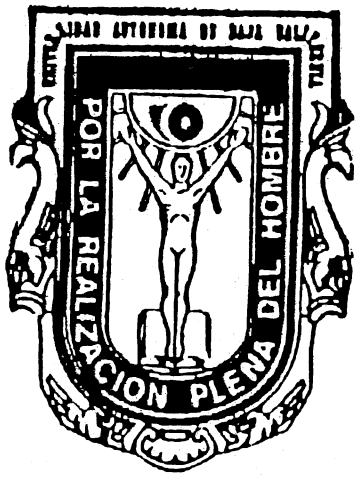
UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA
FACULTAD DE CIENCIAS



OPTIMIZACION DE UNA RED NEURONAL ENTRENADA POR RETROPROPAGACION

TESIS PROFESIONAL COMO REQUISITO PARCIAL

PARA OBTENER EL TITULO DE

LIC. CIENCIAS COMPUTACIONALES

PRESENTA

CLAUDIA ELIZABETH TALAVERA BALBUENA

ENSENADA, B. C. NOVIEMBRE 1992

UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS

OPTIMIZACION DE UNA RED NEURONAL ENTRENADA POR RETROPROPAGACION

TESIS PROFESIONAL

QUE PRESENTA:

CLAUDIA ELIZABETH TALAVERA BALBUENA

APROBADO POR:

DR. LAURA VIANA C.
Presidente

MC. ARMANDO REYES S.
Primer Vocal

MC. FERNANDO ROJAS I.
Secretario

MC. SONIA FAVELA
Suplente

LIC. ANA MARTINEZ
Suplente

Dedicatorias

A mis padres, con todo cariño y respeto.

A mi hermana, que me ha orientado siempre.

A mis familiares y amigos.

A mis maestros.

Agradecimientos

A la Dr. Laura Viana C. por su apoyo constante en la realización de mi tesis, por sus valiosas explicaciones durante el desarrollo de la misma.

A M.C. Fernando Rojas por su importante colaboración en el desarrollo de este trabajo, por sus observaciones y sugerencias en la presentación de mi tesis.

Al M.C. Armando Reyes por su valiosa y desinteresada ayuda en la realización del Servicio Social y conclusión de este trabajo.

A la M.C. Irma Rivera por la confianza y apoyo que siempre me brindó a lo largo de mis estudios de licenciatura.

A la M.C. Sonia Favela por su apoyo ofrecido durante la licenciatura y la orientación en la elaboración de este trabajo.

Al Dr. Mario Farías por sus valiosos consejos, y por su colaboración para continuar estudios superiores.

A la Lic. Ana Martínez por la revisión del trabajo escrito, así como el asesoramiento del equipo necesario para la realización del mismo.

A Sergio Carnalla Gastelum de manera muy especial, por los consejos dados para el desarrollo de este trabajo entre tantos otros.

A la Sra. Guillermina Moreno M. por sus observaciones en la tesis, como su colaboración en la búsqueda y préstamo del material bibliográfico dentro y fuera de la biblioteca del Instituto de Física UNAM.

A C. Agustina Romero A. por el trabajo tan eficiente que realizó en la documentación requerida para la Universidad Autónoma de Baja California.

A todo el personal que labora en el Instituto de Física, quienes de una manera u otra me apoyaron para la terminación de la tesis.

A la Universidad Autónoma de Baja California UABC y al Instituto de Física UNAM Laboratorio Ensenada, por la prestación de las instalaciones durante la realización del trabajo.

Resumen

En esta tesis se realizó una revisión general de la literatura de Redes Neuronales y se analizó un algoritmo que nosotros llamamos "AMNUI". Este algoritmo se basa en retropropagación y permite determinar el número óptimo de unidades de la capa intermedia. Se implantó en varios problemas, analizando el comportamiento del sistema al variar el factor de aprendizaje (η) en la etapa del entrenamiento, optimizando el número de neuronas y logrando el aprendizaje para cada caso.

Asbtract

In this thesis an overview is presented of the Neural Networks' literature and an analysis of an algorithm that we have called "AMNUI" is presented. This algorithm is based on retroprogation and it allows to determine the optimal number of units in the intermediate layer. It was implemented on several problems, and it was analized the behavior of the system upon varying the learning factor (η) throughout the learning phase, optimizing the number of neurons and achieving learning in every case.

Indice

Introducción

Durante las dos últimas décadas se ha realizado un gran esfuerzo en tratar de resolver problemas que se dificultaba encontrar su solución mediante métodos tradicionales de cómputo. Como ejemplo, tenemos problemas tales como el reconocimiento de imágenes y problemas de optimización. Un enfoque nuevo que ha resultado exitoso es utilizar métodos que tratan de simular el razonamiento humano. A este grupo de técnicas y métodos se les llama Inteligencia Artificial (IA).

La IA es utilizada para reproducir la inteligencia natural en la solución de problemas específicos, y básicamente consiste en la exploración de técnicas basadas en la representación del conocimiento en computadoras, para la simulación de un comportamiento inteligente (Rasmus, 1991). Entre las aplicaciones de la IA, se encuentran los llamados Sistemas Expertos (SE), y diversos modelos de Redes Neuronales Artificiales (RNA).

Los SE están constituidos principalmente por dos componentes, un motor inferencial y una base de conocimientos. El motor inferencial realiza el control logístico de una consulta, tomando decisiones basadas en la información proporcionada por las interfaces con el usuario, archivos externos, etc; y utilizando la base de conocimientos, dentro de la cual encuentra la información específica de alguna aplicación en particular, por ejemplo, para diagnóstico médico (Nelsol *et al.*, 1992).

Las RNA surgieron alrededor de los 40's, y consisten de un arreglo de procesadores o neuronas artificiales, que simulan el comportamiento de las neuronas biológicas. Cada neurona consta de dos estados, excitado o inhibido, es decir, estos estados

reflejan si la neurona emite o no una respuesta. Las RNA son un prototipo de la realización del llamado Procesamiento en Paralelo, donde se realizan varias operaciones concurrentemente, es decir, al mismo tiempo. (Müller *et al.*, 1991).

A diferencia de una computadora convencional, una Red Neuronal (RN) no es programable, sino que es entrenada; es decir, tiene que aprender para resolver un problema. El aprendizaje es mediante el entrenamiento, y se logra básicamente a través del uso de ejemplos. Las reglas de aprendizaje para las neuronas describen cómo cada neurona interpreta la información que va y viene a ella y al resto que la rodean (Gary, 1987). El flujo de información entre las neuronas está regulado mediante un valor asociado a cada conexión llamado "peso sinaptico"; una analogía interesante es la regulación del flujo de agua mediante compuertas en un canal, para lograr una distribución adecuada en un campo de irrigación. El valor del peso sináptico se modifica constantemente mediante un algoritmo de aprendizaje, hasta lograr que la red obtenga el comportamiento esperado.

Existen diversas arquitecturas o formas de interconectar los procesadores. Una arquitectura muy popular es la del tipo Perceptrón Simple, donde los procesadores que lo constituyen son acomodados uno a un lado del otro formando una capa, o nivel. Este tipo de arquitectura consiste en dos capas de neuronas, una capa de entrada, y otra capa de salida, en la cual cada elemento de una capa está conectado con todos los elementos de la otra capa. La idea es asociar valores para los estados de cada una de las neuronas de entrada con valores para los estados de las neuronas de salida, de manera que la información de entrada se encuentre distribuida entre las neuronas de la capa de entrada y la información final, en la capa de salida. Durante la etapa de aprendizaje o entrenamiento de la red, la información obtenida a la salida no será la óptima, hasta que los pesos

sean ajustados adecuadamente, ésto es, hasta que la red haya aprendido. Existe un tipo de redes llamadas de Alimentación hacia Adelante, ya que las señales o información fluyen entre las neuronas de una capa y la siguiente, ésto es, en una sola dirección. La arquitectura fue introducida y representada matemáticamente por Warren McCulloch y Walter Pitts en 1943 (Brunak *et al.*, 1990).

Se ha demostrado que el Perceptrón Simple presenta serias dificultades, ya que no es capaz de resolver algunos problemas (Hertz *et al.*, 1991). Fue por eso que se realizó la generalización del Perceptrón Simple, siendo nombrada arquitectura de Multicapas, la cual consiste en una red formada de varias capas, una después de otra. El algoritmo de aprendizaje más utilizado en esta arquitectura es conocido con el nombre de Retropropagación. Este trabajo está enfocado principalmente a este tipo de arquitectura.

El algoritmo de retropropagación, fue desarrollado a mediados de los 80's por Rumelhart, un psicológo de la Universidad de Stanford (CA). Este algoritmo se encuentra categorizado dentro de los que trabajan con el método de Aprendizaje Supervisado, el cual consiste en buscar los pesos sinápticos adecuados de las conexiones, permitiendo definir los valores para las unidades de entrada y de las unidades de salida, bajo una especie de supervisión como la de un "maestro" en casa, por medio de una serie de ejemplos. Cuando los valores de salida deseados son iguales a los de entrada, la red es de tipo autoasociativa, y cuando éstos son diferentes es llamada heteroasociativa.

Un parámetro importante del algoritmo de retropropagación, es el factor de aprendizaje (η), porque determina la intensidad de aprendizaje de la red (más adelante se profundizará en el tema). Este algoritmo es de gran importancia, porque per-

mite la obtención de soluciones a un gran número de problemas, como sucedería en problemas de reconocimiento de patrones, donde los patrones pueden ser imágenes codificadas, de manera que la red tenga la capacidad de reconstruir imagénes incompletas.

Una RN puede ser diseñada con ayuda de una computadora convencional secuencial, la cual consta de un solo procesador (CPU), en la cual la RN se simula, ya que construir la circuiteria (hardware) implicaría un equipo de expertos, tiempo y recursos asociados con la construcción de la misma (Studt, 1991).

Uno de los problemas que se encuentran al diseñar una RN es la dificultad de determinar el número adecuado de unidades de la capa intermedia, en una red compuesta de tres o más capas. Esto es debido a que la cantidad de conexiones que tenga la red dependerá del número de unidades de esta capa. Si tenemos más unidades que las indispensables, el número de conexiones aumentará innecesariamente, haciendo más difícil su construcción y más costoso su diseño. Dada esta dificultad, el presente trabajo revisa el funcionamiento de un algoritmo diseñado por Yoshio Hirose (Hirose *et al.*, 1990), para obtener el número óptimo o adecuado de unidades intermedias (AMNUI).

El presente trabajo consta de los siguientes capítulos. En el capítulo 1 se hará una breve introducción sobre el significado e importancia de las RN, la motivación biológica de este estudio y se mencionarán las bases del primer modelo artificial. En el capítulo 2, nos adentraremos en la Arquitectura tipo Perceptrón, donde analizaremos los algoritmos de aprendizaje para el caso en que las unidades de salida tienen valores discretos y para el caso de respuesta con valores continuos. Entre estos algoritmos estudiaremos el denominado Descenso de Gradiente, el cual es útil

para dar solución a algunos problemas simples, como son las funciones booleanas "Y" (AND) y "O-Exclusivo" (XOR). El capítulo 3 está enfocado a describir el funcionamiento de la arquitectura de Multicapas, haciendo énfasis en el algoritmo de Retropropagación, el cual es utilizado como algoritmo de aprendizaje, así como a mencionar de una manera muy general, algunos problemas resueltos con este tipo de algoritmo. En el capítulo 4, se llevará a cabo el análisis e implantación del algoritmo que ayuda a la determinación del número óptimo de unidades intermedias en tres problemas diferentes. Finalmente en el capítulo 5 se analizará el comportamiento de una RN ya entrenada, ante la presentación de diferentes estados iniciales.

A continuación describiremos los problemas analizados en el capítulo 4. El primer problema, es la solución del problema de la función booleana O-Exclusivo (XOR), formada por una red compuesta de dos unidades de entrada y de una unidad de salida. El segundo problema es un contador de las unidades activas de los patrones de entrada, en el cual el resultado se despliega mediante la activación de únicamente una de las unidades de salida; en este caso los patrones de entrada utilizados fueron los números binarios del (0 ... 15) (Ooyen, 1988). El tercer problema se dividió en dos partes, la primera parte consiste en que la red aprenda las diez primeras letras del abecedario (A ... J) como patrones de entrada, donde cada letra está formada en un arreglo de celdas de 8x8. es decir, 64 unidades de entrada; los patrones de salida constan de una unidad activada y el resto desactivadas, tal cómo sucedió en el problema anterior, pero la posición de la unidad de salida activada dependerá de su correspondiente patrón de entrada. La otra parte del problema consiste en que la red aprenda 36 patrones, compuestos de las 26 letras del abecedario (A ... Z), y de 10 digitos (0 ... 9). En ambos problemas, el número de unidades

de la capa de salida es igual al número de patrones con que se entrena a la red (Hirose *et al*,. 1991). Finalmente, en cada uno de estos problemas se variará el factor de aprendizaje (η), representado básicamente por medio de gráficas, donde se mostrará la relación entre η (eje de las x's) y el número de iteraciones requeridas para que exista convergencia en el algoritmo (eje de las y's).

I. Redes Neuronales

Nosotros sabemos que nuestro cerebro es superior a una computadora digital en muchas tareas, por ejemplo, en la capacidad de entender, recordar y almacenar cierto tipo de información, como por ejemplo, la enorme capacidad de un bebé en el reconocimiento de objetos, en comparación con sistemas basados en Inteligencia Artificial (IA), diseñados para computadoras muy veloces. Podríamos decir que el ser humano va aprendiendo conforme se modifican las interconexiones sinápticas entre sus neuronas; ésto ha sido demostrado experimentalmente en organismos simples.

Una de las aplicaciones de la IA, se está enfocada a la posibilidad de formar Redes Neuronales Artificiales. Como mencionamos en la introducción, estas redes consisten de un conjunto de procesadores o "neuronas" interconectadas en paralelo, que se caracterizan por dos estados de actividad o un valor de actividad que toma un valor continuo dentro de un intervalo. Los diferentes modelos que las describen son extremadamente simplificados comparados con el funcionamiento de las neuronas dentro del cerebro humano, sin embargo en sistemas tan simplificados como éstos ya se han observado características importantes de la memoria humana.

Para poder darnos una idea de las bases utilizadas para la formulación de los primeros modelos de RNA, a continuación mencionaremos la estructura y comportamiento de las neuronas biológicas, así como la importancia de su función en colectividad formando la llamada Red Neuronal.

I.1 La neurona

Desde finales del siglo XVII el hombre tenía un conocimiento muy amplio acerca de la anatomía del cerebro y sus diversas partes. Sin embargo, el conocimiento del sistema nervioso a nivel celular se inició hasta principios de siglo XVIII. En parte, esto fue una consecuencia natural de la falta de técnica e intrumentos apropiados que permitiesen llevar a cabo ciertas observaciones y experimentos.

Hasta finales del siglo XIX, la idea general era que el tejido nervioso es un tejido continuo. Alrededor de 1880 Camilo Golgi observó bajo el microscopio, un corte de tejido nervioso en el cual se había derramado una solución de sales de plata y encontró que la plata se había impregnado en ciertas áreas de tejido, revelando que las células nerviosas son individuales, y delineando la forma de éstas y de sus extensiones (Viana, 1990).

Esta misma técnica fue aplicada ingeniosamente por un doctor español llamado Santiago Ramón y Cajal en 1880, quedando demostrado que las unidades básicas del tejido nervioso son células separadas unas de otras, denominadas neuronas. Con este descubrimiento se abrió el camino hacia la neurociencia moderna (Müller *et al.*, 1991).

La neurona es la célula constituyente del tejido nervioso, capaz de generar y de conducir los impulsos nerviosos. Básicamente está constituída por una masa central que contiene el núcleo, llamado soma, del cual parten prolongaciones llamadas axón y dendritas (figura 1). El axón es una prolongación, a veces extraordinariamente larga y que en ocasiones se ramifica; por otro lado, las dendritas son ramificaciones cortas. La misión de los axones y dendritas es básicamente la de transmitir y recibir

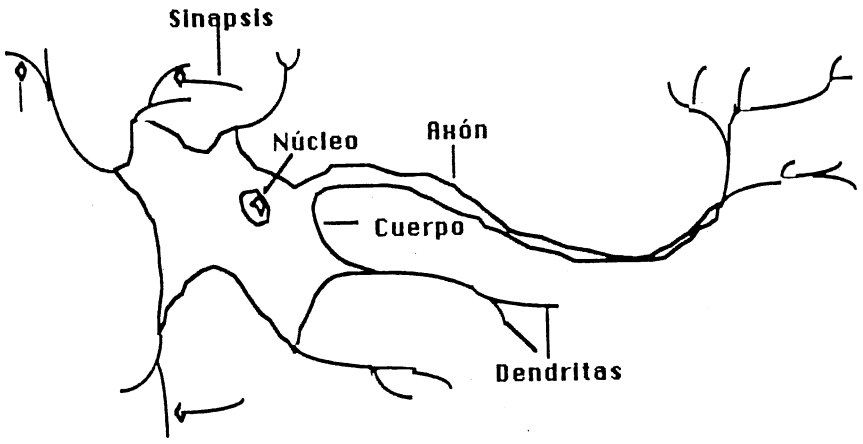


Figura 1: Neurona.

señales respectivamente (Enc., 1981).

Por medio de las dendritas, cáda neurona recibe un gran número de señales provenientes de otras neuronas. Estas señales son sumadas y promediadas; en caso que la intensidad total del estímulo recibido sea mayor a cierto umbral característico, la neurona generará y emitirá una señal eléctrica de respuesta (Viana, 1990). La señal será enviada por el axón hasta llegar a una parte en donde se juntan dos neuronas, conocido como sinapsis o uniones sinapticas, estas uniones sinapticas son establecidas entre el axón y dentrita o entre axón y soma; en donde de nueva cuenta se transmitirá la información hacia otras neuronas mediante un intercambio químico. Finalmente una sinapsis será excitadora o inhibidora, de acuerdo con el

tipo de substancia neurotransmisora que emite (Enc., 1981).

El propósito principal de la neurona es la generación y transmisión de señales eléctricas, teniendo como objetivo, la comunicación con las demás neuronas del sistema nervioso central, para lograr el procesamiento de información que pasa a través de él. A la colectividad de neuronas se le llama "Red neuronal".

Típicamente, el número de conexiones que tiene una neurona va arriba de 10^4, pero varía grandemente dependiendo del tipo de neurona y del sitio donde está colocada dentro del cerebro. Las neuronas se comunican por pulsos cortos, generalmente de un milisegundo de duración, y con una velocidad de casi 500 kilómetros por hora. Dentro del sistema nervioso existen 10^{11} neuronas, sus dimensiones varían entre 0.002 mm y 0.5 mm (Viana, 1990) dependiendo del tipo de neurona que se trate; el número de señales que intercambia es de más de 10^{14} pulsos por segundo.

Desde que nacemos, tenemos la capacidad de almacenar y de asociar la información; este almacenamiento se logra mediante la modificación de las interacciones sinapticas y es mejor conocido como aprendizaje. El aprendizaje es consi-derado como el cambio directo dentro de la estructura del conocimiento. Hebb un psicólogo, propuso que la memoria reside dentro de las conexiones sinapticas del sistema nervioso central, y que todo aprendizaje consiste en un cambio del poder sinaptico de las neuronas (Soucek, 1988). Esta suposición está sustentada por evidencia experimental. La ley de Hebb sirvió como base para la introducción del concepto de aprendizaje en máquinas neuronales a partir de modelos biológicos.

I.2 Historia

Los procesos del pensamiento han sido estudiados desde hace mucho tiempo, las explicaciones técnicas del cerebro y el proceso de pensar fue estudiado por los antiguos filósofos griegos, como Platón (427 - 347 B.C) y Aristóteles (384- 322 B.C.), habiendo este último formulado un conjunto de observaciones sobre memoria humana y asociación (Kohonen, 1988).

En los últimos años ha habido una gran cantidad de esfuerzos para hacer Redes Neuronales Artificiales. Un primer modelo fue sugerido en 1943 por Warren McCulloch, neurocientífico, y Walter Pitts, matemático. Ambos implementaron las bases de la " Computación Neuronal " , y propusieron la teoría general del procesamiento de información basado en redes binarias o elementos de decisión, las cuales son simplemente llamadas "neuronas". McCulloch y Pitts demostraron como dichas redes, en principio, podían realizar una cantidad inimaginable de cálculos, similares al que realizan las computadoras tradicionales. En cierta manera, las redes neuronales poseen un "programa en código", el cual gobierna el proceso de cálculo, contenido en la matriz de pesos ω_{ij}, donde los índices indican que la neurona i excita o inhibe a la neurona j con magnitud ω; esta matriz de pesos regula el proceso de aprendizaje, y es modificada constantemente hasta lograr que la red aprenda, es decir, que la red nos proporcione la solución requerida (figura 2).

El modelado neuronal analítico está en conexión con las teorías psicológicas y las investigaciones neurofisiológicas. Entre estas teorías, podemos mencionar a Hebb en 1949 con su leyes sobre aprendizaje. Hebb propuso la famosa regla de

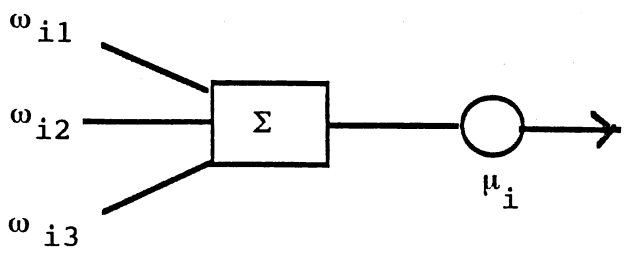


Figura 2: Diagrama esquemático de la neurona de McCulloch - Pitts.

aprendizaje en la cual dice que el proceso de aprendizaje está relacionado con la modificación de las sinapsis. Posteriormente a esta regla se le dió una forma analítica para el almacenamiento de patrones (Soucek, 1988). Se define como patrón a los valores iniciales y en algunos casos los valores esperados en la salida, en la aplicación de un modelo específico en una red neuronal.

Cragg y Temperley en 1954, reformularon la red descrita por McCulloch-Pitts como un sistema de espines magnéticos. Para ésto, hicieron una analogía con un material magnético que consiste de un conjunto de átomos que se comportan como si fuesen pequeños imanes, que interactúan entre sí. En estos casos se dice que los átomos tienen un "momento magnético" diferente de cero, el cual se caracteriza por su magnitud y en la dirección en que está orientado. A estos imanes, se les llama espines magnéticos o simplemente espines (Viana, 1990). Uno de los modelos más utilizados para describir estos sistemas es el modelo de Ising, en el cual, cada espín tiene dos orientaciones posibles, i.e. hacia arriba y hacia abajo, tomando los valores $S_i = \pm 1$ respectivamente (Hertz *et al.*, 1991). Con la ayuda de la Física

Estadística ha sido posible entender el comportamiento colectivo de este sistema magnético y por analogía, así como el comportamiento colectivo de un conjunto grande de neuronas para procesar y almacenar información en el cerebro a un nivel muy simplificado.

