UNIVERSIDAD AUTONOMA DE NUEVO LEON
Facultad de Ingeniería Mecánica y Eléctrica
División de Estudios de Posgrado

PROCESO DE DISTRIBUCION APLICANDO REDES NEURONALES ARTIFICIALES CON SUPERVISION

T E S I S

En Opción al Grado de:

MAESTRO EN CIENCIAS

de la Administración con Especialidad en Sistemas

Por:

ING. ARACELI CAMPOS ORTIZ

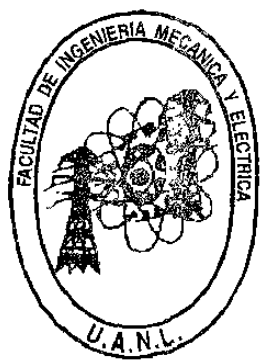
San Nicolas de Los Garza, N. L. Diciembre de 1998

98

Proceso de Distribución, Aplicando Redes Neuronales Artificiales con Supervisión

A. C. O.

UNIVERSIDAD AUTONOMA DE NUEVO LEON
FACULTAD DE INGENIERIA MECANICA Y ELECTRICA
DIVISION DE ESTUDIOS DE POSGRADO



PROCESO DE DISTRIBUCION APLICANDO REDES NEURONALES ARTIFICIALES CON SUPERVISION

TESIS

EN OPCION AL GRADO DE MAESTRO EN CIENCIAS DE LA ADMINISTRACION CON ESPECIALIDAD EN SISTEMAS

POR:

ING. ARACELI CAMPOS ORTIZ

SAN NICOLAS DE LOS GARZA, N.L. DICIEMBRE DE 1998

UNIVERSIDAD AUTONOMA DE NUEVO LEON
FACULTAD DE INGENIERIA MECANICA Y ELECTRICA
DIVISION DE ESTUDIOS DE POSGRADO

Los miembros del comité de tesis recomendamos que la tesis PROCESO DE DISTRIBUCION APLICANDO REDES NEURONALES ARTIFICIALES CON SUPERVISION realizada por el Ing. Araceli Campos Ortiz sea aceptada para su defensa como opción al grado de Maestro en Ciencias de la Administración con especialidad en Sistemas.

El Comité de Tesis

Asesor
Dra. Ada Margarita Alvarez Socarrás

Coasesor
Dr. José Luis Martínez Flores

Coasesor
M.C. Roberto Villarreal Garza

Vo.Bo.
M.C. Roberto Villarreal Garza
División de Estudios de Posgrado

San Nicolás de los Garza, N.L. a 7 de Diciembre de 1998

DEDICATORIA

A Dios

Por permitirme vivir y emprender nuevos retos

Ami familia

Por su incansable apoyo y cooperación

A mi esposo

Por su comprensión y por compartir conmigo este compromiso

A mis asesores

Por compartirme sus conocimientos y experiencias en la realización de esta investigación y por su valiosa amistad.

Y a todas aquellas personas que de alguna u otra manera me brindaron su apoyo y compartieron conmigo tanto mis aciertos como mis errores.

AGRADECIMIENTOS

Expreso mi más sincero agradecimiento a la Dra. Ada M. Alvarez Socarrás asesor de mi tesis y al Dr. José Luis Martínez Flores por su valiosa colaboración e interés para el desarrollo y revisión de este trabajo.

Agradezco también a mis padres y familiares por su apoyo en todas las decisiones que he tomado en mi vida.

En forma muy especial, agradezco a mi esposo por su paciencia y apoyo incondicional durante mis estudios de Maestría y en la realización de este trabajo.

RESUMEN

Araceli Campos Ortiz

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Título del Estudio: PROCESO DE DISTRIBUCION APLICANDO REDES NEURONALES ARTIFICIALES CON SUPERVISION

Número de Páginas: 89

Candidato para el grado de Maestría en Ciencias de la Administración con especialidad en Sistemas

Area de Estudio: Redes Neuronales Artificiales

Propósito y Método del Estudio: El propósito principal de este estudio es implementar un programa simulador que logre adaptar un patrón de distribución de flujo que permita minimizar el costo total que implicará abastecer las necesidades expuestas de oferta/demanda de uno o varios centros fuente-sumidero. Para esto, se aplicó la metodología de redes neuronales artificiales con supervisión. Esta investigación tiene su base en el artículo propuesto por Renzo Perfetti en 1995 en donde expone casos de solución a problemas de flujo aplicando redes neuronales. El programa diseñado en este estudio fue implementado en un software de aplicación computacional llamado Matlab, el cual fue probado con una cantidad considerable de problemas de distribución logrando excelentes resultados en su implementación.

