red en detalle, para relacionar cada red a patrones existentes de clasificación y algoritmos de agrupamiento que son normalmente implementados en computadoras secuenciales Von Neumann y para ilustrar los principios de diseño usados para obtener paralelismo usando elementos de procesamiento tipo neuronal.

III. 3 Red de Hopfield.

La Red Neuronal de Hopfield tiene un remarcado interés, por su trabajo intensivo en diferentes aplicaciones. Esta red puede ser usada como una memoria asociativa y para la solución de problemas de optimización. La Red de Hopfield es normalmente usada con entradas binarias.

Las redes con entradas binarias son las más apropiadas cuando son posibles representaciones binarias exactas, como imágenes en blanco y negro donde los elementos de entrada son valores de pixel, o con texto ASCII donde los valores de entrada pueden representar bits en la representación ASCII de 8 bits de cada caracter. Este tipo de redes es menos apropiada cuando los valores de entradas son continuos, ya que un problema fundamental de representación puede estar direccionado a convertir valores análogos a valores binarios.

En la Figura 15 se muestra una red de Hopfield usada como memoria de contenido direccionable, esta red usando linealidades de limitadores rígidos y entradas y salidas binarias tomando los valores de +1 y -1. La salida de cada nodo es retroalimentada a todos los demás nodos mediante los pesos, esto se denota como t_{ij} .



La operación de esta red, descrita en la Figura 16, es la siguiente:

- Primero, los pesos son puestos usando la "receta" del patrón ejemplo para todas las clases.
- Entonces, en un momento 0 un patrón desconocido es puesto en la red, forzando la salida de la red a empatar con el patrón desconocido.
- Siguiendo esta inicialización, la red itera en pasos de tiempo discreto usando la fórmula dada.
- Se considera que la red ha convergido cuando las salidas no cambian en iteraciones sucesivas.
- El patrón especificado por los nodos de salida después de la convergencia es la salida de la red.

Hopfield y otros han probado que esta red converge cuando los pesos son simétricos ($t_{ij} = t_{ji}$) y los nodos de salida son actualizados asincrónicamente usando las ecuaciones de la Figura 16. Hopfield también demostró que la red converge cuando se usan no linealidades clasificadas similares a la sigmoideal.

Figura 16: Algoritmo de la Red de Hopfield

Paso 1.-Asignar el peso de las conexiones

$$t_{ij} = \begin{cases} \sum_{s=0}^{M-1} w_i^s w_j^s & i \neq j \\ 0, w_i^s, 0 \leq i, j \leq N-1 \end{cases}$$

En esta formula t_{ij} es el peso de la conexión del nodo i al nodo j y x_i el cual puede ser +1 ó -1 es elemento i del ejemplo para la clase s .

Paso 2.-Inicializar con patrones de entrada desconocidos.

$$x_i(0) = x_i, \quad 0 \leq i \leq N-1$$

En esta formula $m(i)$ es la salida del nodo i en el momento t y x_i el cual puede ser +1 ó -1 es el elemento i del patron de entrada.

Paso 3.-Iterar Hasta la Convergencia.

$$x_i(t+1) = f \left[\sum_{j=0}^{N-1} t_{ij} x_j(t) \right], \quad 0 \leq j \leq N-1$$

La funcion f es una no-linealidad de limitador rígido. El proceso se repite hasta que los nodos de salida se mantengan sin cambios al iterar. Los nodos de salida representan entonces al patron de ejemplo que concuerda mejor con la entrada desconocida.

Paso 4.-Repetir (ir al paso 2).

Cuando la red de Hopfield se usa como una memoria asociativa, la salida de la red después de la convergencia se usa directamente como la memoria restaurada completa.

Cuando la red de Hopfield es usada como un clasificador, la salida de la red, después de la convergencia puede ser comparada a los M ejemplos para determinar si este empata con un ejemplo exactamente. Si lo hace la salida es la clase cuyo ejemplo empató con el patrón de salida, si no lo hace entonces ocurre una "no concordancia".

El comportamiento de la red de Hopfield se ilustra en la Figura 17. Una red de Hopfield con 120 nodos y por lo tanto 14,400 pesos, se entrenó para recordar 8 patrones de ejemplos (A). Estos patrones de blanco y negro tipo dígitos contienen 120 pixeles cada uno, los elementos de entrada a la red toman los valores de +1 para un pixel negro y -1 para un pixel blanco. En el ejemplo (B) el patrón para el dígito "3" fue corrupto al invertir aleatoriamente cada bit independientemente, de +1 a -1 y viceversa con probabilidad de 0.25. Este patrón se presentó a la red en tiempo 0.



Se muestran sucesivamente los patrones producidos por la red en los tiempos de 0 a 7. Cada vez que la red itera el patrón se parece más al patrón ejemplo correcto hasta que en la iteración 6 la red converge al patrón para el dígito 3.

La red de Hopfield tiene 2 principales limitaciones cuando se usa como una memoria de contenido direccionable:

- Primero, el número de patrones que puede ser almacenado y recordado exactamente está severamente limitado.
- Segundo, un patrón de ejemplo será inestable si comparte muchos bits en común con otro patrón de ejemplo. Un ejemplo se considera inestable si es aplicado en un momento 0 y la red converge a algún otro ejemplo. Este problema puede ser eliminado, aplicando procedimientos de ortogonalización.

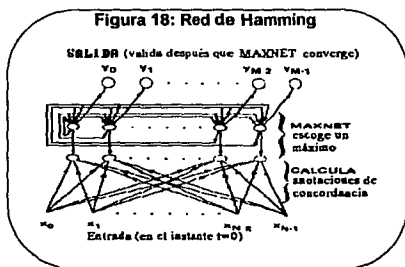
Cuando la red se usa como un clasificador, si muchos patrones son almacenados, la red puede convergir a un patrón nuevo adulterado diferente de todos los patrones de ejemplo y de la misma forma que un patrón adulterado producirá una "no concordancia".

Hopfield demostró, que esto no ocurre frecuentemente cuando los patrones de ejemplo son generados aleatoriamente y el número de clases (M) es menor que 15 veces el número de elementos de entrada o nodos en la red (N). El número de clases es por lo tanto típicamente mantenido bien, debajo de 15N.

III. 4 Red de Hamming.

La red de Hamming es a menudo probada en problemas donde las entradas son generadas seleccionando un ejemplo e invirtiendo valores de bits, aleatoria e independientemente, con una probabilidad dada¹⁰

Este es un problema clásico en teoría de comunicaciones, que ocurre cuando las señales de longitud binaria fija son mandadas a través de un canal simétrico binario sin memoria. El clasificador óptimo con mínimo error, en este caso, calcula la distancia de Hamming al ejemplo y selecciona la clase con la mínima distancia de Hamming. La **distancia de Hamming** es el número de bits en la entrada los cuales no concuerdan con el correspondiente ejemplo de bits.



Una red de Hamming usada para selección se muestra en la Figura 18. Los pesos y umbrales son primero puestos en la subred más baja de forma tal que las anotaciones de concordancia generadas por la salida de los nodos medios sean iguales a la distancia de Hamming al patrón ejemplo. Las anotaciones de concordancia ranquearán entre 0 y el número de elementos en la entrada (N) y son mayores para aquellos nodos con ejemplos que concuerden mejor con la entrada. Umbrales y pesos en la subcapa MAXNET son fijados. Todos los umbrales son puestos a 0 y los pesos de cada nodo a 1. Los pesos entre nodos son inhibitorios con un valor de $-c$ donde $c < 1/M$.

Después de que los pesos y umbrales son fijados, un patrón binario con N elementos se presenta en la parte baja de la red de Hamming. Esta debe de presentar los suficientes valores para permitir las salidas de las anotaciones de concordancia de la subcapa inferior para poner e inicializar los valores de salida de la MAXNET. La entrada es entonces removida y la MAXNET itera hasta que la salida de solamente un nodo, sea positiva. La clasificación es entonces completa y la clase seleccionada es la que corresponde al nodo con una salida positiva.

El comportamiento de la red de Hamming se ilustra en la Figura 19.

¹⁰.-"Redes Neuronales y Sistemas Físicos con habilidades computacionales colectivas emergentes"
Hopfield J.J 1982

Figura 19

ALGORITMO DE LA RED DE HAMMING

Paso 1. Asignar los pesos de conexión y umbrales

En la red inferior:

$$w_{ij} = x_i^j / 2, \quad \theta_j = N/2,$$

$$0 \leq i < N-1, \quad 0 \leq j < N-1$$

En la red superior:

$$v_{kl} = \begin{cases} 1, & l=1 \\ -1, & l=2 \end{cases}, \quad 0 \leq k, l < N/2$$

$$0 \leq k, l < N-1$$

En estas ecuaciones w_{ij} es el peso de la conexión del nodo i al nodo j en la red inferior y θ es el umbral en ese nodo. El peso de conexión del nodo l al nodo k en la red superior es v_{kl} y todos los umbrales en esta red son cero. x_i^j es el elemento i del ejemplo j .

Paso 2. Inicializar con un patrón de entrada desconocido

$$y_j(0) = I_j \left[\sum_{i=0}^{N-1} w_{ij} x_i - \theta_j \right] \quad 0 \leq j < N-1$$

En esta ecuación $y_j(t)$ es la salida del nodo j en la red superior en el tiempo t , x_i es el elemento de entrada i y I_j es el umbral lógico de no-linearidad (véase Figura 12 Capítulo 11). Aquí y en el siguiente paso se asume que la neurona obtiene a este no-linearidad una salida que la salida se satura.

Paso 3. Iterar hasta convergencia

$$y_i(t+1) = I_i \left[y_j(t) - \sum_{k=0}^{N/2-1} v_{ki} y_k(t) \right]$$

Este proceso se repite hasta que converge después de lo cual la salida de solo un nodo permanece positiva.

Paso 4. Devolver al Paso 2

La red de Hamming tiene un número de ventajas obvias sobre la red de Hopfield.

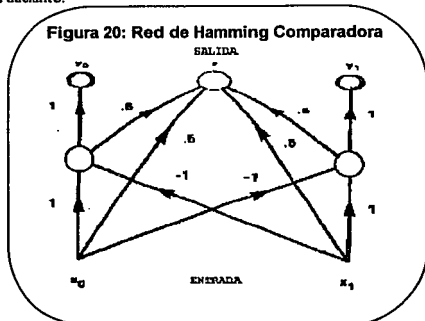
Esta implementa el clasificador óptimo de error mínimo, cuando los errores de bits son aleatorios e independientes y por lo tanto el desempeño de la red de Hopfield debe ser peor o equivalente al de la red de Hamming en tales situaciones. Comparaciones realizadas entre las 2 redes en problemas tales como: reconocimiento de caracteres, reconocimiento de patrones aleatorios y referencias bibliográficas han demostrado esta diferencia en desempeño.

La Red de Hamming requiere menos conexiones que la red de Hopfield. Por ejemplo con 100 entradas y 10 clases la red de Hamming requiere solamente 1,100 conexiones, mientras la red de Hopfield requiere de casi 10,000. Además la diferencia en el número de conexiones requeridas se incrementa cuando el número de entradas se incrementa, porque el número de conexiones en la red de Hopfield crece cuadráticamente con respecto al número de entradas, mientras el número de conexiones en la red de Hamming crece linealmente.

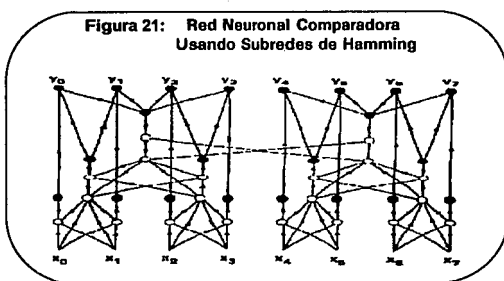
La Red de Hamming también puede ser modificada para ser un clasificador de error mínimo cuando los errores son generados por la inversión de elementos de +1 a -1 y de -1 a +1 asimétricamente con diferentes probabilidades y cuando los valores de elementos de entrada específicos son desconocidos. Finalmente la red de Hamming no sufre de patrones de salida adulterados los cuales pueden producir un resultado de no concordancia.

En los problemas de clasificación ocurre frecuentemente la necesidad de seleccionar ó realizar la entrada con valor máximo. Algunos tipos diferentes de redes neuronales pueden realizar esta operación¹¹. La red de Hamming, descrita arriba, usa inhibición rígida lateral similar a la que se usa en otras redes donde se desea un máximo. Estos diseños crean un tipo de red "el ganador toma todo" cuyo diseño imita el uso de inhibición rígida lateral en las redes neuronales biológicas del cerebro humano.

Son posibles también otras técnicas para tomar el máximo. En la Figura 20 se muestra una subred comparadora. Esta usa un nodo de umbral lógico para tomar el máximo de 2 entradas y entonces alimentar este valor máximo hacia adelante.



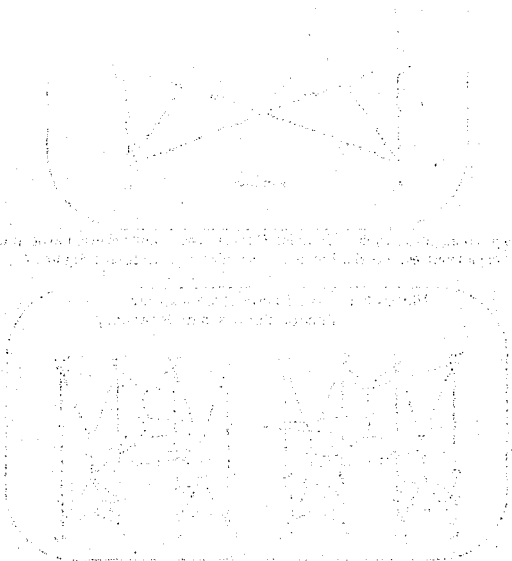
La subred comparadora puede ser dividida en $\log_2(M)$ capas para tomar el máximo de M entradas. Una red que usa estas subredes para escoger el máximo de 8 entradas se presenta en la Figura 21.



¹¹ "Auto-organización y memoria asociativa" Kohonen T. Springer-Verlag Berlin 1984.

Redes de este tipo son útiles cuando el valor máximo debe pasarse, sin alteraciones, a la salida.

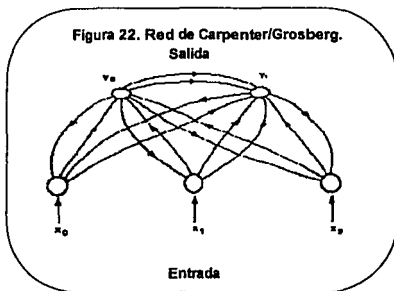
En algunas situaciones no se requiere un máximo y las anotaciones de concordancia deben ser comparadas con un umbral. Esto puede hacerse tomando un arreglo de nodos, limitadores rígidos con umbral interno puesto al valor del umbral deseado. La salida de estos nodos será -1 a menos que las entradas excedan el valor del umbral. Alternativamente, los umbrales pueden ser puestos adaptativamente usando una entrada inhibitoria común a todos los nodos. Este umbral puede ser incrementado o disminuido hasta que la salida de solamente un nodo sea positiva.



III. 5 Clasificador de Carpenter/Grosberg.

Carpenter y Grosberg en el desarrollo de su Teoría de Resonancia Adaptativa¹² implementaron un algoritmo de agrupamiento que es muy semejante al **algoritmo líder de agrupamiento**¹³, el cual forma conjuntos y se entrena sin supervisión. Este algoritmo, selecciona la primera entrada como el ejemplo para el primer grupo. La siguiente entrada se compara con el primer grupo ejemplo, esta sigue al líder y se agrupa con el primero si la distancia al primero es menor que un umbral. De otra forma, este será un ejemplo para un nuevo grupo. Este proceso se repite para todas la demás entradas. El número de grupos, por lo tanto, crece con el tiempo y depende del umbral y la distancia métrica usada para comparar las entradas a grupos ejemplos.

Los principales componentes de la Red de Clasificación de Carpenter/Grosberg, con tres entradas y dos nodos de salida se presentan en la Figura 22.



La estructura de esta red es muy similar a la utilizada por la Red de Hamming. Las anotaciones de concordancia son calculadas usando conexiones de alimentación hacia adelante y el valor máximo es realizado usando inhibición lateral entre los nodos de salida.

Esta red se diferencia de la red de Hamming en que las conexiones de alimentación hacia adelante provienen de los nodos de salida hacia los nodos de entrada. Este mecanismo también permite a los nodos apagar el nodo con el valor máximo y comparar los ejemplos de entrada con la prueba de umbral requerida por el algoritmo líder. Carpenter y Grosberg en su libro describieron esta red, completamente, usando ecuaciones diferenciales no lineales, incluyendo retroalimentación extensiva y demostraron su estabilidad.

¹² "Dinámicas Neuronales de aprendizaje de categorías y reconocimiento"
G.A. Carpenter, S. Grosberg. Ed. Simposium Series AAAS. 1986.

¹³ Se describe en el libro "Algoritmos de Agrupamiento"
J.A. Hartigan Ed. John Wiley, New York. 1975

En la Figura 23 se muestra, la implementación de las ecuaciones diferenciales para el algoritmo de Carpenter/Grosberg.

Figura 23

ALGORITMO DE CARPENTER / GROSBERG

Paso 1. $t_{ij}(0) = 1$
 $h_{ij}(0) = 1/(1+N)$ $0 \leq i \leq N-1, 0 \leq j \leq N-1$
 Establecer ρ , $0 \leq \rho \leq 1$

En estas ecuaciones $h_{ij}(t)$ son los pesos de conexión ascendentes y $t_{ij}(t)$ son los pesos de conexión descendentes entre el nodo de entrada i y el nodo de salida j en el momento t . Estos pesos definen los ejemplos especificados por el nodo de salida j . La fracción ρ es el umbral vigilante al cual indica que tan cerca debe estar una entrada para concordar con un ejemplo almacenado.

Paso 2. Aplicar nueva entrada

Paso 3. Computar anotaciones de concordancia

$$u_j = \sum_{i=0}^{N-1} h_{ij}(t) x_i, \quad 0 \leq j \leq N-1$$

En esta ecuación u_j es la salida del nodo de salida j y x_i es el elemento i de la entrada la cual puede ser

Paso 4. Seleccionar el ejemplo que mejor concuerda

$$u_j^* = \max_j (u_j)$$

Esto se realiza inhibición lateral extensiva como en la NUNNET.

Paso 5. Prueba de vigilancia

$$|X| = \sum_{i=0}^{N-1} x_i$$

$$|T \cdot X| = \sum_{i=0}^{N-1} t_{ij} x_i$$

$$\text{Es } |T \cdot X| / |X| \geq \rho ?$$

NO
= IR AL PASO 6

SI
= IR AL PASO 7

Paso 6. Deshabilitar el ejemplo de mejor concordancia

La salida del nodo de mejor concordancia seleccionada en el paso 4 es puesta (temporalmente a cero y no toma su parte en la minimización del paso 4. Entonces ir al paso 3.

Paso 7. Adaptar el ejemplo de mejor concordancia

$$t_{ij}(t+1) = t_{ij}(t) x_i$$

$$h_{ij}(t+1) = t_{ij}(t) x_i / \sum_{i=0}^{N-1} t_{ij}(t) x_i$$

Paso 8. Repetir paso 2

Primera salida: multiplica cada derivada por el paso 6)

La red se inicializa al poner todos los ejemplos, representados por pesos de conexiones a cero. Además un umbral de concordancia llamado vigilante cuyo valor ranquea entre 0.0 y 1.0 debe ser puesto. Este umbral determina que tan cerca debe estar un nuevo patrón de entrada para ser considerado similar. Un valor cercano a 1 requiere una concordancia cercana y el valor más pequeño aceptará una concordancia pobre.

Nuevas entradas son presentadas secuencialmente a la red, como en la red de Hamming, después de la presentación la entrada es comparada con todos los patrones almacenados para producir anotaciones de concordancia. El patrón con la más alta anotación de concordancia se selecciona usando inhibición lateral. Este se compara con la entrada para calcular el radio del producto punto y del ejemplo de mejor concordancia (con un número de bits en común) dividido por el número de bits de la entrada. Si este número es mayor que el umbral vigilante, entonces la entrada se considera similar y el patrón es actualizado realizando la operación lógica AND entre sus bits y los de la entrada.

Si el radio es menor que el umbral vigilante, se considera diferente de todos los ejemplos y se añade como un nuevo ejemplo. Cada nuevo ejemplo adicional requiere un nodo y 2N conexiones para calcular sus anotaciones de concordancia.

El comportamiento de la Red de Carpenter/Grosberg se ilustra en la Figura 24.

Aquí se asume que los patrones a ser reconocidos son tres patrones de letras C, E y F, que se muestran en el lado izquierdo de la Figura 24. Estos patrones tienen 64 pixels cada uno y toman el valor de 1 cuando son negros y 0 cuando son blancos.

Los resultados presentados se obtienen al poner el umbral vigilante a 0.9, esto forza a crear patrones ejemplo separados para cada letra.

Figura 24
Comportamiento de la Red de Carpenter/Grosberg

Entrada	Patrón Generado
C	C
E	CE
F	CEF
F	CEFF
F	CEFFF

En la Figura 24, el lado derecho muestra los patrones ejemplos formados después de la presentación de cada patrón. Se presentan los 3 patrones, C, E y F y se generan los patrones ejemplo para cada uno de los patrones. Si se presentan patrones de C, E o F, la red reconoce cada uno de ellos como idéntico.