En 1960, había una gran actividad alrededor del grupo de RN de Frank Rosenblatt enfocándose en el problema de encontrar los pesos (ω_{ij}) apropiados de las conexiones para efectuar ciertas tareas computacionales. En aquella época se concentraban en el estudio de las redes llamadas Perceptrones simples, porque se consideraban un modelo simplificado del procesamiento de la información biológica, que aunque muy simplificado, se podía llevar a cabo cualquier tipo de cálculo. Las unidades eran organizadas en capas con conexiones hacia adelante entre una capa y la siguiente. Redes muy parecidas, llamadas Adalines (Elemento Lineal Adaptable), fueron inventadas al mismo tiempo por Widrow y Hoff (Hertz *et al.*, 1991). Más adelante fue necesario la introducción del Perceptrón de Multicapas dado que se encontró, que el perceptrón simple no puede resolver todos los problemas, de esto hablaremos en el capítulo siguiente.

El desarrollo de la teoría de redes neuronales en los 70's se basaba principalmente en el tema de "memoria asociativa". Con este tipo de memoria, la búsqueda de datos se realiza por medio del contenido, ya que la información no está localizada en ninguna localidad específica. El diseño de la circuitería (hardware) es de tipo paralelo, aquí cada procesador realiza una pequeña tarea. Cada procesador consta de memoria, la cual se asocia con las demás memorias de los diferentes procesadores formando una red neuronal, este tipo de funcionamiento es similar al almacenamiento de información llevado acabo en el cerebro humano.

Las redes neuronales autoasociativas que trabajan con este tipo de memoria, almacenan un conjunto de patrones, de tal forma que cuando le presentamos un nuevo patrón ésta va a responder produciendo el patrón almacenado que más se asemeje a él. Un ejemplo de esto, es cuando relacionamos una fotografía inclompleta con una ya memorizada (Hertz *et al.*, 1991).

Probablemente el desarrollo más significativo continúa con el trabajo de Rosenblatt, Hinton y Williams en 1986, quienes decubrieron un algoritmo de aprendizaje que trabaja ajustando los pesos que conectan las unidades en niveles sucesivos de perceptrones multicapas, conocido con el nombre de Retropropagación. (Hertz *et al.*, 1991).

Más adelante explicaremos algunos de los modelos ya mencionados, así como sus características y comportamiento.

II. Redes de Alimentación hacia Adelante

Una Red Neuronal consiste de una colección de elementos llamados unidades de procesamiento, los cuales procesan información de manera colectiva a través de la interconexión entre ellos. Cada unidad envía una señal excitadora o inhibidora hacia las otras unidades con las que está conectada y a cada conexión, le es asociado un número real llamado peso (ω), el cual determina que tan importante y de que tipo es la influencia que tiene una unidad sobre otra (Rumelhart *et al.*, 1986).

El proceso de aprendizaje consiste, como ya se ha mencionado anteriormente, en la determinación de los pesos (ω) en las conexiones de la red, que hacen que la red se comporte de la forma adecuada; ésto hace que la red evolucione dinámicamente hacia ciertos estados específicos. De esta manera el aprendizaje es considerado como un problema dinámico.

Existen dos tipos generales de aprendizaje: supervisado y no supervisado. En el aprendizaje supervisado los cambios en los pesos de las conexiones están hechos directamente como respuesta a una comparación con ejemplos correctos, por otro lado, en el aprendizaje no supervisado estas modificaciones se hacen de acuerdo con un criterio externo fijo. A continuación hablaremos de una posible arquitectura de las redes que consiste en capas, es decir explicaremos la manera en que las neuronas están conectadas unas con otras (Garrido, 1990).

II.1 Redes de capas

Los modelos de redes neuronales que aquí se discuten son usados para almacenar y obtener información. Estas RN llamadas Perceptrones o redes de alimentación hacia adelante, obtienen su nombre debido al flujo de información en una sola dirección entre las distintas capas. Los primeros diseños de este tipo, fueron efectuadas por Rosenblatt en 1962, quien modeló la operación del perceptrón en una computadora IBM 740. Los programas, simulaban las conexiones interneuronales y sus pesos estaban representados por medio de una matriz.

Los perceptrones están compuestos por una o más capas de neuronas formales y constan de tres tipos de unidades: las unidades de entrada, unidades asociativas o intermedias y las unidades de respuesta. Si hacemos una comparación con un sistema biológico entonces las unidades de entrada corresponderían a la entrada inicial sensorial, las unidades asociativas serían las unidades intermedias a través de las cuales fluye la información y las unidades de respuesta representarían la información de salida de la red.

Para saber en qué consiste el aprendizaje de una RN, nos ocuparemos de definir el término patrón. Un patrón μ consiste en un conjunto de valores $\{\xi_k^\mu\}$ o $\{\zeta_i^\mu\}$, para las unidades de la capa de entrada o salida respectivamente. Este conjunto puede estar constituído por valores que representen los dos estados, apagado o encendido, que pueden ser 0 y 1, ó -1 y 1 ó valores continuos acotados. El aprendizaje consiste en lograr que la red realice una asociación entre cada patrón de entrada $\{\xi_k^\mu\}$ y el patrón correspondiente de salida $\{\zeta_i^\mu\}$. Esto es, si la capa de entrada de la red se pone en un estado cercano al patrón $\{\xi_k^\mu\}$, el sistema evolucionará dinámicamente

hasta llegar al patrón $\{\zeta_i^\mu\}$ en la capa de salida. Inicialmente, si la red no ha sido entrenada y presentamos un patrón $\{\xi_k^\mu\}$ de entrada, obtendremos un patrón $\{\theta_i^\mu\}$ de salida, el cual no tendrá nada que ver con $\{\zeta_i^\mu\}$. Una manera de adiestrar a la red, es mediante la modificación constante de la matriz de pesos ω_{ik}, de acuerdo con algún algoritmo de aprendizaje. El estado final que queremos obtener es entonces:

$$\theta_i^\mu = \zeta_i^\mu \tag{1}$$

Para toda unidad $_i$ y cada patrón μ.

Las redes de alimentación hacia adelante pueden estar enfocadas hacia la heteroasociación o la autoasociación . La heteroasociación consiste en establecer una relación entre patrones diferentes, ésto es, los patrones de entrada $\{\xi_k^\mu\}$ son distintos a los patrones de salida $\{\zeta_i^\mu\}$ ($\xi_k^\mu \neq \zeta_i^\mu$). Por otro lado, en la autoasociación, los patrones de entrada $\{\xi_k^\mu\}$ tienen que ser iguales a los patrones de salida ($\xi_k^\mu = \zeta_k^\mu$). Obviamente, para la heteroasociación el número N_S de unidades de salida puede ser mayor o menor que el número N_E de unidades de entrada, en la autoasociación el número de unidades es igual para ambos casos.

A cada conexión entre el elemento i-ésimo de la capa de salida y el elemento k-ésimo de la capa de entrada se le asocia un peso ω_{ik}; tal que puede ser guardado en un vector n - dimensional llamado "vector de pesos", $\omega_{ik} = (\omega_{i1}, \omega_{i2}, \ldots, \omega_{in})$. De igual manera, cada patrón $\{\xi_k^\mu\}$, se va a considerar como un vector N_E-dimensional $\vec{\xi^\mu}$ y otro vector N_S-dimensional $\vec{\zeta^\mu}$ para los patrones correspondientes de salida.

Cada elemento o componente de la red tiene un valor umbral, el cual se puede definir como la mayor señal que éste puede recibir sin que cambie el estado del sistema. Una analogía útil sería cuando empujamos a alguien, se supone que esta persona siente el empujón y se mueve (en el caso de que empujemos fuertemente), pero si empujamos poco, no pasa nada. Entonces debe de existir alguna cantidad de tolerancia, por encima de la cual existe una reacción; si el empujón es menor que esa cantidad, no existe reacción alguna.

El ejemplo más simple de redes de alimentación hacia adelante consta únicamente de una capa de entrada y otra de salida sin elementos intermedios. Este tipo de redes es conocida con el nombre de Perceptrón Simple (figura 3) y su representación matemática es la siguiente:

$$\theta_i = g(h_i) = g(\sum_k \omega_{ik}\xi_k) \tag{2}$$

donde el estado de activación para la unidad $_k$-ésima de entrada es denotada por ξ_k y el estado de la unidad $_i$ de la capa de salida es denotada por θ_i, con $i = \{1, \ldots, n\}$, siendo $g(h_i)$ una función llamada función de activación, que calcula el nuevo valor de respuesta para las unidades de salida. Los pesos entre la unidad i y la unidad k son representados por una matriz ω_{ik}, en la cual los pesos positivos significan que la unidad k excita a la unidad i y los pesos negativos indican que la unidad k inhibe a la unidad i; en el caso de que este peso sea cero, las unidades involucradas no poseen directamente ninguna conexión (Rumelhart *et al.*, 1986).

Para el perceptrón simple, el valor de θ_i^μ, que se obtendrá de las unidades de salida

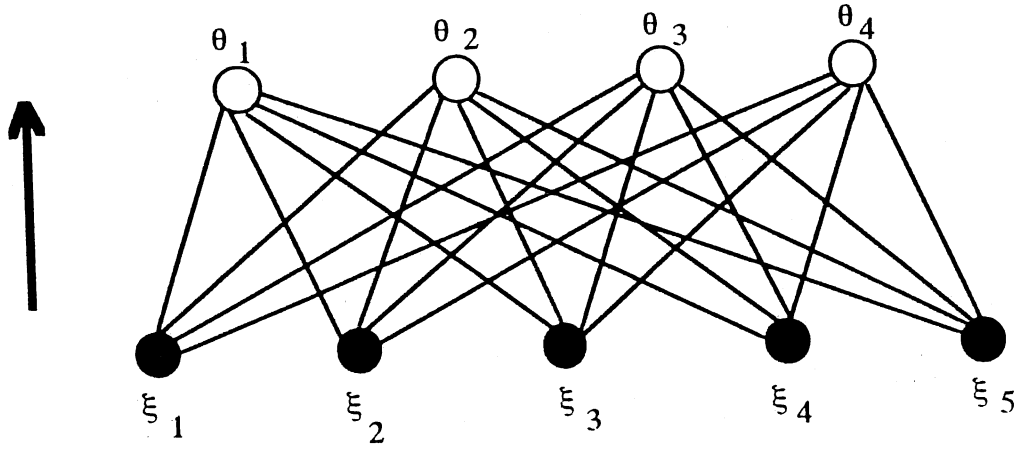


Figura 3: Perceptrón simple.

y estará dado por:

$$\theta_i^\mu = g(h_i^\mu) = g(\sum_k \omega_{ik}\xi_k^\mu), \tag{3}$$

En este tipo de RN, la activación deseada de las neuronas de salida $\{\zeta_i^\mu\}$, es decir los patrones de entrada que se debe aprender, para la señal de entrada $\{\xi_k^\mu\}$ muchas veces está determinada por una función no lineal $g(h)$. La función $g(h)$ puede ser una ley estocástica que por ejemplo, determina la probabilidad de los valores de $\zeta_i^\mu = \pm 1$, o una función determinista, que también puede tomar valores discretos o continuos.

Como ejemplo de lo anterior, tenemos una red en la cual se requerirá de una función

de activación $g(h) = sgn(h)$, donde el patrón de salida ζ_i^μ puede tomar valores de ± 1, esta función de activación está definida de la siguiente manera:

$$sgn(h) = \begin{cases} 1 & \text{si } h \geq 0 \\ -1 & \text{otro caso} \end{cases} \tag{4}$$

Consideremos un sistema simple compuesto de la ecuación (3), lo podemos representar también por medio de notación vectorial, quedándonos:

$$sgn(\vec{\omega} \cdot \vec{\xi^\mu}) = \zeta^\mu \tag{5}$$

Para toda μ.

Esta ecuación tendrá solución si encontramos un vector de pesos $\vec{\omega}$, tal que al proyectarle el patrón ξ^μ tenga el mismo signo que ζ^μ. Esto significa, que necesitamos un plano que se encuentre dentro del espacio ξ y que separe los valores de los componentes de salida positivos ($\zeta = 1$) de los negativos ($\zeta = -1$).

El límite entre las proyecciones positivas y negativas en ω, es el plano $\omega \cdot \vec{\xi} = 0$ que pasa por el origen, y es perpendicular a ω.

Para analizar la ecuación (5) necesitamos tener en cuenta la simetría de la función $sgn(x)$, en donde $-sgn(-x) = sgn(x)$, entonces multiplicamos ambos lados de la ecuación (5) por ζ^μ, dado que ζ^μ puede tomar únicamente los valores de ± 1, donde

la ecuación no se altera, quedándonos de la siguiente manera:

$$\zeta^{\mu} sgn(\vec{\omega} \cdot \vec{\xi^{\mu}} \zeta^{\mu}) = sgn(\vec{\omega} \cdot \vec{\xi^{\mu}}) = \zeta^{\mu} \tag{6}$$

Ahora si representamos $\vec{\xi^{\mu}} \zeta^{\mu} = \vec{X^{\mu}}$, la ecuación cambia a :

$$\zeta^{\mu} sgn(\vec{\omega} \cdot \vec{X^{\mu}}) = \zeta^{\mu} \tag{7}$$

Para que la ecuación (7) se cumpla, tenemos la siguiente condición:

$$\vec{\omega} \cdot \vec{X^{\mu}} > 0 \tag{8}$$

La cual se debe cumplir, sobre todos los patrones de entrada y de salida definidos, para que el problema tenga solución. Representado gráficamente lo anterior, analizaremos la función booleana Y (AND), donde contamos con un perceptrón de dos unidades de entrada ξ_1, ξ_2 y una unidad de salida ζ. Dada la siguiente tabla de verdad para la función Y (AND):

Casos	ξ_1	ξ_2	ζ
I	-1	-1	-1
II	-1	$+1$	-1
III	$+1$	-1	-1
IV	$+1$	$+1$	$+1$

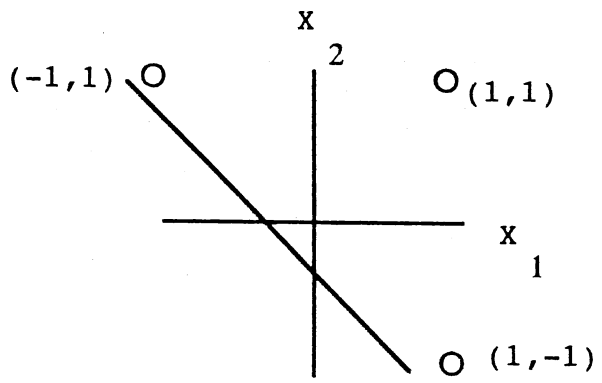


Figura 4: Patrones de la Función booleana Y (AND)

Pri mero obtenemos los valores $\vec{X}^{\mu}$, que son los siguientes: $\vec{X}^{1} = (1,1)$, $\vec{X}^{2} = (1,-1)$, $\vec{X}^{3} = (-1,1)$, $\vec{X}^{4} = (1,1)$ (figura 4). Segundo, entrenamos a la red para obtener los pesos ω apropiados, para que se cumpla la ecuación (8). Una manera de encontrar los valores de los pesos es generalizando la ecuación (5) con unidades de umbral que a continuación explicaremos.

Se dice que un problema puede ser linealmente separable cuando tiene solución por medio de la ecuación (5). Para el caso de que el plano que los separe no está en el origen, definiremos la siguiente expresión matemática:

$$\theta_i = sgn(\sum_{k>0} \omega_{ik}\xi_k - \vartheta) \tag{9}$$

o con notación vectorial:

$$\theta = sgn(\vec{\omega} \cdot \vec{\xi} - \vartheta) \tag{10}$$

Donde ϑ significa el umbral permitido para el sistema. Continuando con el mismo ejemplo, se analizarán las condiciones para saber si el problema de la función booleana Y (AND) es linealmente separable. utilizaremos la función de activación $sgn(h)$, cuya salida será ± 1. La idea es encontrar los pesos ω_{ik} por medio del planteamiento de un sistema de ecuaciones. Si existe al menos una solución a dicho sistema de ecuaciones, podremos decir que el problema tiene solución y puede ser representado por medio del perceptrón simple.

Aplicando la ecuación (9), para los valores de la tabla anterior, obtenemos las condiciones siguientes:

Para el Caso I:

Dados los valores de $\xi_1 = -1$ y $\xi_2 = -1$, tenemos que:

$$\theta = sgn(-\omega_1 - \omega_2 - \vartheta) \tag{11}$$

Para los valores de ξ, definidos en la tabla Y (AND), justificamos lo siguiente:

$$-\omega_1 - \omega_2 - \vartheta < 0$$

Realizando la misma operación, para el resto de los casos, finalmente obtenemos el siguiente sistema de ecuaciones:

$$-\omega_1 - \omega_2 < \vartheta$$

$$-\omega_1 + \omega_2 < \vartheta$$

$$\omega_1 - \omega_2 < \vartheta$$

$$\omega_1 + \omega_2 > \vartheta$$

Mediante el análisis de estas condiciones, concluimos que: tienen una cantidad enorme de soluciones, de las cuales una es suficiente, para que el problema pueda ser resuelto por medio del perceptrón simple. Una de estas soluciones es: $\omega_1 = 1$, $\omega_2 = 1$ y $\vartheta = 1.5$. Si suponemos que $\vartheta = 0$, veremos que sí se cumple aplicando la ecuación (8), por lo tanto el problema de la función booleana Y (AND) tiene solución (figura 5).

Un buen ejemplo de un problema que no puede separarse, sería el caso de la función booleana O-Exclusivo (XOR), su tabla de verdad está definida por:

Casos	ξ_1	ξ_2	ζ
I	-1	-1	-1
II	-1	$+1$	$+1$
III	$+1$	-1	$+1$
IV	$+1$	$+1$	-1

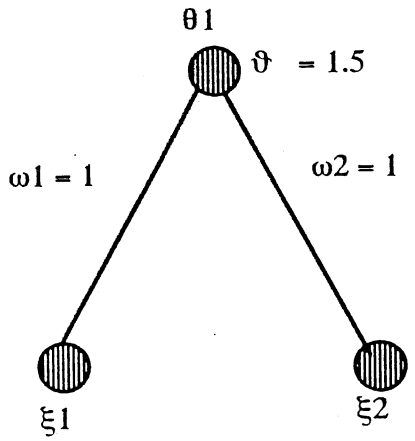


Figura 5: Solución de la Función booleana Y (AND).

Para los casos del O-Exclusivo (XOR), las ecuaciones resultantes son las siguientes:

$$-\omega_1 - \omega_2 < \vartheta$$

$$-\omega_1 + \omega_2 > \vartheta$$

$$\omega_1 - \omega_2 > \vartheta$$

$$\omega_1 + \omega_2 < \vartheta$$

Con la suma de las ecuaciones primera y tercera, obtenemos la condición $\omega_1 > 0$, por otro lado, las ecuaciones segunda y cuarta se tiene que $\omega_1 < 0$. Con estas

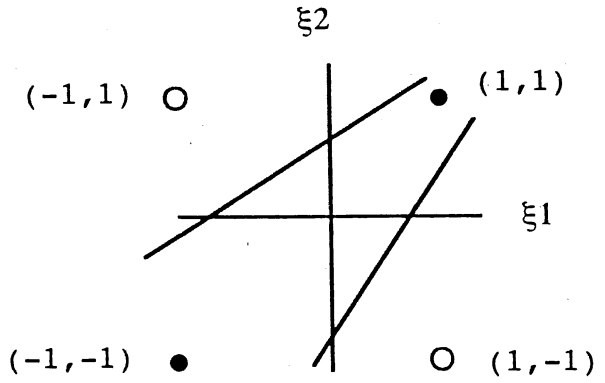


Figura 6: Función booleana O-Exclusivo (XOR).

dos condiciones se comprueba la inconsistencia del sistema. El problema del O-Exclusivo (XOR) es llamado linealmente inseparable, porque no existe una forma de subdividir el espacio de variables de entrada en regiones iguales de salida por una única condición lineal (figura 6), lo cual significa que el problema de la función O-Exclusivo (XOR) no puede ser resuelto por una red de tipo perceptrón simple, siendo uno de los motivos la generalización de este algoritmo. En el capítulo siguiente se analizará este problema, mediante este nuevo enfoque.

II.2 Algoritmos de aprendizaje del Perceptrón simple

Un algoritmo para aprendizaje es un procedimiento para encontrar los pesos apropiados de la red, de manera que se establece una asociación entre los patrones de

entrada y sus correspondientes patrones de salida. Este procedimiento consiste en revisar si la señal de salida es la deseada, basándose en las unidades de entrada. El valor de las conexiones es modificado constantemente mientras que las salidas obtenidas sean diferentes a las deseadas; a este procedimiento de modificación se le llama entrenamiento de la red.

Existen varios algoritmos para lograr este objetivo, entre ellos está el que se basó en los estudios de Hebb llevados a cabo en 1949. Las hipótesis de Hebb fueran que las conexiónes sinapticas entre las neuronas eran reforzadas, cuando las neuronas de entrada y salida estaban activas (Wasserman *et al.*, 1988). Específicamente, la ley de Hebb se interpreta como un cambio en los pesos de las conexiones sinapticas ω_{ik}, debido al conjunto de patrones μ, el cual debe ser un múltiplo del producto de los estados deseados de la neurona k-ésima de entrada e i-ésima de salida. La expresión formal de la regla es:

$$\omega_{ik\{nueva\}} = \omega_{ik\{vieja\}} + \Delta\omega_{ik} \tag{12}$$

donde :

$$\Delta\omega_{ik} = \begin{cases} 2\eta\zeta_i^\mu\xi_k^\mu & \zeta_i^\mu \neq \theta_i^\mu \\ 0 & \zeta_i^\mu = \theta_i^\mu \end{cases} \tag{13}$$

donde el término $\eta\zeta_i^\mu\xi_k^\mu$ es la regla de Hebb y por lo tanto $\Delta\omega_{ik}$ será diferente de cero si la salida obtenida es diferente de la salida deseada ($\zeta_i^\mu \neq \theta_i^\mu$). Recordemos que estamos considerando valores de $\zeta_i^\mu = \pm 1$, $\theta_i^\mu = \pm 1$ por lo que origina que

$\zeta_i^\mu = -\theta_i^\mu$. Esta ecuación puede escribirse como:

$$\Delta\omega_{ik} = \eta(1 - \zeta_i^\mu\theta_i^\mu)\zeta_i^\mu\xi_k^\mu \qquad (14)$$
$$= \eta(\zeta_i^\mu - \theta_i^\mu)\xi_k^\mu$$

El parámetro η es llamado el factor de aprendizaje, porque describe que tan grandes o pequeñas serán las correcciones de los pesos en la red; este factor debe ser escogido de tal manera que obtengamos una convergencia óptima, es decir, en el menor número posible de iteraciones, aplicando la ecuación (14). Es costumbre escoger valores de η menores que 1. En el caso de que queramos satisfacer varias de estas asociaciones estímulo - reacción, esto es que se aprendan varios patrones, podemos simplemente sumar sobre las modificaciones requeridas para cada uno de ellos (Müller *et al.*, 1991).