Contribuciones y conclusiones: El estudio realizado representa una aportación para resolver problemas de distribución, en donde el programa desarrollado logra eficientar el proceso de distribución. Como contribución de ésta investigación se puede mencionar la facilidad de poder resolver problemas en donde los centros fuente no estén totalmente conectados con todos los centros sumidero, así como el poder determinar el mínimo valor de capacidad de distribución en los enlaces fuente-sumidero.

Firma del asesor: 

TABLA DE CONTENIDO

Capítulo **Página**

Capítulo **Página**

INDICE DE FIGURAS

Figura **Página**

Figura **Página**

CAPITULO 1

INTRODUCCION

1.1 Problemática a Tratar

Siempre que pretendemos lograr un objetivo, cualquiera que éste sea, deseamos minimizar los costos que puedan surgir del mismo. Nuestra investigación está fundamentada precisamente en lograr minimizar costos en procesos de distribución aplicando la metodología de Redes Neuronales Artificiales. Los elementos que constituyen la base de este trabajo fueron los antecedentes en problemas de flujos en redes, cuyo propósito es lograr que los elementos concentrados en un punto inicial puedan fluir a través de enlaces hasta lograr llegar a su destino, cuidando siempre elegir la mejor ruta que permita minimizar el costo total del envío.

Los problemas de flujos en redes pueden ser resueltos por técnicas clásicas de Programación Lineal (PL), sin embargo, en aplicaciones de tiempo real, sería mucho más interesante adaptar patrones de flujo, variación en la capacidad o bien fallas en los enlaces en un tiempo de respuesta relativamente corto. En estos casos las Redes Neuronales Artificiales (RNA) son mucho más atractivas ya que permiten realizar este tipo de adaptaciones.

Por la importancia que hoy en día tienen las aplicaciones con redes neuronales y los descubrimientos e investigaciones realizadas hasta el momento surgió el interés por desarrollar este trabajo de tesis, en el cual se pretende diseñar e implementar una red neuronal artificial que permita resolver óptimamente problemas de distribución de flujo. Para ello se utilizarán técnicas de Programación Lineal (especialmente aquellas que son utilizadas en los métodos de transporte), características de las redes neuronales artificiales de retroalimentación, así como algunos aspectos importantes de métodos de optimización. La combinación de todas estas técnicas dan como resultado la implementación de redes neuronales artificiales capaces de solucionar este tipo de problemas.

1.2 Objetivo

El objetivo primordial de esta investigación es el de lograr adaptar un patrón de distribución de flujo que permita minimizar el costo total que implicará abastecer las necesidades expuestas de oferta / demanda de uno o varios centros ***fuente – sumidero***. Para esto se diseñará un sistema simulador que utilizará una Red Neuronal Artificial de distribución, permitiendo adaptarse a las características de cada problema en cuestión.

1.3 Guía de la Tesis

En el capítulo 2 se mostrarán los antecedentes de las Redes Neuronales Artificiales, la introducción a conceptos fundamentales, así como los modelos del sistema neuronal artificial.

El capítulo 3 muestra la aplicación de las Redes Neuronales para resolver problemas de Programación Lineal, así como problemas de optimización combinatoria.

El capítulo 4 presenta el estudio de la tesis, el cual se centra en la implementación de una Red Neuronal Artificial de distribución. El capítulo 5 expone la aplicación e implementación del sistema desarrollado en esta investigación.

Los resultados de esta tesis se muestran en el capítulo 6, posteriormente en el capítulo 7 encontraremos algunas conclusiones de la investigación y recomendaciones a investigaciones futuras en esta área.

Al final de la tesis se encontrarán algunas referencias de la investigación; así como apéndices complementarios a la misma.

CAPITULO 2

ANTECEDENTES

2.1 Introducción

El hombre se ha caracterizado siempre por una búsqueda constante de nuevas vías para mejorar sus condiciones de vida. Estos esfuerzos le han servido para reducir el trabajo en aquellas operaciones en las que la fuerza juega un papel primordial. Los progresos obtenidos han permitido dirigir estos esfuerzos a otros campos, como por ejemplo, a la construcción de máquinas calculadoras que ayuden a resolver de forma automática y rápida determinadas operaciones que resultan tediosas cuando se realizan a mano. Estas máquinas permiten implementar fácilmente algoritmos para resolver multitud de problemas que antes resultaban tediosos de resolver [Catalina 96].