Si se disminuye ligeramente el umbral vigilante y se presenta un patrón para F, alterado en un pixel, la red acepta este patrón como similar al patrón F y degrada este patrón ejemplo al realizar la operación AND en la actualización de pesos. Cuando otra F alterada en un pixel se presenta a la red esta considera el patrón como diferente y añade un nuevo patrón ejemplo. Esto ocurrirá para posteriores patrones alterados de F y generará un crecimiento de patrones alterados de F. Esto demuestra que la Red de Carpenter/Grosberg se desempeña bien con patrones de entrada perfectos, pero aún una pequeña alteración puede causar problemas.

Sin alteraciones, el umbral vigilante puede ser puesto, de forma tal, que 2 patrones muy similares sean considerados diferentes. Con alteraciones, sin embargo, este nivel puede ser muy alto y el número de ejemplos puede crecer muy rápidamente, hasta que todos los nodos disponibles sean usados.

Se necesitan modificaciones para realzar el desempeño de este algoritmo con alteraciones. Estas pueden incluir una adaptación de pesos más ligera y cambiar el umbral vigilante durante el entrenamiento.

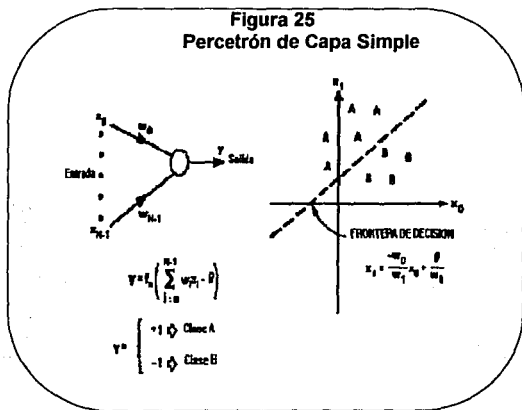
III. 6 Perceptrón de Capa Simple.

El **perceptrón de capa simple**¹⁴, se encuentra clasificado dentro de las redes que pueden usar valores de entrada binarios y continuos.

Rosenblatt en 1959 contempló al perceptrón como un sistema de proceso distribuido en paralelo hecho de una red de funciones sumadoras y multiplicadoras.

Esta red simple generó mucho interés por su habilidad de aprender a reconocer patrones simples. En el reconocimiento de patrones es esencial que el sistema sea capaz de aprender de su experiencia e inferir discriminantes autónomamente.

En la Figura 25 se muestra un perceptrón que decide si una entrada pertenece a una de 2 clases (denotadas por A y B) el nodo calcula una suma de pesos de los elementos de entrada, resta un umbral β y pasa el resultado a través de una no-linealidad de limitador duro tal que la salida y toma un valor de +1 o -1. La regla de decisión responde a la clase A si la salida es +1 ó a la clase B si la salida es -1.



Note que los nodos están colocados de forma tal que la entrada es igual a la salida de todos los nodos. Por lo tanto, esta es una implementación en red de un discriminante lineal, todos los nodos son lineales, así como las uniones.

¹⁴ - "Principios de Neurodinámica"
Rosenblatt R. Ed. Spartan Books New York. 1959

Una técnica para calcular el comportamiento de redes como el perceptrón es dibujar un mapa de la región de decisión creada en un espacio multidimensional, generado por las variables de entrada. Esta región de decisión especifica cuales valores de entrada resultan en una respuesta en una clase A y cuales en una clase B.

Los pesos de las conexiones y los umbrales en un perceptrón pueden ser fijados ó adaptados usando varios algoritmos diferentes.

Describiremos el perceptrón analíticamente.

Describimos el patrón en términos de vectores columna x , así que:

$$x^t = (x_1, x_2, \dots, x_n)$$

Agrupamos los patrones de conjuntos de entrenamiento como renglones para formar una matriz X . Resolver para obtener el discriminante lineal consiste en buscar un vector columna w tal que:

$$Xw = b$$

donde los valores de b son las salidas especificadas por el ejemplo de entrenamiento.

En un problema de clasificación, los valores de b pueden ser especificados como positivos, si x pertenece a la clase A y negativos si x no pertenece a la clase A. En términos de redes esta circunstancia corresponde a tener una salida escalar simple b que puede tener diferentes valores.

Para M patrones en un espacio N -dimensional podemos representar la expresión:

$$Xw = b$$

en forma expandida como :

$$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}$$

y especificar que x_i pertenece a A, si b_i es positivo, y que x_i no pertenece a A, si b_i es negativo.

Sabemos que podemos resolver la ecuación:

$$Xw = b$$

en términos de la pseudoinversa de X , esto es:

$$X^t X w = X^t b$$

y

$$w = (X^t X)^{-1} X^t b$$

ó

$$w = X^T b$$

donde X^T es la pseudo inversa y, en principio, puede también ser obtenida como el limite del valor del proceso.

$$X^T = \lim_{\epsilon \rightarrow 0} (X^T X + \epsilon I)^{-1} X^T$$

En la práctica, sin embargo, se obtienen valores insignificantes si el determinante de $X^T X$ se hace muy pequeño. El procedimiento exacto analítico es impráctico, debido no solo a dificultades analíticas sino porque no parece corresponder a las circunstancias encontradas en las situaciones reales.

Desde una perspectiva formal, del reconocimiento de patrones, podemos contemplar el tener un conjunto fijo de patrones de ejemplos. Sin embargo las circunstancias encontradas por una red neuronal biológica pueden ser diferentes, los patrones pueden ser encontrados secuencialmente, uno a la vez. Esto es a menudo cierto en las tareas de reconocimiento de patrones reales.

En el caso mas realista las ecuaciones apropiadas del discriminante pueden ser:

$$x_p w = b \quad p=1,2,\dots$$

y los requerimientos pueden ser, que un simple conjunto de pesos w sean suficientes para dar el conjunto correcto de salidas b para todos los patrones x_p , $p=1,2,\dots$

El procedimiento original de convergencia del perceptrón, desarrollado por Rosenblat, se describe en la Figura 26.

Figura 26

PROCEDIMIENTO DE CONVERGENCIA PERCEPTIVA

Paso 1. Inicializar Pesos y Umbrales

Establecer $w_i(0)$ ($0 \leq i \leq M-1$) y θ a valores aleatorios pequeños. Aquí $w_i(t)$ es el peso de la entrada i en el tiempo t y θ es el umbral en el modo de salida.

Paso 2. Presentar nueva entrada y salida deseada

Presenta nuevos valores de entrada continuos x_0, x_1, \dots, x_{M-1} acompaña a la salida deseada $d(t)$.

Paso 3. Calcular la salida actual

$$y(t) = f_n \left[\sum_{i=0}^{M-1} w_i(t) x_i(t) - \theta \right]$$

Paso 4. Adaptar Pesos

$$w_i(t+1) = w_i(t) + \eta [d(t) - y(t)] x_i(t),$$

$$0 \leq i \leq M-1$$

$$f(t) = \begin{cases} 1 & \text{si la entrada es de la clase A} \\ -1 & \text{si la entrada es de la clase B} \end{cases}$$

En estas ocasiones η es una fracción de ganancia positiva menor que 1 y $d(t)$ es la salida correcta deseada para la entrada actual. Nota que los pesos son actualizados si la decisión correcta es hecha por la red.

Paso 5. Repetir el paso 2

Primero los pesos de las conexiones y los valores de los umbral son inicializados a un valor aleatorio pequeño, diferente de cero. Entonces se aplica una nueva entrada con N elementos de valores continuos y se calcula la salida. Los pesos de las conexiones son adaptados únicamente cuando ocurre un error usando la fórmula:

$$w_i = w_i(t) + \eta [d(t) - y(t)] x_{i(t)}$$

$$0 \leq i \leq N - 1$$

$$d(t) \begin{cases} +1 & \text{si la entrada es de la clase A} \\ -1 & \text{si la entrada es de la clase B} \end{cases}$$

Esta fórmula incluye un término de incremento (η) que ranquea de 0.0 a 1.0 y controla el rango de adaptación. Este término de incremento debe ser ajustado para satisfacer los requerimientos conflictivos de adaptación rápida para cambios reales en las distribuciones de entrada y promedios de entradas pasadas para dar una estimación estable de los pesos.

El algoritmo del perceptrón lineal y la versión iterativa de Widrow-Hoff recomiendan encontrar w a través de las siguientes reglas:

Asignar

$$w_i = \text{Arbitrario}$$

y

$$w_{k+1} = w_k + \rho (b^k - w_k^t x^k) x^k$$

donde b^k es el valor apropiado de b para el patrón x^k y puede ser +1 para patrones A y -1 para patrones ~A, y x^k es el patrón que esta siendo considerado en el paso k .

La expresión:

$$w_{k+1} = w_k + \rho (b^k - w_k^t x^k) x^k$$

es una regla para actualizar el vector de pesos hasta que todos los vectores de patrones sean clasificados correctamente. El procedimiento es para añadir una cierta cantidad de x^k a w^k si x^k no se clasifica correctamente.

El factor de proporcionalidad es

$$\rho (b^k - w_k^t x^k) x^k$$

el cuál es cero o un valor muy pequeño si x^k se clasifica correctamente. En la mayoría de los casos de interés es imposible satisfacer todas las ecuaciones:

$$w_k^t x^k = b^k$$

de esta forma las correcciones nunca se detienen. La convergencia puede asegurarse, si ρ se decrementa con k ; esto hace que $\rho \rightarrow 0$ conforme proceda la iteración.

Sin embargo la convergencia obtenida de esta forma es artificial y no necesariamente da una w valida que clasifique todos los patrones correctamente.

La expresión:

$$w_{k+1} = w_k + \rho (b^k - w_k^t x^k) x^k$$

puede ser reescrita como

$$\Delta w = \eta \delta x$$

donde la cantidad δ es igual a $(b^k - w_k^t x^k)$, la diferencia entre la salida deseada b y la salida actual producida por la red, esto es wx , siendo x el patrón de entrada.

Por lo tanto, la expresión:

$$\Delta w = \eta \delta x$$

ó **regla delta simple** establece que los cambios en el vector de pesos debe ser proporcional a δ y al patrón de entrada.

Esta expresión es conocida como la **regla delta**.

Desde un punto de vista gráfico, los algoritmos del perceptrón consisten en actualizar el vector de pesos, considerando cada patrón mal clasificado de nuevo y añadir una fracción de cada patrón mal clasificado al vector de pesos. Esta práctica se continua hasta que todos los patrones sean clasificados correctamente ó hasta que esté claro que el procedimiento fallará al convergir a una solución satisfactoria.

El perceptrón forma 2 regiones de decisión separadas por un hiperplano. En la figura 26 se muestra el caso cuando hay 2 entradas y el hiperplano forma una línea. En este caso las entradas arriba de la línea frontera llevan a una respuesta de clase A y las entradas debajo de esta frontera llevan a una respuesta de la clase B. Como puede verse la ecuación de la línea limite depende de los pesos de las conexiones y los umbrales.

Rosenblatt demostró que si las entradas presentadas para las 2 clases son separables (si éstas caen en lados opuestos de un hiperplano), entonces el procedimiento de convergencia del perceptrón converge y posiciona al hiperplano de decisión entre estas 2 clases.

Un problema con el procedimiento de convergencia del perceptrón es que los limites de decisión pueden oscilar continuamente cuando las entradas no son separables y las distribuciones se enciman.

Una modificación al procedimiento de convergencia del perceptrón forma la solución de la menor media cuadrada (LMS por sus siglas en inglés Least Mean Square). Esta solución minimiza el error medio cuadrado entre la salida deseada de una red tipo perceptrón y la salida actual. El algoritmo que forma la solución LMS es llamado el **algoritmo Widrow-Hoff** ó **algoritmo LMS**.

El algoritmo LMS es idéntico al procedimiento de convergencia del perceptrón, excepto que la no linealidad del limitador rigido se hace lineal ó se reemplaza por una no linealidad de umbral lógico. Los pesos son entonces corregidos por una proporción que depende de la diferencia entre la entrada actual y la salida deseada.

Un clasificador que utilice el algoritmo de entrenamiento LMS puede usar salidas deseadas de 1 para la clase A y 0 para la clase B. Durante la operación la entrada sería asignada a la clase A solamente si la salida estuviera arriba de 0.5.

Las regiones de decisión formadas, son parecidas a las formadas por el clasificador Gaussiano de máxima probabilidad el cuál asume que las entradas no están correlacionadas y las distribuciones para las diferentes clases difieren solamente en valores medios.

Si se involucra más de una salida, entonces la solución será una matriz de pesos en lugar de un vector de pesos y las salidas requeridas también formarán una matriz. El procedimiento se mantiene idéntico.

Así mismo se puede generalizar la variante Widrow-Hoff del procedimiento de convergencia del perceptrón para aplicarse a M clases. Esto requiere de una estructura idéntica a la red de Hamming y una regla de clasificación que seleccione la clase correspondiente al nodo con la salida máxima. Durante la adaptación, se pueden poner los valores de salida deseados a 1 para la clase correcta y a 0 para las demás clases.

En el caso lineal no hay ventaja en tener más de una capa de nodos. Esto indica que, las complicaciones de una capa interna no son útiles. El producto matricial de 2 matrices lineales sigue siendo una matriz y la naturaleza de la solución la misma.

Esto significa que:

$$XW_1W_2=B$$

puede también escribirse como:

$$XW=B$$

y resolver iterativamente para una simple W es tan fácil como ciclar a través de 2 pasos sucesivos para resolver W_1 y W_2 .


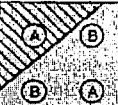



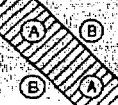



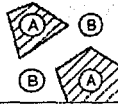
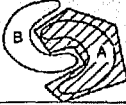
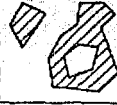
La estructura del perceptrón puede ser usada para implementar tanto clasificadores Gaussianos de máxima probabilidad ó clasificadores que usen el algoritmo de entrenamiento del perceptrón ó una de sus variantes. La elección depende de la aplicación.

El algoritmo de entrenamiento del perceptrón no asume nada concerniente a la forma ni en las bases de las distribuciones pero se enfoca en el error que ocurre donde las distribuciones se enciman. Esto lo hace más robusto que las tecnicas clásicas y trabaja bien cuando las entradas son generadas por procesos no lineales y son fuertemente asimétricas y no Gaussianas. Los clasificadores Gaussianos asumen, de forma muy fuerte, las formas y las bases de las distribuciones, y son más apropiados cuando las distribuciones son conocidas y concuerdan mejor con lo que el Gaussiano da por hecho.

El algoritmo de adaptación definido por el procedimiento de convergencia del perceptrón es fácil de implementar y no requiere almacenar ninguna otra información que la presentada en los pesos y los umbrales. El clasificador Gaussiano puede ser más adaptativo, pero requiere almacenar más información extra y los cálculos son más complejos.

Ni el clasificador Gaussiano ni el procedimiento de convergencia del perceptrón son apropiados cuando las clases no pueden ser separadas por un hiperplano. En el primer renglón de la **Figura 27** se presentan 2 de estas situaciones.

Figura 27: Regiones Típicas de Clasificación

Estructura	Tipo de Región de Decisión	Problema de OR-Exclusivo	Clases con regiones Mezcladas	Forma de región más general
Capa Simple 	Medio Plano Limitado por un Hiperplano			
Dos Capas 	Convexa Abierta o Regiones Cerradas			
Tres Capas 	Arbitraria (Complejidad limitada por el número de nodos)			

Los contornos etiquetados con A y B son las distribuciones de entrada, cuando hay 2 valores de entrada continuos para la red. Las áreas sombreadas son las regiones de decisión creadas por el perceptrón de capa simple.

Las distribuciones de las 2 clases para el problema de OR-Exclusivo son disjuntas y no pueden ser separadas por una simple línea recta.

Si el conjunto B en la parte inferior izquierda se toma como el origen del espacio bidimensional entonces la salida del clasificador debe ser alta solamente si una de las entradas, pero no ambas, son altas. Esto generaría la región de decisión mostrada en el primer recuadro. Las regiones de decisión para el segundo problema presentado están mezcladas y tampoco pueden ser separadas por una simple línea recta.

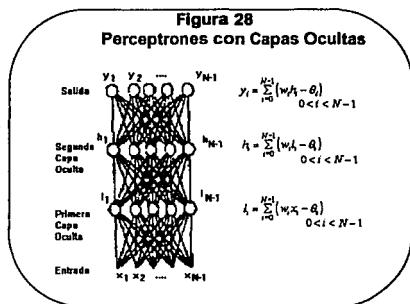
Un perceptrón lineal sin capas internas es incapaz de dar un discriminante que resuelva el problema de OR-Exclusivo o problema de paridad. Esto significa que el perceptrón lineal no puede aprender automáticamente el discriminante que clasifique correctamente los patrones de paridad non o par.

Este problema fué utilizado para demostrar la debilidad del perceptrón por Paper y Minsky en 1969. Situaciones como esta se presentan cuando parámetros tales como perfiles de frecuencias de fonemas particulares son empleados para el reconocimiento del lenguaje.

III. 7 Perceptrón Multicapa.

Los perceptrones de multicapa son redes con alimentación hacia adelante con una ó mas capas de nodos entre las capas de entrada y salida. Estas capas ocultas contienen nodos que no están directamente conectados con los nodos de las capas de entrada y salida.

En la Figura 28 se ilustra esquemáticamente la arquitectura de este tipo de redes o perceptrones con capas ocultas. Los perceptrones de multicapa superan muchas de las limitaciones de los perceptrones de capa simple, pero no fueron empleados debido a la falta de un algoritmo de aprendizaje eficiente, como lo demostró Nilsson en 1965 en su descripción de su máquina de multicapas.



En general estas redes se hacen de conjuntos de nodos arreglados en capas. Las salidas de los nodos en una capa son transmitidos a los nodos en otra capa a través de conexiones que amplifican, atenúan o inhiben tales salidas mediante factores de peso. Excepto por los nodos de la capa de entrada, la entrada de la red a cada nodo es la suma de las salidas de los pesos de los nodos en la capa anterior. Cada nodo se activa de acuerdo con la entrada al nodo, la función de activación del nodo y el umbral del nodo. Este proceso puede poseer una muy extensa labor de aprendizaje y no siempre fácilmente completado.

Las capacidades de los perceptrones multicapa radican en las no-linealidades usadas dentro de los nodos. Si los nodos fueran elementos lineales, entonces una red de capa simple con la elección apropiada de los pesos podría duplicar exactamente los cálculos realizados por una red de multicapa.

En la primera columna de la Figura 27 se ilustran las capacidades de perceptrones de una, dos y tres capas que usan no-linealidades de limitador rígido. La segunda columna indica el tipo de región de decisión que puede ser formada con diferentes redes. Las siguientes 2 columnas presentan ejemplos de regiones de decisión que pueden ser formadas por el problema de OR-Exclusivo y un problema con regiones mezcladas. La última columna da ejemplos de la región de decisión más general que puede formarse.

Como se discutió anteriormente un perceptrón de capa simple forma regiones de decisión de medio plano. Un perceptrón de 2 capas puede formar cualquier región convexa, posiblemente no cerrada, a través del espacio de las entradas. Tales regiones incluyen polígonos convexos algunas veces llamados **cascos convexos**. Las regiones convexas no cerradas se muestran el centro de la Figura 27.

Aquí el término **convexo** significa que cualquier línea uniendo puntos en la frontera de una región pasa únicamente a través de los puntos de la región. Las regiones convexas se forman mediante intersecciones de regiones de medio plano formadas por cada nodo en la primera capa del perceptrón multicapa. Cada nodo en la primera capa se comporta como un perceptrón de capa simple y tiene una salida alta solamente para puntos de un lado del hiperplano formado por sus pesos y umbrales.

Si los pesos a un nodo de salida de los N nodos de la primera capa son todos 1.0 y el umbral en el nodo de salida es $N-\epsilon$ donde $0 < \epsilon < 1$, entonces la salida del nodo será alta solamente si las salidas de todos los nodos de la primera capa son altas.

Esto corresponde a realizar la operación lógica AND en el nodo de salida y obtener como resultado una región de decisión que es la intersección de todas las regiones de medio plano formadas en la primera capa. Las intersecciones de estos medios planos forman regiones convexas como se describió antes. Estas regiones convexas tiene a lo más tantas caras como nodos en la primera capa.

Este análisis da una visión al problema de seleccionar el número de nodos a usar en un perceptrón de 2 capas. El número de nodos debe ser lo suficientemente grande para formar una región de decisión que sea tan compleja como se requiera por el problema dado. Este número no debe ser tan grande que los muchos pesos requeridos no puedan ser confiablemente estimados de los datos disponibles de entrenamiento. Por ejemplo, 2 nodos son suficientes para resolver el problema de OR-Exclusivo como se muestra en el segundo renglón de la Figura 27, sin embargo, con un perceptrón de 2 capas ningún número de nodos podrá separar las clases con regiones mezcladas.

Un perceptrón de 3 capas puede formar regiones de decisión arbitrariamente complejas y separar las clases mezcladas como se muestra en la parte baja de la Figura 27.

Esto se puede probar por construcción. La prueba depende de particionar la región de decisión deseada en pequeños hipercubos (cuadros donde hay 2 entradas). Cada hipercubo requiere $2N$ nodos en la primera capa (4 nodos cuando hay 2 entradas) uno por cada lado del hipercubo, y uno en la segunda capa que realiza la operación lógica AND con las salidas de los nodos de la primera capa. Las salidas de los nodos de la segunda capa estarán altas solamente para las entradas dentro de cada hipercubo. Los hipercubos son asignados a la región propia de decisión conectando la salida de cada nodo de la segunda capa solamente al nodo de salida correspondiente a la región de decisión dentro de la que está el nodo del hipercubo y realizando la operación lógica OR en cada nodo de salida.