II.2.1 Ejemplos de problemas de optimización

En el transcurso de los años, el estudio de los métodos de optimización ha tenido un incremento constante en diversas áreas, como en la ingeniería, ciencias computacionales etc. Entre los problemas estudiados en estas áreas, se encuentran los del tipo de optimización combinatoria, los cuales constan de un conjunto de configuraciones o espacio de configuraciones y de una función compuesta de muchas variables independientes, llamada Función de Costo, la cual asigna un número real a cada configuración. La idea es encontrar el máximo o mínimo valor obtenido de la función de costo, la cual representa el grado de efectividad del sistema (Laarhoven *et al.*, 1988).

Como ejemplo a problemas de optimización combinatoria podemos mencionar el problema del acomodo de conferencistas, en un conjunto de casas ubicadas dentro de una universidad. Por comodidad, las personas son divididas en grupos dependiendo de los temas que dominan, para reducir el tiempo que tardarían en llegar a las conferencias deseadas. El problema es tratar de minimizar el número de casas asignadas a estos grupos, procurando la comodidad de sus visitantes. En este problema, el número de factores necesarios para solucionarlo son demasiados, entre ellos están el tiempo que duran las conferencias de un área en particular, la capacidad de cada casa, etc (Rayward *et al.*, 1979). Otro problema complejo, que hace uso de una función de costo, el problema consiste en el diseño de un circuito integrado compuesto de N elementos que no caben dentro de una pastilla (chip). Debido a que el número de elementos que lo constituyen es muy grande, tiene que ser repartido en dos pastillas. El problema consiste en tratar que el número de alambres a conectar entre ambas sea mínimo, entre todas las configuraciones obtenidas apartir de una función de costo definida para el problema (Hertz *et al.*, 1990).

La solución de estos problemas, dependen de la técnica de optimización para máximizar o minimizar la función de costo. Una técnica de optimización empleada para minimizar el porcentaje de error en el aprendizaje de redes tipo perceptrón, es el algoritmo de Descenso de Gradiente que emplea una función de costo para tratar de minimizar el error existente entre la salida obtenida de la red durante el paso de aprendizaje y la salida que deseamos obtener. Para mayores detalles de este algoritmo a continuación se describirá su funcionamiento.

II.2.2 Algoritmo de aprendizaje por Descenso de Gradiente

Al estudiar la regla de aprendizaje anterior, se consideró a $g(h) = sgn(h)$ como función de activación, la cual es una función discreta. Pero ahora, estudiaremos el caso de unidades con valores continuos, definidas por una función $g(h)$ continua y diferenciable en h. Una de las ventajas de la utilización de estas unidades, es la construcción de una función de costo $E[w]$, mediante la cual se mide el grado de la efectividad del sistema. Esto se logra por medio de una superficie de pendientes, dada por una función diferenciable en un espacio de pesos $w = \{\omega_{ij}\}$.

Una forma eficaz para llegar a los valores adecuados de los pesos, es usar la técnica por Descenso de Gradiente, la cual minimiza el error del sistema derivando la función de costo con respecto a los $\{\omega_{ij}\}$ de la red. La función de costo nos da una evaluación de la desviación existente entre el valor esperado en las unidades de salida y los valores obtenidos en la práctica. Al ir minimizando este error, la red va aprendiendo los patrones, es decir, las salidas obtenidas se van acercando a los valores esperados.

Para una RN cuya función de activación $g(h)$ es diferenciable, la respuesta de la red puede estar representada por la ecuación (3). La función de costo, utilizada para encontrar los pesos ω_{ik} de la red, se define por la siguiente expresión matemática:

$$E[\omega_{ik}] = \frac{1}{2}\sum_{i\mu}[\zeta_i^\mu - \theta_i^\mu]^2 \tag{15}$$

donde $\theta_i^\mu = g(\sum_k \omega_{ik}\xi_k^\mu)$, es el valor de la unidad i-ésima de salida cuando a la red

se le presenta el patrón de entrada $\{\xi_k^\mu\}$ durante el entrenamiento. De tal manera que podemos escribir lo anterior como:

$$E[\omega_{ik}] = \frac{1}{2}\sum_{i\mu}[\zeta_i^\mu - g(\sum_k \omega_{ik}\xi_k^\mu)]^2 \tag{16}$$

Con ayuda de la función de costo $E[\omega]$, obtenemos la desviación entre las salidas ζ_i^μ esperadas y las salidas obtenidas θ_i^μ, el propósito principal es que $E[\omega]$ adquiera su valor mínimo, es decir, se vaya acercando a cero.

Para lograr lo anterior, se modifica cada ω_{ik}, por una cantidad $\Delta\omega_{ik}$ proporcional al gradiente de E, en la configuración presente. A este tipo de aprendizaje se le llama Descenso de Gradiente.

Cabe notar que no existe una solución única, porque estamos hablando de una superficie de N dimensiones y la superficie puede presentar una serie de "cuencas", donde cada cuenca representa una posible solución, por lo tanto, se pueden encontrar muchas soluciones que a la vez satisfacen las condiciones requeridas.

Para minimizar el valor E, calculamos el gradiente con respecto a las conexiones ω_{ik}, obteniendo:

$$\Delta\omega_{ik} = -\eta\frac{\partial E}{\partial\omega_{ik}} = \eta\sum_{i\mu}[\zeta_i^\mu - \theta_i^\mu]g'(h_i^\mu)\xi_k^\mu \tag{17}$$

Donde el factor de aprendizaje η está multiplicado por un signo negativo, que nos acerca al mínimo buscado. Si el valor de η es demasiado pequeño, el acercamiento

hacia el mínimo es muy lento, sin embargo, si es demasiado grande puede llevar a oscilaciones que impidan la convergencia del algoritmo. Por otro lado, el valor ideal de η depende del problema particular a considerado.

La función de activación utilizada para este tipo de unidades, puede ser cualquier función que sea sigmoide y diferenciable; la función de activación más utilizada es $g(h) = tanh(\beta h)$, cuya derivada puede ser expresada en términos de sí misma, en esta expresión β nos indica la pendiente de la función de activación cuando h=0.

En el siguiente capítulo, describiremos con más detalle las redes de multicapas en donde se analizará un algoritmo de aprendizaje llamado Retropropagación.

III. Perceptrones de Multicapas

Como se mencionó en el capítulo anterior, las serias limitaciones que presenta el Perceptrón Simple, hicieron que se generalizara con la inclusión de una o más capas intermedias. En este capítulo estudiaremos este tipo de Perceptrón llamado, Perceptrón de Multicapas. Existe un algoritmo de aprendizaje llamado "Retropropagación", aplicado en redes de alimentación hacia adelante, el cual es utilizado generalmente en los Perceptrones de Multicapas. En este capítulo se analizará en que consiste este algoritmo y algunas aplicaciones. A continuación veremos la manera de resolver la función booleana O-Exclusivo (XOR), descrita en el capítulo anterior con la ayuda de una capa intermedia.

III.1 Solución del problema del O-Exclusivo (XOR)

La regla de aprendizaje de Retropropagación, es aplicada en redes compuestas de muchas capas. En este tipo de algoritmo de aprendizaje se suscita el fenómeno de retroalimentación, esto es, la necesidad de revisar los datos obtenidos y la situación en que se encuentra la red, para continuar con el siguiente paso.

El algoritmo se basa principalmente en el cambio de los pesos sinapticos ω_{pq} con la utilización de un conjunto de pares de patrones $\{\xi_k^\mu\}$ y $\{\zeta_k^\mu\}$, para ejemplificarlo utilizaremos una red compuesta de tres capas, en la forma en que explicaremos a continuación.

Para solucionar el problema del O-Exclusivo (XOR), mencionado en el capítulo anterior, se utiliza una RN formada de dos unidades en cada una de las capa de

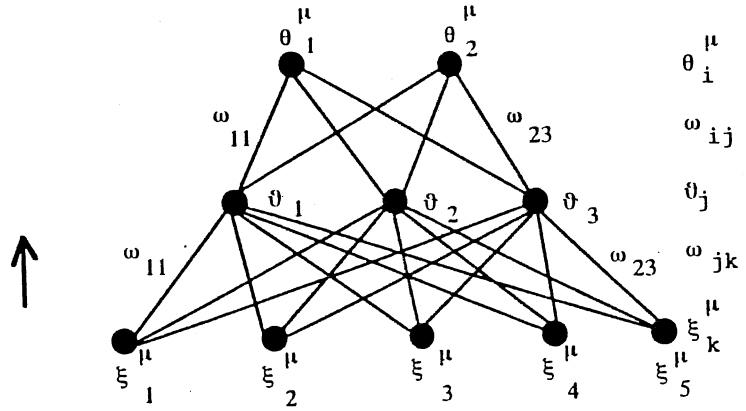


Figura 7: Red neuronal de 3 capas.

entrada e intermedia y una unidad en la capa de salida. Para este caso la notación utilizada será la siguiente: para las unidades de entrada utilizaremos $\{\xi_k^\mu\}$, los valores para las unidades de salida $\{\theta_i^\mu\}$ y las correspondientes a las capas intermedias estarán denotadas por ϑ_j. Las conexiones ω_{jk} son los pesos que van de las unidades de entrada a las unidades intermedias, y las ω_{ij}, van de las unidades intermedias a las unidades de salida (figura 7).

En la figura 7, ξ_k^μ representa el valor de la unidad k-ésima en el patrón μ-ésimo, tomando valores binarios (0/1, ó ± 1) o continuos. Para el caso de una RN compuesta de tres capas, su representación matemática, para cuando las unidades

intermedias j reciben a las unidades de entrada k es:

$$\vartheta_j^\mu = g(h_j^\mu) = g(\sum_k \omega_{jk}\xi_k^\mu) \tag{18}$$

Por otro lado, las unidades i de salida reciben las señales provenientes de las unidades j intermedias, para producir la salida final:

$$\theta_i^\mu = g(h_i^\mu) = g(\sum_j \omega_{ij}\vartheta_j^\mu), \tag{19}$$

de manera que θ_i^μ estará dada por la composición de funciones.

Para solucionar el problema de la función booleana O-Exclusivo (XOR), aplicaremos las ecuaciones anteriores, con una función de activación $g(h) = sgn(h)$, en el cual los valores para los patrones de entrada y salida son ± 1 ($\in \Re$). Una de las muchas soluciones posibles generada es: $\omega_{jk} = \omega, \vartheta_1 = \omega$, $\vartheta_2 = -\omega$, para la capa entrada - intermedia, $\omega_{11} = -2\omega$, $\omega_{12} = \omega$ y $\vartheta_i = 2\omega$, para la capa intermedia - salida. Dado que el problema del O-Exclusivo (XOR), es un problema simple, la solución puede encontrarse mediante el análisis de las ecuaciones, pero para problemas más complicados, esto no es posible. Para solucionar estos problemas requerimos el uso de algún algoritmo de aprendizaje.

Con la solución anterior obtenemos que las conexiones de la primera unidad de la capa intermedia son excitadoras, (figura 8) porque la respuesta proveniente

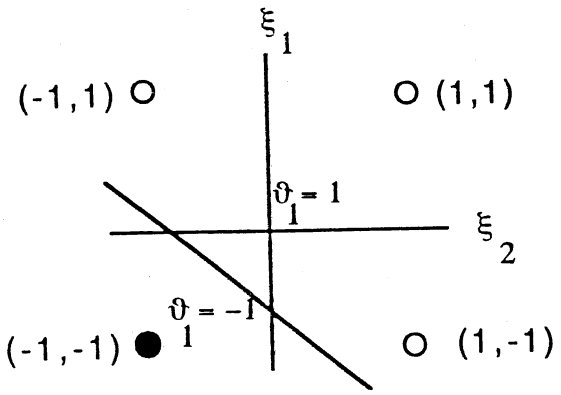


Figura 8: Estados de la primera unidad de la capa intermedia.

de los patrones de entrada es de 1 en tres de sus salidas. Sin embargo, en la segunda unidad de la capa intermedia, predomina en sus conexiones el poder de lo inhibitorio -1 (figura 9). Los valores de la capa intermedia, sirven de entrada para obtener una solución satisfactoria al problema del O-Exclusivo (XOR) (figura 10) (Touretzky, 1989).

III.2 Aprendizaje por Error de Retropropagación

El algoritmo por Error de Retropropagación fue desarrollado alrededor de 1985. Este algoritmo se basa en la generalización del método de Descenso de Gradiente, estudiado en el capítulo anterior para el Perceptrón Simple. A continuación, se explicará el algoritmo para una red formada de tres capas, donde sus valores de

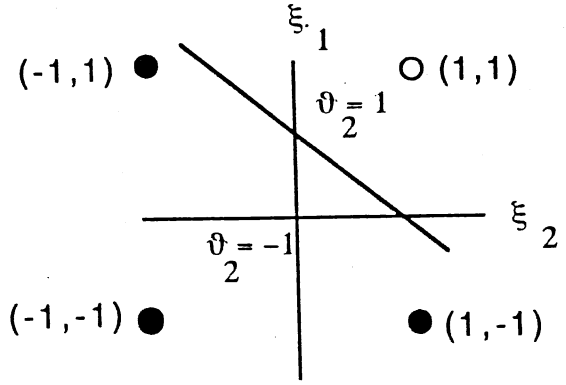


Figura 9: Estados de la segunda unidad de la capa intermedia.

entrada son continuos. Su función de costo está definida por la siguiente expresión matemática:

$$E[\omega] = \frac{1}{2} \sum_{\mu i} [\zeta_i^\mu - \theta_i^\mu]^2 \tag{20}$$

Dado que esta función es diferenciable, aplicamos la regla por Descenso de Gradiente, donde la idea principal es encontrar los mínimos de la suferficie definida por la ecuación (20), calculando el gradiente de E con respecto a los pesos, para buscar el incremento apropiado de cada uno. Como primer paso, calculamos los

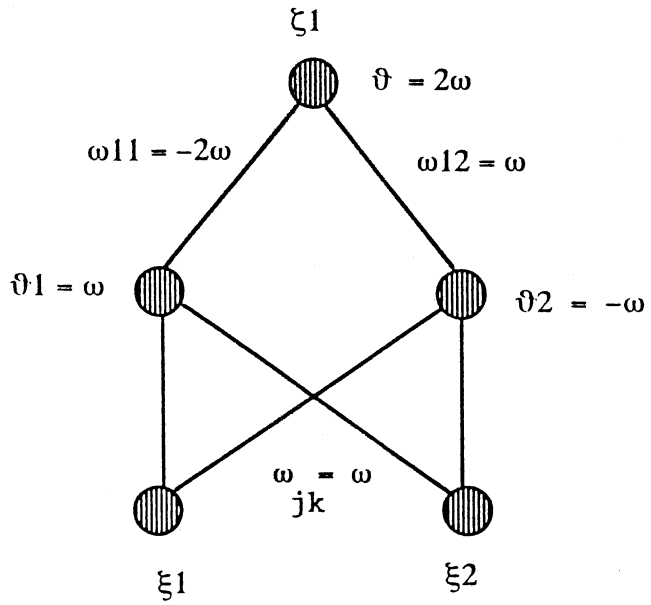


Figura 10: Solución de la Función booleana O-Exclusivo (XOR).

incrementos de las unidades de salida a las intermedias de la siguiente manera :

$$\Delta\omega_{ij} = -\eta\frac{\partial E}{\partial\omega_{ij}} = \eta\sum_{i\mu}[\zeta_i^\mu - \theta_i^\mu]g'(h_i^\mu)\vartheta_j^\mu, \tag{21}$$

donde $g'(h)$ es la derivada de g con respecto a su argumento, h_i^μ recibe las señales provenientes de la unidades intermedias j a las unidades de salida i (ecuación 19) y ϑ_j^μ recibe señales de las unidades de entrada k a las unidades de salida j (ecuación 18) .

El siguiente paso consiste en hacer el mismo cálculo, para el valor de las unidades intermedias como función de las señales recibidas por las unidades de entrada, utilizando:

$$\Delta\omega_{jk} = -\eta\frac{\partial E}{\partial\omega_{jk}} = \eta\sum_{i\mu}[\zeta_i^\mu - \theta_i^\mu]g'(h_i^\mu)\omega_{ij}g'(h_j^\mu)\xi_k^\mu. \tag{22}$$

Para este algoritmo, la función de activación puede ser la misma que se utilizó en el Perceptrón Simple , cuando se realizó el algoritmo de Descenso de Gradiente. Esta función es $g(h) = tanh(\beta h)$.

III.3 Error y Momentum

El algoritmo de Descenso de Gradiente en la ecuación (20), define el error existente, entre la salida real y la salida esperada. Un criterio para determinar la convergen-

cia del algoritmo, es dar un parámetro que sirve como cota del error, esto es, si el error total de todos los patrones es mayor que la cota propuesta, el algoritmo vuelve a calcular el error por medio de la función de costo y los incrementos de los pesos, en caso contrario decimos que el error total converge, consecuentemente la red aprendió.

Como ya mencionamos anteriormente, la convergencia del algoritmo de Descenso de Gradiente puede ser muy lenta, devido a que no se conoce el valor ideal de η para un problema específico. Por lo cual, se ha propuesto una modificación de este algoritmo (Rumelhart *et al.*, 1991) la cual consiste en añadir un término extra (α) llamado "momentum" a la prescripción para $\Delta\omega$. De esta manera, dos parámetros importantes requeridos para este algoritmo son: el factor de aprendizaje η, utilizado para determinar que tan grandes o pequeñas serán las correciones de los pesos y, el momentum α, para acelerar el paso de convergencia. Con lo cual $\Delta\omega$ quedará dado por la siguiente expresión matemática:

$$\Delta\omega(n) = -\eta\frac{\partial E}{\partial\omega} + \alpha\Delta\omega(n-1) \tag{23}$$

Donde n denota el número iteración, durante el paso de aprendizaje. Como podemos ver el término momentum es multiplicado por el incremento calculado de un paso anterior.

En algunas ocasiones, en la superficie donde aplicamos el método del gradiente, existen valles con pendientes muy pronunciadas. Cuando el valor de η, es muy grande, tardaríamos mucho en llegar al fondo del valle, debido a que se efectuarían

muchos " rebotes ", en contra de sus paredes, lo cual no garantiza encontrar un mínimo.

Para solucionar este problema, es necesario la intervención del parámetro α, la idea es multiplicar en cada conexión ω_{ij} por el momento, esto es, que pueda cambiar la dirección de la fuerza de descenso hacia el valle, de manera que las distancias sean incrementadas, consecuentemente el error converge más rápido, es decir, que la red aprenderá en menos tiempo.

Sin embargo, no existe ninguna receta para determinar los valores α y η, únicamente dependen de las condiciones de cada problema en particular, y de la experiencia personal del programador.

III.4 Ejemplos de Retropropagación

Este tipo de algoritmo se ha aplicado exitosamente en la solución de muchos problemas. Como ejemplos se tiene los siguientes:

III.4.1 NETtalk

La meta del proyecto NETtalk (red parlante) fue la de entrenar una red para pronunciar textos en inglés, realizado por Sejnowski y Rosenberg en 1987. La entrada de la red consiste en 7 caracteres consecutivos, presentados en una ventana que rastreaba el texto, la salida esperada, era un código de fonemas, que puede ser alimentado por un generador de voces.

La arquitectura empleada fue de 7 x 29 entradas, codificando 7 caracteres (incluyendo puntuación), 80 unidades intermedias y 26 unidades de salida, codificando fonemas, la red fue entrenada con 1024 palabras con sus respectivos fonemas, obteniendo una voz entendible.

III.4.2 Compresión de Imágenes

En los últimos años, se ha estado tratando de resolver el problema de transmitir grandes cantidades de información, con restricciones en el número de canales y tiempo de transmisión, a la vez minimizar el error que pudiera introducirse en el proceso transmisión-recepción. Uno de los problemas más comunes es en el manejo de imágenes, por ejemplo, en el televisor de alta definición debido a que no se puede transmitir información a cada pixel, hay que codificar esa información, transmitirla y decodificarla para desplegarla.

Para solucionar este problema se han utilizado las RN con aprendizaje supervisado, haciendo por ejemplo, iguales las entradas con las salidas (autoasociación), de manera que se pueda tomar la señal comprimida de la capa intermedia, obtenida del proceso de codificación. En las conexiones entre las unidades de entrada e intermedias se hace la codificación y la decodificación en las conexiones de las unidades intermedias a las de salida. Para llevar acabo este proceso, es necesario que el número de patrones sea igual al número de unidades de la capa de entrada y de salida. Así como, el número de unidades de la capa intermedia sea menor que el número de unidades del resto de las capas.

Este método fue desarrollado por Cottrell en 1987, la red que más estudiaron

tenía una entrada de una región de la imagen de 8 x 8 pixeles, con 16 unidades intermedias. Fue entrenada con Retropropagación, sobre regiones seleccionadas aleatoriamente de una imagen particular, con 150,000 pasos de entrenamiento. Los resultados fueron buenos, pero al utilizar imágenes muy variadas, el rendimiento bajo, aunque todavía fueron aceptables.

IV. Realización de una RN

El algoritmo más utilizado para aprendizaje en RN cuya arquitectura es de Multicapas es el algoritmo de Retropropagación, ya estudiado en capítulos anteriores. Aunque este algoritmo funciona bastante bien no está libre de problemas, ya que durante el proceso de aprendizaje, algunas veces el estado de la red se queda "atrapado" en algún mínimo y por lo tanto, no proporciona la solución requerida.

Un problema muy importante en el diseño de una red, es la determinación del número óptimo de unidades intermedias. Si el número de unidades utilizadas es muy pequeño, tenemos la posibilidad de que la red nunca aprenda, sin embargo, si es más grande de lo necesario, entonces el número de conexiones, el costo del sistema y la cantidad de dificultades técnicas aumentarían inútilmente. Un ejemplo extremo sería intentar resolver el problema del O-Exclusivo (XOR), descrito anteriormente, con 100 unidades en la capa intermedia.