Existe actualmente una tendencia a establecer un nuevo campo de las ciencias de la computación que integraría los diferentes métodos de resolución de problemas que no pueden ser descritos fácilmente mediante un enfoque algorítmico tradicional. Estos métodos, de una forma u otra, tienen su origen en la emulación, más o menos inteligente, del comportamiento de los sistemas biológicos.

Los desarrollos actuales de los científicos se dirigen al estudio de las capacidades humanas como una fuente de nuevas ideas para el diseño de las nuevas máquinas. Así, la inteligencia artificial es un intento por descubrir y describir aspectos de la inteligencia humana que pueden ser simulados mediante máquinas. Esta disciplina se ha desarrollado fuertemente en los últimos años teniendo aplicación en algunos campos como visión artificial, demostración de teoremas y procesamiento de información expresada mediante lenguajes humanos [Catalina 96].

Aunque este nuevo campo todavía no está perfectamente definido, ya se han establecido diferentes términos para denotarlo. En cualquier caso, se trata de una nueva forma de computación que es capaz de manejar las imprecisiones e incertidumbres que aparecen cuando se trata de resolver problemas relacionados con el mundo real (reconocimiento de formas, toma de decisiones, etc.), ofreciendo soluciones robustas y de fácil implementación. Para ello, se dispone de un conjunto de metodologías, como la lógica difusa, las **redes neuronales**, el razonamiento aproximado, los algoritmos genéticos, la teoría de caos y la teoría de aprendizaje. Aunque, en principio, se trata de enfoques diferentes, existe una tendencia a buscar combinaciones entre ellos, para lo cual son cada vez más frecuentes los encuentros entre expertos en cada una de estas disciplinas, lo cual permitirá la consolidación de este campo de las ciencias de la computación y, en consecuencia, el progreso en el desarrollo de nuevas tecnologías de procesamiento de la información.

En este capítulo se introduce precisamente una de estas formas de computación, la basada en la utilización de *redes neuronales artificiales.* Con las redes neuronales se intentará expresar la solución de problemas complejos, no como una secuencia de pasos, sino como la evolución de unos sistemas de computación inspirados en el funcionamiento del cerebro humano, y dotados por tanto de cierta "inteligencia", los cuales son la combinación de una gran cantidad de elementos simples de procesamiento

(*neuronas*) interconectados que, operando de forma masivamente paralela, consiguen resolver problemas relacionados con el reconocimiento de formas o patrones, predicción, codificación, clasificación, control y optimización.

Además, se da una panorámica histórica de la evolución de las redes neuronales, su relación con la inteligencia artificial y sus posibles aplicaciones. También se aborda, de forma general, el tema de la implementación práctica de las redes neuronales, desde las neurocomputadoras de propósito general y especial. En este sentido, es importante destacar que, aunque se pueden desarrollar aplicaciones mediante programas de simulación, codificando algoritmos de funcionamiento y aprendizaje, la verdadera potencia de las redes neuronales se pone de manifiesto precisamente mediante su implementación física en hardware con múltiples elementos de proceso que permitan explorar masivamente el paralelismo inherente a estos sistemas de computación.

2.2 Antecedentes Históricos

Diseñar y construir máquinas capaces de realizar procesos con cierta inteligencia han sido los principales objetivos y preocupaciones de los científicos a lo largo de la historia. De los intentos realizados en este sentido se han llegado a definir las líneas fundamentales para la obtención de máquinas inteligentes. En un principio, los esfuerzos estuvieron dirigidos a la obtención de autómatas, en el sentido de máquinas que realizan, con más o menos éxito, alguna función típica de los seres humanos. Pero esto no era más que el resultado del desarrollo técnico de la habilidad mecánica de los constructores de tales artefactos. Sin embargo, en esta misma línea se sigue

investigando hoy en día con herramientas enormemente sofisticadas y con resultados realmente sorprendentes, de forma que actualmente existen diversas maneras de realizar procesos similares a los inteligentes y que podemos encontrar dentro de la denominada inteligencia artificial.