Una operación lógica OR será realizada si los pesos de conexión de la segunda capa oculta a la capa de salida son 1 y los umbrales en los nodos de salida son 0.5. Este procedimiento de construcción se puede emplear para formar regiones convexas arbitrariamente formadas en lugar de pequeños hipercubos y es capaz de generar las regiones no convexas y desconectadas mostradas en la parte baja de la Figura 27.

Este análisis demuestra que no se requieren más de 3 capas en una red tipo perceptrón con alimentación hacia adelante porque una red de 3 capas puede generar regiones de decisión arbitrariamente complejas. Esto también provee un visión de seleccionar el número de nodos para usar en un perceptrón de 3 capas.

El número de nodos en la segunda capa debe ser mayor que 1 cuando las regiones de decisión estén desconectadas ó mezcladas y puedan ser formadas de una región convexa. El número de nodos en la segunda capa requiere que en el peor de los casos sea igual al número de regiones desconectadas en las distribuciones de entrada.

El número de nodos en la primera capa debe ser suficiente para proveer 3 o más filos para cada área convexa generada por cada nodo de la segunda capa. Debe, por lo tanto, ser 3 veces mayor que el número de nodos en la segunda capa.

Este análisis se centra primariamente en perceptrones de multicapa con una salida cuando se usan no-linealidades de limitador rígido. Comportamientos similares se presentan en perceptrones de multicapa con múltiples nodos de salida, cuando se usan no-linealidades sigmoidales y la regla de decisión es para seleccionar el nodo de salida con la mayor salida. El comportamiento de estas redes es más complejo porque las regiones de decisión están limitadas por curvas suaves en lugar de segmentos de líneas y el análisis es por lo tanto más difícil y necesitan otro algoritmo de entrenamiento.

III. 7.1 Perceptrón de Retropropagación (BPN).

Las redes con no-linealidades sigmoidales y regla de decisión para seleccionar el nodo de salida con la mayor salida. Estas redes pueden ser entrenadas con el algoritmo de retropropagación.

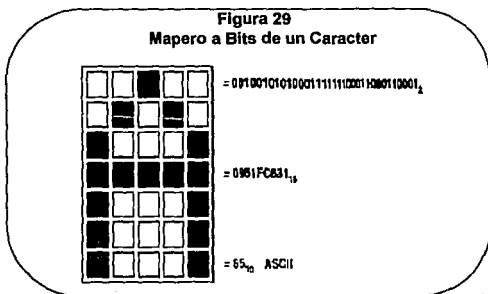
Existen muchas aplicaciones computacionales que no son resolubles mediante el uso de procesamiento secuencial. Aplicaciones que deban realizar algunas traducciones de datos complejas, que no tienen predefinida una función de mapeo para describir el proceso de traducción o aquellos donde los datos de entrada se presentan un poco distorsionados o contaminados, son ejemplos de este tipo de aplicaciones. Ilustremos un problema que a menudo se presenta cuando se pretende automatizar aplicaciones complejas de reconocimiento de patrones.

Consideremos el diseño de un programa de computadora que debe traducir de una matriz binaria de 5x7 la imagen mapeada a bit de un caracter alfanumérico, a su valor equivalente ASCII de 8 bits.

Este problema básico aparenta ser trivial en primera instancia.

Como no existe una función matemática que realice la traducción deseada e indudablemente el implementar una correlación pixel por pixel tomaría mucho tiempo hombre-máquina, la mejor solución algorítmica será implementar una tabla de búsqueda (Tabla Look-Up).

La tabla look-up para resolver este problema será un arreglo lineal, unidimensional, de pares ordenados, cada uno conteniendo 2 elementos. El primer elemento del arreglo será el equivalente numérico del código de patrón de bits generado por mover los 7 renglones de la matriz, a un solo renglón y considerar el resultado como un número binario de 35 bits. El segundo es el código ASCII asociado al caracter. El arreglo contendrá el mismo número de pares ordenados como caracteres a convertir. (Figura 29)



El algoritmo necesario para realizar el proceso de conversión se parecerá al siguiente:

```
Function Translate(Input : long integer;  
                 LUT : AElement[] ) return ascii;  
{ Realiza la conversión matriz-píxel a carácter ascii }  
var  
Table : AElement;  
Found : Boolean;  
i : integer;  
  
Begin  
Table := LUT;      { Localiza la tabla de Traducción }  
Found := False;   { Traducción no encontrada }  
For i:=1 to length(Table) do { Para todos los elementos en la tabla }  
  If Table[i]=Input Then  
    Begin  
      Found:=True;  
      Exit;      { Traducción Encontrada;Salir del loop }  
    End;  
  If Found Then Table[i].ascii  
  Else return:=0;  
End;
```

La solución mediante tabla look-up corre razonablemente rápido y es fácil de mantener, pero hay muchas situaciones reales en las que no se puede emplear este método.

Consideremos el mismo proceso de conversión pixel-imagen a ASCII en un medio ambiente más realista. Supongamos que nuestro scanner altera un bit aleatorio en la matriz pixel-imagen de salida, debido a ruido cuando se lee la imagen. Este error de un solo pixel puede causar que nuestro algoritmo no nos regrese ningún valor ó regrese un código ASCII equivocado, ya que la concordancia entre el patrón de entrada y el patrón objetivo debe ser exacto. Ahora consideremos el monto de software adicional y la cantidad de tiempo hombre-máquina que se debe añadir al algoritmo de tabla look-up para darle la habilidad a la computadora de "adivinar" a que carácter corresponde la imagen alterada.

Errores de un solo bit son relativamente fáciles de encontrar y corregir. Errores multibit incrementan el grado de dificultad de acuerdo como se incrementa el número de bits de error. Para complicar las cosas aún más ¿Cómo compensaría nuestro software una imagen alterada si una "O" parece una "Q" o una "E" parece una "F"?

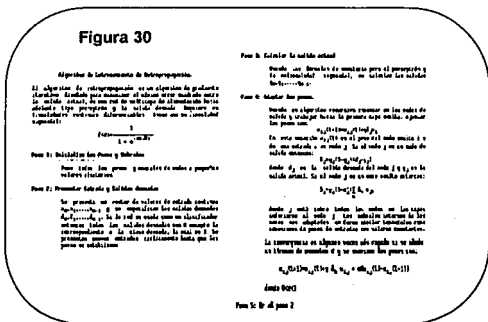
Si nuestro sistema de conversión de caracteres tiene que producir una salida exacta todo el tiempo se necesitaría emplear una extraordinaria cantidad de tiempo de CPU para eliminar el ruido del patrón de entrada antes de intentar realizar la traducción a ASCII.

Una solución a este dilema es tomar ventaja de la naturaleza en paralelo de las redes neuronales, para reducir el tiempo requerido por un procesador secuencial para realizar el mapeo.

Además, el tiempo de desarrollo del sistema se reduce porque la red puede aprender del algoritmo propio sin que nadie lo haya deducido con anterioridad y que pueda examinar todos los pixels en la imagen en paralelo. Idealmente un sistema tal no tiene que ser explícitamente programado, sino que debe de adaptarse el mismo para aprender las relaciones entre un patrón de conjuntos de ejemplo y ser capaz de aplicar las mismas relaciones a nuevos patrones de entrada. Este sistema debe ser capaz de fijarse en las características de una entrada arbitraria, que se asemeje a otros patrones vistos previamente de forma tal que reconozca las características del patrón e ignore los pixels de distorsión.

Este sistema se conoce como: **Red de Retropropagación.**

El algoritmo de retropropagación es una generalización del algoritmo LMS. Este usa una técnica de búsqueda de gradiente descendente para minimizar una función de costo igual a la diferencia cuadrada de medias entre la salida actual de la red y la salida deseada. (Figura 30)

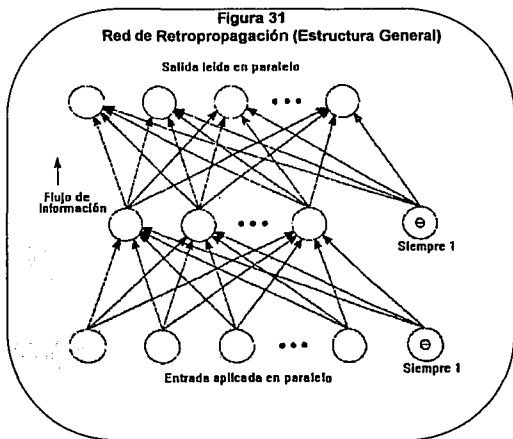


El algoritmo se describe de la siguiente forma:

- Para comenzar la red debe aprender un conjunto predefinido de pares de ejemplos entrada-salida usando un ciclo de 2 fases propagación-adaptación.
- Después se aplica un patrón de entrada como estímulo a la primera capa de unidades de la red.
- El patrón de entrada se propaga, a través de cada capa superior hasta que se genera una salida. Este patrón de salida se compara entonces a la salida deseada y una señal de error se calcula para cada unidad de salida.
- Las señales de error son transmitidas en retroceso (de donde se obtiene su nombre: retropropagación) desde la capa de salida, a cada nodo en la capa intermedia que contribuye directamente a la salida. Sin embargo cada unidad en la capa intermedia recibe únicamente una porción de la señal de error total, basada aproximadamente, en la contribución relativa que la unidad hace a la salida original.

Este proceso se repite, capa por capa, hasta que cada nodo en la red ha recibido una señal de error que describe su contribución relativa al error total. Basándose en la señal de error recibida los pesos de las conexiones se actualizan en cada unidad para causar que la red converja a un estado que permita que todos los patrones de entrenamiento sean codificados.

El significado de este proceso es que, cuando la red se entrena, los nodos en la capa intermedia se organizan ellos mismos de forma tal que diferentes nodos aprenden a reconocer diferentes características del espacio total de entradas.

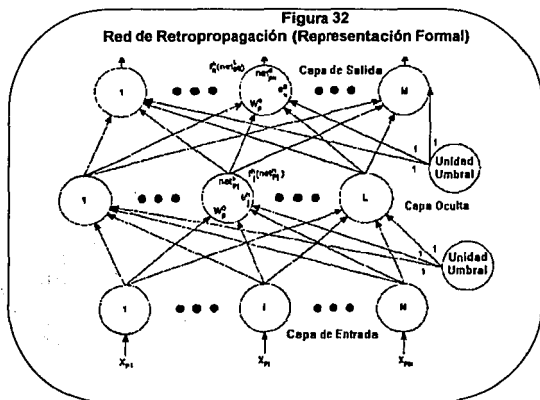


Después del entrenamiento cuando se presenta un patrón de entrada arbitrario que puede estar alterado o incompleto, las unidades en las capas ocultas de la red responderán con una salida de activación si la nueva entrada contiene un patrón que se asemeje a las características, que las unidades individuales aprendieron a reconocer durante el proceso de entrenamiento. Inversamente, las unidades en las capas ocultas tienden a inhibir su salida si el patrón de entrada no contiene la característica que aprendieron a reconocer. Como la señal se propaga a través de las diferentes capas en la red, el patrón de actividad presente en cada capa superior puede pensarse como un patrón de características que pueden ser reconocidas por unidades en las capas subsiguientes. (Figura 31)

El patrón de salida generado puede pensarse como un mapa de características que provee una indicación de muchas diferentes combinaciones de características en la entrada.

El efecto total de este comportamiento es que la red de retropropagación provee un medio efectivo para permitir, a un sistema de cómputo, reconocer patrones de datos que pueden estar incompletos o alterados y reconocer patrones de una entrada parcial.

A una red neuronal se le llama red de mapeo si es capaz de computar algunas relaciones funcionales entre sus entradas y sus salidas. La utilidad de esto radica en que, en muchos casos, necesitamos realizar un mapeo muy complicado, donde no sabemos como definir la relación funcional, pero conocemos ejemplos del mapeo correcto. En este tipo de situaciones, el poder de las redes neuronales para descubrir sus propios algoritmos es extremadamente útil.



Describamos de manera formal y matemática la red de retropropagación.

La red de retropropagación es una red multicapa de alimentación hacia adelante, cuyas capas están completamente interconectadas. De este modo no hay conexiones de retroalimentación (Figura 32) y no hay conexiones que permitan a una capa pasar directamente a una capa posterior. Se permite más de una capa oculta.

Supóngase que tenemos un conjunto de P pares de vectores, $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_m)$ los cuales son ejemplos de un mapeo funcional:

$$y = \phi(x); x \in R^N, y \in R^M$$

Se necesita entrenar la red de forma tal que aprenda una aproximación de:

$$0 = y' = \phi'(x)$$

Las relaciones que estamos tratando de mapear son comúnmente no linealidades multidimensionales así que empleamos una versión iterativa del método de mínimos cuadrados (LMS), llamada técnica de paso descendiente.

Para comenzar revisemos las ecuaciones de proceso de información. Un vector de entrada $x_p = (x_{p1}, x_{p2}, \dots, x_{pm})^t$, se aplica a la capa de entrada de la red. Los componentes de un patrón de entrada constituyen las entradas a los nodos en la capa i .

Las unidades de entrada distribuyen los valores a las unidades de capa oculta. La entrada a la j -ésima capa oculta es:

$$net_j = \sum_{i=1}^N w_{ji}^h o_i + \theta_j^h$$

Donde w_{ji} es el peso en las conexiones en la i -ésima unidad de entrada θ_j^h es el término de umbral. El índice superior h se refiere a cantidades en la capa oculta. Asume que la activación de este nodo es igual a la entrada de la red, entonces la salida de un nodo en la capa j es:

$$i_{pj} = f_j^h(net_{pj}^h)$$

Las ecuaciones para los nodos de salida son:

$$net_{pk}^o = \sum_{j=1}^L (w_{kj}^o i_{pj}) + \theta_k^o$$

y las correspondientes salidas son:

$$o_{pk} = f_k^o(net_{pk}^o)$$

donde f es la función de activación. El índice superior o se refiere a las cantidades en las capas de salida.

El conjunto inicial de valores de pesos representa una primera aproximación a los pesos apropiados para el problema. A diferencia de otros métodos la técnica que empleamos aquí no depende de hacer una buena primera aproximación. Los pesos pueden ser pequeños valores aleatorios entre ± 0.5 ; asimismo para el término del umbral, de forma tal que dé una unidad en la ecuación de entrada a la red.

Es una práctica común tratar este umbral como otro peso que está conectado a una unidad ficticia que siempre da como salida 1. Esto funciona de la siguiente forma:

En la ecuación:

$$net_{pk}^o = \sum_{j=1}^L (w_{kj}^o i_{pj}) + \theta_k^o$$

Tomando las definiciones $\theta_k^o = w_{k(L+1)}^o$ y $i_{p(L+1)} = 1$ tenemos:

$$net_{pk}^o = \sum_{j=1}^{L+1} w_{kj}^o i_{pj}$$

Así θ_i' es tratado como un peso. Otra posibilidad es remover el término de umbral en conjunto. Su uso es opcional.

Se describe a continuación el procedimiento básico para entrenar la red:

- Se aplica un vector de entrada a la red y se calculan los correspondientes valores de salida.
- Se comparan las salidas actuales con las salidas correctas y se determina la medida del error.
- Se determina en que dirección (+ o -) cambia cada peso para reducir el error.
- Se determina la cantidad en la cual cada peso cambia.
- Se aplican las correcciones a los pesos.
- Se repiten los pasos anteriores con todos los vectores de entrenamiento, hasta que el error para todos los vectores en el conjunto de entrenamiento sea reducido a un valor aceptablemente pequeño.

El algoritmo de retropropagación ha sido probado con problemas determinísticos tales como el problema de OR-Exclusivo, en problemas relacionados a síntesis y reconocimiento de lenguaje y a problemas relacionados con reconocimiento visual de patrones y caracteres. Se ha encontrado que funciona bien en muchos casos y que encuentra buenas soluciones a los problemas propuestos.

Una demostración del poder de este algoritmo fue dada por Sejnowski¹⁴. El entrenó un perceptrón de 2 capas con 120 unidades ocultas y mas de 20,000 pesos para convertir letras a reglas de transcripción fonética (NetTalk). La entrada eran palabras de 7 letras y la salida era la transcripción fonética de estas.

Este perceptrón tuvo un rango de error del 5% con un entrenamiento de 1024 palabras.

¹⁴ - T. Sejnowski, C. Rosemberg

* NETalk: Una red en paralelo que aprende a leer" John Hopkins University, 1986.

III. 7.2 Regla Delta Generalizada.

Las redes semilineales de alimentación hacia adelante como la reportada por Rumelhart, Hinton y William, en 1986, han demostrado ser sistemas efectivos para discriminantes de aprendizaje de un cuerpo de ejemplos.

La arquitectura del sistema es idéntica a la red de retropropagación excepto que la función de activación para los nodos es una función analítica. Este algoritmo de aprendizaje para la red de retropropagación está dado por una ley de cambio de peso iterativa llamada **Regla LMS** o **Regla Delta Generalizada**.

Se conoce como Regla Delta Generalizada, porque es una generalización de la Regla Delta usada para perceptrones de multicapa.

Cambiaremos un poco la notación usada con la red de retropropagación para trabajar con la Regla Delta Generalizada.

En la Regla Delta Generalizada los componentes de un patrón de entrada constituyen las entradas a los nodos en la capa i .

La entrada a la red a un nodo en la capa j es:

$$net_j = \sum w_{ji}^h o_i$$

La salida del nodo j es:

$$o_j = f(net_j)$$

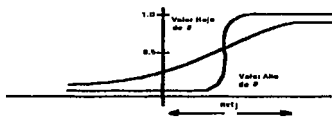
donde f es la función de activación.

Para una función de activación sigmoideal tenemos:

$$o_j = \frac{1}{1 + e^{-(net_j - \theta_j)/\theta_o}}$$

En esta expresión el parámetro θ_j sirve como un umbral. El efecto de un θ_j positivo es el de cambiar la función de activación a lo largo de la parte positiva del eje horizontal x , y el efecto de θ_o es el de modificar la forma de la sigmoide. Un valor bajo de θ_o tiende a hacer que la sigmoide tome las características de una unidad de umbral lógico, un valor grande de θ_o da como resultado una función de variación más gentil. El efecto de esto se ilustra en la Figura 33.

Figura 33 Umbral Sigmoideal con Alteraciones de θ



Continuando con nuestra descripción del proceso computacional tenemos que para los nodos en la capa k la entrada es:

$$net_k = \sum w_{ij}^h o_j$$

y las correspondientes salidas son:

$$o_k = f(net_k)$$

En la fase de entrenamiento se presenta el patrón $x_p = \{i_p^n\}$ como entrada y hacemos que la red ajuste el conjunto de pesos en todas las conexiones y también todos los umbrales en los nodos, de forma tal que las salidas deseadas t_{pk} sean obtenidas en los nodos de salida. Una vez que este ajuste sea completado presentamos un par de x_p y $\{t_{pk}\}$ y hacemos que la red aprenda la asociación también. De hecho hacemos que la red encuentre un simple conjunto de pesos y umbrales que satisfagan todos los pares (entrada, salida) que le sean presentados. Este proceso puede tener una muy extenuante tarea de aprendizaje y no siempre es satisfactoriamente completado.

En general, las salidas $\{o_{pk}\}$ no serán las mismas que el objetivo o valores deseados $\{t_{pk}\}$.

Para cada patrón el cuadrado del error es:

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

y el error promedio del sistema es:

$$E = \frac{1}{2} \rho \sum_p \sum_k (t_{pk} - o_{pk})^2$$

donde el factor de $1/2$ se inserta por conveniencia matemática para el cálculo posterior de derivadas. Como en el resultado final aparece una constante arbitraria, la presencia de este factor no afecta la respuesta.

En el algoritmo original de Regla Delta Generalizada formulado en 1986 por Rumelhart, Hinton y Williams para el aprendizaje de pesos y umbrales, el procedimiento para aprender el conjunto correcto de pesos consiste en variar el error en una forma calculada para reducir el error E_p , tan rápido como sea posible. A menudo es conveniente imaginar a E_p como una superficie en espacio de pesos.

Para determinar la dirección en la cual cambian los pesos, calculamos el negativo del gradiente de E_p , ∇E_p , con respecto a los pesos, w_{ij} .

En general se obtienen diferentes resultados dependiendo de si se saca el gradiente buscado, en espacio de pesos, en las bases funcionales de E_p ó E . En el primer caso las correcciones a los pesos son hechas secuencialmente, basándose en que el aprendizaje se obtiene de la secuencia de los patrones, uno a la vez. El segundo procedimiento es factible para reconocimiento de patrones adaptativos, pero es diferente al proceso correspondiente que se realiza en las redes neuronales biológicas. Aquí se realiza una verdadera búsqueda por gradientes para errores mínimos del sistema; debe estar basada en la minimización de la expresión:

$$E = \frac{1}{2} \rho \sum_p \sum_k (t_{pk} - o_{pk})^2$$

Para la Regla Delta Generalizada el caso de interés es el primero.