En este trabajo se analizará un algoritmo propuesto por Yoshio Hirose (Hirose *et al.*, 1991), que consiste en cambiar el número de neuronas intermedias de la red dinámicamente (AMNUI, algoritmo para modificar el número de unidades intermedias). Cuando la red se atrapa en la búsqueda de un mínimo, una nueva neurona intermedia se aumenta. El proceso de aprendizaje sigue porque a la red se le aplica el algoritmo de retropropagación las veces requeridas, modificando constantemente los pesos de la misma. Una vez que se ha logrado encontrar un mínimo deseado, continúa la etapa de reducción de neuronas, la cual consiste en quitar neuronas de la capa intermedia para ver si se puede obtener un error pequeño

con menos neuronas. Este procedimiento se repite, acercándonos más al mínimo deseado. Si la red es de nuevo atrapada, en un estado con un error mayor al logrado anteriormente, el algoritmo desecha esta información y recupera el estado de la red antes de ser atrapada, lo cual proporciona el número óptimo de neuronas intermedias.

Para revisar la efectividad de este método, se realizó la simulación de redes neuronales para tres problemas. En estos problemas se varió el factor de aprendizaje (η), para analizar el comportamiento del sistema y poder determinar la cantidad óptima de neuronas para la capa intermedia con mayor efectividad.

La primera simulación consitió en diseñar una red que aprendiera la función booleana O-Exclusivo (XOR), donde el número de neuronas de entrada es 2, y de salida es 1 (Hirose *et al.*, 1991). La simulación se llevó acabo en una Computadora Personal (PC), 386 SX, 16 HMZ de velocidad, 1MB de memoria, monitor VGA y Sistema Operativo MS-DOS.

El segundo problema, consiste en diseñar un contador de las neuronas activas, o bits diferentes de cero, de los patrones de entrada; de manera que la red empleada consta de 4 neuronas de entrada y 5 neuronas de salida. En este caso, los patrones de entrada son los 16 números binarios que se pueden obtener como combinaciones de cuatro bits (0 ... 15) y sus patrones correspondientes de salida, los cuales se obtienen al sumar el número de neuronas activas de los patrones de entrada (Ooyen, 1992). Esto es, "cuenta" el número de bits diferentes de cero presentes en el número binario (n) y como respuesta se activa únicamente la n-ésima unidad del patrón de salida. De manera que si el patrón de entrada se encuentra con una neurona activa, en el patrón de salida se activará únicamente la primera neurona de salida, cuando

en el patrón de entrada se encuentran dos neuronas activas, en el patrón de salida se activará la neurona que se encuentra en la segunda posición, etc. En el caso de que las neuronas del patrón de entrada estén todas inactivas, lo cual corresponde el número cero en notación binaria, el estado de la quinta neurona del patrón de salida se activará. El equipo utilizado para la simulación fue diferente al anterior, debido a la necesidad de mayor memoria y rapidez de procesamiento que la de las PC's, utilizando un equipo Multiusuarios Hewlett Packard (HP9000/835S), de capacidad para 16 usuarios, 24MB de RAM, dos discos duros 307MB y Sistema Operativo UNIX.

El tercer problema que abordamos es un ejercicio de reconocimiento de patrones consistente en la identificación de caracteres alfanuméricos. Este problema es muy importante debido a que puede ser utilizado como un primer paso para la transformación de textos escritos a lenguaje oral, la copia de textos impresos hacia una computadora, etc. Este problema se dividió en dos partes, la primera parte consiste en entrenar a una RN con 10 patrones de entrada, correspondientes a las 10 primeras letras del abecedario (A ... J), para la n-ésima letra del alfabeto ($1 \leq n \leq 10$), el patrón de salida correspondiente será aquel en el cual únicamente se active la n-ésima neurona. La segunda parte consistió en realizar el mismo experimento, pero con 36 patrones de entrada, correspondientes a 26 letras del abecedario (A ... Z), más 10 digitos (0 ... 9), los patrones de salida son similares a los descritos en la primera parte. Cabe mencionar, que el número de patrones de entrada, en ambos ejercicios, es igual al número de neuronas de la capa de salida (Hirose *et al.*, 1991). En este caso se trabajó con un equipo multiusuarios, descrito en el problema anterior y en una estación de trabajo Hewlett Packard Apolo (HP9000/425t), con capacidad para dos usuarios, 20MB de RAM, un disco

duro de 200MB y Sistema Operativo UNIX .

IV.1. Descripción del algoritmo AMNUI

Para el aprendizaje de las redes a diseñar, se utilizó el algoritmo de Retropropagación. Como ya explicamos en capítulos anteriores, este algoritmo requiere de dos elementos: una función de costo y de una función de activación. La función de costo se define por la siguiente expresión matemática:

$$E[\omega] = \frac{1}{2}\sum_{\mu i}[\zeta_i^\mu - \theta_i^\mu]^2, \tag{24}$$

donde ζ_i^μ es el valor de la i-ésima unidad del patrón de salida esperado y θ_i^μ es la el valor de la unidad i-ésima de salida de la red cuando le presentamos la unidad i-ésima ξ_i^μ del patrón de entrada, las variables ζ_i^μ, θ_i^μ y ξ_i^μ pueden tomar valores continuos en un intervalo de [-1,1]. La función de activación está definida por la siguiente expresión matemática:

$$g(h) = \frac{1}{2}\{1 + tanh(\beta h)\} \tag{25}$$

donde tanh es la tangente hiperbólica, $h = \sum_j \omega_{ij} g(\sum_k \omega_{jk}\xi_k^\mu)$, la cual representa la suma de todas las señales provenientes de las neuronas de las capas de entrada e intermedias para dar la respuesta a las neuronas de salida. Siendo β el factor que da valor de g(h) cerca de h=0.

Como ya mencionamos anteriormente, el algoritmo de aprendizaje por Retropropagación, se basa en la búsqueda de mínimos, sin embargo el estado del sistema puede ser atrapado en un mínimo no global, es decir un minímo local y por lo tanto no deseado, del que no pueda salir. Una solución a este problema es la aumentar una nueva neurona en la capa intermedia la cual modifique la forma de la función de costo y pueda dar una salida hacia los mínimos más profundos. El error total (E) obtenido de la función de costo, se utiliza como criterio para establecer la condición de aumentar o disminuir neuronas en la capa intermedia.

La figura 11, muestra el diagrama de flujo del algoritmo AMNUI utilizado para la realización de las simulaciones de las redes neuronales. Como primer paso (1), se definen los datos iniciales de la RN, que consisten en el número de las neuronas de la capa de entrada y de la capa de salida, los patrones con los que se va a entrenar, y las matrices pesos $\{\omega_{ij}\}$ cuyas componentes se van escogiendo de manera aleatoria, dentro del intervalo de [-1,1] con la misma probabilidad y se calcula el error inicial utilizando la ecuación (24). La red es entrenada por medio del algoritmo de retropropagación, obteniendo el error E sobre todos los patrones (2), de acuerdo con la función de costo. Después de cada cálculo del error E se revisa si el sistema convergió esto es, si $E < \varepsilon$, siendo ε el parámetro de convergencia escogido. En caso de que el algoritmo no haya convergido aún (3), se repite el entrenamiento al menos de que el número de iteraciones sea un múltiplo de 100 (4). Si la cantidad de $\frac{(E(n)-E(n-100))}{E(n)}$ es menor del 1% (5), dado que $E > \varepsilon$, se considerará que el estado del sistema se encuentra atrapado en un mínimo local de la función de costo con un valor alto, por lo que se opta por aumentar una neurona intermedia (6). Por otro lado si $\frac{(E(n)-E(n-100))}{E(n)} < 1\%$ se continúa con el proceso de entrenamiento (2).

Una vez que E es menor al parámetro de convergencia (ε) (3), los pesos $\{\omega_{ij}\}$ se

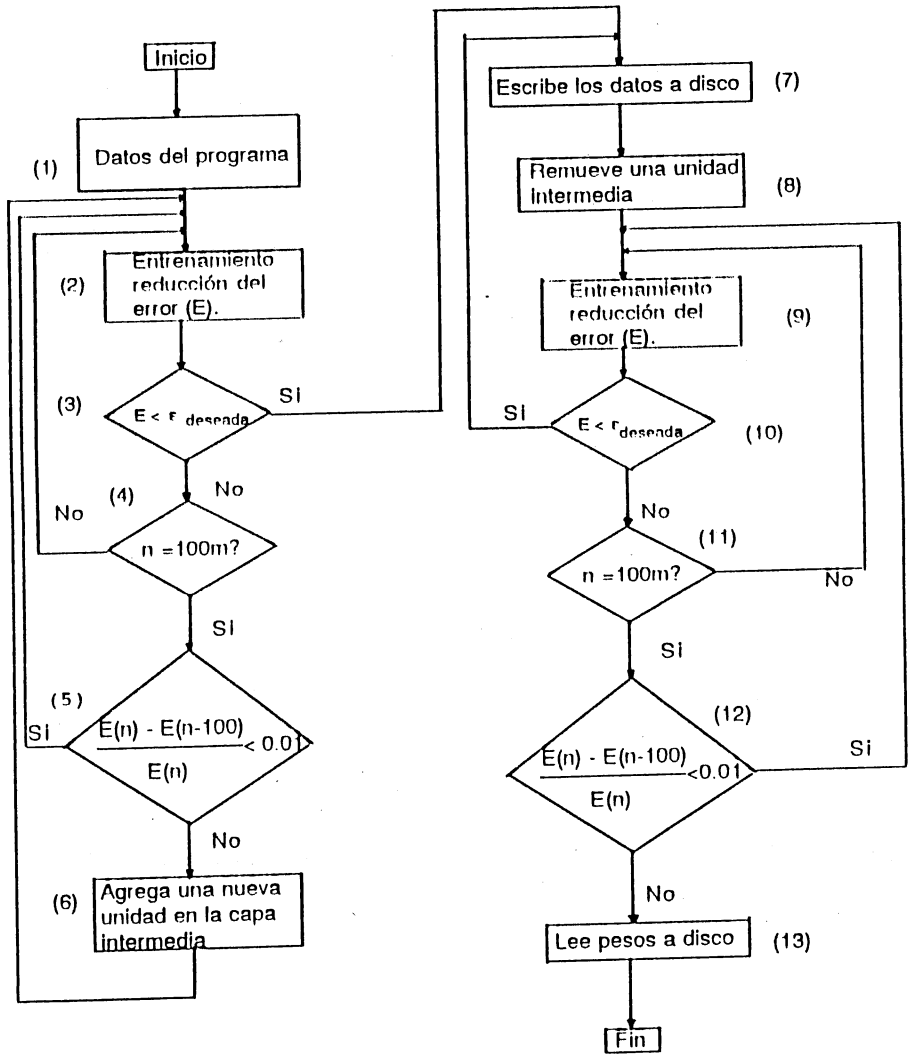


Figura 11: Diagrama de Flujo del algoritmo para encontrar el número óptimo de neuronas en la capa intermedia (AMNUI). Para mayor explicación ver texto.

guardan en un archivo de datos (7) y se remueve una neurona de la capa intermedia (8). Con ésto, el valor de E vuelve a aumentar, pero ahora con diferentes condiciones iniciales más cercanas al mínimo que estamos buscando y la red se entrena de nuevo (9). Este procedimiento se repite mientras que se continue reduciendo E (10). En caso de que el valor de E no se reduzca, este se revisa cada 100 correcciones de pesos (11), si el valor de $\frac{(E(n)-E(n-100))}{E(n)} < 1\%$ (12) se vuelven a corregir los pesos $\{\omega_{ij}\}$ (9), en caso contrario el algoritmo lee los pesos almacenados del archivo (13), y el algoritmo termina (Hirose *et al.*, 1991).

IV.2 Ejemplos.

En este trabajo se ejemplifica el algoritmo anterior, variando el factor de aprendizaje (η); en todos los problemas se escogieron los valores de $\beta = 0.5$ y $\alpha = 0.2$, debido a que se encontró que el algoritmo funcionaba satisfactoriamente para estos valores. Para graficar el error obtenido de la función de costo [ecuación (24)], se transformó a error porcentual $E\%$, es decir, el mayor error posible corresponde a 100 %. En el caso del problema del O-Exclusivo (XOR), en que los valores de los patrones están compuestos de números ± 1 reales, este error fue calculado por medio de la fórmula $E\% = \frac{E}{N*P} * 100$, donde N es el número de unidades de salida y P el número de patrones. Para el resto de los problemas, los valores de los patrones fueron 0.0's y 1.0's, en donde calculó el error máximo posible. Como ya sabemos, cada patrón de salida solo tenía una neurona activa y el resto inactivas. Dado que el máximo error obtenido para cada estado de la neurona es diferente, se tubieron que calcular por separado. Este error máximo fue obtenido de la función de costo antes definida, pero si elevarla al cuadrado, para ambos casos. Después

se realizó el cálculo de $E\%$ de manera similar al problema del O-Exclusivo (XOR), sustituyendo el nuevo valor en el lugar de $N * P$. Con el objeto de representar gráficamente los patrones utilizados para el entrenamiento de las redes simuladas, en las figuras se utilizaron círculos blancos para denotar a las neuronas activas y círculos oscuros para las neuronas inactivas.

IV.2.1 O-Exclusivo (XOR)

El primer problema en el cual aplicamos el algoritmo AMNUI descrito anteriormente fue el diseño de una red que funciona como el circuito lógico O-Exclusivo (XOR). Los patrones utilizados son los correspondientes a la tabla de valores de este problema mostrada en la figura 12, donde cabe recordar que se utilizaron valores reales ± 1 y se consideraron 9 valores para η en el intervalo de $\eta = [0.2, 2.3]$. El algoritmo completo fue aplicado para cada valor de η, obteniendo el número óptimo de neuronas de la capa intermedia y el número de iteraciones requeridas. Para todos los valores $\eta \leq 2$, se obtuvieron dos unidades en la capa intermedia, por otro lado, para $\eta > 2$, el error fue muy grande y su disminución muy lenta, ocasionando que tanto el número de iteraciones como el de neuronas en la capa intermedia creciera, sin llegar nunca a la convergencia del algoritmo ni a la etapa de reducción de las neuronas (figura 13). En la figura 14 se muestra el valor de $\eta = 2.0$, en donde el tiempo requerido para que la red aprendiera fue el mínimo obtenido durante los experimentos. Los números interiores muestran el número de neuronas de la capa intermedia durante la ejecución de esa etapa del algoritmo y las rayas continuas denotan el un intervalo en donde se aumentaron o se disminuyeron neuronas. Nótese como al añadir una nueva neurona durante la primer

Función booleana O-Exclusivo (XOR)	
Patrones Entrada	Patrones de salida
● ●	●
● ○	○
○ ●	○
○ ○	●

Figura 12: Conjunto de patrones que forman a la función booleana O-Exclusivo (XOR). Donde los círculos oscuros representan las neuronas inactivas (-1) y los círculos blancos las neuronas activas (1).

etapa del algoritmo, el error disminuye abruptamente. En la iteración 560 se realiza la primera reducción en la capa intermedia y la última reducción en la iteración 630, aumentándose el error nuevamente a poco menos del 50%. Cabe mencionar, que los datos mostrados en esta gráfica fueron adquiridos cada 10 iteraciones.

IV.2.2 Contador de neuronas activas.

El segundo problema con el cual se estudia el AMNUI ee un contador de neuronas activas en la capa de entrada. Los patrones de entrada para este problema, son los 16 números binarios que se pueden obtener de la combinación de cuatro bits. En este problema los valores de los patrones a aprender están constituidos por 0's y 1's reales (figura 15) y se utilizaron 9 valores para η situados en un intervalo

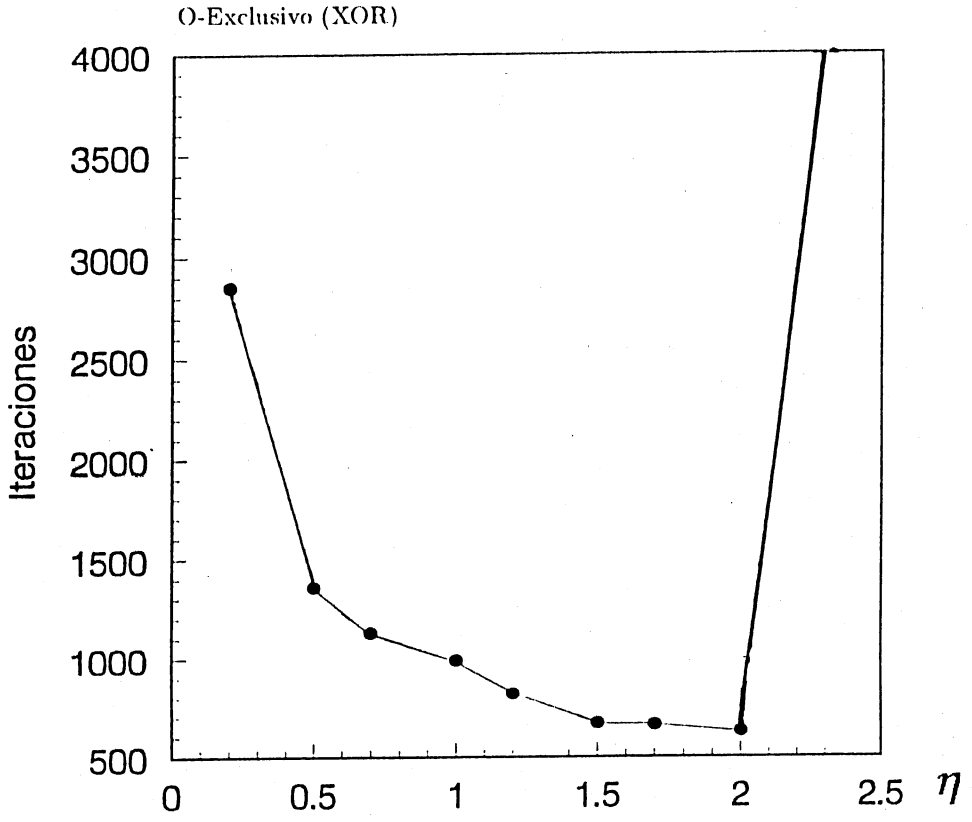


Figura 13: Función booleana O-Exclusivo (XOR). Esta gráfica muestra el número total de iteraciones desde el principio hasta el final del algoritmo como función del parámetro de aprendizaje η, con $\alpha = 0.2$ y $\beta = 0.5$. El número óptimo de unidades intermedias encontrado es 2. Como podemos ver, para valores $\eta > 2$ el algoritmo no converge.

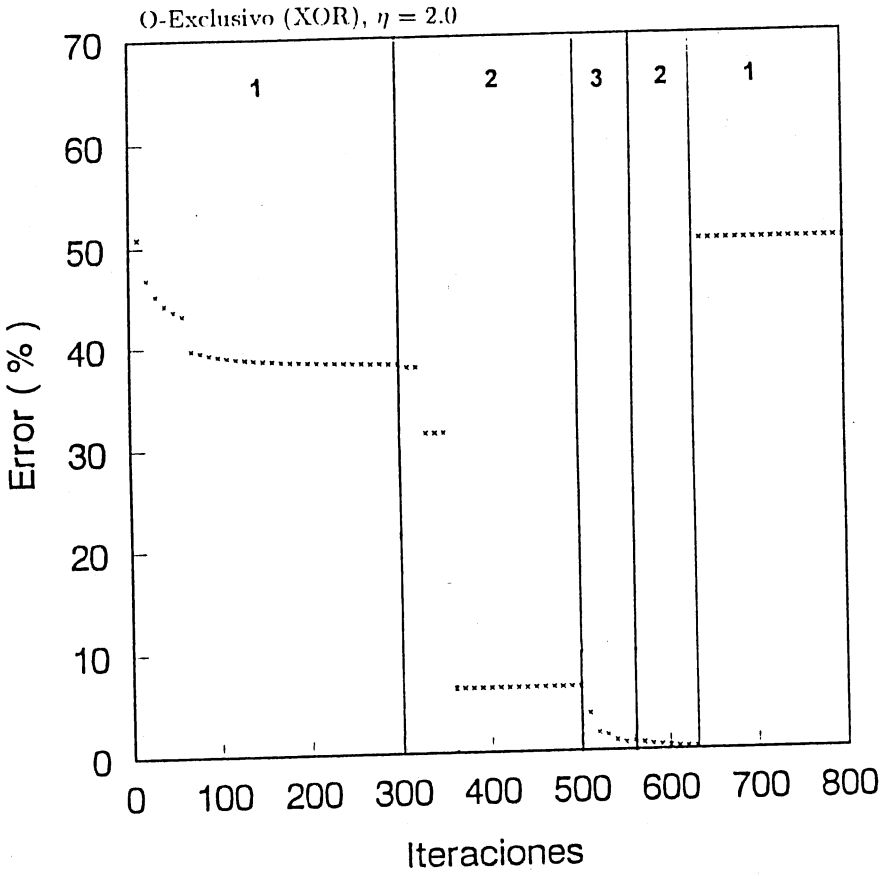


Figura 14: Función booleana O-Exclusivo (XOR). Esta gráfica muestra el valor del error porcentual (%) como función del número de iteraciones. Los números en el interior de la gráfica muestran el número de unidades intermedias de la red en el un intervalo denotado por líneas continuas. El valor de $\eta = 2.0$, $\alpha = 0.2$ y $\beta = 0.5$. Nótese como el error es casi el 50%, cuando la capa intermedia sólo tiene una neurona.

de $\eta = [0.1, 0.9]$. Al ejecutar la simulación, pudimos observar que para todos los valores de η el sistema convergió, es decir, la red aprendió, sin embargo, en la figura 16 podemos ver, que para valores $\eta < 0.1$, el sistema tiene posibilidades de que no exista convergencia, ya que el número de iteraciones es grande, lo que significa que el mínimo local de la función de costo requiera de más búsqueda. El número óptimo de neuronas intermedias obtenidas al variar η fue cuatro. En la figura 17a), se muestra una gráfica del error porcentual ($E\%$) en función del número de iteraciones, con $\eta = 0.8$,. Además se observa muy claramente cómo el sistema queda atrapado en estados con error alto y cómo el añadir una unidad en la capa intermedia contribuye a la disminución del error. Nótese que al principio del algoritmo el error disminuye rapidamente, ya que en la iteración 0 éste es del 18.89%, y para la iteración 10 que se encuentra en la figura, toma un valor de 9.90%. Dado que la etapa de reducción de neuronas de la capa intermedia, no puede ser observado claramente en la figura 17a), este un intervalo se amplificó en la figura 17b). Cabe notar que la etapa de reducción de neuronas comenzó apartir de la iteración 2385 y terminó en la iteración 2398, quedando finalmente cuatro neuronas en la capa intermedia. En la figura 17 se ve claramente que para $\eta = 0.8$ el sistema muestra oscilaciones en el error obtenido de la función de costo, con el objeto de reducir dichas oscilaciones se utilizó un valor de η menor ($\eta = 0.7$), los resultados obtenidos se muestran en la figura 18a). El intervalo en donde se encuentra la etapa de reducción se amplificó en la figura 18b). Los datos presentados en las figuras anteriores, fueron adquiridos cada 10 iteraciones del algoritmo.