Sin embargo, a pesar de disponer de herramientas y de lenguajes de programación diseñados expresamente para el desarrollo de máquinas inteligentes, existe un problema de fondo que limita enormemente los resultados que se pueden obtener : estas máquinas se implementan sobre ordenadores basados en la filosofía de funcionamiento expuesta inicialmente por Van Neumann, y se apoyan en una descripción secuencial del proceso de tratamiento de la información. El elevado nivel de desarrollo de estos ordenadores, por espectacular y complejo que haya llegado ha ser, no deja de seguir la línea antes expuesta: una máquina puramente mecánica que es capaz de realizar tareas mecánicas (de cálculo, ordenación o control) de forma increíblemente rápida, pero incapaz de obtener resultados aceptables cuando se trata de tareas sencillas, por ejemplo, para un ser humano (reconocimiento de formas, habla, etc.).

Alan Turing, en 1936, fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación; sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch y Walter Pitts, quienes, en 1943, lanzaron una teoría acerca de la forma de trabajar de las neuronas [Hilera 95]. Ellos modelaron una red neuronal simple mediante circuitos eléctricos.

En 1957, Frank Rosenblatt comenzó el desarrollo del Perceptrón. El Perceptrón es la más antigua red neuronal, y se usa hoy en día de varias formas en el reconocimiento de patrones [Hilera 95].

En 1959, Bernard Widrow y Marcial Hoff, desarrollaron el modelo ADELINE (ADAptative LINear Elements). Esta fue la primera red neuronal aplicada a un problema real (filtros adaptivos para eliminar ecos en las líneas telefónicas) [Hilera 95].

Uno de los mayores investigadores de las redes neuronales desde los años 60 hasta nuestros días es Stephen Grossberg. A partir de su extenso conocimiento filosófico, ha escrito numerosos libros y desarrollado modelos de redes neuronales [Hilera 95].

En 1969 surgieron críticas que frenaron el crecimiento que estaban experimentando las investigaciones sobre redes neuronales, pero en 1982, coincidieron numerosos eventos que hicieron resurgir el interés por las redes neuronales. John Hopfield presentó su trabajo sobre redes neuronales, en el cual describe con claridad y rigor matemático una red a la que ha dado su nombre [Catalina 96]. En el siguiente capítulo abundaremos en las características de la Red Hopfield.

Como ejemplo del resurgir de la investigación sobre redes neuronales, podemos destacar la labor patrocinada por la Oficina de Tecnología Táctica de la Agencia de Proyectos de Investigación Avanzada del Departamento de Defensa de Estados Unidos (DARPA/TTO) y llevada a cabo en el Instituto Tecnológico de Massachussets de octubre de 1987 a febrero de 1988. El resultado de dicho estudio apareció en el libro Neuronal Network Study, el cual constituye una revisión del estado actual de la tecnología de redes neuronales hasta febrero de 1988, así como sus posibles aplicaciones al área de defensa y otras áreas, tales como clasificación de patrones, robótica (especialmente control de trayectorias), visión artificial, procesamiento de señales y aplicaciones al habla.

2.3 Conceptos Fundamentales y Modelos del Sistema Neuronal Artificial

2.3.1 Neuronas Biológicas y sus Modelos Artificiales.

La teoría y modelado de las redes neuronales artificiales está inspirada en la estructura y funcionamiento de los sistemas nerviosos, donde la neurona es el elemento fundamental. Una neurona es una célula viva y, como tal, contiene los mismos elementos que forman parte de todas las células biológicas. Además, contiene elementos característicos que las diferencian. En general, una neurona consta de un cuerpo celular de aproximadamente 5 a 10 micras de diámetro esférico, del que salen una rama principal, el axón, y varias ramas más cortas, llamadas dendritas. A su vez, el axón puede producir ramas en torno a su punto de arranque, y con frecuencia se ramifica extensamente cerca de su extremo.

Una de las características que diferencia a las neuronas del resto de las células vivas, es su capacidad de comunicarse. En términos generales, las dendritas y el cuerpo celular reciben señales de entrada; el cuerpo celular las combina e integra y emite señales de salida. El axón transporta esas señales a los terminales axónicos, que se encargan de distribuir información a un nuevo conjunto de neuronas. Por lo general, una neurona recibe información de miles de otras neuronas, y a su vez, envía información a miles de neuronas más. Se calcula que en el cerebro humano existen 10^{15} conexiones, la figura 1 muestra el esquema general de una neurona biológica.