Trabajando con E_p , y omitiendo el subíndice p por conveniencia escribimos la expresión del error como:

$$E = \frac{1}{2} \sum_k (t_k - o_k)^2$$

Se logra una convergencia hacia valores más exactos de los pesos y los umbrales, tomando cambios incrementales de Δw_{ij} proporcionales a $-\partial E / \partial w_{ij}$, esto es:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

Sin embargo, E , el error se expresa en términos de las salidas o_k , cada una de las cuales es la salida no lineal del nodo k . Esto es:

$$o_k = f(\text{net}_k)$$

donde net_k es la entrada al k -ésimo nodo y por definición es la suma lineal de los pesos de todas las salidas de la capa anterior:

$$\text{net}_k = \sum w_{kj} o_j$$

La derivada parcial $-\partial E / \partial w_{ij}$ puede ser evaluada usando la regla de la cadena:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \text{net}_k} \frac{\partial \text{net}_k}{\partial w_{ij}}$$

Usando la expresión:

$$\text{net}_k = \sum w_{kj} o_j$$

Obtenemos:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \text{net}_k} \sum w_{kj} o_j = o_j$$

Definimos ahora:

$$\delta = \frac{\partial E}{\partial \text{net}_k}$$

y escribimos la expresión:

$$\Delta w_{ij} = \eta \delta_k o_j$$

Esta es una expresión de forma similar a la Regla Delta Simple para perceptrones de multicapa, dado en la expresión:

$$\Delta w = \eta \delta x$$

Para calcular:

$$\nabla = \delta = -\frac{\partial E}{\partial \text{net}_k}$$

usamos la regla de la cadena para expresar la derivada parcial en términos de 2 factores, uno expresa el rango de cambio del error con respecto a la salida o_k y el otro expresa el rango de cambio de la salida del nodo k con respecto a la entrada a ese mismo nodo. Esto es:

$$\delta_k = \frac{\partial E}{\partial \text{net}_k} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial \text{net}_k}$$

Los 2 factores son obtenidos como sigue:

$$\frac{\partial E}{\partial o_k} = -(t_k - o_k)$$

$$\frac{\partial o_k}{\partial net_k} = f'(net_k)$$

De lo cual obtenemos:

$$\nabla_k = \delta_k = (t_k - o_k) f'(net_k)$$

para cualquier nodo k capa de salida, y tenemos:

$$\Delta w_{kj} = \eta (t_k - o_k) f'_k'(net_k) = \eta \delta_k o_j$$

Son diferentes las circunstancias si los pesos no afectan los nodos de salida directamente. Así escribimos:

$$\begin{aligned} \Delta w_{\mu j} &= -\eta \frac{\partial E}{\partial w_{\mu j}} \\ &= -\eta \frac{\partial E}{\partial net_j} \frac{\partial net_j}{\partial w_{\mu j}} \\ &= -\eta \frac{\partial E}{\partial net_j} o_{\mu} \\ &= \eta \left[\frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \right] o_{\mu} \\ &= \eta \left[\frac{\partial E}{\partial o_j} \right] f'_j'(net_j) o_{\mu} \\ &= \eta \delta_j o_{\mu} \end{aligned}$$

Sin embargo el factor $\partial E / \partial o_j$ no puede ser evaluado directamente. De hecho lo escribimos en términos de cantidades que son conocidas y otras cantidades que pueden ser evaluadas.

Específicamente escribimos:

$$\begin{aligned} -\frac{\partial E}{\partial o_j} &= -\sum_k \frac{\partial E}{\partial net_k} \frac{\partial net_k}{\partial o_j} \\ &= \sum_k \left[-\frac{\partial E}{\partial net_k} \right] \frac{\partial}{\partial o_j} \sum_m w_{km} o_m \\ &= \sum_k \left[-\frac{\partial E}{\partial net_k} \right] w_{kj} \\ &= \sum_k \delta_k w_{kj} \end{aligned}$$

Vemos que, en este caso

$$\delta_j = f_j'(net_j) \sum_k \delta_k w_{kj}$$

Esto es, las deltas en nodos internos pueden ser evaluadas en términos de las deltas de una capa superior. De esta forma, comenzando en la capa superior (capa de salida) podemos evaluar δ_k usando la expresión:

$$\delta_k = (t_k - o_k) f_k'(net_k)$$

y podemos propagar los errores hacia atrás a las capas inferiores.

Sumarizando y usando el subíndice adicional p para denotar el número de patrones obtenemos:

$$\Delta_{pj} w_{kj} \Rightarrow \eta \delta_{pj} o_{pj}$$

Si los nodos j son nodos de la capa de salida entonces tenemos:

$$\delta_{pj} = (t_{pj} - o_{pj}) f_j'(net_{pj})$$

Sin embargo si los nodos j son nodos internos, entonces necesitamos evaluar δ_{pj} en términos de las deltas en una capa superior, es decir evaluar un término de error para las capas ocultas, esto es:

$$\delta_{pj} = f_j'(net_{pj}) \sum_k \delta_{pk} w_{kj}$$

En particular, si

$$o_j = \frac{1}{1 + e^{-\sum_i w_{ji} a_i}}$$

entonces:

$$\frac{\partial o_j}{\partial net_j} = o_j (1 - o_j)$$

y las deltas están dadas por las 2 expresiones siguientes:

$$\delta_{pk} = (t_{pk} - o_{pk}) o_{pk} (1 - o_{pk})$$

$$\delta_{pj} = o_{pj} (1 - o_{pj}) \sum_k \delta_{pk} w_{kj}$$

para las capas de salida y para las unidades en las capas ocultas respectivamente.

Entonces las ecuación de actualización de pesos en la capa de salida y en las capas ocultas son:

$$w_{kj}(t+1) = w_{kj} + \eta \delta_{pj} t_{pj}$$

$$w_{ji}(t+1) = w_{ji} + \eta \delta_{pj} x_i$$

para la capa de salida y las capas ocultas respectivamente.

Generalizando, esta regla de actualización de pesos está dada por:

$$w(t+1) = w(t) + 2\mu \varepsilon_k x_{ki}$$

Donde μ es una constante positiva, x_{ki} es la i -ésima componente del k -ésimo vector de entrenamiento y ε_k es la diferencia entre la salida actual y el valor correcto, $\varepsilon_k = (\delta_k - y_k)$.

Esta es la **Regla Delta Generalizada**.

Note que los umbrales θ_j son aprendidos de la misma forma que los otros pesos. Simplemente imagine que θ_j es el peso de una unidad que siempre tiene como salida el valor de la unidad.

También note que la derivada $\partial o_j / \partial net_j = o_j(1-o_j)$, alcanza su máximo para $o_j=0.5$ ya que $0 \leq o_j \leq 1$, se aproxima a su mínimo cuando o_j se acerca a 0 ó a 1. Ya que el cambio en el peso es proporcional a esta cantidad, esta claro que los pesos que están conectados a unidades en su rango medio, la mayoría son cambiados.

En algún sentido, estas unidades son aún impenetrables y no se sabe si encienden o apagan. Los pesos cambian rápidamente bajo estas condiciones y esta característica probablemente contribuye a la estabilidad del procedimiento de aprendizaje.

Es importante notar que para la función de activación dada por:

$$o_j = \frac{1}{1 + e^{-\sum_i w_{ji} o_i}}$$

un nodo no puede tener valores de salida de 0 ó 1, sino valores de pesos infinitamente largos negativos ó positivos. Por lo tanto, en el modo de aprendizaje, los valores de 0.9 y 0.1 pueden bastar para especificar valores binarios del objetivo de salida. En el aprendizaje w_{ji} , es buena práctica calcular $\Delta_p w_{ji}$ para cada patrón en el conjunto de patrones de entrenamiento y tomar:

$$\Delta w_{ji} = \sum \Delta_p w_{ji}$$

El procedimiento de aprendizaje por lo tanto consiste, en la red que comienza apagada, con un conjunto de valores de peso aleatorios, escogiendo un patrón del conjunto de entrenamiento y usando ese patrón como entrada evaluando las salidas de una forma de alimentación hacia adelante.

Los errores en las salidas generalmente serán bastante grandes, por lo que es necesario hacer cambios en los pesos usando el procedimiento de retropropagación (BPN) la red calcula $\Delta_p w_{ji}$ para todos los w_{ji} en la red para ese particular p . Este procedimiento se repite para todos los patrones en el conjunto de entrenamiento hasta obtener el resultado Δw_{ji} para todos los pesos de esa presentación.

Se hacen las correcciones a los pesos y las salidas son de nuevo evaluadas en forma de alimentación hacia adelante. Discrepancias entre los valores actuales y las salidas objetivo resultan de nuevo en la evaluación de los cambios de los pesos.

Después de que se completa la presentación de todos los patrones, en el conjunto de entrenamiento, se obtiene un nuevo conjunto de pesos y nuevas salidas son evaluadas de forma de alimentación hacia adelante.

La red puede ser hecha, para seguir el error del sistema y también los errores para patrones individuales. En un ejercicio de aprendizaje exitoso el error del sistema se decrementará con el número de iteraciones y el procedimiento convergerá a un conjunto estable de pesos, el cual solamente exhibirá pequeñas fluctuaciones en valor cuando nuevo aprendizaje sea intentado.

Hay algunos otros problemas que debemos tener en mente cuando implementamos tales redes. Existe por ejemplo la pregunta de como debe elegirse el valor de η . Este no es un problema nuevo o inusual; esto es común a todos los métodos de paso descendiente de funciones de mínimo local. Como puede esperarse, un gran valor de η corresponde a aprendizaje rápido pero puede también resultar en oscilaciones.

Rummelhart, Hinton y Williams sugirieron que las expresiones:

$$\begin{aligned}\Delta w_{kj} &= \eta (t_k - o_k) f_k' (net_k) o_j = \eta \delta_k o_j \\ \Delta_p w_{ij} &= \eta \delta_{pj} o_{pi}\end{aligned}$$

pueden ser modificadas para incluir una búsqueda del término momentum η . Esto es:

$$\Delta w_{ij}(n+1) = \eta (\delta_j o_i) + \alpha \Delta w_{ij}(n)$$

donde la cantidad $(n+1)$ es usada para indicar el $(n+1)$ -ésimo paso y α es una constante proporcional. El segundo término en la expresión es usado para especificar que el cambio en w_{ij} en el $(n+1)$ -ésimo paso debe ser de alguna forma similar al cambio obtenido en el n -ésimo paso. De esta forma alguna inercia se interconstruye y el *momentum* en el rango del cambio se conserva en algún grado. Exámenes del error del sistema E , sobre un gran número de pasos en un acercamiento iterativo a una solución generalmente mostrará que una α finita tiende a suavizar las oscilaciones pero puede también servir para disminuir el rango de aprendizaje.

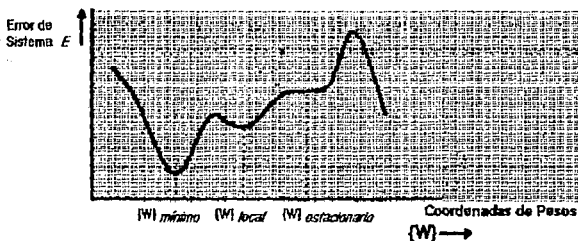
Si las correcciones de los pesos son obtenidas después de la presentación de cada patrón, el método no es verdaderamente un procedimiento de búsqueda de gradiente. En adición el valor de η necesita ser pequeño de otra forma grandes excursiones pueden realizarse en el espacio peso.

Note que a la red no se le debe permitir comenzar con un conjunto de pesos iguales. Se ha demostrado que no es posible proceder de una configuración tal a una configuración de pesos desiguales, aún si a la segunda le corresponde el más pequeño error del sistema.

Otro problema concierne a la pregunta si el sistema puede estar atrapado en algún mínimo local o aún en algún punto estacionario, o quizá oscila entre tales puntos. Bajo tales circunstancias el error del sistema se mantiene grande, sin tomar en cuenta de cuantas iteraciones son realizadas. Esta situación se describe esquemáticamente en la Figura 34.

En esta figura se muestra el error del sistema como una función en espacio peso. Esperamos aprender los pesos, en el punto $\{w\}_{\text{mínimo}}$ pero puede estar atrapado en el punto $\{w\}_{\text{local}}$ o aún en el punto $\{w\}_{\text{estacionario}}$.

Figura 34 Función Espacio Peso



En reconocimiento de patrones y en aprendizaje los procesos "gemelos" de generalización y especialización son muy importantes.

La generalización permite a un sistema de reconocimiento de patrones funcionar competentemente a través de un espacio de patrones ejemplos. La especialización permite a tal sistema reestablecerse de sus errores y probarse el mismo.

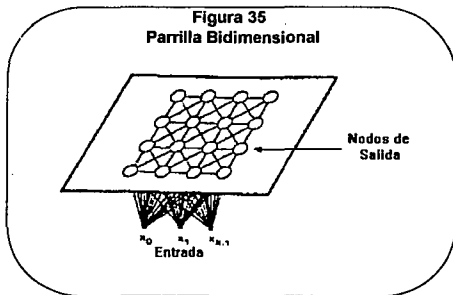
La Regla Delta Generalizada esta lejos de ser perfecta o aún perfectamente entendida.

III. 8 Mapa de Características de Auto-Organización de Kohonen.

Un importante principio organizativo de los canales sensoriales en el cerebro es que la distribución de las neuronas esta ordenada y a menudo refleja algunas características físicas del estímulo externo que está siendo sentido. Por ejemplo en cada nivel del canal auditivo, las células y las fibras están ordenadas anatómicamente en relación a la frecuencia, la cual da la mayor respuesta en cada neurona. Esta organización tonotópica en el canal auditivo se extiende hasta la corteza auditiva.

Dado que gran parte de la organización de bajo nivel esta genéticamente predeterminada, es probable que algo de la organización en los niveles altos sea creada durante el aprendizaje por algoritmos los cuales fomentan la auto-organización.

Kohonen¹³ presenta un algoritmo el cual produce lo que él llamó **Mapa de Características de Auto-Organización**, que es similar a lo que ocurre en el cerebro. El algoritmo de Kohonen crea un vector cuantizador, ajustando los pesos de las entradas a nodos comunes a M nodos de salida colocados en una parrilla bidimensional, como se muestra en la Figura 35.



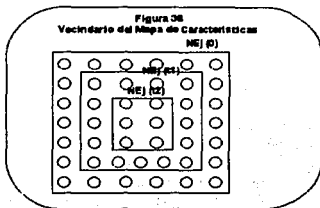
Los nodos de salida están extensamente interconectados con muchas conexiones locales. Los vectores de entrada de valores continuos son presentados secuencialmente sin especificar la salida deseada. Después de que suficientes vectores de entrada son presentados, los pesos serán agrupados específicamente o vectores centrales que ejemplifican el espacio de entradas tal que el punto de la función de densidad de los vectores centrales tiende a aproximar la función de densidad de probabilidad de los vectores de entrada.

Además, los nodos serán organizados de forma tal que los nodos topológicamente cercanos sean sensibles a entradas físicamente similares.

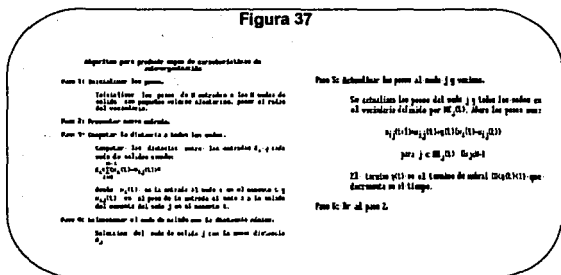
Los nodos de salida serán entonces ordenados de una forma natural. Esto puede ser importante en sistemas complejos con muchas capas de procesamiento ya que esto puede reducir la longitud de las conexiones intercapas.

¹³ T. Kohonen, *Self-Organization and Associative Memory*. Springer-Verlag 1984.

El algoritmo que forma el mapa de características requiere un vecindario definido alrededor de cada nodo como se muestra en la Figura 36. Este vecindario decremente ligeramente en tamaño con respecto al tiempo.



El algoritmo de Kohonen se describe en la Figura 37.



Los pesos entre los nodos de entrada y salida son inicialmente puestos a pequeños valores aleatorios y se presenta una entrada. Se calcula la distancia entre las entradas y todos los nodos. Si los pesos de los vectores son normalizados para tener una longitud constante (la suma del cuadrado de los pesos de todas las entradas a cada salida son idénticas) entonces el nodo con la mínima distancia Euclídea puede encontrarse usando la red de la Figura 36 para formar el producto punto de las entradas y los pesos.

La selección del nodo con la distancia mínima cambia a un problema de encontrar el nodo con el valor máximo. Este nodo puede ser seleccionado usando inhibición lateral extensiva. Una vez que se selecciona este nodo, los pesos a este y a todos los demás nodos en su vecindario son modificados para hacer que estos nodos respondan mejor a la entrada actual.

Este proceso se repite para posteriores entradas. Los pesos eventualmente convergen y son fijados después de que el término de umbral se reduce a 0.

Este algoritmo puede trabajar relativamente bien con patrones alterados, porque el número de clases es fijo, los pesos se adaptan lentamente y la adaptación se detiene después del entrenamiento.

Este algoritmo es así mismo un cuantizador de vector viable, cuando el número de conjuntos deseados puede ser especificado antes del uso y la cantidad de datos de entrenamiento es relativamente grande, con respecto al número de conjuntos deseados.

Los resultados, sin embargo, pueden depender del orden de presentación de los datos de entrada para pequeñas cantidades de datos de entrenamiento.

CAPITULO IV.

EL PROBLEMA DE LA REPRESENTACION DE LA INFORMACION EN NEUROCOMPUTACION.

Resumen

Actualmente se considera a los cerebros biológicos (principalmente al cerebro humano) y a su memoria como los más complicados sistemas de procesamiento de información que existen en la naturaleza.

Además de ser capaces de almacenar una cantidad inconmensurable de información, tienen la característica de hacer inferencias con la información que ya poseen; disponen de un sinnúmero de formas para recuperar la información almacenada; realizan sus operaciones a velocidades increíbles; tienen funciones de análisis y síntesis y son capaces de olvidar selectivamente.

Pero aún no conocemos completamente la forma en que los cerebros biológicos almacenan la información, se han propuesto varios modelos sobre la forma del almacenamiento de la información. El más aceptado es el conocido como **Modelo de Holograma**¹⁴, el cual es análogo a la teoría de los hologramas en Física, este modelo propone que la información no se almacena en una sola célula nerviosa, sino que la información se multiplica en toda la corteza cerebral.

Al problema de encontrar la forma en que el cerebro almacena la información se le ha dado el nombre de **Representación de Información**.

En Neurocomputación el problema de la Representación de la Información se extiende a simular en computadoras la forma en que los cerebros biológicos realizan el almacenamiento de la información.

¹⁴. - Propuesto al final de los 60's por el Doctor Karl H. Pribram.

IV.1 El problema de la representación de la información en Neurocomputación.

El tema de **representación** es central en la investigación tanto en las ciencias cognitivas como en la Neurocomputación. Básicamente se refiere a como los sistemas neurocomputacionales (y por analogía los humanos) codifican en forma abstracta, la información externa en una forma comprimida pero conservando todas sus características para su posterior utilización.

Una Red Neuronal se define¹⁷ como un sistema de cómputo hecho de algunos elementos de proceso simples, altamente interconectados, los cuales procesan información debido a su estado dinámico de respuesta a estímulos externos.

Una computadora serial, en su forma más básica, es un procesador central simple, altamente complejo, que puede direccionar y tener acceso a un arreglo de localidades de memoria.

Los datos y las instrucciones son almacenadas en las localidades de memoria.

El procesador toma estas instrucciones y cualquier dato requerido por las instrucciones, de las localidades de memoria, ejecuta la instrucción y almacena cualquier resultado obtenido, en una localidad de memoria específica. De esta forma podemos ir a una localidad específica de memoria, revisar su contenido y saber si se trata de un dato o de una instrucción.

Todo sistema de cómputo serial (y aún uno en paralelo) es esencialmente secuencial: Todo ocurre en una secuencia determinística de operaciones, establecidas por un algoritmo, diseñado por un programador humano.

El programador de Redes Neuronales no especifica un algoritmo a ser ejecutado por cada elemento de proceso. De hecho el programador especifica interconexiones, funciones de transferencia y leyes de entrenamiento de la red, entonces aplica las entradas necesarias a la red y la deja reaccionar.

A diferencia del sistema serial tradicional, una Red Neuronal no es ni secuencial ni necesariamente determinística. Además no tiene arreglos de memoria separados para almacenar datos. Los elementos que componen una Red Neuronal no son unidades de procesamiento central altamente complejas, de hecho, una Red Neuronal esta compuesta de muchos elementos de proceso simples, que típicamente hacen un poco mas que la simple suma de los pesos de todas sus entradas. La Red Neuronal no ejecuta una serie de instrucciones, ésta responde en paralelo a las entradas que le son presentadas.

Las redes neuronales no ejecutan programas, sino que se "comportan" dada una entrada específica. Estas "reaccionan", se "organizan", "aprenden" y "olvidan".

Los resultados no se almacenan en localidades específicas de memoria, sino que estos persisten en el estado total de la red después de que se ha alcanzado alguna condición de equilibrio.

¹⁷ - Definición de Richard Nielsen, Inventor de Transputador Comercial.

De esta forma, el conocimiento (aprendizaje, organización y reacción a estímulo) dentro de una red neuronal es más una función de la estructura o arquitectura de la red, que el contenido de una dirección de memoria particular.

Una definición simple y generalmente aceptada es la siguiente: el aprendizaje es un cambio en la estructura del conocimiento de un sistema, que le permite realizar efectivamente una tarea dada. Cuando se trata de aprender una tarea, se repite esta un cierto número de veces hasta que, en las neuronas, se establecen una serie de conexiones que logran el aprendizaje.

Los organismos vivos deben ser capaces de adaptarse a un medio ambiente en constante cambio. Su supervivencia puede depender de su habilidad para cambiar y adaptarse a su medio, esto es de su capacidad de aprender. El aprendizaje en los organismos vivos produce cambios químicos, anatómicos y fisiológicos en sus cerebros, algunos temporales y otros permanentes.