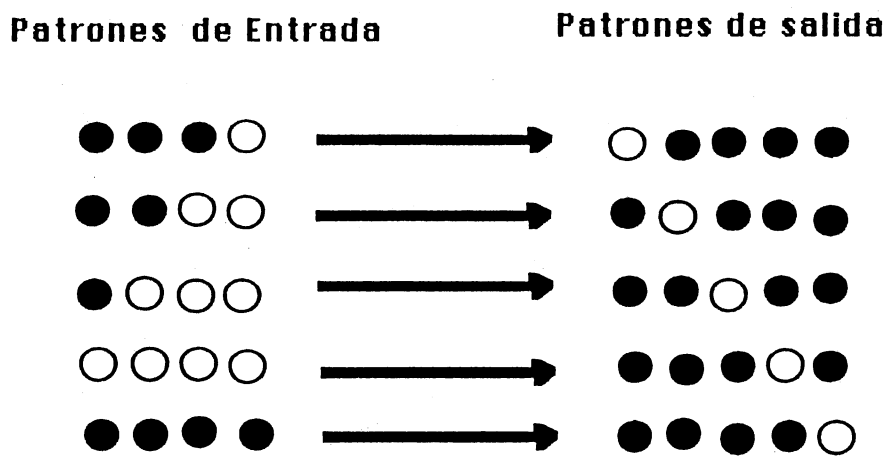


Figura 15: Ejemplos de patrones utilizados para entrenar a la RN, para el problema del contador de neuronas. Si hay n neuronas activas se activará la n-ésima neurona del patrón de salida. En el caso de que ninguna neurona este activa se activará la neurona colocada en la quinta posición del patrón de salida. Los círculos oscuros representan la neuronas inactivas, y los círculos blancos neuronas activas.

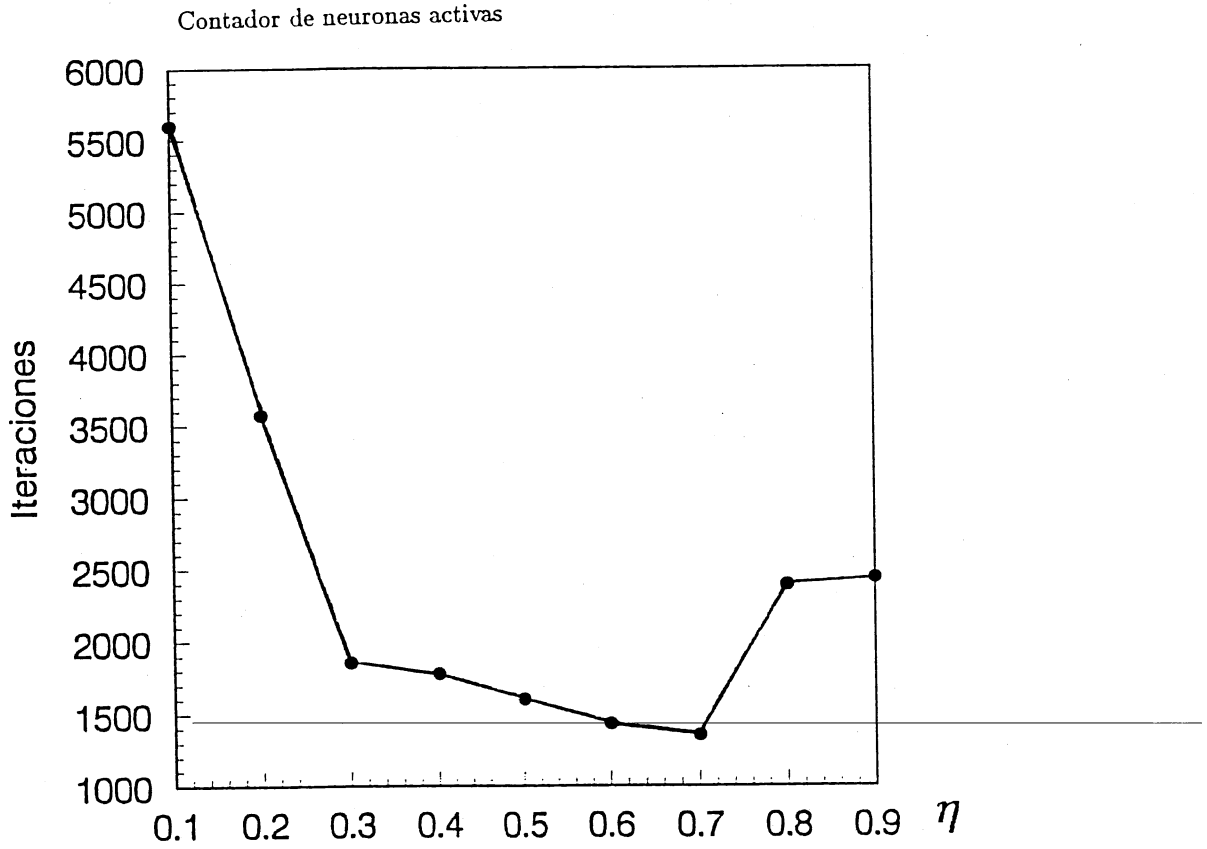


Figura 16: Contador de neuronas activas. Esta gráfica muestra el número total de iteraciones desde el principio hasta el final del algoritmo como función de parámetro de aprendizaje η, con $\alpha = 0.2$ y $\beta = 0.5$. El número óptimo de unidades intermedias encontrado es de 4. Como podemos ver, para todos los valores η el algoritmo converge.

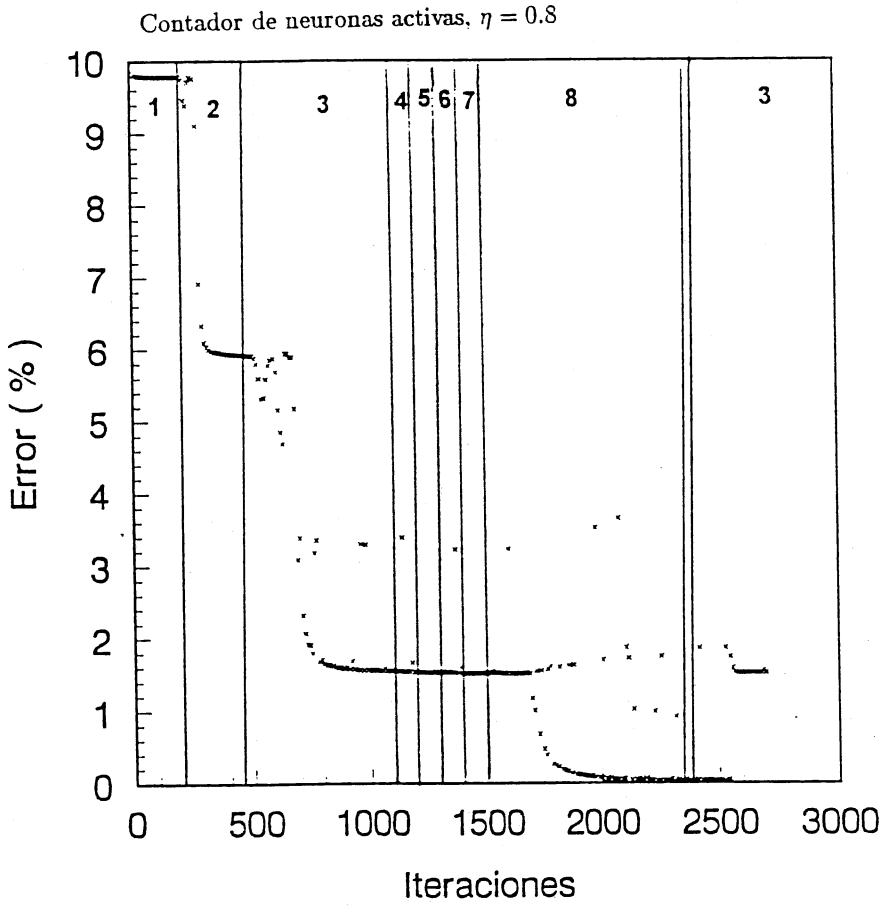


Figura 17: a) Contador de neuronas activas. Esta gráfica muestra el valor del error porcentual (%) como función del número de iteraciones. Los números en el interior de la gráfica muestran el número de unidades intermedias de la red en el intervalo denotado por líneas continuas. El valor de $\eta = 0.8$, $\alpha = 0.2$ y $\beta = 0.5$. En la gráfica 17b) se amplifica la región de la etapa de reducción del algoritmo.

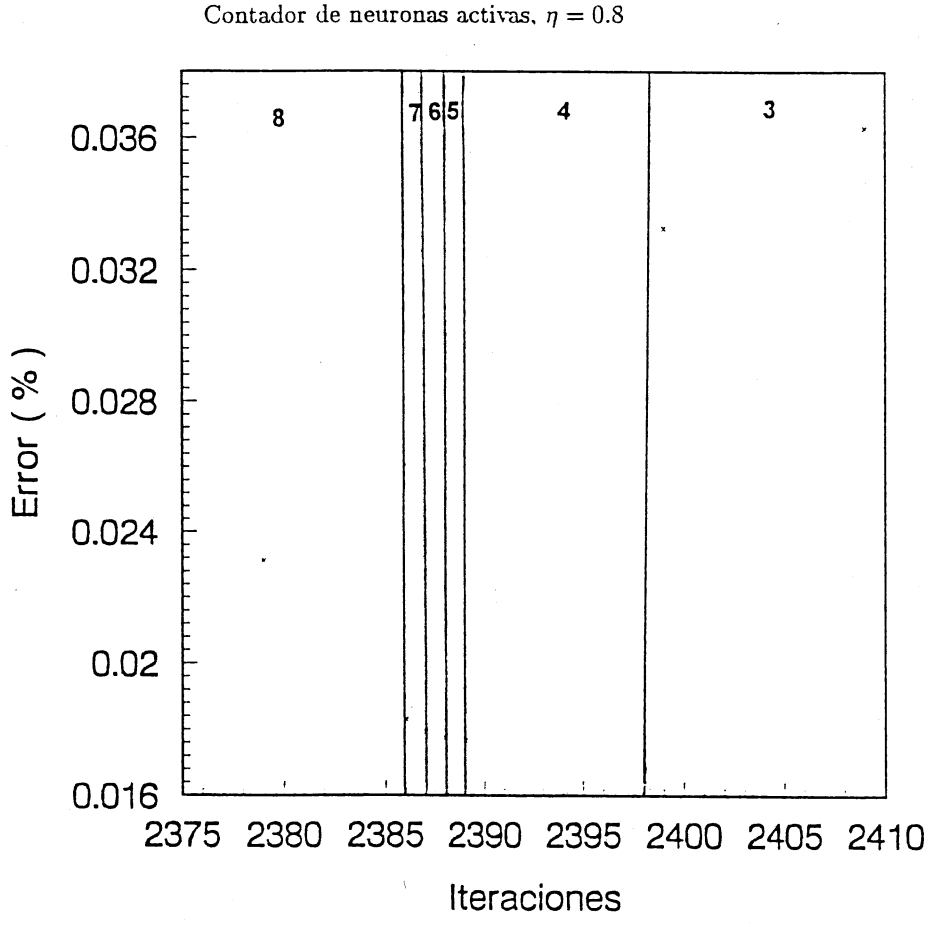


Figura 17: b) Contador de neuronas activas. Esta gráfica muestra el valor del error porcentual (%) como función del número de iteraciones. Los números en el interior de la gráfica muestran el número de unidades de la capa intermedia conforme avanzó la etapa de reducción del algoritmo. El intervalo denotado por líneas continuas es la cantidad de iteraciones en donde el número de unidades no cambió. Esta gráfica es una amplificación de la figura 17a), con un valor de $\eta = 0.8$, $\alpha = 0.2$ y $\beta = 0.5$.

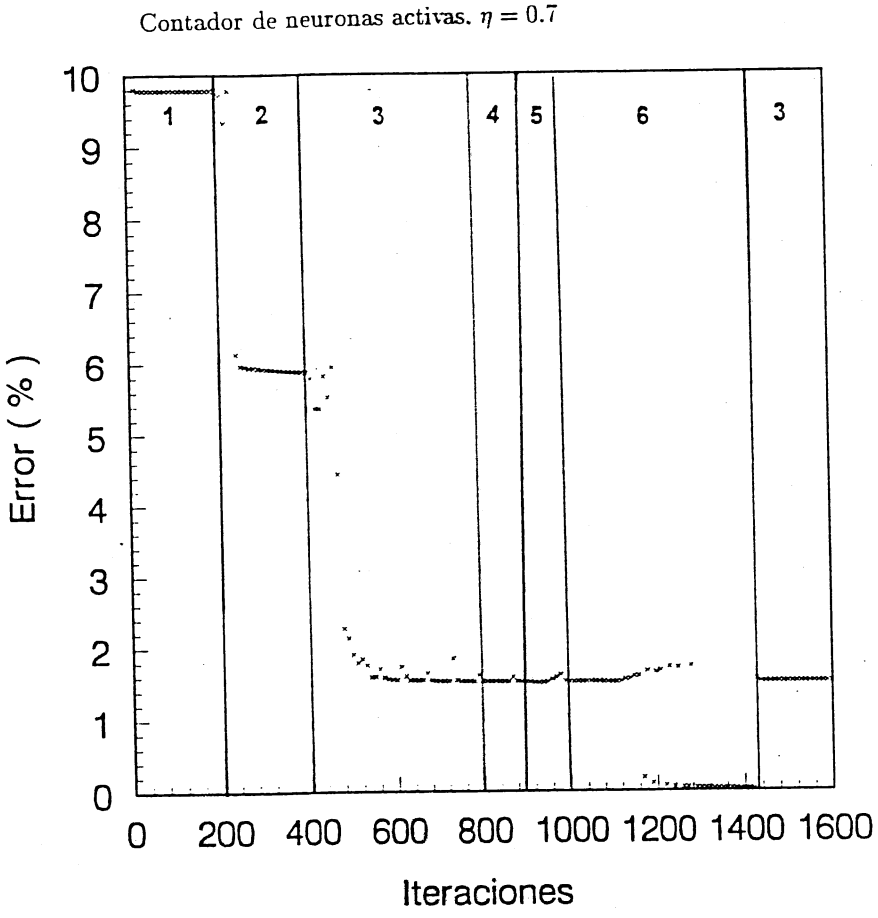


Figura 18: a) Contador de neuronas activas. Esta gráfica muestra el valor del error porcentual (%) como función del número de iteraciones. Los números en el interior de la gráfica muestran el número de unidades intermedias de la red en el intervalo denotado por líneas continuas. El valor de $\eta = 0.7$, $\alpha = 0.2$ y $\beta = 0.5$. En la gráfica 18b) se amplifica la región de la etapa de reducción del algoritmo, ya que ella existen varias reducciónes en la capa intermedia.

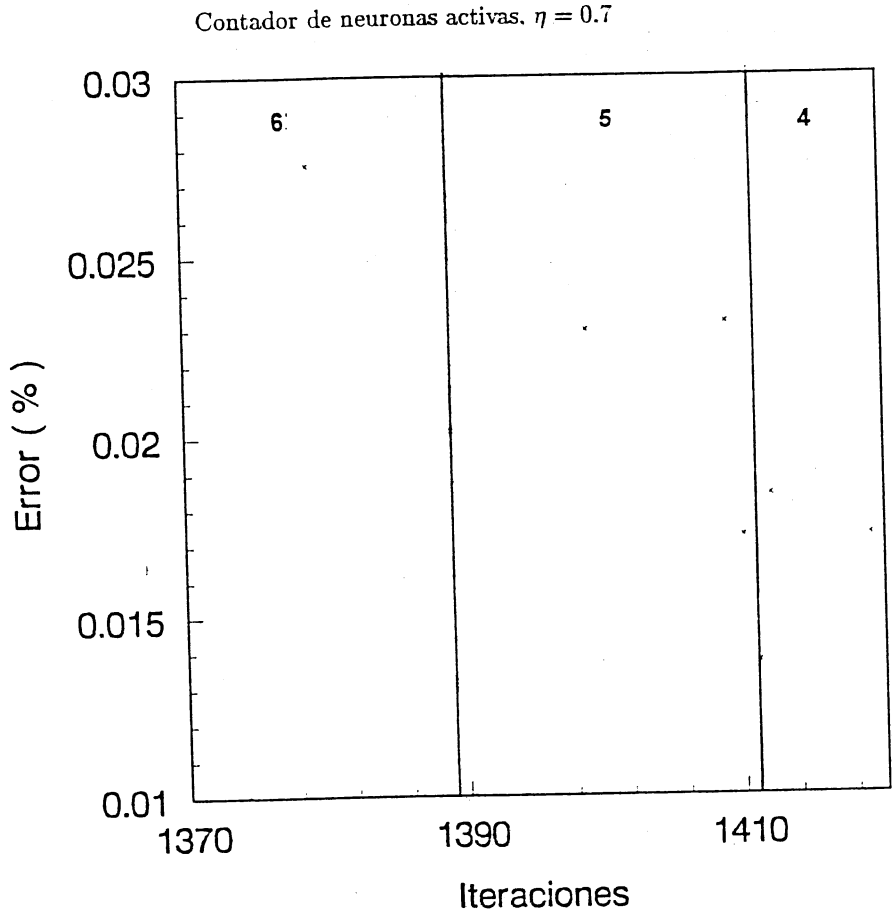


Figura 18: b) Contador de neuronas activas. Esta gráfica muestra el valor del error porcentual (%) como función del número de iteraciones. Los números en el interior de la gráfica muestran el número de unidades intermedias conforme avanzó la etapa de reducción del algoritmo. El intervalo denotado por líneas continuas es la cantidad de iteraciones en donde el número de unidades no cambió. Esta gráfica es una amplificación de la figura 18 a), con un valor de $\eta = 0.7$, $\alpha = 0.2$ y $\beta = 0.5$.

IV.2.3 Patrones alfanuméricos

La última simulación, se divide en dos partes, una de ellas consiste en entrenar a la red para reconocer las 10 primeras letras del abecedario $(A \ldots J)$ como patrones de entrada y la segunda parte, con 36 patrones compuestos por caracteres alfanuméricos. Los patrones a aprender están constituidos por valores 0's y 1's reales, sin embargo, se espera que la red generalice y reconozca letras escritas en tonos de grises. En la figura 19, podemos ver tres patrones de entrada y su correspondientes patrones de salida, donde la neurona activa del patrón de salida, indica el orden alfabético de la letra en el patrón de entrada. Lógicamente, el número de neuronas en el patrón de salida debe ser igual al número de patrones de entrada para ambas partes.

El algoritmo se modificó para las dos partes de la simulación. El valor de E se revisó cada 50 iteraciones en lugar de cada 100 (pasos 4 y 11 en el diagrama de flujo) y, el parámetro del 1% (pasos 5 y 12) utlizando como criterio, necesario para aumentar neuronas en la capa intermedia y para terminar el algoritmo, fue reemplazado por el valor de 0.5%.

Para el problema de las diez letras, se utilizaron 9 valores para η en el intervalo de [0.1,0.9]. Al realizar el entrenamiento con los diez patrones de letras, en la figura 20 se observa que para valores de $\eta > 0.7$ el algoritmo no convergió, es decir, las neuronas de la capa intermedia aumentaron enormemente sin llegar a reducir el valor del error E obtenido de la función de costo. Sin embargo, con el resto de los valores se obtuvo que el número óptimo de neuronas en la capa intermedia es 5. En la figura 21 muestra una gráfica del error porcentual $E\%$

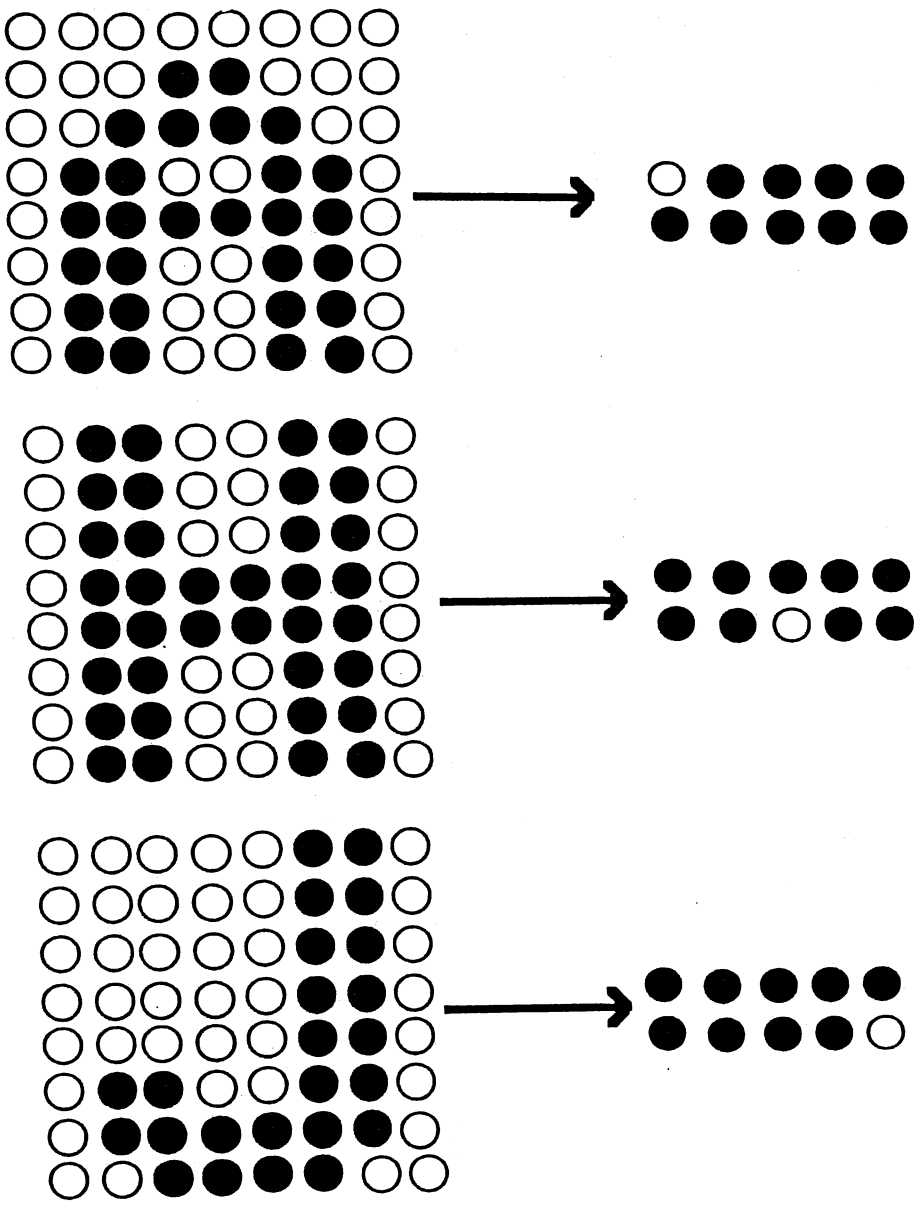


Figura 19: Diez letras. En los dibujos se muestra tres ejemplos de un patrón de entrada y su correspondiente patrón de salida, requeridos para entrenar a la RN. El entrenamiento consiste en que la red aprenda 10 patrones de entrada compuestos de las 10 primeras letras del abecedario (A ...J). El patrón de salida consta únicamente de una neurona activa y el resto inactivas, en donde la posición de la unidad activa (de arriba hacia abajo, y de izquierda a derecha) dependerá del orden alfabético de cada patrón de entrada.

en función del número de iteraciones para $\eta = 0.5$, éste valor no es el de menor número de iteraciones mostrado en la figura anterior, pero representa claramente el comportamiento del sistema. Los números que se encuentran dentro de los intervalos de las líneas continuas son el número de neuronas en la capa intermedia. Dados los resultados, tenemos que en la iteración 0 en $E\%$ fue igual a 35.52%, dada la eficiencia del algoritmo bajo el error a 27.2% para la siguiente iteración y a 8.27% diez iteraciones después. También podemos observar que al ir añadiendo una nueva neurona el error disminuye rápidamente, como en el caso de los intervalos donde se encuentran 3 y 5 neuronas en la capa intermedia. Cabe notar, que solo se ejecutó una reducción durante el algoritmo, proporcionando finalmente cinco unidades en la capa intermedia. Los datos graficados fueron adquiridos cada 10 iteraciones del algoritmo.