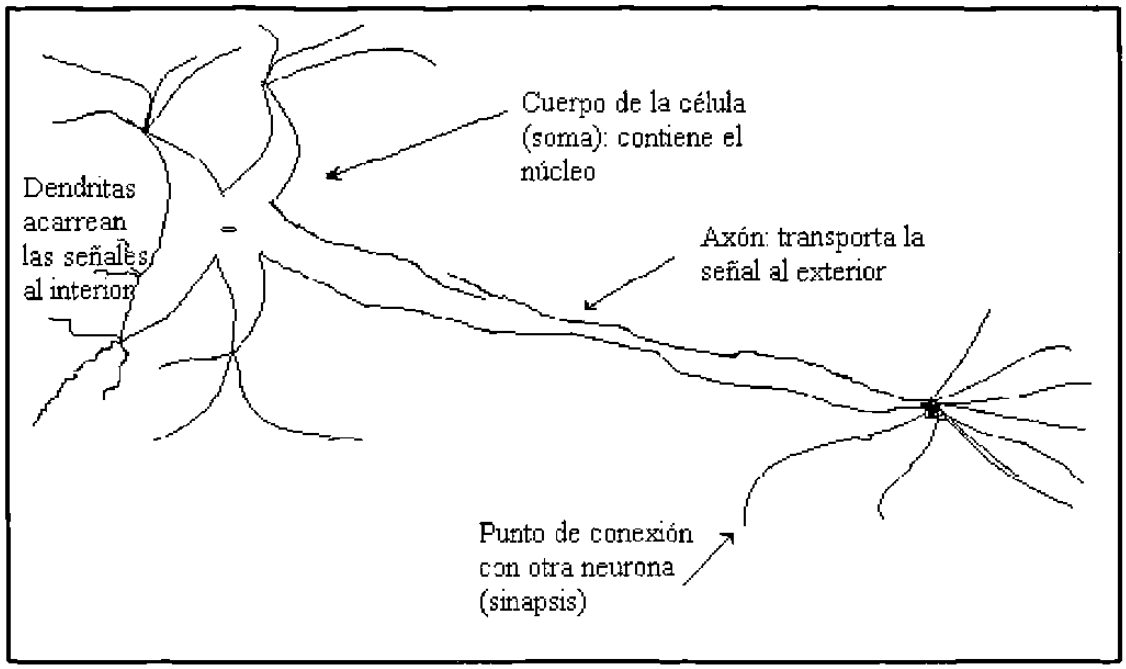


Figura 1. Forma General de una Neurona Biológica

- Modelo Neuronal de McCulloch Pitts.

La primera definición formal de un modelo neuronal sintético basado altamente en la consideración simplificada del modelo biológico descrito anteriormente, fue formulada por McCulloch Pitts (1943). El modelo de neuronas de McCulloch Pitts [Zurada 92] es mostrado en la figura 2. Las entradas x_i, desde $i = 1,2,...,n$ son 0 o 1, dependiendo de la ausencia o presencia de la entrada impulsada en el instante k. La señal de salida de la neurona es denotada como θ. La regla de salida de este modelo es definido como sigue:

$$\theta = \begin{cases} 1 \quad \text{si} \quad \sum_{i=1}^{n} w_i * x_i^k \geq T \\ 0 \quad \text{si} \quad \sum_{i=1}^{n} w_i * x_i^k < T \end{cases}$$

donde k = 0,1,2,... denota el instante de tiempo discreto, y w_i es el peso conectado a la entrada i . Note que w_i = +1 por la excitación de sinapsis, w_i = -1 por la inhibición de la sinapsis de este modelo y T es el valor de umbral de la neurona, que necesita ser excedido por la suma de pesos de la salida de la neurona.

A través de este modelo neuronal, se pueden llevar a cabo operaciones lógicas Not, Or y And, conociendo y seleccionando apropiadamente los pesos y umbrales correspondientes.

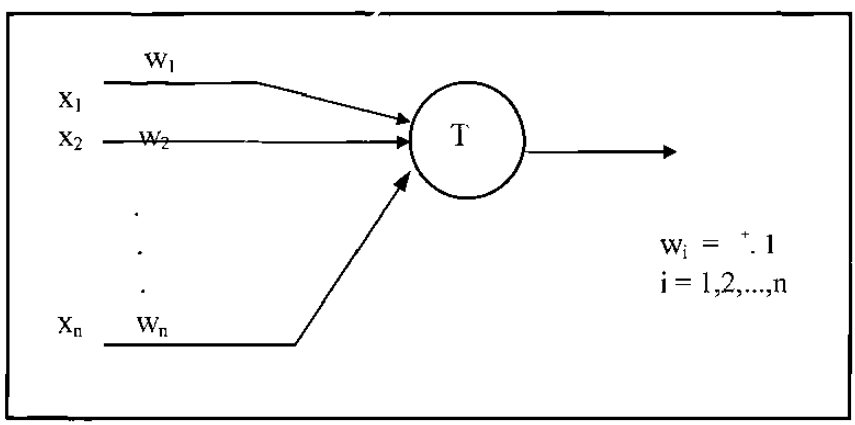


Figura 2. Modelo de McCulloch Pitts.