Construir máquinas que aprendan ha sido una de las principales metas de la investigación en Inteligencia Artificial.

Difícilmente podemos decir que hemos creado un sistema de aprendizaje inteligente cuando escribimos un programa para resolver un problema abstracto (Ejemplo: el problema de las Torres de Hanoi). Cualquier cambio en el sistema cuando el programa exitosamente resuelve el problema, ocurre en el programador y no en el programa.

Las redes neuronales ofrecen una forma más natural de simular los procesos cognitivos e inteligentes.

El aprendizaje en una Red Neuronal significa encontrar un conjunto apropiado de pesos. Cuando una red neuronal aprende como realizar una tarea dada, después de un cierto número de intentos, ocurren cambios internos en el sistema. Comenzando con un conjunto aleatorio de pesos, el aprendizaje cambiará estos pesos a un nuevo conjunto, que almacena el conocimiento necesario para realizar la tarea aprendida.

Aún no entendemos totalmente los mecanismos cognitivos usados en el cerebro humano y por lo tanto, aún no podemos emularlos completamente. Como resultado nuestras redes neuronales artificiales reflejan nuestra ignorancia.

Los procesadores de primer nivel (periférico) tales como procesos sensores involucrados en visión y audición son mejor entendidos y nuestras redes neuronales son mejores para simularlos. No sabemos aún como poner Redes Neuronales para generar procesos inductivos y cognitivos de alto nivel.

IV.2 Hipótesis de representación.

El conocimiento dentro de una red neuronal no es almacenado dentro de una localidad de memoria específica, en cambio, el conocimiento se distribuye y almacena, en toda la red, tanto en la forma de interconectar los elementos de proceso, como en la importancia que se da (o valores de peso) a cada entrada de los elementos de proceso.

La entrada a la red de un patrón dispara representaciones neuronales para sacar la información que se conoce, como por ejemplo un patrón ya aprendido o parte de este. Esto produce un patrón de actividad inestable que gradualmente es alterado por la dinámica de la red.

Colectivamente y en paralelo las neuronas de la red comienzan una lucha que llevara a restaurar un patrón estable, el patrón ya aprendida.

La fuerza de las sinapsis artificiales gobierna que patrones de actividad son estables y también determina lo que el modelo es capaz de recordar. Estas fuerzas o pesos deben ser establecidas en el entrenamiento. Aquellos patrones de actividad que la red debe ser capaz de recordar deben ser los mas reforzados ya que tales patrones deben ser los mas estables.

Esta naturaleza distribuida de la información memorizada al mismo tiempo asegura la robustez de la memoria. Esto implica, que aun si muchas de las conexiones de la red, el refuerzo mínimo, el cual corresponde a una memoria particular no varia mucho. Aun después de una destrucción extensiva de una red neuronal es posible que esta recuerde lo que ha memorizado mas o menos perfectamente.

La prueba del teorema del aprendizaje del perceptrón (Rosenblat 1969) demostró que un perceptrón podía aprender cualquier cosa que pudiera ser representada. Es necesario diferenciar entre aprendizaje y representación. Representación se refiere a la habilidad de un red neuronal (un perceptrón en este caso) de simular una función específica. El Aprendizaje requiere la existencia de un procedimiento sistemático para ajustar los pesos de la red para producir la función a representar.

Hilbert fue uno de los primeros en preguntarse si es posible tener una función general que representara y realizara otras funciones matemáticas. Kolmogorov (1957) contestó que bajo ciertas limitaciones, es posible obtener tales funciones e implementarlas en un sistema de cómputo.

Los resultados obtenidos al inicio de los años 80's, mediante el uso de Redes Neuronales ha demostrado que, un sistema de cómputo es capaz de aprender a realizar tareas y utilizar esta información para obtener resultados.

Hetch-Nielsen (1987) y otros han generalizado la teoría de Kolmogorov hacia la representación de la información en neurocomputación. De esta forma, ahora la pregunta que se realiza es: Como un sistema de computo, que utiliza Redes Neuronales, aprende a realizar una tarea y utiliza esta información para obtener resultados.

Una Red Neuronal con una función con más de un mínimo, tiene por lo tanto más de una memoria. El recuerdo de una memoria particular necesariamente comienza un proceso dentro del área de atracción para una memoria particular.

La información de algunas memorias es codificada en algunos conjuntos de pesos sinápticos. Esto se investiga mediante la forma de la representación de la información en una Red Neuronal, sobre todo bajo la hipótesis de que la representación se realiza en las capas internas.

Hanson (1990) al hacer una revisión del problema de representación en Redes Neuronales consideró que en general hay tres hipótesis de representación:

- La **localista**, que indica que son ciertas partes locales de la red donde se codifica la información.
- La **generalizada**, que indica que son ciertas áreas extensas de la red donde se localiza la información.
- La **global**, que indica qué es el comportamiento total el que representa la información en la red.

Tomando como base estas 3 hipótesis de representación, nuestro análisis se realizará para averiguar si el comportamiento de una red neuronal es local, generalizado o global.

Estudiaremos el comportamiento matemático de las capas internas de una red neuronal del tipo Perceptron Multicapa de Retropropagación, con algoritmo de aprendizaje Regla Delta Generalizada.

**ESTA TESIS NO DEBE
SALIR DE LA BIBLIOTECA**

CAPITULO V.

APLICACION DEL ALGORITMO REGLA DELTA GENERALIZADA PARA EL APRENDIZAJE SUPERVIZADO.

Resumen

Se utilizara el Algoritmo de Regla Delta Generalizada como base para realizar la investigación del porque funcionan las Redes Neuronales.

Se retomaran la mayoría de los conceptos utilizados en los capitulos anteriores para poder dar una explicación clara y concisa del funcionamiento e implementación del Algoritmo Regla Delta Generalizada para el Aprendizaje Supervisado en el estudio de la representación del conocimiento.

Este capitulo será por si mismo una de las partes fundamentales de la tesis ya que la implantación y uso del Algoritmo Regla Delta Generalizada para el Aprendizaje Supervisado en la Representación del conocimiento, desarrollará una herramienta tanto para el presente estudio como para investigaciones posteriores en este u otros campos.

V.1 Implementación del algoritmo Regla Delta Generalizada.

Se implementó el algoritmo Regla Delta Generalizada en un programa, que permite almacenar en un archivo el valor de los pesos de las conexiones entre los elementos de proceso.

El algoritmo se implemento en lenguaje C por sus grandes capacidades matemáticas, velocidad, sencillez y velocidad.

El procedimiento utilizado para la implementación del algoritmo de la Regla Delta Generalizada expresado en forma de pseudocódigo general es el siguiente:

1. -Establecer la Arquitectura de la Red.
2. -Establecer los parámetros propios del algoritmo Regla Delta Generalizada.
3. -Aplicar el vector de entrada $X_p = (x_{p1}, x_{p2}, \dots, x_{pn})$
4. -Calcular los valores de entrada de la red a las unidades de la capa oculta:

$$net_{pj} = \sum_{i=1}^N w_{ji} x_{pi} \theta_j$$

5. -Calcular las salidas de la capa oculta:
$$i_{pj} = f_j(net_{pj})$$
6. -En la capa de salida, calcular los valores de entrada de cada unidad:

$$net_{pk} = \sum_{j=1}^N w_{kj} x_{pj} + \theta_k$$

7. -Calcular los valores de salida de la capa de salida:
$$o_{pk} = f_k(net_{pk})$$
8. -Calcular los términos de error de las unidades de salida:
$$\delta_{pj} = (y_{pk} - o_{pk}) f'_k(net_{pk})$$
9. -Calcular los términos de error de las unidades de la capa oculta.

$$\delta_{pj} = f'_j(net_{pj}) \sum_{k=1}^M \delta_{pk} w_{kj}$$

Note que los términos de error en las unidades de las capas ocultas son calculados antes de que los pesos de las capas ocultas sean actualizado

10. -Se procede a actualizar los pesos en la capa de salida:
$$w_{kj}(t+1) = w_{kj}(t) + \eta \delta_{pk} i_{pj}$$

11. -Actualizar los pesos en la capa oculta:
$$w_{ji}(t+1) = w_{ji}(t) + \eta \delta_{pj} x_i$$

El orden de la actualización de los pesos en una capa individual no es importante. Se debe de calcular el término de error:

$$E_p = 1/2 \sum_{k=1}^M \delta_{pk}^2$$

ya que esta cantidad es la medida de que tan bien la red esta aprendiendo. Cuando el error es aceptablemente pequeño para cada uno del par de vectores de entrenamiento, el entrenamiento puede detenerse.

V.2 Aplicación del algoritmo Regla Delta Generalizada para aprendizaje supervisado.

Se utilizó el procedimiento recomendado por Pao (1989) que es un programa optimizado de la Regla Delta Generalizada. Este programa permite cambiar el valor de los siguientes parámetros:

- 1.- Número de Elementos de Proceso.
- 2.- Número de Capas Internas.
- 3.- Número de Elementos de Proceso por Capa Interna.
- 4.- Número de Elementos de Proceso en la Capa de Salida.

Una característica importante de este programa es que además permite definir el valor de los parámetros propios de la Regla Delta Generalizada que son:

- Porcentaje de aprendizaje Eta (η)
- Porcentaje de momentum Alfa (α).
- Valor de la máxima tolerancia del error total e individual.
- Número máximo de iteraciones.

Como se explicó antes, el código se encuentra realizado en lenguaje C, implementado en un compilador Turbo C 2.0 de Borland. El algoritmo y notación usados son los mismos empleados en el Capítulo III.

El programa provee código para las siguientes tareas:

- 1.-Especificar la arquitectura de la red.
- 2.-Aprender pesos y umbrales con el uso de conjuntos de patrones de entrenamiento.
- 3.-Usar la red para obtener valores de salida para nuevos patrones, tanto para propósitos de clasificación o para estimación de valores de atributos asociados.

El programa permite al usuario especificar el número de unidades de entrada y salida, el número de capas ocultas y número de unidades en cada capa oculta.

Después de que se construye la red, toma lugar el aprendizaje en la red con un conjunto dado de ejemplos de entrenamiento.

Después del aprendizaje, toda la información relevante a la estructura de la red, incluyendo pesos y umbrales son almacenados en archivos.

Las salidas pueden generarse para nuevos patrones leyendo el archivo y reconstruyendo la red.

El código fuente obtenido de la implementación del algoritmo se encuentra en la sección de Anexos. Las Rutinas empleadas para el sistema de ventanas se encuentran detalladas en el archivo "ventanas.h", otras rutinas se encuentran en el archivo "util.c", ambos programas se expondrán también en esa sección.

El procedimiento experimental consistió en darle diferentes valores numéricos a la red, donde dada una serie de valores, debe obtener otros valores mediante una función de transformación no conocida por la red.

La tarea consiste en aprender una transformación y generar, dada una nueva serie de valores de entrada, una serie de valores en base a la función de transformación no conocida. Las tareas a aprender se dividieron en 3 tipos: Contigüidad, Simetría y Paridad.

El problema de contigüidad es cuando a la red se le presenta una cadena de N dígitos en la capa de entrada y el sistema tiene que distinguir entre las entradas de acuerdo al número de grupos (Por ejemplo: Bloques continuos) de +1's. Para este problema se resolvió el problema de "2 contra 3" usando todas las posibles entradas que tengan grupos de 2 o 3 unos como nuestro patrón de entrenamiento. Se utilizó para esto una arquitectura N: N: 1. Donde N es el número de elementos de proceso en cada capa.

En el problema de simetría la salida debe ser +1 (o un valor alto), si el patrón de entrada es simétrico alrededor de su centro y -1 (o un valor bajo) de otra forma. El número mínimo de unidades en las capas ocultas para resolver este problema es 2 y el número de capas ocultas debe ser $N > 2$.

En el problema de paridad se requiere que la salida sea +1 (valor alto) para números pares de bits y -1 (valor bajo) de otra forma. Este problema es computacionalmente más difícil que los otros 2, ya que la salida es sensible a un cambio en el estado, de cualquier unidad de entrada.

Para resolver este problema se usaron redes con una arquitectura N:2N:1 como las seleccionadas por Tesauro y Janssen.

A su vez estas tareas numéricas se dividieron en 2:

Aquellas que en una cantidad X de ensayos (o iteraciones de la Regla Delta Generalizada) la red aprende y otras donde no aprendió.

Después de realizar esto, los valores de los pesos sinápticos de la red serán analizados¹⁷.

¹⁷.-No se incluyen los resultados de los procedimientos experimentales por la siguiente razón: Para un experimento pequeño, una red neuronal 3:3:1 con 8 patrones de ejemplos, resolviendo un problema de paridad y ejecutando 2,000 iteraciones se obtienen 3,024,000 datos numéricos reales de 12 dígitos de longitud.

CAPITULO VI.

ANALISIS DE RESULTADOS.

Resumen

Los resultados se analizaran en base a la teoría matemática del algoritmo de Regla Delta Generalizada y además se analizaran mediante una técnica fractal.

Se necesita la aplicación de una técnica fractal por la cantidad de datos a analizar, por el tipo de datos que se obtienen, por comportamiento mismo de las redes neuronales y por el tipo de estudio que se realiza.

Del análisis de los datos obtenidos por la aplicación intensiva del Algoritmo Regla Delta Generalizada, podrán hacerse evaluaciones con respecto al grado y la forma de aprendizaje de las Redes Neuronales.

El análisis en sí deberá de dar conceptos de evaluación para entender como y cuando aprenden las Redes Neuronales.

Estos conceptos serán posteriormente muy importantes para poder obtener conclusiones posteriores.

VI.1 Análisis e Interpretación de resultados.

Los resultados obtenidos del uso extensivo del programa Regla Delta Generalizada deben ahora de ser analizados para poder obtener alguna información acerca de lo que ocurre dentro de la red neuronal y sobre la forma en la que ésta manipula la información para lograr el aprendizaje.

Este análisis estará enfocado a resolver alguna de las 3 hipótesis planteadas con respecto al problema de la representación de información.

Una vez analizados los datos, los resultados que arroje el análisis deberán de ser interpretados para poder concluir algo con respecto a alguna de las hipótesis de la representación de la información.

Es esperable, partiendo de las hipótesis planteadas que, se presente alguna de las siguientes situaciones:

- La red aprende debido a la codificación de información en ciertas partes de la red.
- La red aprende debido a la codificación generalizada entre algunos de los elementos de la red.
- La red aprende debido a la codificación global de la red.

VI.2 Técnicas de análisis de representación.

Existen diversas técnicas para el análisis de la representación tales como:

- Análisis de Correlaciones.
- Análisis Fractal.
- Análisis de Convergencia.
- Análisis Caótico.

Se optó por los métodos de análisis fractal y análisis de convergencia.

El análisis fractal se adoptó por las siguientes razones:

- Utiliza métodos con muy sólidos fundamentos matemáticos.
- Utiliza métodos de alto desempeño computacional.

El análisis de convergencia se adoptó por las siguientes razones:

- El algoritmo de la Regla Delta Generalizada se basa en la búsqueda de un punto óptimo global de la función de aprendizaje dada sobre un espacio de pesos, mediante una técnica de gradiente descendiente.
- No existe un teorema que asegure la convergencia del algoritmo Regla Delta Generalizada: como en cualquier procedimiento de minimización se puede encontrar un mínimo local que no corresponda a la solución, como se explico en el Capitulo III.

VI.3 Análisis Fractal.

Para el análisis de los datos se desarrollo un programa capaz de generar una gráfica fractal tridimensional de la función de aprendizaje, la cual se coloco sobre un espacio de pesos, funcionando sobre los pesos obtenidos en el aprendizaje de red.

De esta forma se puede observar la forma que adopta la función de aprendizaje.

La forma en que se comporta la red se ajusta, de forma muy semejante a la forma en que se comporta una gráfica fractal, en la mayoría de los casos y dentro de algunos límites.

Los valores obtenidos en la red mediante una serie de funciones se ajustan a una gráfica fractal.

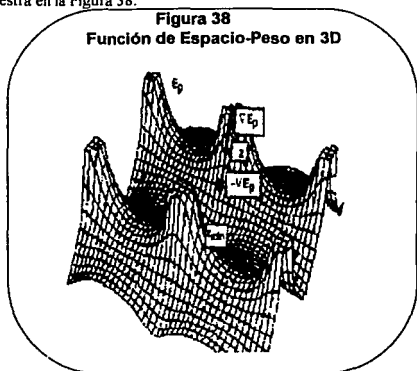
De esta forma se puede observar el comportamiento de la red para los diversos casos, con diversas arquitecturas y se aprecia al mismo tiempo la relación que guardan entre sí los valores de las interconexiones de la red.

También se podrá apreciar lo que ocurre con la función de aprendizaje y las conexiones cuando la red aprende, y cuando la red no aprende.

VI.4 Análisis de convergencia.

El análisis de convergencia se realizó de la siguiente forma:

- En la técnica de gradiente descendente de la función de aprendizaje, calculamos el gradiente descendente de E_p , con respecto a los pesos w . Entonces ajustamos el valor de los pesos de tal forma que el error total se reduzca, si colocamos la función de aprendizaje como una superficie sobre un espacio de pesos como se muestra en la Figura 38.



- Dada la gráfica fractal de la función de aprendizaje se realiza sobre ella el seguimiento del gradiente descendente, con respecto a los pesos obtenidos, durante el proceso de aprendizaje, para averiguar de que forma se llega a la convergencia, es decir, de que forma se obtiene el mínimo global de la función de aprendizaje y que relación tienen los pesos obtenidos en el proceso de aprendizaje, de esta forma se averiguará simultáneamente en que forma se obtiene la convergencia y la relación que guardan los pesos en la red con el aprendizaje, para poder así dar respuesta a alguna de las hipótesis planteadas.

De esta forma las 2 técnicas de análisis se interrelacionan y permiten una interpretación más eficiente y segura de lo que ocurre dentro de la red.

El código de este análisis se implementó, de la misma forma y para asegurar su compatibilidad con el Algoritmo Regla Delta Generalizada en Borland Turbo C Versión 2.0. Este código también se encuentra en la sección Anexos.

La presentación gráfica se realizó, por razones de portabilidad, únicamente para los 2 principales sistemas gráficos actuales CGA y VGA. Además de que los parámetros de la gráfica fractal y de visualización son modificables para la mayor facilidad de interpretación de los resultados.

CONCLUSIONES

La idea central del desarrollo, implementación y uso, de las redes neuronales es presentar a una red neuronal un conjunto de patrones de entrenamiento, patrón por patrón, y definir una salida específica para cada uno de ellos; si la salida es equivocada se cambiara la representación interna del patrón mediante el ajuste de los valores en sus interconexiones o pesos. Esto asegura que solo la representación correcta se almacenara y la representación interna de un patrón específico podrá ser regenerada.

El algoritmo de retropropagación usa unidades básicas con variables continuas y una función de respuesta sigmoide de entrada y salida. Una función de error la cual mide la desviación de la salida actual con respecto a la salida deseada, es minimizada por el algoritmo.

El aprendizaje se logra ajustando los pesos, para encontrar una buena representación interna, o sea el estado tomado por la capa interna en respuesta a la entrada presentada. A esto se le denomina manipulación de la representación interna.

Las redes de retropropagación tienden a desarrollar relaciones internas entre los nodos para organizar los datos de entrenamiento en clases de patrones. Esto sugiere el tipo de comportamiento que presenta la red para lograr el aprendizaje.

Esta idea de manipular la representación interna, mediante el ajuste de los pesos denota que el algoritmo necesita una unidad que conozca el estado de todas las neuronas y todos los pesos.

El aprendizaje de los pesos usando la Regla Delta Generalizada, ajusta el peso w y debe de conocer el estado de la neurona fuente j y el estado deseado de la neurona objetivo y . El estado correcto de las capas ocultas es desconocido. El realizar este ajuste depende del estado de la unidad que conoce el estado de todas las neuronas y todos los pesos, esto a su vez indicara si es necesario seguir con el aprendizaje, si el error es mayor que lo tolerado o si se puede terminar el aprendizaje, gracias a un error pequeño.

Esto puede denominarse como un "control global" de la red neuronal. Esto nos da la idea de que todas las unidades de en la capa interna están de alguna forma asociadas con características específicas del patrón de entrada como resultado del entrenamiento. De esta forma la red encuentra una representación interna que la capacita para generar la salida deseada cuando se le presenta una entrada. Asi redes de retropropagacion de acuerdo a las características que comparten con los patrones de entrada. Generalizando este concepto obtendremos que la red para obtener un aprendizaje debe de presentar un **comportamiento global**.

El significado de esto es que, para lograr el aprendizaje todas las conexiones de la red influyen de manera similar, interactuando entre sí, y que la representación interna se realiza en todas las conexiones de la red.

Los resultados experimentales dieron este mismo resultado, de forma que cuando la red lograba un aprendizaje, los pesos de las conexiones presentaban un valor similar y dentro de un rango reducido, de forma muy diferente a lo que ocurría en los casos cuando la red no aprendía. La función de aprendizaje para estos casos presenta pocos mínimos locales y permite al gradiente descendiente localizar el mínimo global de la función.

Cuando la red alcanza una solución aceptable (aprende) no hay garantía de que se haya alcanzado el mínimo global. Sin embargo si la solución es aceptable no importa si es un mínimo local o el mínimo global de la función.