Finalmente la segunda parte de la tercera simulación requirió de más tiempo de procesamiento, ya que el número de patrones de entrada y el número de neuronas de salida fueron de 36. Este problema se trabajó con 11 valores para η en el intervalo de [0.01,0.6], sin embargo, en la figura 22 se observa que solo para valores de $0.03 \leq \eta \leq 0.2$ el algoritmo convergió, y que para el resto de los valores que no se encuentran dentro de la figura, sistema no convergió. El número óptimo de neuronas obtenidas al variar η fue de 6 y en algunos caso de 7. En la figura 23, se graficó el $E\%$ en función al número de iteraciones, para un valor de $\eta = 0.2$, como se puede obsevar en la figura anterior para éste valor no se obtuvo el número mínimo de iteraciones. pero con él podemos visualizar el comportamiento del sistema. Nótese que el error se va reduciendo conforme el número de neuronas en la capa intermedia crece, y cómo el $E\%$ se va reduciendo conforme el algoritmo itera. Tenemos que en la primer iteración el $E\%$ ed igual al36.59%, en la siguiente

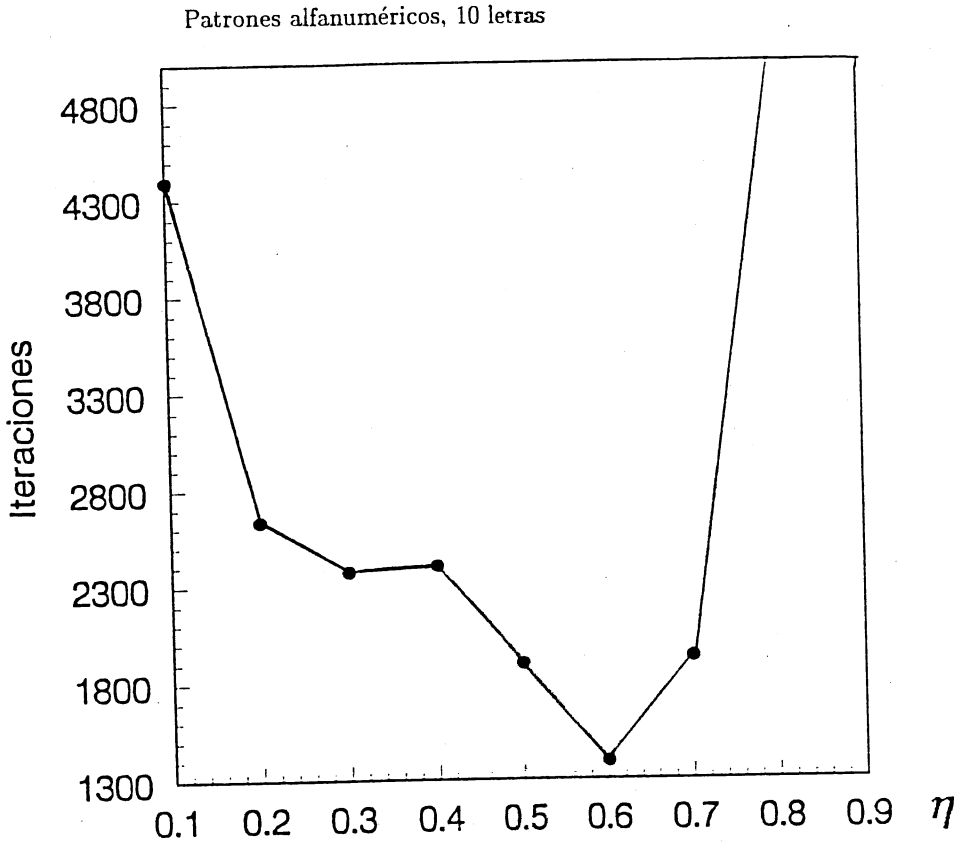


Figura 20: Diez letras. Esta gráfica muestra el número total de iteraciones desde el principio hasta el final del algoritmo como función de parámetro de aprendizaje η, donde $\alpha = 0.2$ y $\beta = 0.5$. El número óptimo de unidades intermedias encontrado es de 5. Los valores de $\eta > 0.7$, que no se aprecian en la gráfica el algoritmo no converge.

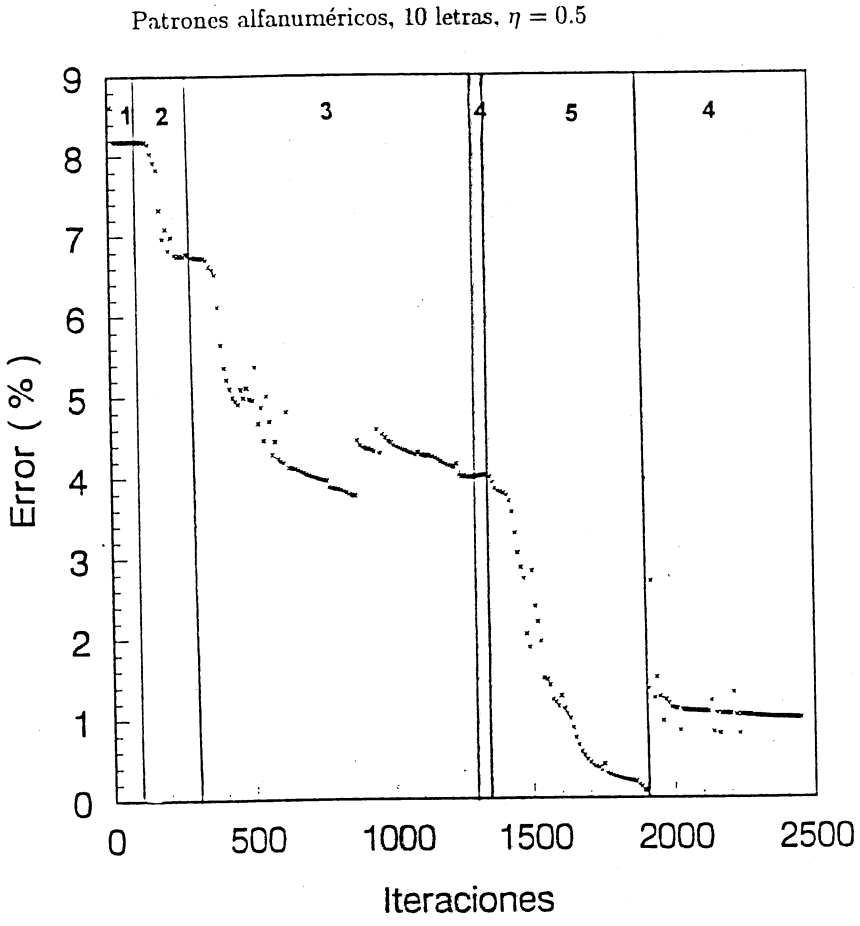


Figura 21: Diez letras. Esta gráfica muestra el valor del error porcentual (%) como función del número de iteraciones. Los números en el interior de la gráfica muestran el número de unidades intermedias de la red en el intervalo denotado por líneas continuas. El valor de $\eta = 0.5$, $\alpha = 0.2$ y $\beta = 0.5$. Como podemos ver, sólo se presenta una reducción en la capa intermedia con la que finaliza el algoritmo.

iteración este error disminuyó a 21.47% y en la décima tenía un valor de 2.72%, llegando al mínimo en la iteración 2649 con 6 neuronas en la capa intermedia, sólo se disminuyó una neurona. Los datos representados en esta figura, fueron adquiridos cada 50 iteraciones del algoritmo.

Como se observó, con la utilización del algoritmo AMNUI podemos darnos una idea de las condiciones necesarias para obtener una solución a diferentes problemas que se pretenden construir con este tipo de arquitectura de Redes Neuronales. Como mencionamos anteriormente, este algoritmo nos ayuda a encontrar el número óptimo de unidades intermedia. Cuando decimos el número óptimo, solo nos referimos a obtener una mejor manera de estimar el número de unidades de la capa intermedia, con las que la red pueda se entrenada para obtener una solución, con un número mínimo de conexiones entre sus neuronas.

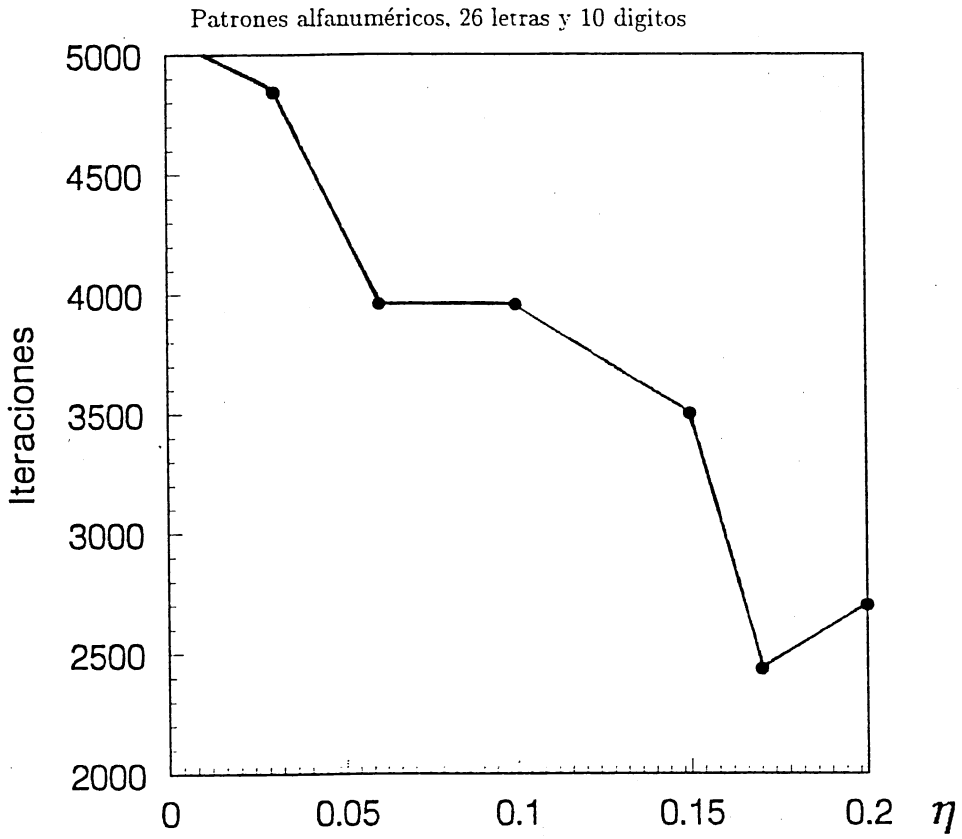


Figura 22: Patrones alfanuméricos. Esta gráfica muestra el número total de iteraciones desde el principio hasta el final del algoritmo como función de parámetro de aprendizaje η, donde $\alpha = 0.2$ y $\beta = 0.5$. El número óptimo de unidades intermedias encontrado es de 6 y en algunos casos de 7. Para valores de $\eta > 0.2$ y $\eta < 0.03$ el algoritmo no converge.

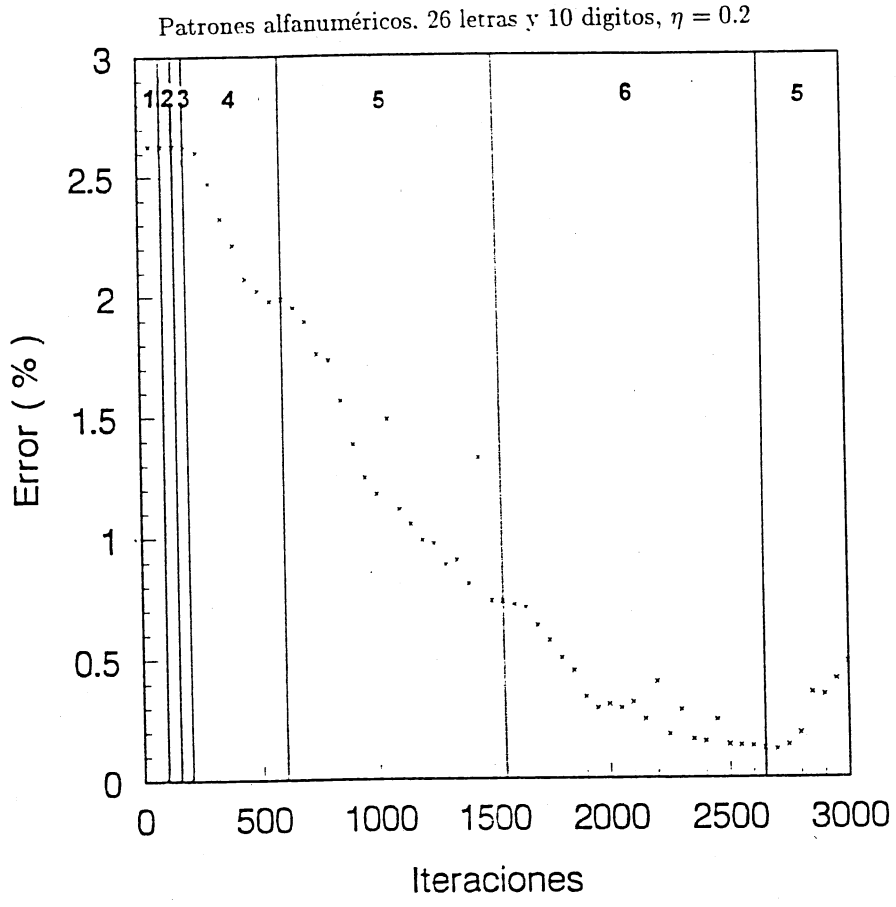


Figura 23: Patrones alfanumericos. Esta gráfica muestra el valor del error porcentual (%) como función del número de iteraciones. Los números en el interior de la gráfica muestran el número de unidades intermedias de la red en el intervalo denotado por líneas continuas. El valor de $\eta = 0.2$, $\alpha = 0.2$ y $\beta = 0.5$. Como podemos ver, sólo presenta una reducción de la capa intermedia con la que finaliza el algoritmo.

V. Desempeño de una RN entrenada

En los capítulos anteriores, se analizó cómo se realizaba el entrenamiento de una RN, describiendo los algoritmos de Retropropagación y AMNUI, siendo este último necesario para encontrar el número óptimo de unidades intermedias. Una vez que la red es entrenada, se espera que la red generalice, ésto es, que sea capaz de reconocer, un conjunto de valores de entrada que se asemejen a cualquiera de los patrones aprendidos.

Para validar el funcionamiento de la RN, es necesario la utilización de matrices de pesos $(\omega)'s$, obtenidas al finalizar el entrenamiento. En el caso de un RN compuesta por 3 capas, las $\{\delta_k^\mu\}$ son los valores para las unidades de entrada, (en lugar de $\{\xi_k^\mu\}$ denotadas en los capítulos anteriores, porque la idea es que la red entrenada pueda se validada con un conjunto de estados de entrada diferentes), los valores para las unidades de salida $\{\theta_i^\mu\}$ y las correspondientes a la capa de intermedia estarán denotadas por ϑ_j^μ. El procedimiento consiste en substituir los valores de los pesos (ω) en las siguientes ecuaciones:

$$\vartheta_j^\mu = g(h_j^\mu) = g(\sum_k \omega_{jk}\delta_k^\mu) \tag{26}$$

donde las unidades intermedias j reciben a las unidades de entrada k. Por otro lado, las unidades i de salida reciben las señales provenientes de las unidades j

intermedias para producir la salida, por medio de la ecuación:

$$\theta_i^\mu = g(h_i^m u) = g(\sum_j \omega_{ij} \vartheta_j^\mu) \tag{27}$$

donde $g(h)$ es la función de activación.

Para ejemplificar el funcionamiento de las ecuaciones anteriores, se validó la RN utilizada para reconocer las 10 letras del abecedario $(A \ldots J)$, donde a la n-ésima letra del alfabeto le corresponde como patrón de salida aquel en el cual únicamente se active la n-ésima neurona. Cada letra está compuesta por un arreglo de 8x8, es decir, 64 unidades de entrada, 5 unidades en la capa intermedia y 10 en la capa de salida, siendo los valores de los patrones 0's y 1's reales, para $\eta = 0.3$ y $\beta = 0.5$.

Se analizó el comportamiento del sistema con varias letras. Para representar los valores de los patrones de salida obtenidos, se colocaron dentro de las unidades de salida, debido a que no se contaba con sufientes tonos de grises para ejemplificar la diversidad de decimales obtenidos. La RN se verificó con todas las letras que había sido entrenada. En la figura 24 muestra la letra B como patrón de entrada y los valores obtenidos como salida de la red. Nótese que en la segunda unidad de salida cuyo valor es de 0.951 es muy cercano a uno, lo que significa que la neurona deseada está activada y el resto de los valores se acercan a cero. Este resultado fue similar para los patrones restantes, es decir la red aprendió el conjunto de datos utilizados para su entrenamiento. Como era de esperarse, para el caso de que el estado inicial de entrada no se parecia a algún patrón definido, la red no fue capaz de reconocerlo, como sucedió en el caso de la letra Z mostrado en la figura 25.

Otro caso importante que analizamos fue como responde el sistema cuando se le

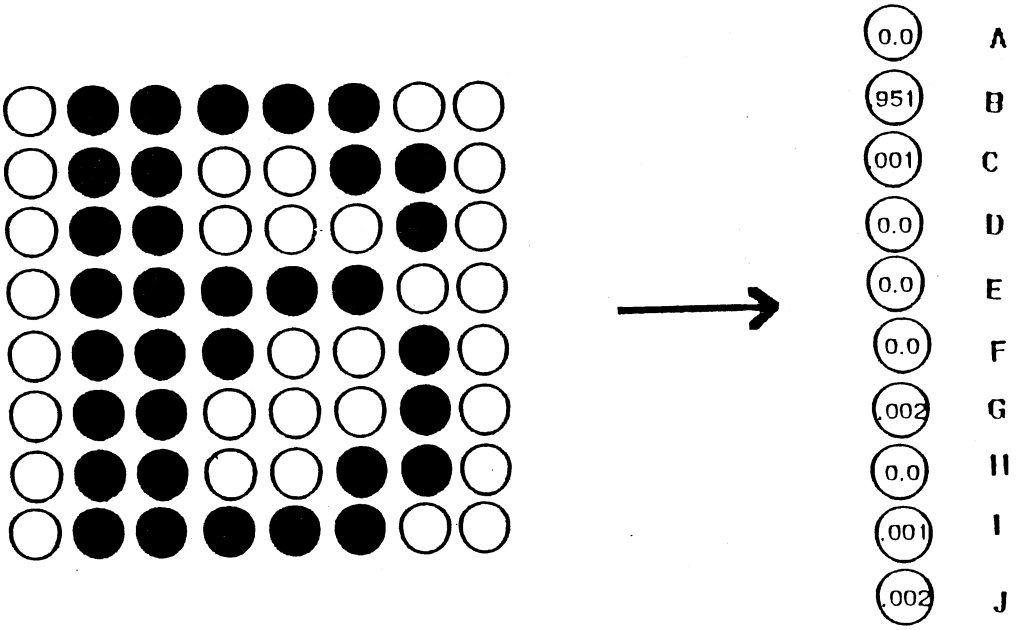


Figura 24: Diez letras. Ejemplo de un estado inicial de entrada, compuesto de la letra B, donde en su correspondiente salida se activará únicamente la segunda unidad, de arriba hacia abajo. Como podemos observar, la red aprendió la letra, ya que fue una de las utilizadas para el entrenamiento. Cabe notar que los valores de 0.0, fueron calculados hasta 10^{-3}.

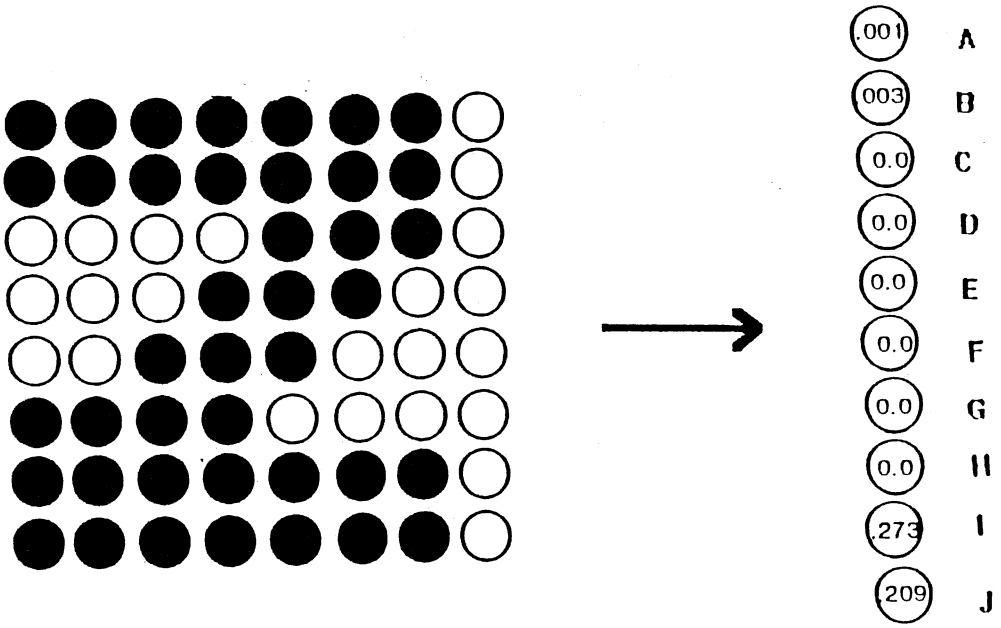


Figura 25: Diez letras. Ejemplo de un estado inicial de entrada, compuesto de la letra Z, donde la salida obtenida no se asemeja a ningún patrón de salida previamente definido. Como podemos observar, la red no aprendió a reconocer letras que no se parecian a los patrones de entrada con los que se habia entrenado. Cabe notar que los valores de 0.0, fueron calculados hasta 10^{-3}.