2.3.2 Modelos de Redes Neuronales Artificiales

El componente mínimo de una red neuronal es una neurona o elemento de procesamiento. Es un dispositivo que transforma (en el soma, o cuerpo celular) varias señales de entrada (por las dendritas) en una única salida (por el axón). Las entradas pueden proceder de otras neuronas, o bien, del exterior. La salida, asimismo, puede

transferirse a otras neuronas o funcionar como señal de salida a la red, en cuyo caso el comportamiento es ligeramente diferente en cuanto a las funciones que se le aplican o el uso final que se hace de ella.

Las señales de entrada se encuentran moduladas por un factor, llamado peso, que gradúa la importancia de la conexión existente entre la neurona receptora y el emisor de la señal (generalmente otra neurona). Una neurona es, en realidad, un procesador con una capacidad limitada de cómputo, restringida a un conjunto elemental de instrucciones (sumas y productos) y una memoria para almacenar pesos y activaciones.

En el caso más sencillo, la activación de una unidad en un determinado instante es la suma de las actividades de las unidades con las que está conectada, ponderándolas por los pesos correspondientes. En las redes más elaboradas, la activación será alguna función compleja de las señales que recibe.

- Arquitectura de una red

Para definir totalmente una red neuronal no basta con describir el comportamiento individual de sus componentes (neuronas), sino que hay que especificar, además, las interconexiones existentes entre ellas. Las neuronas se agrupan en **capas**, cada una de ellas con un conjunto de neuronas de número variable y comportamiento similar, constituyendo varias capas una **red neuronal.** Cada capa está conectada a la inmediata posterior total o parcialmente, excepto la última capa, que constituye la salida total de la red neuronal. Existen tres **tipos** de capas [Sierra 95]:

1.- **Capa de entrada.** El número y tipo de neuronas que constituyen esta capa depende de los datos de entrada al problema.

2.- **Capas intermedias.** Pueden ser más de una, dependiendo del tipo y complejidad del problema que va a resolver la red. Mediante el tratamiento adecuado de estas capas se consiguen las propiedades de generalización, extracción de características, adaptabilidad, etc., que hacen muy interesante el trabajo de las redes neuronales.

3.- **Capa de salida.** El número de neuronas de esta capa depende del formato esperado de salida de la red.

Las diferentes formas de distribuir, conectar e interrelacionar estos tres tipos de capas, junto al tipo de neuronas que constituye cada una de ellas, nos van a definir los diferentes **paradigmas** de red existentes.

En la operativa habitual de construcción y manejo de la red existen cuatro pasos a seguir:

- **Fase de conceptualización.** Partiendo del problema que hay que resolver se comprueba que su solución natural se obtiene a través de una red neuronal, y se estudia qué modelo de red de los existentes se ajustan más al problema.

- **Fase de diseño.** Se determina la arquitectura de la red, el tipo de elemento de procesamiento (función de transferencia) y el algoritmo de aprendizaje a utilizar.

- **Implementación.** Una red neuronal encargada de realizar una tarea deberá ser entrenada para su correcta realización, es decir, la arquitectura y los valores de los pesos sinápticos habrán de ser los adecuados para suministrar la respuesta correcta cuando se les presenta una entrada determinada. Al proceso en el que se determina esa configuración óptima se le denomina **entrenamiento**, y se realiza a través de la presentación de ejemplos. Se le presentarán a la red un conjunto de ejemplos de entradas asociadas con la correspondiente respuesta, y se modificarán paulatinamente

los pesos sinápticos para que cada vez sea menor la discrepancia promedio entre las respuestas de la red y las correctas.

Esta fase está dividida, a su vez, en dos pasos:

1. Se elige un conjunto significativo de entrenamiento, se selecciona el entorno de desarrollo adecuado y se entrena a la red.
2.- Una vez que ha finalizado el proceso de entrenamiento, se comprueba que su comportamiento es el esperado con un número determinado de casos de ejemplo diferentes de los utilizados en el entrenamiento.

- **Mantenimiento**. Se integra la red neuronal en el sistema de información donde va a operar habitualmente. Se hace un seguimiento del funcionamiento con casos reales.