Nuestros resultados demuestran además que en la mayoría de los casos donde no se obtiene un aprendizaje o donde el aprendizaje se torna muy lento, la función tiene una gran cantidad de mínimos locales que le impiden al gradiente descendiente llegar o acercarse al mínimo global de la función. Para algunos, se presenta el caso extremo donde la función es totalmente descendiente y no tiene un punto mínimo, lo que confirma que no existe un teorema que asegure la convergencia del algoritmo.

Otro resultado obtenido de este análisis (y demostrado por muchos investigadores) es el de que la arquitectura de la red ejerce una gran influencia en la forma en que la red aprende, en si se presenta a la red el mismo problema, con diferentes arquitecturas, en algunos casos lo resuelve y en otros no. Esto nos indica que el uso de nodos excesivos (superfluos) para la red, puede crear conflicto en el comportamiento global de la red.

Esto refuerza el resultado obtenido acerca del comportamiento global de la red.

Notas Finales.

El papel actual de las redes neuronales, es aparentemente, ser compañero de los sistemas tradicionales y no su reemplazo. Las neurocomputadoras han sido principalmente diseñadas como coprocesadores trabajando en conjunción con otros sistemas. Estos operan llamando subrutinas o procedimientos cuando se encuentra una aplicación de red neuronal.

Las redes neuronales deben ser usadas solamente para resolver problemas para los cuales, las alternativas tradicionales no existen o son insatisfactorias de cualquier forma. Si existe una alternativa perfectamente satisfactoria olvidese de las redes neuronales.

Antes de intentar resolver un problema mediante redes neuronales, se debe analizar el problema, para averiguar si se tiene una alternativa de solución para resolver el problema. Se debe investigar:

- 1.-Si existe un algoritmo para resolver el problema, si el algoritmo (de existir este) es efectivo.
- 2.-Si se pueden especificar reglas if-then para resolver el problema y si existe un experto capaz de ayudar a definir esta reglas.

Si la respuesta a alguna de estas cuestiones es afirmativa se tiene un problema de programación o un problema de sistema experto.

Al resolver un problema mediante redes neuronales se debe de analizar tipo de problema que se trate para seleccionar de esta forma el modelo de red neuronal que resuelva mejor el problema. La mayor parte de los que se inician en redes neuronales, eligen por lo regular el algoritmo de retropropagación que aunque es muy útil y fácil de implementar para muchos casos, pero este algoritmo no funciona correctamente para resolver algunos problemas.

Un análisis correcto del tipo de problema a resolver y los valores que este puede tomar facilitaran su solución mediante la elección correcta del mejor modelo para su solución.

ANEXOS

Código fuente obtenido al implementar el algoritmo Regla Delta Generalizada.

```
/*.....*/
/* Regla Delta Generalizada */
/* Implementación Realizada en Turbo C versión 2.0 */

/* Autor: */
/* Aguilar Castillejos Raul Antonio */
/*.....*/

#include <dos.h>
#include "string.h"
#include "process.h"
#include "stdio.h"
#include "math.h"
#include "ctype.h"
#include "conio.h"
#include "time.h"

/* Definicion para funcionar en VAX */

#ifndef VAX
#include "alloc.h"
#endif

/* Definicion de constantes */

#define NMXUNIT 25 /* Número máximo de unidades en una capa */
#define NMXHLR 5 /* Número máximo de capas ocultas */
#define NMXOATTR 25 /* Número máximo de salidas mostradas */
#define NMXINP 45 /* Número máximo de pruebas de entrada */
#define NMXIATTR 25 /* Número máximo de entradas mostradas */
#define SEXIT 3 /* Salida con buen ,éxito */
#define RESTRT 2 /* Reinicio */
#define FEXIT 1 /* Salida en caso de falla */
#define CONTNE 0 /* Continúa el cálculo */
```

```
/* Declaracion de variables */
```

```
float eta; /* Porcentaje de Aprendizaje */
float alpha; /* Porcentaje de momentum */
float err_curr; /* Error del sistema normalizado */
float maxe; /* Error maximo permitido del sistema */
float maxep; /* Error maximo permitido del diseño */
float sigma[NMXHLR];
float *wtptr[NMXHLR+1];
float *outptr[NMXHLR+2];
float *errptr[NMXHLR+2];
float *delw[NMXHLR+1];
float target[NMXINP] [NMXOATTR];
float input[NMXINP] [NMXIATTR],ep[NMXINP];
float outpt[NMXINP] [NMXOATTR];
```

```
int nunit[NMXHLR+2],nhlayer,ninput,ninattr,noutattr;
int result,cnt,cnt_num;
int nsnew,nsold;
int fplot10;
```

```
char task_name[20],buf[20];
```

```
FILE *fp1,*fp2,*fp3,*fp4,*fopen();
```

```
/* Sistema de ventanas */
```

```
#define NUMV 6
#include "b:ventanas.h"
#include "b:util.c"
```

```
/* Implementación de Funciones */
```

```
/* Generacion de Numeros Aleatorios */
```

```
long randseed = 568731L;
int random()
{
    randseed = 15625L * randseed + 22221L;
    return(( randseed >> 16 ) & 0x7FFF);
}
```

```
/* Localidades dinamicas promedio de la red */
```

```
init()
```

```
{  
  int len1,len2,i;  
  float *p1,*p2,*p3,*p4;  
  
  len1 = len2 = 0 ;  
  nunit[nhlayer+2] = 0 ;  
  for (i=0 ; i < (nhlayer + 2) ; i++)  
  {  
    len1 += ( nunit[i]+1 ) * nunit[i+1] ;  
    len2 += nunit[i] + 1 ;  
  }  
}
```

```
p1 = (float *) calloc(len1+1,sizeof(float)) ; /* Pesos */  
p2 = (float *) calloc(len2+1,sizeof(float)) ; /* Salida */  
p3 = (float *) calloc(len2+1,sizeof(float)) ; /* Errores */  
p4 = (float *) calloc(len1+1,sizeof(float)) ; /* D E L W */
```

```
/* Inicializacion de apuntadores */
```

```
wtptr[0] = p1 ;  
outptr[0] = p2 ;  
errptr[0] = p3 ;  
delw[0] = p4 ;
```

```
/* Inicializacion de mas apuntadores */
```

```
for (i=1 ; i < ( nhlayer + 1 ) ; i++)  
{  
  wtptr[i] = wtptr[i-1] + nunit[i] * (nunit[i-1]+1) ;  
  delw[i] = delw[i-1] + nunit[i] * (nunit[i-1]+1) ;  
}
```

```
for (i=1 ; i < ( nhlayer + 2 ) ; i++)  
{  
  outptr[i] = outptr[i-1] + nunit[i-1] + 1 ;  
  errptr[i] = errptr[i-1] + nunit[i-1] + 1 ;  
}
```

```

/* Inicializacion de umbrales de salida */
for (i=0 ; i < ( nhlayer + 1 ) ; i++)
{
    *(outptr[i] + nunit[i]) = 1.0; }
}

/* Inicializacion de los pesos. Rango rand[-0.5,0.5] */
initwt()
{
    int i,j ;

    for (j=0 ; j < nhlayer + 1 ; j++)
        for (i=0 ; i < (nunit[j]+1) * nunit[j+1] ; i++)
        {
            *(wptr[j]+i) = random() / pow(2.0,15.0) - 0.5 ;
            *(delw[j] + i) = 0.0 ;
        }
}

/* Especificacion de la arquitectura de la red */
set_up()
{
    int i ;
    clrscr();
    cprintf(" Parametros de aprendizaje\n\r");
    gotoxy(4,3);
    strcpy(buf,"0.9");
    cprintf("Porcentaje de Momento ETA ( Default = 0.9 ) : ");
    lee(69,11,5,buf,2);
    eta = atof(buf);
    gotoxy(4,5);
    strcpy(buf,"0.7");
    cprintf("Porcentaje de Aprendizaje ALPHA ( Default = 0.7 ) : ");
    lee(69,13,5,buf,2);
    alpha = atof(buf);
    gotoxy(4,7);
    strcpy(buf,"0.01 ");
    cprintf("Error Mximo Total ( Default = 0.01 ) : ");
    lee(62,15,10,buf,2);
    maxe = atof(buf);
    gotoxy(4,9);
    strcpy(buf,"0.001 ");
}

```

```

printf("Error Mximo Individual ( Default = 0.001): ");
lee(62,17,10,buf,2);
maxep = atof(buf);
gotoxy(4,11);
strcpy(buf,"1000 ");
printf("Numero mximo de iteraciones ( Default = 1000 ): ");
lee(69,19,5,buf,2);
cnt_num = atoi(buf);
clrscr();
printf("  Arquitectura de la red");
gotoxy(4,3);
strcpy(buf," ");
printf("Número de capas ocultas : ");
lee(48,11,3,buf,2);
nhlayer = atoi(buf);
for (i=0 ; i<nhlayer ; i++)
{
  strcpy(buf," ");
  gotoxy(9,5+i);
  printf("Número de Unidades para la capa oculta %d :",i+1 );
  lee(66,13+i,3,buf,2);
  nunit[i+1] = atoi(buf);
}
fcloseall();
gotoxy(4,7+i);
strcpy(buf,"1");
printf("Crear archivo de Error [Tipo (1) o de tipo (0)]:");
lee(68,15+i,2,buf,2);
fplot10 = atoi(buf);
clrscr();
printf("Peso – Valor Umbral - Valor Iter; Ep. Err:");
for(i=1;i<3*4+1;i++)
{
  gotoxy(3,1+i);
  printf("%d ",i);
}
for(i=1;i<3+2;i++)
{
  gotoxy(21,1+i);
  printf("%d ",i);
}
nunit[nhlayer+1] = noutattr;
nunit[0] = ninattr;

```

```

/* Lectura de archivo para arquitectura suffix_v.dat */
dread (taskname)
char *taskname;
{
    int i,c ;
    char var_file_name[20];
    strcpy(var_file_name,taskname);
    strcat(var_file_name,"_v.dat");
    if ((fp1 = fopen(var_file_name,"r")) != NULL)
    {fscanf(fp1, "%d%d%d%P%P%d%P", &ninput,&noutattr,&ninattr,&eta,
        &alpha,&nhlayer,&cnt_num);
        for ( i=0 ; i < nhlayer+2 ; i++)
            fscanf(fp1, "%d", &nunit[i]);
    }
    if (( c = fclose(fp1)) != 0 )
        fcloseall();
    return(c);
}

/* lectura de archivo de pesos suffix_w.dat */

```

```

wthead(taskname)
char *taskname;
{
    int i,j,c ;
    char wt_file_name[20];
    strcpy(wt_file_name,taskname);
    strcat(wt_file_name,"_w.dat");
    if (( fp2 = fopen(wt_file_name,"r")) != NULL)
    { for ( i=0 ; i < nhlayer + 1 ; i++)
        {
            for ( j=0 ; j < (nunit [i] + 1) * nunit[i+1] ; j++)
            {
                fscanf(fp2, "%f", (wtptr[i]+j));
            }
        }
    }
    if (( c = fclose(fp2)) != 0 )
        fcloseall();
    return(c);
}

```

```
/* Crea archivo de arquitectura y valores de aprendizaje suffix_v.dat */
```

```
dwrite(name)
char *name;
{
    int i,j,c;
    char file_name[20];
    strcpy(file_name,name);
    strcat(file_name,"_v.dat");
    if((fp1 = fopen(file_name,"w+")) == NULL)
        fprintf("\n No puede abrirse el archivo de datos. ");
    else{
        fprintf(fp1," %u %u %u %f%f%u %u\n",ninput,noutattr,
                ninattr,eta,alpha,
                nhlayer,cnt_num);

        for (i=0 ; i < nhlayer+2 ; i++)
            {
                fprintf(fp1,"%d ",nunit[i]);
            }
        fprintf(fp1,"\n%d %f", cnt,err_curr);
        fprintf(fp1,"\n");
        for ( i=0 ; i < ninput ; i++)
            {
                for ( j=0 ; j<noutattr ; j++)
                    fprintf(fp1,"%f ",outpt[i][j]);
                fprintf(fp1,"\n");
            }
        }
    if(( c = fclose(fp1)) != 0 )
        fprintf("\n No puede cerrarse el archivo");
return(c);
}
```

```
/* Crea archivo de pesos,entradas aprendidas suffix_w.dat */
```

```
wtwrite(taskname)
char *taskname;
{
    int i,j,c,k;
    char wt_file[20];
    strcpy(wt_file,taskname);
    strcat(wt_file,"_w.dat");
    if(( fp2 = fopen(wt_file,"w")) == NULL)
        fprintf("\n No se puede abrir el archivo de datos. ");
    else
```

```

{
k=0;
for (i=0; i<nhlayer+1; i++)
    for (j=0; j < (nunit[i] + 1) * nunit[i+1]; j++)
    {
        if ( k==8 )
        {
            k = 0;
            fprintf(fp2,"n");
        }
        fprintf(fp2,"%f",*(wptr[i] + j));
        k++;
    }
}
if ((c = fclose(fp2)) !=0 )
    fprintf("n No puede cerrarse el archivo %d ",c);
fcloseall();
}

/* Impresion de archivos */
dfprint(taskname)
char *taskname;
{
    int i,j,c,k;
    char var_file_name[20];
    char wt_file_name[20];
    fcloseall();
    strcpy(var_file_name,taskname);
    strcat(var_file_name,"_v.dat");
    if (( fp1 = fopen(var_file_name,"r") == NULL)
        fprintf("n No puede abrirse el archivo de datos. ");
    else{
        fprintf(stdprn, "n\nr Valores de aprendizaje\nr");
        fprintf(stdprn, " Ejemplos Salidas Entradas Eta Alpha Capas Ocultas Iteraciones\nr");
        fprintf(stdprn, " %u %u %u %f%f %u %u\nr ",
                    ninput,noutattr,
                    ninattr,eta,alpha,
                    nhlayer,cnt_num);

        fprintf(stdprn,"n\nr Arquitectura\nr ");
        for (i=0; i < nhlayer+2; i++)
        {
            fprintf(stdprn,"%d ",nunit[i]);
        }
        fprintf(stdprn,"n\nr");
    }
}

```



```

fprintf(stdprn, "\nIteraciones Alcanzadas Error Obtenido\n\n");
fprintf(stdprn, "\n %d %f\n\n", cnt, err_curr);
fprintf(stdprn, "\n\n");
fprintf(stdprn, "Resultados Obtenidos\n\n");
for ( i=0 ; i < ninput ; i++ )
{
    for ( j=0 ; j < noutattr ; j++ )
        fprintf(stdprn, "%f ", outpt[i][j]);
        fprintf(stdprn, "\n\n");
}
}
if (( c = fclose(fp1) ) != 0 )
    cprintf("\n No puede cerrarse el archivo");
strcpy(wt_file_name, taskname);
strcat(wt_file_name, "_w.dat");
if (( fp2 = fopen(wt_file_name, "r") ) == NULL)
    cprintf("\n No puede abrirse el archivo de datos. " );
else{
    fprintf(stdprn, "\n\n");
    fprintf(stdprn, "Pesos Entradas Aprendidas\n\n");
    k=0;
    for ( i=0 ; i < nlayer+1 ; i++ )
        for ( j=0 ; j < (nunit[i] + 1) * nunit[i+1] ; j++ )
        {
            if ( k=8 )
            {
                k = 0;
                fprintf(stdprn, "\n\n");
            }
            fprintf(stdprn, "%f ", *(wtptr[i] + j) );
            k++;
        }
        fprintf(stdprn, "\n\n");
    }
if ((c = fclose(fp2) ) != 0 )
    cprintf("\n No puede cerrarse el archivo %d ", c);
fcloseall();
activa_v(2);
return(c);
}

```

```
/* Calcula la base de la red para la entrada del ejemplo i */
```

```
void forward(i)
```

```
{  
  int m,n,p,offset ;  
  float net ;
```

```
/* Nivel de entrada para el calculo de salida */
```

```
for ( m=0 ; m < ninattr ; m++)  
  *(outptr[0]+m) = input[i][m];
```

```
/* Capa oculta y de salida para calcular la salida */
```

```
for ( m=1 ; m < nhlayer+2 ; m++)  
{  
  for ( n=0 ; n < nunit[m] ; n++)  
  {  
    net = 0.0;  
    for ( p=0; p<nunit[m-1]+1 ; p++)  
    {  
      offset = ( nunit[m-1]+1 ) * n + p ;  
      net += *(wptr[m-1]+offset)*(*(outptr[m-1]+p));  
    }  
    fprintf(fp4,"%f\n",net);  
    *(outptr[m]+n) = 1 / (1+exp(-net));  
  }  
  for ( n=0; n<nunit[nhlayer+1];n++)  
    outpl[i][n] = *(outptr[nhlayer+1]+n);  
}
```

```
/* Checar condiciones de fin de aprendizaje */
```

```
int introspective (nfrom,nto)  
int nfrom;  
int nto;
```

```
{  
  int i,flag;
```

```
/* maxima iteracion? */
```

```
if (cnt > cnt_num) return (FEXIT);
```

```
/* Error suficientemente pequeno para cada ejemplo? */
```

```
nsnew = 0;
flag = 1;
for ( i = nfrom ; i < nto ) && ( flag == 1 ) ; i++)
{
    if ( ep[i] <= maxep ) nsnew++;
    else flag = 0 ;
}
if ( flag == 1 ) return (SEXIT);
```

```
/* Error suficientemente pequeno para el sistema total? */
```

```
if ( err_curr <= maxe ) return (SEXIT);
return(CONTNE);
}
```

```
/* Se tratan las entradas como pesos de enlace de nodos virtuales */
```

```
/* Salidas valores unitarios */
```

```
int rumelhart(from_snum,to_snum)
int from_snum;
int to_snum;
```

```
{
    int i,j,k,m,n,p,offset,index;
    float out;
    char *err_file = "criter.dat";
    char *prn_file = "values.dat";
    nsold = 0;
    cnt = 0;
    result = CONTNE;
    fcloseall();
    if ((fp4=fopen(prn_file,"w+")) == NULL)
        fprintf("No puedo abrir el archivo de error");
    if (fp1ot10)
        if ((fp3=fopen(err_file,"w+")) == NULL)
            fprintf("No puedo abrir el archivo de error");
    do
    {
        err_curr = 0.0 ;
```

```

/* Para cada ejemplo */
for ( i=from_snum; i < to_snum ; i++ )
{
/* Calculo de abajo hacia arriba */

forward(i);

/* Propagacion del error hacia abajo. Nivel de salida del error */

for ( m=0 ; m<nunit[nhlayer+1] ; m++ )
{
out = *(outptr[nhlayer+1] + m );
*(errptr[nhlayer+1]+m) = (target [i][m] -out) * (1-out) * out;
}

/* Errores de la capa oculta y de entrada */

for ( m=nhlayer+1 ; m>=1 ; m-- )
{
for ( n=0 ; n < nunit[m-1]+1 ; n++ )
{
*(errptr[m-1]+n) = 0.0 ;
for ( p=0 ; p<nunit[m] ; p++ )
{
offset = ( nunit[m-1]+1) * p + n ;
*(delw[m-1]+offset) = eta * ( *(errptr[m]+p)
*( *(outptr[m-1]+n)
+ alpha * *(delw[m-1]+offset));
*(errptr[m-1]+n) += *(errptr[m]+p) * *(wtptr[m-1]+offset));
}
*(errptr[m-1]+n) = *(errptr[m-1]+n) * ( 1- *(outptr[m-1]+n)
*( *(outptr[m-1]+n)));
}
}

/* Cambio de pesos */

for ( m=1 ; m < nhlayer + 2 ; m++ )
{
for ( n = 0 ; n < nunit[m] ; n++ )
{
for ( p = 0 ; p < nunit[m-1]+1 ; p++ )
{

```

```

offset = ( nunit[m-1]+1 ) * n + p;
*(wtptr[m-1]+offset) += *(delw[m-1]+offset);
if ( m == 1 )
{
    if ( p == 3 )
    {
        gotoxy(24,2+n);
        printf("%f",*(wtptr[m-1]+offset));
    }
    else
    {
        gotoxy(7,2+(3*n)+p);
        printf("%f",*(wtptr[m-1]+offset));
    }
}
if ( m == 2 )
{
    if ( p == 3 )
    {
        gotoxy(24,2+p);
        printf("%f",*(wtptr[m-1]+offset));
    }
    else
    {
        gotoxy(7,1+offset);
        printf("%f",*(wtptr[m-1]+offset));
    }
}
}
}
ep[i] = 0.0;
for (m=0; m < nunit[nhlayer+1]; m++)
{
    ep[i] += fabs((target[i][m] - *(outptr[nhlayer+1]+m)));
    gotoxy(41,2+i);
    printf("%f", ep[i]);
}
err_curr += ep[i] * ep[i];
}

```

```

/* Normalizacion del error del sistema */
err_curr = 0.5 * err_curr / ninput ;
gotoxy(53,1);
printf("%f",err_curr);
/* Grabar errores para dibujar el error del sistema */

if (fplot10==1)
fprintf(fp3,"%ld, %2.9f\n",cnt,err_curr);
cnt++;
gotoxy(39,1);
printf("%d",cnt);

/* Checa condicion de fin de aprendizaje */

result = introspective(from_snum,to_snum);
}

while (result==CONTNE);

/* Actualizar salidas con pesos cambiados */

fcloseall();
tecla();
clrscr();
for ( i=from_snum ; i < to_snum ; i++)
forward(i);
for ( i=0 ; i < nhlayers+1 ; i++)
{
index = 0;
for ( j=0 ; j < nunit[i+1] ; j++)
{
printf("\n\r Peso entre unidad %d de capa %d" j,i+1);
printf(" y unidades de capa %d\n\r",i);
for (k=0;k<nunit[i];k++)
{
printf(" %f",*(wtptr[i] + index++));
}
printf("\n\r La unidad de entrada %d de la capa %d es %f",
j,i+1,*(wtptr[i] + index++));
}
}
tecla();
clrscr();
printf(" Salidas de la Red\n\r");