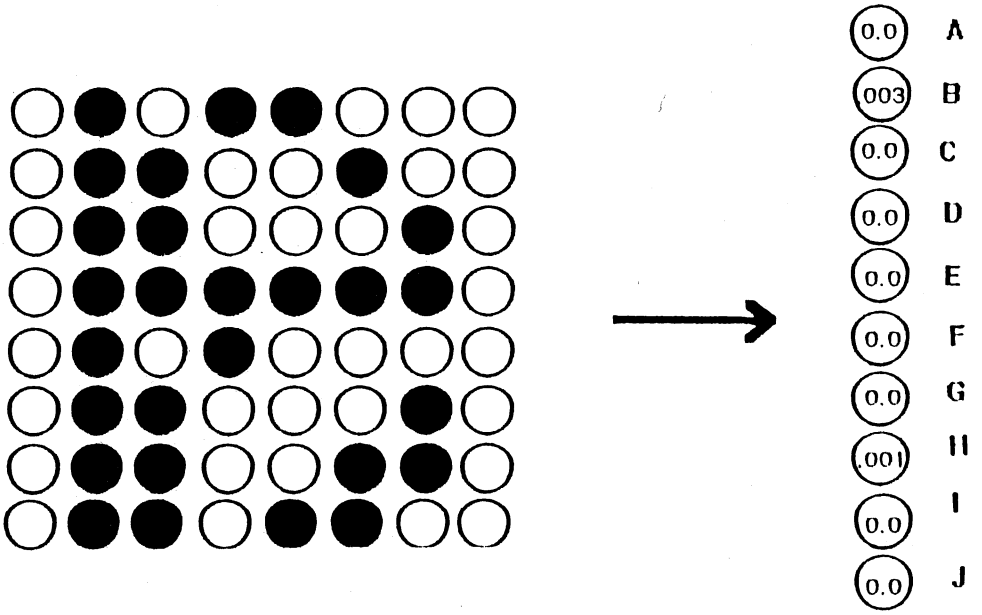


Figura 26: Diez letras. Ejemplo de un estado inicial de entrada, compuesto de la letra B, donde 6 unidades de patrón de entrada original fueron cambiadas de ceros (círculos negros) a unos (círculos blancos). Como podemos observar los valores obtenidos de salida no son los deseados, ya que la segunda unidad de salida (de arriba hacia abajo) no obtuvo un buen valor para que se activará. Nótese que solo la octava y segunda posición contiene valores altos, dado a los resultados podemos decir que la red no reconoció la letra. Los valores de 0.0, fueron calculados hasta 10^{-3}.

proporciona a la red letras borrosas. Una de las letras a modificar fue la letra B mostrada en la figura 26, donde los valores de 6 unidades cuyos valores eran de cero (círculos negros) fueron cambiados por unos (círculos blancos). Para este caso, todos los valores de las unidades de salida, fueron todos cercanos a cero, es decir, la red no reconoció la letra. Más sin embargo, cambiando 3 unidades de la misma letra (figura 27), los resultados fueron satisfactorios, ya que la salida deseada se obtuvo, con un valor de 0.941 en la segunda unidad de salida y el resto de los valores cercanos a ceros.

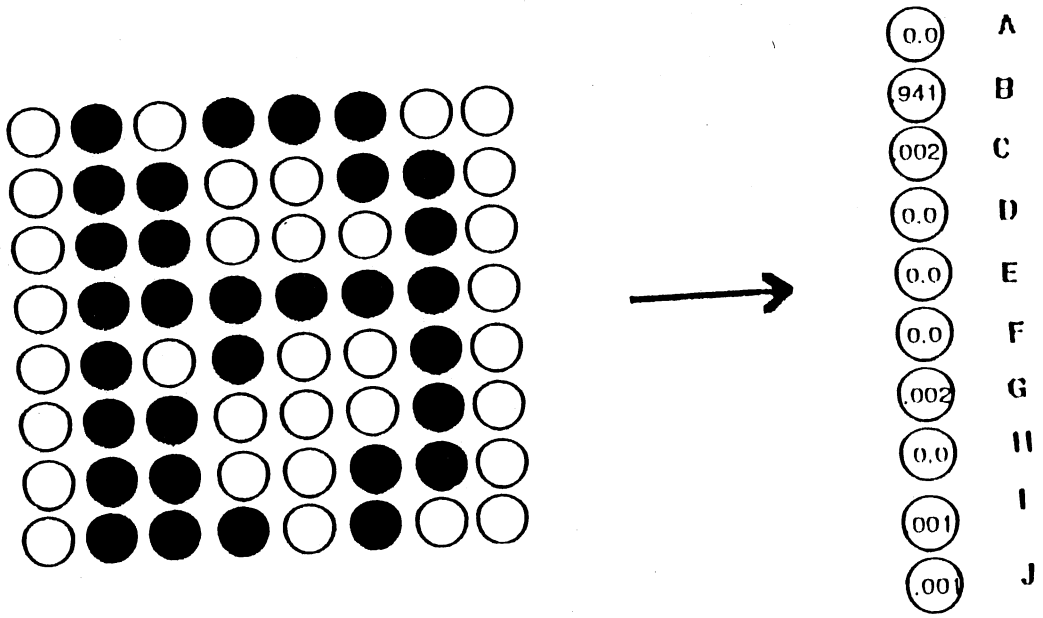


Figura 27: Diez letras. Ejemplo de un estado inicial de entrada, compuesto de la letra B, donde 3 unidades de patrón de entrada original fueron cambiadas de ceros (círculos negros) a unos (círculos blancos). Como podemos observar, la segunda unidad de las unidades de salida se activo, y el resto inactivas, siendo la salida esperada, es decir, la red reconoció la letra. Cabe notar que los valores de 0.0, fueron calculados hasta 10^{-3}.

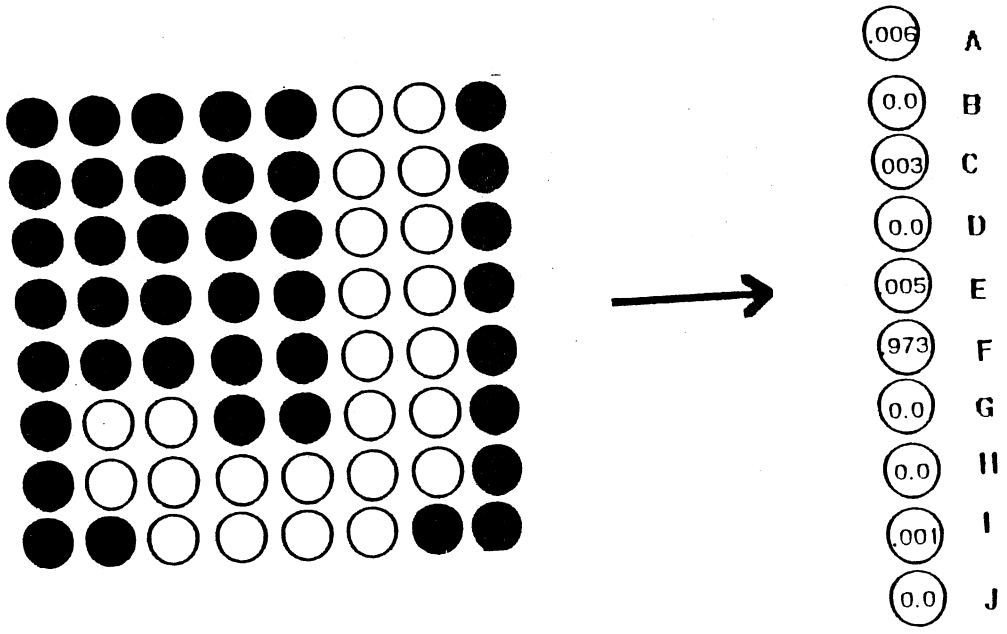


Figura 28: Diez letras. Ejemplo de un estado inicial de entrada, compuesto de la letra J, donde los valores de la salida obtenida fue la activación de la sexta unidad, posicionada de arriba hacia abajo, correspondiente a la letra F. Como podemos observar las letra J uno de los antipatrones, esto es, el patrón de entrada original tiene el fondo blanco (unos) y la letra oscura (ceros). Cabe notar que los valores de 0.0, fueron calculados hasta 10^{-3}.

En otro caso estudiado, se concideró como reacciona la red ante la presentación de los antipatrones. Esto es, los patrones de entrada con los que se llevó acabo el entrenamiento tenían el fondo blanco (unos) y la letra oscura (ceros), de manera que se presentó como estado inicial una letra en blanco y fondo negro. El resultado obtenido fue que la red no reconoció los estados iniciales, ya que los valores de las unidades de salida correspondía a un letra del conjunto de datos utilizados para el entrenamiento. Como fue para el caso de la letra J (figura 28) que obtuvo valores en sus unidades de salida de la letra F, esto es, activandose la sexta unidad de salida con un valor de 0.973 y el resto apagadas.

Como últimos casos, se analizaron letras con fondo en gris (0.5), con el resto negro (0.0) y viceversa. Para el caso de que el fondo de las letras era gris, se obtuvo un 80% de éxito con 10 estados iniciales con los que se validó la red, ya que la unidad de salida correspodiente a cada estado inicial, se activó satisfactoriamente. Los valores de ésta unidad de salida fueron valores pequeños, como sucedió en el caso de la letra A mostrada en la figura 30. Uno de los mejores resultados fue la letra J (figura 29), ya que el valor de la décima posición fue de 0.892, y el resto apagadas. Por otro lado, para el caso contrario, la red reconoció un 1% de los estados iniciales proporcionados. En la figura 31 se muestra la única letra que la red reconoció, siendo la letra F, donde la unidad de salida activada es la que se encuentra en la sexta posición. Un ejemplo de un estado que no reconoció fue la letra E (figura 32) que si contiene una unidad activada, pero en la sexta posición, la cual corresponde a la salida de la letra F.

Como ya sabemos, la red debe reconocer los patrones utilizados para entrenerla. Por los ejemplos anteriores, la red tiene poca capacidad de reconocimiento ante los diferentes estados iniciales de entrada. Esta capacidad está relacionada con un número de patrones con los que la red fue entrenada ya que si el número es muy grande, las unidades de la capa intermedia obtenidas con el algoritmo AMNUI lo sarán también.

En este trabajo solo se analizó el comportamiento e importancia del factor de aprendizaje (η), más sin embargo no se profundizó en saber el comportamiento del sistema, al acotar más el intervalo en el que ocurrieron las oscilaciones, o lo que secedería al variar los parámetros α y β. Estos son algunos ejemplos, que serían importantes de analizar en trabajos posteriores.

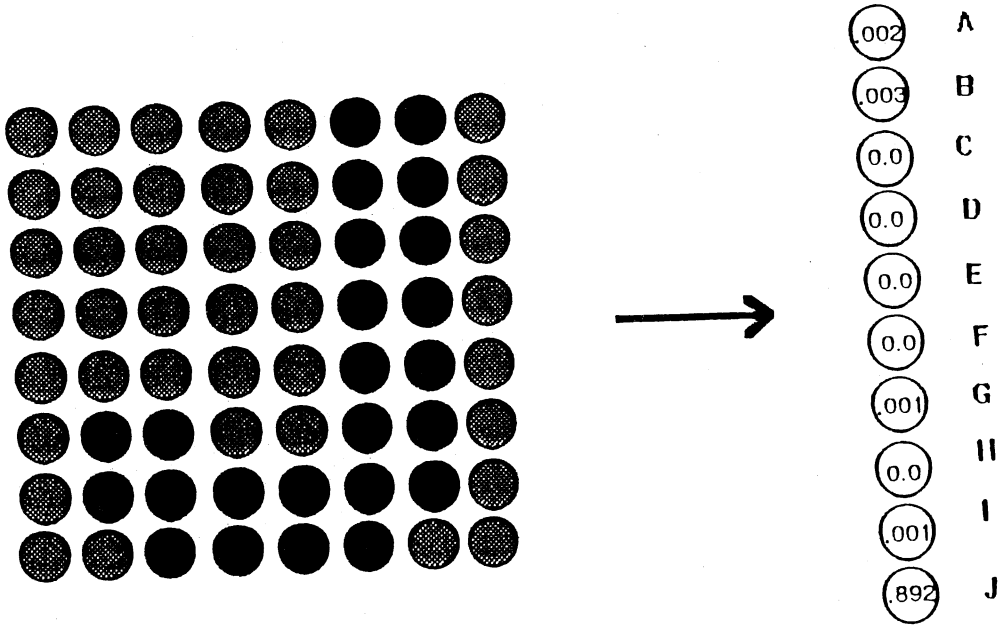


Figura 29: Diez letras. Ejemplo de un estado inicial de entrada, compuesto de la letra J, donde el patrón de salida obtenido fue la activación de la decima posición de arriba hacia abajo. Como podemos observar que el fondo de la letra es en tono gris (0.5) y el resto en negro (0). Cabe notar que los valores de 0.0, fueron calculados hasta 10^{-3}.

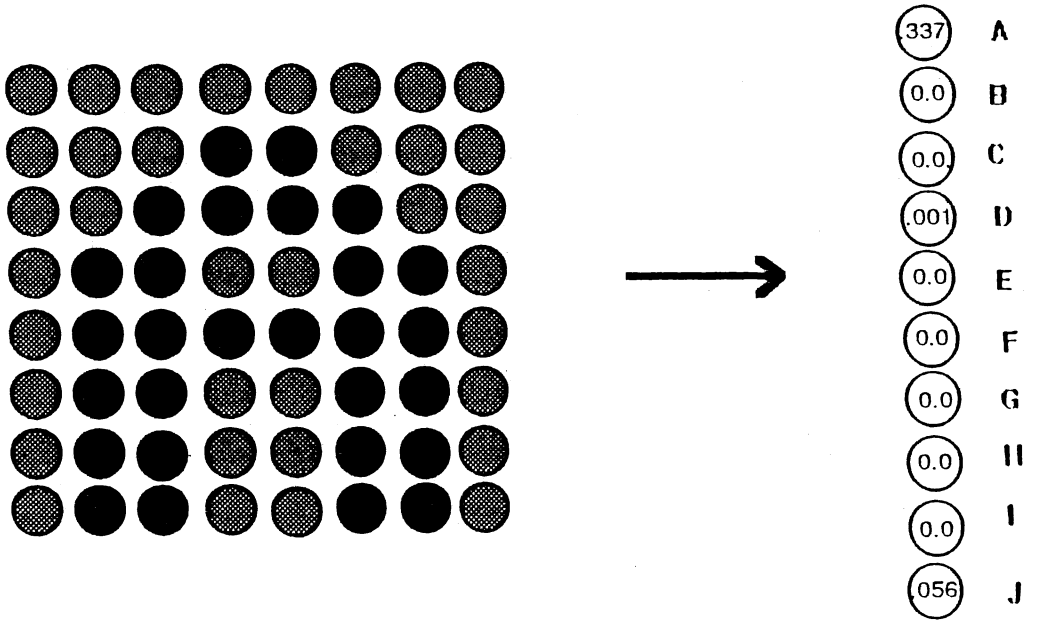


Figura 30: Diez letras. Ejemplo de un estado inicial de entrada, compuesto de la letra A, donde el patrón de salida obtenido fue la activación de la decima posición de arriba hacia abajo. Como podemos observar que el fondo de la letra es en tono gris (0.5) y el resto en negro (0). Cabe notar que los valores de 0.0, fueron calculados hasta 10^{-3}.

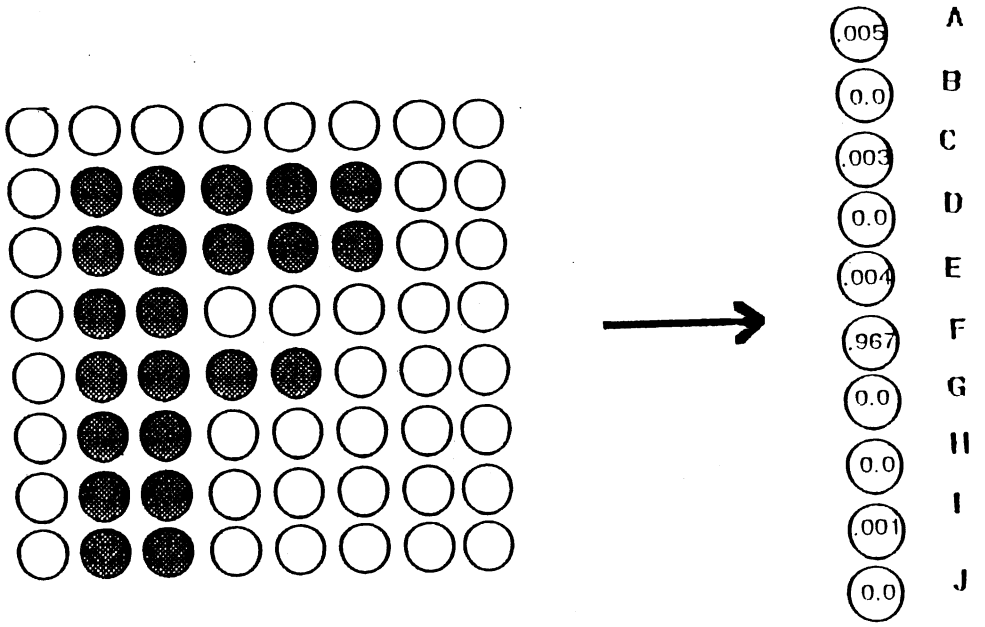


Figura 31: Diez letras. Ejemplo de un estado inicial de entrada, compuesto de la letra F, donde el patrón de salida obtenido fue la activación de la sexta posición de arriba hacia abajo. Como podemos observar que el fondo de la letra es blanco (1) y el resto en tono gris (0.5). Cabe notar que los valores de 0.0, fueron calculados hasta 10^{-3}.

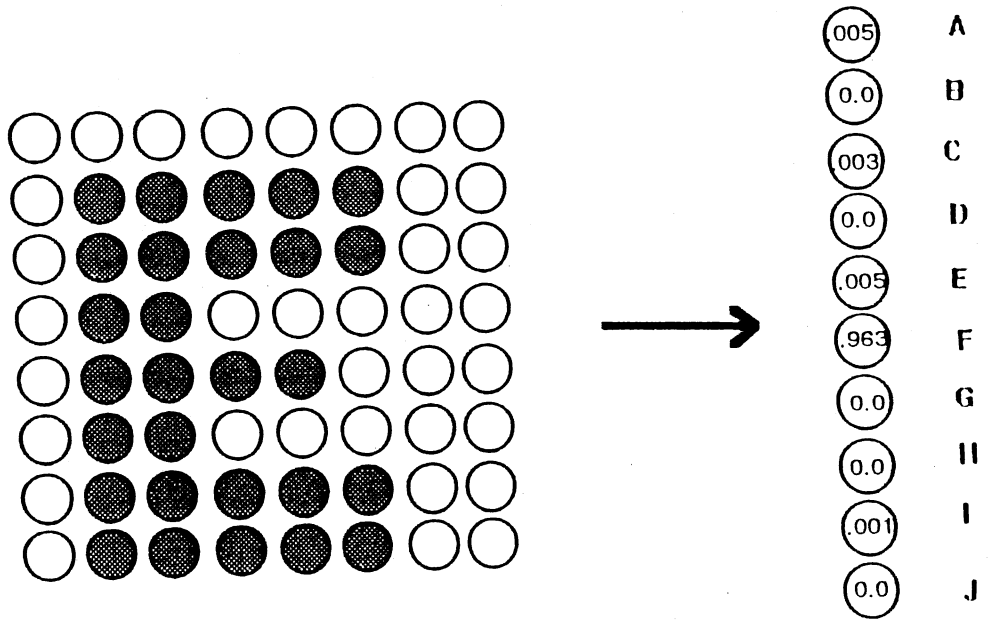


Figura 32: Diez letras. Ejemplo de un estado inicial de entrada, compuesto de la letra E, donde el patrón de salida obtenido fue la activación de la sexta posición de arriba hacia abajo. Como podemos observar que el fondo de la letra es blanco (1) y el resto en tono gris (0.5). Cabe notar que los valores de 0.0, fueron calculados hasta 10^{-3}.

VI Conclusiones.

De los resultados obtenidos se puede concluir lo siguiente:

a) Con el uso del algoritmo de Retropropagación durante la etapa de aprendizaje, la red no se encuentra libre ser atrapada en la búsqueda de algún mínimo. Una solución a este problema es utilizar el algoritmo AMNUI necesario para encontrar el número óptimo en unidades de la capa intermedia, para una RN formada de tres capas.

b) El algoritmo AMNUI, está fuertemente ligado con el valor óptimo del factor de aprendizaje (η), ya que si no es el adecuado, el algoritmo no pasa la etapa de añadir constantemente unidades a la capa intermedia, teniendo como consecuencia las disminuciones del error son lentas, sin llegar nunca a la solución requerida.

c) Para la determinación del valor del factor de aprendizaje (η) no existe, por ahora, ninguna receta, lo único es una búsqueda detallada del valor óptimo, variando η constantemente dentro de un intervalo definido. Se comprobó que para valores pequeños o valores grandes dependiendo del problema, no existe convergencia ocasionando que el algoritmo nunca llegue a su terminación.

d) Dada las condiciones de los problemas a simular y la función de activación, el parámetros η utilizado en el algoritmo AMNUI debe ser el adecuado, ya que para ciertos valores, se pueden obtener buenos resultados pero con un número grande de oscilaciones del error obtenido por la función de costo. Una solución a este problema fué el de diminuir un poco el valor η, reduciendose el número de oscilaciones en el algoritmo.

e) El número de unidades intermedias obtenido del algoritmo AMNUI, dependen del número de patrones con los cuales la red va hacer entrenada. Si el número de patrones a aprender es muy grande, la cantidad de unidades de la capa intermedia será mayor a comparación de una RN que tenga aprender pocos patrones.

f) Una vez que la red es entrenada, tiene alguna capacidad de reconocer un conjunto de estados iniciales de entrada que se asemejen a cualquiera de los patrones aprendidos.

Bibliografía

Brunak, S., Lautrup, B., Neural Networks Computer with Intuition, World Scientific, (1990), 117.