Estos cuatro pasos se realizan iterativamente hasta que la red neuronal trabaje en el entorno de instalación tal y como se espera [Sierra 95].

El conjunto de una o varias redes neuronales, con las interfaces con el medio exterior (de entrada y salida), forman un **sistema neuronal**. En él pueden incluirse otros subsistemas, que pueden no ser de tipo neuronal, como sucede en las redes expertas, simbiosis entre un sistema experto y una red neuronal.

2.3.3 Características del Procesamiento Neuronal

Una de las principales características en una red neuronal es su capacidad de aprendizaje, razón por la que vamos a profundizar en ella. El aprendizaje permite que la red modifique su propia estructura (matriz de pesos) adaptándola hasta conseguir un algoritmo de ejecución. Este algoritmo se construye a partir de las características

extraídas de los ejemplos con los que se realiza el entrenamiento. Aquí convendría hacer una distinción entre el concepto de **aprendizaje** y el de **entrenamiento.**

El aprendizaje consiste en hacer cambios en los pesos de las conexiones entre las neuronas de la red hasta conseguir la respuesta deseada. Podemos decir que el entrenamiento es el procedimiento por el cual la red aprende y que el aprendizaje es el resultado final de ese proceso, y también que el primero es un procedimiento externo a la red, y el segundo, un procedimiento o actividad interna.

En estos sistemas, la información no se almacena en un emplazamiento físico único, ya que, como hemos dicho, la capacidad de almacenamiento individual es muy limitada. Por el contrario, la información aparece distribuida por toda la estructura de la red, concentrándose en las uniones de los distintos elementos. Esta es la base del procesamiento distribuido; las funciones proporcionadas en estos sistemas vienen dadas por la complejidad del sistema, más que por la aportación individual de cada elemento.

El aprendizaje en una Red Neuronal Artificial puede darse de dos formas:

a) Aprendizaje Supervisado

b) Aprendizaje No Supervisado

La diferencia fundamental entre ambos estriba en la existencia, o no, de un agente externo (supervisor) que controle el proceso de aprendizaje de la red. Considerando la naturaleza del proyecto a desarrollar en esta tesis se analizará el aprendizaje supervisado.

El aprendizaje supervisado se caracteriza porque el proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo que determina la respuesta que debería generar la red a partir de una entrada determinada. El supervisor comprueba la salida de la red y en el caso de que ésta no coincida con la deseada se procederá a

modificar los pesos de las conexiones, con el fin de conseguir que la salida obtenida se aproxime a la deseada.

- Tipos de Asociación entre la Información de entrada y salida

Existen dos formas primarias de realizar esta asociación entre entrada/salida, las cuales se corresponden con la naturaleza de la información almacenada en la red. Una primera sería la denominada heteroasociación, que se refiere al caso en el que la red aprende parejas de datos [(A_1, B_1), (A_2, B_2), . . ,(A_N, B_N)], de tal forma que cuando se presente cierta información de entrada A_i, deberá responder generando la correspondiente salida asociada B_i. La segunda se conoce como autoasociación, donde la red aprende cierta información A_1, A_2, ..A_N, de tal forma que cuando se le presenta una información de entrada realizará una autocorrelación, respondiendo con uno de los datos almacenados, el más parecido al de entrada.

Estos dos mecanismos de asociación dan lugar a dos tipos de redes neuronales: las redes heteroasociativas y las autoasociativas. Una red heteroasociativa podría considerarse aquella que computa cierta función, que en la mayoría de los casos no podrá expresarse analíticamente, entre un conjunto de entradas y un conjunto de salidas, correspondiendo a cada posible entrada una determinada salida. Por otra parte, una red autoasociativa es una red cuya principal misión es reconstruir una determinada información de entrada que se presenta incompleta o distorsionada (le asocia el dato almacenado más parecido).

- Representación de la información de entrada y salida

Las redes neuronales pueden también clasificarse en función de la forma en que se representa la información de entrada y las respuestas o datos de salida. Así, en un gran número de redes, tanto los datos de entrada como de salida son de naturaleza analógica; es decir, son valores reales continuos, normalmente estarán normalizados y su valor absoluto será menor que la unidad. Cuando esto ocurre, las funciones de activación de las neuronas serán también continuas, del tipo lineal o sigmoidal (figura 3). Otras redes, por el contrario, solo admiten valores discretos o binarios {0,1} en su entrada, generando también unas respuestas en la salida de tipo binario. En este caso, las funciones de activación de las neuronas serán de tipo escalón (figura 4).