```

```

for (i=0 ; i<ninput ; i++)
    for (j=0 ; j<noutattr; j++)
    {
        printf("Prueba %d Salida %d = %f objetivo %d = %f\n\r",
            i,j,output[i][j],j,target[i][j]);
    }
gotoxy(10,18);
printf(" Se ejecutaron %d iteraciones.\n\r",cnt);
gotoxy(10,20);
printf("El error del sistema normalizado es de %f\n\r",err_curt);
return(result);
}

/* Sesion interactiva con el usuario */

user_session()
{
    int i,j,k,showdata;
    char fnam[20] ,dtype[20];

    printf(" Inicio de Sesion de Aprendizaje\n\r ");

    /* Lee tarea task_name.dat */
    gotoxy(4,3);
    strcpy(task_name,"");
    printf(" Nombre de la tarea: ");
    lee(40,11,4,task_name,3);
    gotoxy(4,5);
    printf(" Numero de unidades de entrada: ");
    lee(50,13,3,buf,2);
    ninattr=atoi(buf);
    gotoxy(4,7);
    strcpy(buf," ");
    printf(" Numero de unidades de salida: ");
    lee(50,15,3,buf,2);
    noutattr=atoi(buf);
    gotoxy(4,9);
    strcpy(buf," ");
    printf(" Numero de ejemplos de entrada: ");
    lee(50,17,3,buf,2);
    ninput=atoi(buf);
    strcpy(fnam,task_name);
    strcat(fnam,".dat");
    gotoxy(4,11);
    printf(" El archivo es %s",fnam);
}

```

```

if ((fp1 = fopen(fnam,"r"))==NULL)
    cprintf("\n\rNo existe el archivo %s.",fnam);
else
{
    gotoxy(4,13);
    strcpy(dtype,"Si");
    cprintf(" Desea ver los datos [ Si / No ]: ");
    lee(54,21,3,dtype,1);
    showdata = ((dtype[0] == 's' || dtype[0] == 'S'));
    if(showdata){
        clrscr();
        cprintf(" Entrada      Caracteristicas      Salida");
        }
    for (i=0; i<ninput;i++){
        for (j=0; j<ninattr;j++){
            fscanf(fp1,"%f",&input[i][j]);
            gotoxy(5,2+i);
            if(showdata) printf("%d",i);
            gotoxy(10+(j*10),2+i);
            if(showdata)printf("%f",input[i][j]);
        }
        for (j=0; j<noutattr;j++){
            fscanf(fp1,"%f",&target[i][j]);
            if (showdata)
            {
                gotoxy(45,2+i);
                printf("%f",target[i][j]);
            }
        }
    }
    if ((k=fclose(fp1)) != 0)
        cprintf("\n\rNo puedo cerrar el archivo",k);
}
fcloseall();
if(showdata)tecla();
return k;
}

/* Funcion principal de aprendizaje*/

learning()
{
    int result;

    result=user_session();
    clrscr();
}

```



```

if(result!=-1)
{ clrscr();
  set_up();
  init();
  do
  { initwt();
    result = runelchar(0,ninput);
  }
  while (result == RESTRT);
if ( result == FEXIT )
{
  cprintf("Se alcanzo el numero maximo de iteraciones.\n\r");
  cprintf("Pero fallo al disminuir el error del sistema.");
  fcloseall();
  tecla();
}
if (result == EXIT) tecla();
fclose(fp1);
fcloseall();
dwrite(task_name);
wtwrite(task_name);
pon(18,4,30,4,GREEN,YELLOW,"Aprendizaje");
activa_v(2);
}
else
{ pon(18,4,30,4,GREEN,YELLOW,"Aprendizaje");
  activa_v(2);
}
fcloseall();
}

```

/* Funcion generadora de salidas. */

```
output_generation()
```

```

{
  int i,m,nsample,result;
  char ans[10];
  char dfile[20];

```

/* Leer archivo de datos si no existe */

```

cprintf(" Generacion de Salidas" );
gotoxy(4,3);
cprintf("La tarea actual es: %s\n\r",task_name);
gotoxy(4,5);

```

```

strcpy(ans,"No ");
cprintf(" Trabajar en una Tarea diferente [Si/No]: ");
lee(62,13,2,ans,1);
strcpy(buf," ");
if ((ans[0] == 's') || (ans[0] == 'S'))
{
gotoxy(4,7);
result=0;
cprintf("Teclea el nombre de la tarea: ");
lee(60,15,6,buf,3);
strcat(buf,".dat");
result=dread(buf);
if(result==-1)
{ gotoxy(6,9);
cprintf("No puede abrirse el archivo");}
}
else
{
init();
wread(buf);

```

/* Creacion de los datos de entrada */

```

gotoxy(4,9);
strcpy(dfile," ");
cprintf("Nombre del archivo de proceso:");
lee(52,17,6,dfile,3);
strcat(dfile,".dat");
gotoxy(4,11);
cprintf("Archivo de proceso: %s",dfile);
if ((fp1=fopen(dfile,"r")) == NULL)
{ gotoxy(4,12);
cprintf("No puede abrirse el archivo."); }
else{
gotoxy(4,11);
strcpy(buf," ");
cprintf("Numero de ejemplos para proceso: ");
lee(52,19,3,buf,2);
nsample=atoi(buf);
tecla();
clrscr();
cprintf(" Salidas de la Red");
for (i=0 ; i<nsample ; i++)
for (m=0 ; m<ninattr ; m++)
fscanf(fp1,"%f",&input[i][m]);

```

```
/* Calculo de generacion de salidas */
```

```
for (i=0 ; i<nsample ; i++)
```

```
{  
    forward(i);
```

```
    for (m=0 ; m<noutattr; m++)
```

```
        {  
            fprintf(stdout, "Prueba %d Salida %d = %f", i, m, *(outptr[nhlayer+1]+m));
```

```
        }  
    fprintf(stdout, "\n\nLas salidas se han generado.");
```

```
}
```

```
if ((i=fclose(fp1)) != 0)
```

```
{ gotoxy(4,13);
```

```
    fprintf(stdout, "No puede cerrarse el archivo");}
```

```
fcloseall();
```

```
tecla();
```

```
pon(34,4,46,4,BLUE,LIGHTGRAY,"Resultados");
```

```
activa_v(2);
```

```
/* Programa principal */
```

```
main()
```

```
{ char key="";
```

```
saludo();
```

```
portada();
```

```
while(key!=0x2d){
```

```
    key=getch();
```

```
    switch(key){
```

```
        case 0x1e: { pon(18,4,30,4,GREEN,YELLOW+BLINK,"Aprendizaje");
```

```
                    activa_v(2);
```

```
                    learning();
```

```
                    /* Alt A */
```

```
                    break;}
```

```
        case 0x13: { pon(34,4,46,4,BLUE,LIGHTGRAY+BLINK,"Resultados");
```

```
                    activa_v(2);
```

```
                    output_generation();
```

```

/* Alt R */
break;
case 0x17: { pon(48,4,60,4,WHITE,BLACK+BLINK,"Impresiones");
activa_v(2);
cprintf("\n\nImprimiendo resultados:");
dprintf(task_name);
tecla();
/* Alt I */
pon(48,4,60,4,WHITE,BLACK,"Impresiones");
break;}
case 0x22: { pon(64,4,76,4,MAGENTA,WHITE+BLINK,"Graficas");
activa_v(2);
window(1,1,80,25);
system("grafica");
cierra_v(2);
cierra_v(0);
window(1,1,80,25);
textbackground(WHITE);
clrscr();
portada();
tecla();
/* Alt G */
pon(64,4,76,4,MAGENTA,WHITE,"Graficas");
activa_v(2);
break;}
default: {}
cprintf("\n\nRegreso el control a DOS. \n");
delay(500);
cierra_v(2);
delay(1250);
cierra_v(1);
delay(1250);
cierra_v(0);
delay(1250);
window(1,1,80,25);
textbackground(BLACK);
textcolor(WHITE);
clrscr();
fcloseall();
}

```

Fin de Archivo

Código fuente obtenido al implementar el Análisis Fractal 3-D.

```
/*.....*/  
/* Analisis fractal */  
/* Autor: Raul Antonio Aguilar Castillejos */  
/*.....*/
```

```
#include <graphics.h>  
#include <bios.h>  
#include "math.h"  
#include "stdio.h"
```

```
#define NOMBRE "values.dat"  
#define MAX 33  
#define MAXLEVEL 5  
#define MAX_NUM 8584  
#define PI 3.1415926535897932385  
#define TRUE 0  
#define FALSE -1
```

```
/* Variables de estudio */
```

```
float H,ETA,ALFA,eta;  
float x[MAX][MAX];  
float numeros[MAX_NUM];
```

```
/* Variables para el graficador */
```

```
int graphdriver,graphmode; /* Variables del sistema Grafico */  
int L,DIST; /* Valores de visualización */  
int xx,yy,zz; /* Coordenadas del eje X */  
int xy,yy,zy; /* Coordenadas del eje Y */  
int xz,yz,zz; /* Coordenadas del eje Z */  
int xs,ys; /* Coordenadas para impresión */  
int CX,CY; /* Coordenadas iniciales del centro */  
int scrdist; /* Distancia del visor a la pantalla */  
int x_x,y_x; /* Extremo del eje X */  
int x_y,y_y; /* Extremo del eje Y */  
int x_z,y_z; /* Extremo del eje Z */  
int addition; /* Para algoritmo completo */  
int num; /* Contador vector de números */  
float theta,phi,rho; /* Coordenadas esféricas */
```

```

/* Variables globales para calculo */

float sintheta, costheta;
float sinphi, cosphi;
float va, vb, vc, vf, vg, vi, vj, vk, vl; /* Variables globales para calculo */
float cual; /* Indicador de Tecla */
FILE *arch,*fopen(); /* Apuntador a archivo */

/* Inclusion de rutinas de graficos */

#include "graf.h"

float learn()
{
float res;
res = 1/(1+exp(-numeros[num]));
num++;
return (res);
}

float f3 (float eta,float x0,float x1,float x2)
{
float res;

res = (x0+x1+x2)/3+eta*learn();
return(res);
}

float f4 (float eta,float x0,float x1,float x2,float x3)
{
float res;

res = (x0+x1+x2+x3)/4+eta*learn();
return(res);
}

void MidPointFM2D()
{
int N,D,d,stage,X,y;

num = 0;
N = (int)pow((float)2.0,(float)MAXLEVEL);
eta=ETA;
x[0][0]=eta*learn();
x[0][N]=eta*learn();
x[N][0]=eta*learn();
}

```

```

x[N][N]=eta*learn();
D=N;
d=N/2;

for (stage=1;stage<=MAXLEVEL;stage++)
{
    eta=eta*pow(0.5,0.5*H);
    for (X=d;X<=N-d;X=X+D)
        for (y=d;y<=N-d;y=y+D)
            x[X][y]=f4(eta,x[X+d][y+d],x[X+d][y-d],x[X-d][y+d],x[X-d][y-d]);
    if (addition == TRUE)
        for (X=0;X<=N;X=X+D)
            for (y=0;y<=N;y=y+D)
                x[X][y]=x[X][y]+eta*learn();

    eta=eta*pow(0.5,0.5*H);
    for (X=d;X<=N-d;X=X+D)
    {
        x[X][0]=f3(eta,x[X+d][0],x[X-d][0],x[X][d]);
        x[X][N]=f3(eta,x[X+d][N],x[X-d][N],x[X][N-d]);
        x[0][X]=f3(eta,x[0][X+d],x[0][X-d],x[d][X]);
        x[N][X]=f3(eta,x[N][X+d],x[N][X-d],x[N-d][X]);
    }

    for (X=d;X<=N-d;X=X+D)
        for (y=D;y<=N-d;y+=D)
            x[X][y]=f4(eta,x[X][y+d],x[X][y-d],x[X+d][y],x[X-d][y]);

    for (X=D;X<=N-d;X=X+D)
        for (y=d;y<=N-d;y=y+D)
            x[X][y]=f4(eta,x[X][y+d],x[X][y-d],x[X+d][y],x[X-d][y]);

    if (addition == TRUE)
    {
        for (X=0;X<=N;X=X+D)
            for (y=0;y<=N;y=y+D)
                x[X][y]=x[X][y]+eta*learn();
        for (X=d;X<=N-d;X=X+D)
            for (y=d;y<=N-d;y=y+D)
                x[X][y]=x[X][y]+eta*learn();
    }
    D=D/2;
    d=d/2;
}
}

```

```

/* Se calculan los puntos de la grafica segun los par metros */
calcula ()
{
    setcolor(WHITE);outtextxy(2,190,"Preparando nuevos datos ...");
    MidPointFM2D();
    setcolor(BLACK);outtextxy(2,190,"Preparando nuevos datos ...");
    setcolor(WHITE);outtextxy(2,190,"Cambiando las escalas ...");
    scale();
    setcolor(BLACK);outtextxy(2,190,"Cambiando las escalas ...");
}

```

```

suma (int alfa)
{int i;

for (i=0;i<8584;i++)
    numeros[i]=numeros[i]+alfa;
}

```

```

flechas_u_d (float *cual)
{int tecla=0;

```

```

while (tecla!=20224)
{
    pon_tit(WHITE);
    tecla = bioskey(0);
    pon_tit(BLACK);
    switch(tecla)
    {
        case 18432 : /* Flecha hacia arriba */
            *cual = *cual + 0.01;
            break;
        case 20480 : /* Flecha hacia abajo */
            *cual = *cual - 0.01;
            break;
    }
    pon_tit(WHITE);
}
}

```



```

/* Giros en el espacio y zooms */
custom_angles ()
{
  int tecla=0;
  setcolor(WHITE); outtextxy(2,190,"Giros y acercamientos");
  while (tecla!=20224)
  {
    init_transf();
    ejes();bases(CYAN);pon_tit(WHITE);
    tecla = bioskey(0);
    b_ejes();bases(BLACK);pon_tit(BLACK);
    switch(tecla)
    {
      case 18432 : /* Flecha hacia arriba */
        phi = phi + PI/180;
        break;
      case 20480 : /* Flecha hacia abajo */
        phi = phi - PI/180;
        break;
      case 19712 : /* Flecha hacia la derecha */
        theta = theta + PI/180;
        break;
      case 19200 : /* Flecha hacia la izquierda */
        theta = theta - PI/180;
        break;
      case 18688 : /* Page up */
        scrdist = scrdist + 10;
        break;
      case 20736 : /* Page Dwn */
        scrdist = scrdist - 10;
        break;
      case 11875 : /* Mover el centro "c"*/
        while (tecla != 20224)
        {
          init_transf();
          ejes();bases(CYAN);pon_tit(WHITE);
          tecla = bioskey(0);
          b_ejes();bases(BLACK);pon_tit(BLACK);
          switch (tecla)
          {
            case 18432 : if (CY > 10)          if (CY>0)
                          CY = CY - 10;
                          break;
            case 20480 : if (CY < 470)      if (CY>0)
                          CY = CY + 10;
          }
        }
      }
    }
  }
}

```

```

        break;
    case 19712 : if (CX < 630)
        CX = CX + 10;
        break;
    case 19200 : if (CX > 10)
        CX = CX - 10;
        break;
    }
}
tecla = 0;
break;
}
}
ejes();bases(CYAN);pon_tit(WHITE);
setcolor(BLACK); outtextxy(2,190,"Giros y acercamientos");
}

```

param ()

/* Opciones para cambios en los parametros de la función fractal h, alfa, eta. */

```

{
    int tecla,alfa;
    tecla = 0;
    ejes();bases(CYAN);pon_tit(WHITE);
    setcolor(WHITE); outtextxy(2,190,"Cambios a parametros de la funcion");
    tecla = bioskey(0);
    switch(tecla)
    {
        case 8051 : /* (s) ETA */
            do
                flechas_u_d(&ETA);
            while (!(ETA > 0.00));
            break;
        case 12909 : /* (m) ALFA */
            alfa = ALFA;
            flechas_u_d(&ALFA);
            suma (ALFA-alfa);
            break;
        case 9064 : /* (h) H */
            do
                flechas_u_d(&H);
            while (!(H > 0.00));
            break;
        case 8550 : /* (f) Addition */
            while (tecla != 20224)
            {

```

```

pon_tit(BLACK);
if (addition == TRUE)
    addition = FALSE;
else
    addition = TRUE;
pon_tit(WHITE);
tecla = bioskey(0);
}
tecla = 0;
break;
}
setcolor(BLACK); outtextxy(2,190,"Cambios a parametros de la funcion");
}

despliega ()
{
    int tecla=0;

    cleardevice();
    init_transf();
    ejes();bases(CYAN);pon_tit(WHITE);
    curva(LIGHTMAGENTA);
    while (tecla != 20224)
        tecla = bioskey(0);
}

grapher ()
{
    int tecla=0;

    inicializa();
    while (tecla != 283) /* Salida con ESC */
    {
        init_transf();
        ejes();bases(CYAN);pon_tit(WHITE);
        tecla = bioskey(0);
        switch (tecla)
        {
            case 4978 : /* (R) Rotaciones, zoom */
                custom_angles ();
                break;
            case 6512 : /* (P) Parametros */
                param();
                break;
            case 8292 : /* (D) Despliega */

```

```

        despliega();
        break;
    case 12654: /* (N) Se genera nuevos numeros base*/
        setcolor(WHITE); outtextxy(2,190,"Generando numeros base ...");
        genera();
        setcolor(BLACK); outtextxy(2,190,"Generando numeros base ...");
        break;
    case 11875: /* (C) Para calculo de nuevos puntos en la grafica */
        calcula();
        break;
    }
}
closegraph();
}

genera ()
{int i;
char *file = "values.dat";
if((arch=fopen(file,"r")) == NULL)
    cprintf("No puedo abrir el archivo de datos");
ALFA=0;
for (i=0;i<MAX_NUM;i++){
    fscanf(arch,"%f\n",&numeros[i]);
    ALFA = ALFA + numeros[i];
}
ALFA=(ALFA/MAX_NUM)*-10;
ETA=(ALFA*ALFA)*MAXLEVEL;
H=ETA/ALFA;
}

main()
{
    num = 0;
    addition = FALSE;
    genera();
    grapher();
    fcloseall();
}

```

Fin de Archivo

Rutinas Extra - Librerías (Headers)

```
/*.....  
Estas archivo contiene las rutinas necesarias para la preparación y creación de  
ventanas.  
*.....*/
```

```
struct ventana
```

```
{  
char x1, y1, x2, y2, color_f, sombra, color_m, color_t;  
char ach, lrg, tx1, tx2, ty1, ty2, guarda, color_te;  
char tipo_m, pos_t, titulo[78], forma_d;  
void *area_d;  
} vent[NUMV];  
dib_margen( dir )  
struct ventana *dir;  
{  
register int i;  
char h, v, t_der, t_izq, esq_sd, esq_si, esq_ji, esq_id, div_t;  
char signo[2];
```

```
*(signo+1) = dir->color_m + 16*dir->color_f;
```

```
switch( dir->tipo_m )
```

```
{  
case 'A':  
h = 196;  
v = 179;  
div_t = 196;  
t_der = 180;  
t_izq = 195;  
esq_sd = 191;  
esq_ji = 192;  
esq_id = 217;  
esq_si = 218;  
break;  
case 'B':  
h = 205;  
v = 186;  
div_t = 205;  
t_der = 185;  
t_izq = 204;  
esq_sd = 187;  
esq_ji = 200;  
esq_id = 188;
```

```

        esq_si = 201;
        break;
    case 'C':
        h = 196;
        v = 179;
        div_t = 32;
        t_der = 179;
        t_izq = 179;
        esq_sd = 191;
        esq_ii = 192;
        esq_id = 217;
        esq_si = 218;
        break;
    case 'D':
        h = 205;
        v = 186;
        div_t = 32;
        t_der = 186;
        t_izq = 186;
        esq_sd = 187;
        esq_ii = 200;
        esq_id = 188;
        esq_si = 201;
    }
    signo[0] = div_t;
    for ( i=dir->x1+1; i<dir->x2; i++) puttext( i, dir->y1+2, i, dir->y1+2, &signo[0] );
    signo[0] = h;
    for ( i=dir->x1+1; i<dir->x2; i++)
    {
        puttext( i, dir->y1, i, dir->y1, &signo[0] );
        puttext( i, dir->y2, i, dir->y2, &signo[0] );
    }
    signo[0] = v;
    for ( i=dir->y1+1; i<dir->y2; i++)
    {
        puttext( dir->x1, i, dir->x1, i, &signo[0] );
        puttext( dir->x2, i, dir->x2, i, &signo[0] );
    }
    signo[0] = esq_si; puttext( dir->x1, dir->y1, dir->x1, dir->y1, &signo[0] );
    signo[0] = esq_sd; puttext( dir->x2, dir->y1, dir->x2, dir->y1, &signo[0] );
    signo[0] = t_der; puttext( dir->x2, dir->y1+2, dir->x2, dir->y1+2, &signo[0] );
    signo[0] = t_izq; puttext( dir->x1, dir->y1+2, dir->x1, dir->y1+2, &signo[0] );
    signo[0] = esq_ii; puttext( dir->x1, dir->y2, dir->x1, dir->y2, &signo[0] );
    signo[0] = esq_id; puttext( dir->x2, dir->y2, dir->x2, dir->y2, &signo[0] );
}