Enciclopedia de la Ciencia y Técnica, **6**, Ed. Danae S.A., (1981).

Gary, J., BYTE, **12**, (1987), 183.

Garrido, L., Lecture notes in Physic Statistical Mechanics of Neural Netwoks, Springer - Verlang, (1990), 3.

Hertz, J., Krogh, A., Palmer, R., Introduction to the theory of Neural Computation, Addison-Wesley publishing Company, (1991), 89-115.

Hirose, Y., Yamashita, K., Hijiya, S., Neural Networks, **4**, (1991), 61.

Kohonen, T., Neural Networks, Pergamon Press, **1**, (1988), 3.

Laarhoven., Peter, J.M., E.H.L.Aarts., Simulated Annealing: Theory and Applications,(1988),3.

Müller, B., Reinhardt, R., Phys of Neural Networks, Neural Networks (An Introduction), Springer-Verlag, (1991), 45.

Nelson, M., Illingworth, W., A Practical Guide to Neural Net, Addison-Wesley Publishing, (1991), 82.

Ooyen, A., Neural Networks, **5**, (1992), 465

Rasmus, D., BYTE, **16**, (1991), 281.

Rayward, S., V. J., Combinatorial Optimization II,(1979),53.

Rumelhart, D., E., Hinton, G., E., Williams, R., J., Nature, **323**, (1991), 533.

Rumelhart, D., McClelland, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, **1**, Foundations MIT, Cambridge, (1986), 318.

Soucek, B., Soucek, M., Neural and Massively Parallel Computers (the sixth generation), A wiley Interscience Publishing Company, (1988), 213.

Studt, T., Research Developmet (RD), **33**, (1991), 36.

Touretzky, D., S., Pomerleaun Dean A., BYTE, **14**, (1989), 227.

Viana, L., Memoria Natural y Artificial, La ciencia desde México, **88**, Ed. Fondo de cultura económica (1990), 95.

Wasserman, D., Schwartz, T., IEEE Expert, **3**, (1988), 10.

Apéndice

Descripción del programa

Simulación de una Red Neuronal cuya arquitectura es de Multicapas, la red consta de 3 capas, una capa de neuronas de entrada, una capa de neuronas intermedias que va aumentando o disminuyendo, hasta obtener el número de neuronas óptimas y de una capa de neuronas de salida. El código del programa esta escrito en lenguaje C, y puede ser ejecutado en un equipo multiusuarios o en una computadora personal.

Descripción de las Funciones

Declaración de variables globales:

nent : Número de neuronas de entrada.
nint : Número de neuronas intermedias.
nsal : Número de neuronas de salida.
beta : Pendiente requerida en la función de activación.
eta : Factor de aprendizaje.
alfa : Momento.
MAX_ENT : Número máximo de neuronas de la capa de entrada.
MAX_INT : Número máximo de neuronas de la capa intermedia.
MAX_SAL : Número máximo de neuronas de la capa de salida.
MAX_P : Número máximo de patrones.

RAN,RAN1 : Generación de números aleatorios.
xi[][] : Patrones de la capa de entrada.
xi_act[] : Patrón actual de la capa de entrada.
zeta[][] : Patrones de la capa de salida.
zsal[] : Patrón actual de la capa de salida.
went[][] : Matriz de pesos de la capa de entrada - intermedia.
wsal[][] : Matriz de pesos de la capa de intermedia - salida.
ghi[] : Derivada de la capa intermedia - salida.
ghj[] : Derivada de la capa entrada - intermedia.
vj[] : Resultado de aplicar la función de activación
oi[] : Resultado esperado de la red.
patron : Número de patrones con los que la red va ha ser entrenada.

Función Aprende():

Encuentra los valores adecuados para la matriz de pesos de la red, por medio del algoritmo de aprendizaje de Retropropagación. Revisa mediante la función de costo, si se deben de aumentar o disminuir neuronas a la red en la capa intermedia.

dwent[][] : Matriz de pesos temporal, capa entrada - salida.
dwsal[][] : Matriz de pesos temporal, capa de intermedia - salida.
si[] : Paso intermedio para obtener los incrementos en la matriz de pesos de la capa intermedia - salida.
sj[] : Paso intermedio para obtener los incrementos en la matriz de pesos de la capa entrada - intermedia.
error_act : Error en la iteración actual.
error_ant: Error cada 100 iteraciones.
*ptr1 : Apuntador al archivo de salida.
*ptr2 : Apuntador al archivo de pesos.
i,j,k,ip,it : Contadores.
itmax : Número máximo de iteraciones.
alto : Bandera de aviso cuando el algoritmo termina.
temp : Temporal para guardar el número de neuronas intermedias.
sal,agrega: Banderas de aviso para aumentar o disminuir neuronas en la capa intermedia.
nombre1[] : Nombre del archivo de salida.
nombre2[] : Nombre del archivo de pesos.

```
void Aprende(){
int   i,j,k,agrega,ip,it,alto,sal,itmax=5000,temp;
float  error_act, error_ant; /*   Función de Costo */
float  dwent[MAX_INTER][MAX_ENT+1],
        dwsal[MAX_SAL][MAX_INTER+1],
        si[MAX_SAL],
        sj[MAX_INTER];
FILE *ptr2,*ptr1;
char nombre1[30],nombre2[30];

       printf("Nombre del archivo de salida:");
       scanf("%s",&nombre1);
       printf("Nombre del archivo de pesos: ");
       scanf("%s",&nombre2);

       agrega=alto=sal=0;
       printf("Aprendiendo.....\n");
       srand((unsigned int)time(NULL));
       xi_act[nent]=vj[nint]=0;

        for (i=0; i<nint; i++) for (j=0; j<=nent; j++) went[i][j] =RAN1;
        for (i=0; i<nsal; i++) for (j=0; j<=nint; j++) wsal[i][j]=RAN1;

        for (i=0; i<nint; i++) for (j=0; j<=nent; j++)  dwent[i][j] =0.;
        for (i=0; i<nsal; i++) for (j=0; j<=nint; j++) dwsal[i][j]=0.;
        if((ptr1=fopen(nombre1,"w")) == NULL){
           printf("Archivo %s no puede ser abierto....\n",nombre1);
          exit(1);
        }else{
       for (it=0; it<itmax; it++){
         error_act= 0.;
         for (ip=0; ip<patron; ip++){
           if (ip==0){  /* Initializa el coeficiente del momento */
         for (i=0; i<nint; i++) for (j=0; j<=nent; j++)  dwent[i][j] *=alfa;
         for (i=0; i<nsal; i++) for (j=0; j<=nint; j++)  dwsal[i][j]*=alfa;
       }
      Patron_inicio(ip);
        /* Realiza la progación hacia adelante */
        Propaga();
        /* Desviación entre la salida esperada y la actual */
       for (i=0; i<nsal; i++)
        error_act +=pow((zsal[i]-o[i]),2);
```

```
        /* Calcula las Deltas (incrementos) */

       for (i=0; i<nsal; i++)
         si[i]=ghi[i]*(zsal[i]-o[i]);
        for (j=0; j<nint; j++){
          sj[j] = 0.0;
    for (i=0; i<nsal; i++)  sj[j] += si[i]*wsal[i][j];
        sj[j] *=ghj[j];
       }

       /* Calcula las correciones para cada patrón */
for (j=0; j<nint; j++)        for (k=0; k<=nent; k++)
       dwent[j][k] += eta*sj[j]*xi_act[k];

for (i=0; i<nsal; i++) for (j=0; j<=nint; j++)
       dwsal[i][j] +=eta*si[i]*vj[j];


/* Calcula los nuevos coeficientes sinápticos*/
  if (ip==(patron-1)){
    for (i=0; i<nint; i++) for (j=0; j<=nent; j++)
       went[i][j] += dwent[i][j];
    for (i=0; i<nsal; i++) for (j=0; j<=nint; j++)
       wsal[i][j] += dwsal[i][j];
      }
  } /* ip = 0 ... (patrón-1) */

 error_act /=2.0;

    /* Imprime la  desviación esperada */
   printf("It: %3d Error: %f\n",it+1,error_act);
  if (error_act<=0.01){
      agrega++;
      printf("CONVERGE, iteración %3d\n",it);
      sal=1;
      if(!agrega){
        error_act=0.0;
        agrega=1;
      }
      if(agrega){
      /* Escribiendo los pesos a un archivo */
       if((ptr2=fopen(nombre2,"w")) == NULL){
         printf("Archivo %s no puede ser abierto....\n",nombre2);
         exit(1);
```

```
        }else{
            temp = nint;
          printf("Guardando el valor de la capa %d\n",temp);
          for (i=0; i<temp; i++) for (j=0; j<=nent; j++)
           fprintf(ptr2,"%f ",went[i][j]);
          for (i=0; i<nsal; i++) for (j=0; j<=temp; j++)
           fprintf(ptr2,"%f ",wsal[i][j]);
           fclose(ptr2);
          /* Quitando neuronas */
          if(nint>1){ /* Si la capa intermedia es mayor que uno */
           nint--;
           printf("\nQuitando neurona en la capa intermedia\n");
          }else {
          printf("Solo queda una neurona\n");
          alto=1;
           }
          }
         }
      }else sal=0;

if((fmod(it,10) == 0)) fprintf(ptr1,"%d  %f\n",it,error_act);
if(!sal){
  if(it==0) error_ant = error_act;
  if((fmod(it,100) == 0)&& (it>0)){
   if(fabs((error_act-error_ant)/error_ant) < 0.01){
    if(agrega == 0){
      nint++;
      went[nint][nent]=RAN1;
      wsal[nsal][nint]=RAN1;
        printf("Adiciona neurona en la capa intermedia \n");
     } /* Lee pesos de archivo y el sistema acaba */
    if(agrega>0){
      if((ptr2=fopen(nombre2,"r")) == NULL){
          printf("Archivo %s no puede ser abierto....\n",nombre2);
          exit(1);
          } else {
          printf("Leyendo valores de la capa %d\n",temp);
          for (i=0; i<temp; i++) for (j=0; j<=nent; j++)
          fscanf(ptr2,"%f",&went[i][j]);
          for (i=0; i<nsal; i++) for (j=0; j<=temp; j++)
            fscanf(ptr2,"%f",&wsal[i][j]);
          fclose(ptr2);
          nint=temp;
          alto=1;
```

```
        } /* else archivo*/
        } /* else agrega */
        }/* Comparación de error */
        error_ant=error_act;
       }/* De cada 100 iteraciones*/
       if(alto) break;
       } /* it = 0 ... (itmax-1) */
     }
    }
fclose(ptr1);
return;
}
```

Función Propaga() :

Realiza la propagación de la señal, através de todas las capas de la red, aplicando la función de activación y calculando a su vez la derivada de esta.

act : Sumatoria de las neuronas de la capa de entrada - intermedia o salida - intermedia segun sea el caso multiplicadas por su correspondiente valor de peso.
excita: Valor de la función de activación para las capas de entrada - intermedia y intermedia - salida.
i,j : Contadores.

```
void Propaga(){

float act,excita;
int   i,j;

/* Propaga la señal de la capa de entrada a la capa intermedia */
 for (j=0; j<nint;j++){
 act = 0.;
 for (i=0; i<nent; i++)  act += went[j][i]*xi_act[i];
 excita = Disparo(act); /*  Función de  activación*/
 vj[j] = excita;
 ghj[j]= Ddisparo(excita);
}
 /* Propaga la señal de la capa intermedia a la capa de salida */
```

```
 for (j=0; j<nsal; j++){
  act = 0.;
  for (i=0; i<nint; i++)  act += wsal[j][i]*vj[i];
  excita = Disparo(act);
  oi[j] = excita; /* Salida de la red */
  ghi[j]= Ddisparo(excita);
 }
return;
}
```

Función Disparo():

Contiene la función de activación utilizada durante el paso de aprendizaje.

e : Calcula la función tangente hiperbolica.
x : Valor acumulado de la propagación de la señal hacia adelante.

```
float Disparo(float x){
float e;

 if (fabs(x)<15./beta){
   e = exp(2.*beta*x);
   return 0.5*(1+(e-1.)/(e+1.));
  } else{
    if (x>=0)  return  1.;
     else      return -1.;
 }
}
```

Función Ddisparo():

Contiene la derivada de la función de activación durante el paso de aprendizaje.

y : Valor de la función de activación necesario para calcular la derivada.

```
float Ddisparo(float y){
        return 0.5*beta*(1.-y*y);
}
```

Función Patron_inicio():

Escoge el patrón requerido en el paso de aprendizaje.

ip : Número del patrón actual, al cual se va a calcular el error.
i : Contador.

```
void Patron_inicio(int ip){
int   i;

 for (i=0; i<nent; i++)  xi_act[i] = xi[ip][i];
 for (i=0; i<nsal; i++)  zsal[i] = zeta[ip][i];
return;
}
```

Función Resultados():

Despliega en pantalla los valores de las conexiones apropiados para la capa de entrada - intermedia y de la capa intermedia - salida.

i,j,k : Contadores.

```
void Resultados(){
int i,j,k;

  printf("\Pesos  de la capa de Entrada --> Intermedia\n");
  for (j=0; j<nint; j++){
```

```
    printf("\nWENT(%2d,k)   k= ",j);
    for (k=0; k<nent; k++) printf(" %2d:%8.3f",k,went[j][k]);
  }
   printf("\n\nPesos de la capa Intermedia -->Salida");
   for(i=0;i<nsal;i++){
    printf("\nWSAL(%2d,i)  i= ",i);
    for (j=0; j<nint; j++) printf(" %2d:%8.3f",j,wsal[i][j]);
  }
}
```

Función main():

Programa principal evoca a las funciones de entrenamiento de la red presentando finalmente los resultados esperados.

```
main(){

  Aprende();
  Resultados();
  return;
}
```

Programa Principal

```
/ *
      Simulación de una Red Neuronal tipo perceptron, inicia con una
neurona en la capa intermedia. La red consta de 3 capas, en la capa intermedia
va aumentando disminuyendo hasta obtener el número de neuronas óptimas
para resolver el problema de la función booleana XOR.
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <ctype.h>

#define   MAX_ENT      2
#define    MAX_INTER 100
```

```
#define   MAX_SAL      1
#define   MAX_P        4

#define   RAN     ((double) rand()*3.05184e-5)
#define   RAN1    (2.*RAN-1.)

              /*  Variables Globales */
int nent=2,nsal=1,nint=1,patron=4;
float xi_act[MAX_ENT+1],vj[MAX_INTER+1],o[MAX_SAL];
float xi[MAX_P][MAX_ENT] = {{1,1},{-1,1},{1,-1},{-1,-1}},
       zeta[MAX_P][MAX_SAL] = {-1,1,1,-1},
       zsal[MAX_SAL],ghj[MAX_INTER],ghi[MAX_SAL],
       went[MAX_INTER][MAX_ENT+1],
       wsal[MAX_SAL][MAX_INTER+1];
float beta=0.5,eta= 2.0,alfa=0.2;

/* Declaración de  Funciónes*/
void   Patron_inicio(int),Aprende(),Propaga(),Resultados();
float  Disparo(float), Ddisparo(float);

/*  Programa principal */
main(){
       clrscr();
       Aprende();
       Resultados();
}

void Aprende(){
int   i,j,k,agrega,ip,it,alto,sal,itmax=5000,temp;
float  error_act, error_ant; /*  Función de Costo */
float  dwent[MAX_INTER][MAX_ENT+1],
        dwsal[MAX_SAL][MAX_INTER+1],
        si[MAX_SAL],
        sj[MAX_INTER];
FILE *ptr2,*ptr1;
char nombre1[30],nombre2[30];

    printf("Nombre del archivo de salida:");
    scanf("%s",&nombre1);
    printf("Nombre del archivo de pesos: ");
    scanf("%s",&nombre2);
   agrega=alto=sal=0;
    printf("Aprendiendo.....\n");
        /*Inicializa el generador de números aleatorios */
```

```
   srand((unsigned int)time(NULL));
   xi_act[nent]=vj[nint]=0;

   for (i=0; i<nint; i++) for (j=0; j<=nent; j++) went[i][j] =RAN1;
   for (i=0; i<nsal; i++) for (j=0; j<=nint; j++) wsal[i][j]=RAN1;

   for (i=0; i<nint; i++) for (j=0; j<=nent; j++)  dwent[i][j] =0.;
   for (i=0; i<nsal; i++) for (j=0; j<=nint; j++) dwsal[i][j]=0.;
   if((ptr1=fopen(nombre1,"w")) == NULL){
      printf("Archivo %s no puede ser abierto....\n",nombre1);
      exit(1);
  }

else{
  for (it=0; it<itmax; it++){
   error_act= 0.;
   for (ip=0; ip<patron; ip++){
     if (ip==0){  /* Initializa el coeficiente del momento */
        for (i=0; i<nint; i++) for (j=0; j<=nent; j++)  dwent[i][j]*=alfa;
        for (i=0; i<nsal; i++) for (j=0; j<=nint; j++)  dwsal[i][j]*=alfa;
  }
   Patron_inicio(ip);
      /* Realiza la progación hacia adelante */
  Propaga();
  /* Desviación entre la salida esperada y la actual */
   for (i=0; i<nsal; i++) error_act +=pow((zsal[i]-o[i]),2);


       /* Calcula las Deltas (incrementos) */
   for (i=0; i<nsal; i++)
      si[i]=ghi[i]*(zsal[i]-o[i]);
    for (j=0; j<nint; j++){
     sj[j] = 0.0;
     for (i=0; i<nsal; i++)  sj[j] += si[i]*wsal[i][j];
        sj[j] *=ghj[j];
   }

      /* Calcula las correciones para cada patrón */
  for (j=0; j<nint; j++) for (k=0; k<=nent; k++)
      dwent[j][k] += eta*sj[j]*xi_act[k];

 for (i=0; i<nsal; i++) for (j=0; j<=nint; j++)
      dwsal[i][j] +=eta*si[i]*vj[j];
```

```
/* Calcula los nuevos coeficientes sinápticos */
 if (ip==(patron-1)){
    for (i=0; i<nint; i++) for (j=0; j<=nent; j++)
       went[i][j] += dwent[i][j];
    for (i=0; i<nsal; i++) for (j=0; j<=nint; j++)
       wsal[i][j] += dwsal[i][j];
      }
  } /* ip = 0 ... (patron-1) */

error_act /=2.0;

/* Imprime la  desviación  entre el patrón esperado y el actual */
  printf("It: %3d Error: %f\n",it+1,error_act);
  if (error_act<=0.01){
    agrega++;
     printf("CONVERGE, iteración %3d\n",it);
    sal=1;
    if(!agrega){
      error_act=0.0;
     agrega=1;
      }
    if(agrega){
       /* Escribiendo los pesos a un archivo */
       if((ptr2=fopen(nombre2,"w")) == NULL){
        printf("Archivo %s no puede ser abierto....\n",nombre2);
      exit(1);
       }else{
         temp = nint;
         printf("Guardando el valor de la capa %d\n",temp);

         for (i=0; i<temp; i++)      for (j=0; j<=nent; j++)
          fprintf(ptr2,"%f ",went[i][j]);
         for (i=0; i<nsal; i++) for (j=0; j<=temp; j++)
          fprintf(ptr2,"%f ",wsal[i][j]);
          fclose(ptr2);
              /* Quitando neuronas */
          if(nint>1){ /* Si la capa intermedia es mayor que uno */
           nint--;
            printf("\nQuita neurona en la capa intermedia\n");
          }else {
            printf("Solo queda una neurona \n");
           alto=1;
            }
        }
```

```
        }
    }
    else sal=0;

if((fmod(it,100.0) == 0)) fprintf(ptr1,"%d  %f\n",it,error_act);
if(!sal){
    if(it==0) error_ant = error_act;
    if((fmod(it,100,0) == 0)&& (it>0)){
     if(fabs((error_act-error_ant)/error_ant) < 0.01){
    if(agrega == 0){
        nint++;
        went[nint][nent]=RAN1;
        wsal[nsal][nint]=RAN1;
        printf("Adiciona neurona en la capa intermedia%d\n",nint);
 } /* Lee pesos de archivo y el sistema acaba */
  if(agrega>0){
     if((ptr2=fopen(nombre2,"r")) == NULL){
        printf("Archivo %s no puede ser abierto....\n",nombre2);
        exit(1);
        }else{
        printf("Leyendo valores de la capa %d\n",temp);
        for (i=0; i<temp; i++) for (j=0; j<nent; j++)
           fscanf(ptr2,"%f",&went[i][j]);
        for (i=0; i<nsal; i++) for (j=0; j<temp; j++)
           fscanf(ptr2,"%f",&wsal[i][j]);
        fclose(ptr2);
        nint=temp;
        alto=1;
           } /* else archivo*/
         } /* else agrega */
         }/* Comparación de error */
         error_ant=error_act;
        }/* De cada 100 iteraciones*/
        if(alto) break;
     } /* it = 0 ... (itmax-1) */
    }
  }
fclose(ptr1);
return;
}
```

```
void Propaga(){
float act;
float excita;
int   i,j;

/* De la capa de entrada a la capa intermedia */
        for (j=0; j<nint;j++){
         act = 0.;
         for (i=0; i<=nent; i++)  act += went[j][i]*xi_act[i];
           excita = Disparo(act);
            vj[j] = excita;
           ghj[j] = Ddisparo(excita);
         }
         /* De la capa intermedia a la capa de salida */
         for (j=0; j<nsal; j++){
          act = 0.;
         for (i=0; i<=nint; i++)  act += wsal[j][i]*vj[i];
          excita = Disparo(act);
          o[j]   = excita;
          ghi[j]= Ddisparo(excita);
         }
return;
}

float Disparo(float x){
float e;

        if (fabs(x)<15./beta){
          e = exp(2.*beta*x);
          return 0.5*(1+(e-1.)/(e+1.));
        }else {
        if (x>=0.)  return  1.;
        else        return -1.;
}
}
```

```
float  Ddisparo(float y){
        return 0.5*beta*(1.-y*y);
}

void Patron_inicio(int ip){
int   i;

                for (i=0; i<nent; i++)  xi_act[i] = xi[ip][i];
                for (i=0; i<nsal; i++)  zsal[i] = zeta[ip][i];
return;
}



void Resultados(){
int j,i,k;
                        /* La red aprende los patrones */
        printf("\Pesos de la capa de Entrada --> Intermedia\n");
        for (j=0; j<nint; j++){
         printf("\nWENT(%2d,k)    k= ",j);
         for (k=0; k<nent; k++) printf(" %2d:%8.3f",k,went[j][k]);
 }
        printf("\n\nPesos de la capa Intermedia -->Salida");
        for (i=0; i<nsal; i++){
         printf("\nWSAL(%2d,i)   i= ",i);
         for (j=0; j<nint; j++) printf(" %2d:%8.3f",j,wsal[i][j]);
        }
        getch();
}
```