```

```

esc_titulo ( dir )
struct ventana *dir;
{
    int ld, prov, cdy;

    ld = strlen ( dir->titulo );
    prov = dir->ach - ld;

    if ( dir->tipo_m == 'A' || dir->tipo_m == 'B' )
        cdy = 2;
    if ( dir->tipo_m == 'C' || dir->tipo_m == 'D' )
        cdy = 1;
    textcolor ( dir->color_t );
    switch (dir->pos_t)
    {
        case 'I':
            gotoxy ( 2, cdy );
            cputs ( dir->titulo );
            break;

        case 'D':
            gotoxy ( prov, cdy );
            cputs ( dir->titulo );
            break;

        case 'C':

            if ( cdy == 2 )
            {
                if ( prov%2 == 0 ) gotoxy ( prov / 2 + 1, cdy );
                else gotoxy ( prov / 2 + 2, cdy );
                cputs ( dir->titulo );
            }

            else
            {
                if ( prov%2 == 0 ) gotoxy ( prov / 2 + 0, cdy );
                else gotoxy ( prov / 2 + 1, cdy );
                cputs ( " " );
                cputs ( dir->titulo );
                cputs ( " " );
            }
    }
}

```

```

pon_sombra ( dir )
struct ventana *dir;
{
    int i,j;
    char simbolo[2];

    for (i=dir->y1+1; i<=dir->y2+1; i++)
        for (j=1; j<3; j++)
            {
                gettext( dir->x2+j, i, dir->x2+j, i, &simbolo[0]);
                *(simbolo+1)=LIGHTGRAY+16*BLACK;
                puttext( dir->x2+j, i, dir->x2+j, i, &simbolo[0]);
            }
    for (i=dir->x1+2; i<=dir->x2+1; i++)
        {
            gettext( i, dir->y2+1, i, dir->y2+1, &simbolo[0]);
            *(simbolo+1) = LIGHTGRAY + 16*BLACK;
            puttext( i, dir->y2+1, i, dir->y2+1, &simbolo[0]);
        }
}

crear_v(nv)
int nv;
{
    int i, a_b, pm, sx=0, sy=0;
    struct ventana *dir;
    dir = &vent[nv];

    if( dir->sombra == 1 )
        {
            sx = 2;
            sy = 1;
        }
    if((dir->area_d = malloc((dir->ach+sx)*(dir->lrg+sy)*2)) == NULL )
        {
            printf("Insuficiente memoria...\n");
            exit(1);
        }
    gettext( dir->x1, dir->y1, dir->x2+sx, dir->y2+sy, dir->area_d );

    switch( dir->forma_d )
    {
        case 'L':
            if ( (dir->x2 + dir->x1) % 2 == 0 ) a_b = 0;
            else a_b = 1;
    }
}

```



```

pm = ( dir->x1 + dir->x2 ) / 2;

for ( i = 0; pm-i >= dir->x1; i++ )
{
    delay(25);
    window( pm-i, dir->y1, pm+a_b+i, dir->y2 );
    textbackground( dir->color_f);
    clrscr();
}
break;

case 'l':
    window( dir->x1, dir->y1, dir->x2, dir->y2 );
    textbackground( dir->color_f);
    clrscr();
    break;

case 'P':
    for ( i=0; dir->y1+i <= dir->y2; i++ )
    {
        delay(45);
        window( dir->x1, dir->y1, dir->x2, dir->y1 + i );
        textbackground( dir->color_f);
        clrscr();
    }
}

dib_margen ( dir );
esc_titulo ( dir );
if( dir->sombra == 1 ) pon_sombra ( dir );
}

cierra_v(nv)
int nv;
{
    int sx=0, sy=0;
    struct ventana *dir;
    dir = &vent[nv];

if ( dir->sombra == 1 )
{
    sx=2;
    sy=1;
}
    puttext( dir->x1, dir->y1, dir->x2+sx, dir->y2+sy, dir->area_d );
    free(dir->area_d);
}

```

```

area_v( dir )
struct ventana *dir;
{
    dir->ach=dir->x2-dir->x1+1;
    dir->lrg=dir->y2-dir->y1+1;
}
area_t( dir )
struct ventana *dir;
{
    dir->tx1 = dir->x1+1;
    dir->ty1 = dir->y1+1;
    dir->tx2 = dir->x2-1;
    dir->ty2 = dir->y2-1;

    if ( (dir->tipo_m == 'A') || (dir->tipo_m == 'B') )
        dir->ty1 = dir->y1+3;
}
activa_v(nv)
int nv;
{
    struct ventana *dir;
    dir = &vent[nv];

    window( dir->tx1, dir->ty1, dir->tx2,dir->ty2 );
    textbackground( dir->color_f);
    textcolor( dir->color_te );
    clrscr();
}
iniclzvent(nv, x1, y1, x2, y2, sombra, frm)
char nv, x1, y1, x2, y2, sombra, frm;
{
    vent[nv].x1 = x1;
    vent[nv].x2 = x2;
    vent[nv].y1 = y1;
    vent[nv].y2 = y2;
    vent[nv].forma_d = frm;
    vent[nv].sombra = sombra;
}
caractvent(nv, clrfnd, clrmgn, clrttl, clrtxt, tpmpst, ttl)
char nv,clrfnd, clrmgn, clrttl, *tpmpst, *ttl;
{
    vent[nv].color_f= clrfnd;
    vent[nv].color_m = clrmgn;
    vent[nv].color_t = clrttl;
}

```

```

vent[nv].color_te = clrtxt;
vent[nv].tipo_m = tpmpst[0];
vent[nv].pos_t = tpmpst[1];
strcpy ( vent[nv].titulo, ttl);
area_v(&vent[nv]);
area_t(&vent[nv]);
}
tm_crsr(ini,fin)
char ini,fin;
{
union REGS r;
r.h.ah = 1;
r.h.ch = ini;
r.h.cl = fin;
int86(0x10, &r, &r);
}

```

Fin de Archivo

/* Rutinas Gráficas --- graph.h */

```

inicializa (void)
{
int graphdriver,graphmode;

clrscr();
graphdriver = DETECT;
initgraph(&graphdriver,&graphmode," ");
cleardevice();
if(graphdriver==CGA)
{ CX=320;
CY=65;
L=1024;
DIST=405*5;}
else if(graphdriver==VGA)
{ CX=320;
CY=260;
L=2048;
DIST=410*5; /* 820*5 */
}
/* Inicialización de ngulos */
theta = PI/4; phi = PI/3; rho = 5000;

```

```

/* Inicialización de coordenadas de los ejes */
xx = L; yx = 0; zx = 0;
xy = 0; yy = L; zy = 0;
xz = 0; yz = 0; zz = L;
/* Distancia del usuario a pantalla */
scredist = DIST;
}

```

```

init_transf(void) /* Orientación del Octante */

```

```

{
  sintheta = 1-sin(theta); costheta = 1-cos(theta);
  sinphi = 1-sin(phi); cosphi = cos(phi);
  va = -1*sintheta; vb = costheta;
  ve = -1*costheta*cosphi; vf = -1*sintheta*cosphi;
  vg = sinphi; vi = -1*costheta*sinphi;
  vj = -1*sintheta*sinphi; vk = -1*cosphi;
  vl = rho;
}

```

```

view_transf(float xw,float yw,float zw)

```

```

{
  float xe,ye,ze;

  xe = va*xw+vb*yw;
  ye = ve*xw+vf*yw+vg*zw;
  ze = vi*xw+vj*yw+vk*zw+vl;
  xs = CX-(scredist*xe/ze);
  ys = CY-(scredist*ye/ze);
}

```

```

eje_x (int color)

```

```

{
  setcolor (color);
  line (CX,CY,xs,ys);
  outtextxy(xs+2,ys-2,"X");
  x_x = xs; y_x = ys;
}

```

```

eje_y (int color)

```

```

{
  setcolor (color);
  line (CX,CY,xs,ys);
  outtextxy(xs+2,ys-2,"Y");
  x_y = xs; y_y = ys;
}

```

```

eje_z (int color)
{
    setcolor (color);
    line (CX,CY,xs,ys);
    outtextxy(xs+2,ys-2,"Z");
    x_z = xs; y_z = ys;
}

bases (int color)
{
    int mas,x1,y1,x2,y2,x3,y3,x4,y4;

    if (y_z > CY)
        mas = 50;
    else
        mas = -50;
    setcolor (color);
    /* Base XY */
    view_transf (L,L,0);
    line(x_x,y_x,xs,ys);
    line(x_y,y_y,xs,ys);
    /* Base doble XY*/
    view_transf (0,0,mas); x1 = xs; y1 = ys;
    line(CX,CY,x1,y1);
    view_transf (L,0,mas); x2 = xs; y2 = ys;
    view_transf (L,0,0);
    line(x2,y2,xs,ys); line(x1,y1,x2,y2);
    view_transf (L,L,mas); x3 = xs; y3 = ys;
    view_transf (L,L,0);
    line(xs,ys,x3,y3); line(x2,y2,x3,y3);
    view_transf (0,L,mas); x4 = xs; y4 = ys;
    view_transf (0,L,0);
    line(xs,ys,x4,y4); line(x1,y1,x4,y4);
    line(x3,y3,x4,y4);
    /* Base XZ */
    view_transf (L,0,L);
    line(x_x,y_x,xs,ys);
    line(x_z,y_z,xs,ys);
    /* Base YZ */
    view_transf (0,L,L);
    line(x_y,y_y,xs,ys);
    line(x_z,y_z,xs,ys);
}

```

ejes (void)

```
{
    view_transf(xx,yx,zx); eje_x(RED);
    view_transf(xy,yy,zy); eje_y(GREEN);
    view_transf(xz,yz,zz); eje_z(BLUE);
}
```

b_ejes (void)

```
{
    view_transf(xx,yx,zx); eje_x(BLACK);
    view_transf(xy,yy,zy); eje_y(BLACK);
    view_transf(xz,yz,zz); eje_z(BLACK);
}
```

pon_tit (int color)

```
{
    int g_theta,g_phi;
    char grados[5];
    float num;
```

setcolor(color);

g_theta= (theta*360)/(2*PI);

g_phi = (phi*360)/(2*PI);

outtextxy(2,1,"é : "); outtextxy(2,10,"i : ");

outtextxy(2,20,"L : "); outtextxy(2,30,"X : ");

outtextxy(2,40,"Y : "); outtextxy(2,50,"H : ");

outtextxy(2,60,"á : "); outtextxy(2,70,"æ : ");

outtextxy(2,80,"F : ");

itoa(g_theta,grados,10); outtextxy(30,1,grados);

itoa(g_phi,grados,10); outtextxy(30,10,grados);

itoa(scrdist,grados,10); outtextxy(30,20,grados);

itoa(CX,grados,10); outtextxy(30,30,grados);

itoa(CY,grados,10); outtextxy(30,40,grados);

num = H; gcvt(num,5,grados); outtextxy(30,50,grados);

num = ALFA; gcvt(num,5,grados); outtextxy(30,60,grados);

num = ETA; gcvt(num,5,grados); outtextxy(30,70,grados);

if (addition == TRUE)
 outtextxy(30,80,"C");

else

outtextxy(30,80,"S");

}

```
busca ()
```

```
{  
  int i,j;  
  
  cual = x[0][0];  
  for (i=0;i<MAX;i++)  
    for (j=0;j<MAX;j++)  
      if (x[i][j]>cual)  
        cual = x[i][j];  
}
```

```
scale ()
```

```
{  
  int i,j;  
  
  busca ();  
  if (cual!=0.0)  
    for (i=0;i<MAX;i++)  
      for (j=0;j<MAX;j++)  
        x[i][j] = (x[i][j]*L)/cual;  
}
```

```
curva (int color) /* Curva fractal */
```

```
{  
  int x1,y1,x2,y2,i,j;  
  
  setcolor(color);  
  for (i=0;i<MAX;i++)  
    for (j=0;j<MAX;j++)  
      {  
        x1 = xs; y1 = ys;  
        view_transf ((int)(i*(L/MAX)),(int)(j*(L/MAX)),(int)x[i][j]);  
        x2 = xs; y2 = ys;  
        if (j==0)  
          {  
            x1 = x2;  
            y1 = y2;  
          }  
        line(x1,y1,x2,y2);  
      }  
  for (j=0;j<MAX;j++)  
    for (i=0;i<MAX;i++)  
      {  
        x1 = xs; y1 = ys;  
        view_transf ((int)(i*(L/MAX)),(int)(j*(L/MAX)),(int)x[i][j]);
```

```
x2 = xs; y2 = ys;
if (i==0)
|
|  x1 = x2;
|  y1 = y2;
|
line(x1,y1,x2,y2);
|
```

Fin de Archivo

Ejemplos típicos de Proceso:

A.R.D.G.	Aprendizaje	Resultados	Impresiones	Gráficas
TESIS				

Peso - Valor	Umbral - Valor	Her:16	Ep. Err:0.007365
1 -0.361233	1 0.473047	0.305915	
2 -0.100747	2 -0.206061	0.440050	
3 -0.473344	3 -0.063670	0.427163	
4 0.017568	4 0.195500-0.059361	0.0445100.161749	
5 0.054939		0.307464	
6 -0.269915		0.440396	
7 0.349055		0.426936	
8 -0.478029		0.415100	
9 -0.421933			
10 0.264994			
11 -0.337962			
12 0.442155			

A.R.D.G.	Aprendizaje	Resultados	Impresiones	Gráficas
TESIS				

Salidas de la Red

Prueba 0 Salida 0 = 0.510790 objetivo 0 = 0.900000
 Prueba 1 Salida 0 = 0.510741 objetivo 0 = 0.100000
 Prueba 2 Salida 0 = 0.510735 objetivo 0 = 0.100000
 Prueba 3 Salida 0 = 0.510689 objetivo 0 = 0.900000
 Prueba 4 Salida 0 = 0.510606 objetivo 0 = 0.100000
 Prueba 5 Salida 0 = 0.510757 objetivo 0 = 0.900000
 Prueba 6 Salida 0 = 0.510750 objetivo 0 = 0.900000
 Prueba 7 Salida 0 = 0.510701 objetivo 0 = 0.100000

Se ejecutaron 51 iteraciones.
 El error del sistema normalizado es de 0.006280
 Se alcanzó el número máximo de iteraciones.
 Pero falló al disminuir el error del sistema.

Datos de entrada usados:

0000.9
 0010.1
 0100.1
 0110.9
 1000.1
 1010.9
 1100.9
 1110.1

Paquetes de software usados:

- Word 6.0
- NeuroShell 2.0
- BrainMaker 1.0
- Chi-Writer 4.0
- PaintBrush
- PaintShop
- Windows
- MS-DOS 6.0
- OS/2 2.1

Nota: El proceso y las graficas obtenidas pueden verse consultando revisando una copia de los programas con el autores,

GLOSARIO

Anotación de Concordancia:

Valor que representa la cantidad de características que concuerdan en un patrón presentado a la red y uno aprendido previamente.

Aprendizaje:

Actividad que permite a una red neuronal recordar las características de un patrón específico.

Axón:

Vía de comunicación entre las neuronas, que permite el paso de sus impulsos. Se conoce como **conexión** en las neuronas artificiales.

Capa:

Estructura disjunta en la que se encuentran colocadas las neuronas en un modelo de red neuronal.

Clasificador:

Algoritmo capaz de seleccionar un elemento y clasificarlo en una de varias clases que se usen para realizar la clasificación.

Clasificador Tradicional:

Programa empleado a partir de los 70's para clasificar y/o agrupar elementos en una de varias clases.

Conexionismo:

Estudio de los modelos de representación del cerebro biológico. También conocido como: Neurofisiología, Neurocomputación, ó Sistemas Neuromórficos.

Deadrita:

Contacto mediante los cuales una neurona biológica se interconecta con otras.

Entrada:

Datos que se le proporcionan a una red neuronal para comenzar un proceso de **entrenamiento** o de **reconocimiento**. Esto también se conoce como **datos de entrada**. Estos datos pueden estar dados de forma **binaria** o **continua**, dependiendo del tipo de red, aplicación y propósito.

Entrenamiento:

Proceso mediante el cual se ajustan los **pesos** y **umbrales** que toma la red de forma tal que esta pueda recordar, posteriormente, las características que forman un patrón.

Función de Transferencia:

Función que determina cuando se enviarán los datos de una neurona a otra dentro de una red neuronal.

Lineas de Entrada:

Los canales por donde entra la información a una red neuronal.

Lineas de Salida:

Los canales por donde sale la información de una red neuronal.

Leyes de Entrenamiento:

Conjunto de reglas que rigen la forma en que se realizara el ajuste de los pesos y umbrales en la red neuronal durante el proceso de entrenamiento.

Memoria:

Lugar donde se almacenan los datos. En las computadoras tradicionales la memoria y el proceso de los datos se realizan en lugares separados. En las redes neuronales y en los cerebro humanos la memoria y el proceso de los datos se realizan en el mismo lugar.

Neuro-computadora:

Equipo de computo diseñado de forma que su unidad de procesamiento central sea una cantidad muy grande de pequeños elementos de proceso, todos interconectados y trabajando en paralelo para que de esta forma sea capaz de presentar las mismas características de las redes neuronales.

Neurofisiología:

Ciencia que estudia el comportamiento del sistema nervioso.

Neurona:

Celula mediante la cual se envian los impulsos nerviosos en los seres vivos.

Neurona Artificial:

Elemento de proceso simple con m entradas y una sola salida. Imita la forma en que las neuronas biologicas procesan la información.

No-Linealidad:

Las neuronas son deterministas y no vacilan en si su estado es activo o inactivo. El repentino cruce del umbral, el cual es la base de las desiciones claramente es una expresion de no-linealidad. La no-linealidad es una característica de las neuronas biológicas y es la razón de nuestra capacidad para tomar desiciones.

Patrón:

Conjunto de características que la red neuronal debe aprender.

Periodo de Adición Latente:

Tiempo que una neurona tarda en procesar la información.

Preprocesador:

Sistema que realiza una fragmentación o traducción de los datos de forma que estos puedan ser procesados por un siguiente proceso con mayor facilidad.

Redes Neuronales:

Forma en la que las neuronas se interconectan formando una malla o red para intercambiar mensajes. Las redes neuronales artificiales no se interconectan automaticamente como en el caso de las neuronas biologicas sino que debe de establecerse su forma de interconexión y su funcion de transferencia.

Retardo Sináptico:

Tiempo que se tarda una neurona en enviar su información por una conexión sináptica.

Sinápsis:

También se conoce como valor de pesos ó pesos. Valor que proporciona información sobre alguna de todas las características de un patrón para producir un estímulo en alguna neurona y poder recordar un patrón.

Sistema Experto:

Sistema que imita el proceso de deducción humana para llegar a un resultado, necesita ser reprogramado y de un experto capaz de dar los datos para hacer su base de conocimientos.

Soma:

Cuerpo celular de las neuronas biológicas.

Topología:

Forma en que se interconectan los nodos de una red neuronal.

Transputer:

Computadora especializada diseñada para trabajar en paralelo.

Umbral:

Función de no-linealidad que provoca una respuesta de actividad o inactividad de una neurona dependiendo del estímulo que reciba.

BIBLIOGRAFIA.

N. Baba.
Neural Networks.
Vol 2. (1989)

Cervantes Francisco.
Redes Neuronales para Computación.
Gazeta U.N.A.M.
9 de febrero de 1989.

K.C. Clarke.
Computer and Graphics.
Addison Wesley Publishing Company Inc.

Figueroa Nazuno J.G, Romero Bastida, Vargas Medina y Flores Garcia (1990)

Langevin equations and the formal foundations of Neural Networks.
International Joint-Conference on Neural Networks.
Washington, D.C., Enero 15-19, 1990.

Hanson Stephen, Burr David.
What connectionist models distributions learn: Learning and representation
in connectionist networks.
Behavioral and Brain Sciences (1990).

Hovel Davod.
The Science of computing.
Addison Wesley Publishing Company Inc.

Kaindl Herman
A.I. Magazine (1988).

Lippman R.P.
IEEE ASSP Magazine. Vol. 4 pp 4-22.(1987).

Nilsson Nils J.
Learning Machines
McGraw Hill Company.(1965).

Pao Yoh-Han.
Adaptive Pattern Recognition and Neural Networks.
Addison Wesley Publishing Company Inc.(1989).

D.E. Rumelhart, and R.J. Williams.
Learning Internal Representation by Error Propagation in Parallel distributed processing:
Explorations on the Microstructures of Cognition.
MIT Press (1986).

Rumelhart, D., and McClelland, J.
Parallel Distributed Processing. Cambridge, Mass.
The MIT Press, 1986.

Arbib A. Michael A.
Cerebros, Máquinas y Matemáticas.
Alianza Univesidad.

Burrascano, P.
Learning Vector Quantization for the Probabilistic Neural Network.
IEEE Trans. on Neural Networks, July 1991, 2, 458-461.

Caudill, M.
The Kohonen Model.
Neural Network Primer. AI Expert, 1990, 25-31.

Nishikawa, Y., Kita, H., and Kawamura, A.
NN/I: A Neural Network Which Divides and Learns Environments.
Proceedings of the International Joint Conference on Neural Networks.
January 1990, 1, I-684 to Y-687.

Simpson, P.
Artificial Neural Systems. New York, N.Y.
Pergamon Press, 1990.

Wasserman, P.:
Neural Computing, Theory and Practice. New York, N.Y.:
Van Nostrand Reinhold, 1989.