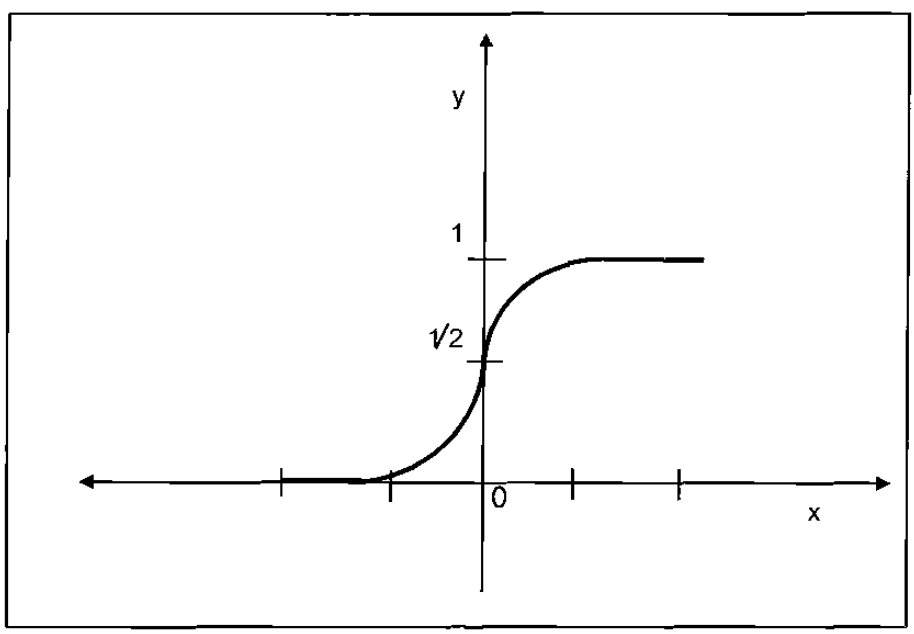


Figura 3. Función de Activación Sigmoidal : $y = 1/(1 + e^{-x})$

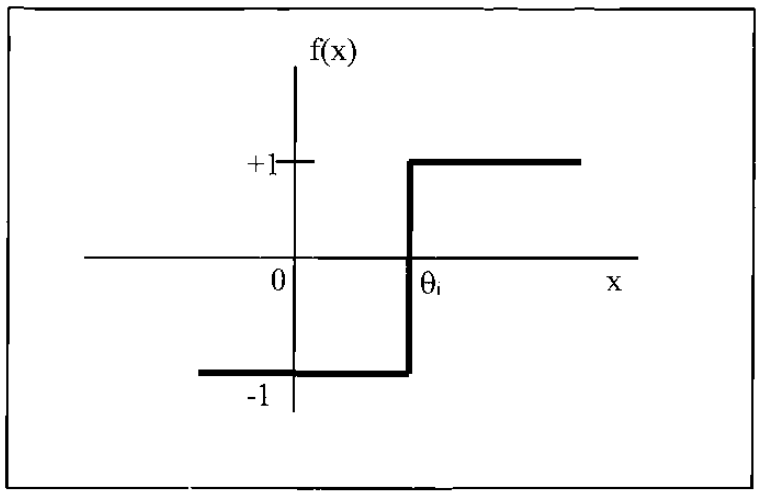


Figura 4. Función Tipo Escalón

Donde :

$$f(x) = \begin{cases} +1 \quad \text{para } x > \theta_i \\ -1 \quad \text{para } x < \theta_i \end{cases}$$

2.4 Procedimiento de operación con Redes Neuronales

Los Sistemas Neuronales Artificiales surgen con la idea de tomar las características esenciales de la estructura neuronal del cerebro para crear sistemas que lo mimeticen en parte, mediante sistemas electrónicos o mediante simulación por computadora, aprovechando sus propiedades de cálculo. Estos sistemas están compuestos por multitud de procesadores simples que operan sobre la base de reconocimiento de patrones y que pueden adquirir, almacenar y utilizar conocimiento experimental, obtenido a partir de ejemplos.

Esta forma de adquirir el conocimiento es una de sus características más destacables : no se programa de forma directa, como en los sistemas expertos, sino que se adquiere a partir de ejemplos, por ajuste de parámetros de las neuronas mediante un algoritmo de aprendizaje.

Sistemas expertos y redes neuronales se asemejan en cuanto al objetivo de dar forma al conocimiento, pero son radicalmente opuestos en cuanto a cómo aspiran a conseguirlo [Serrano 96]. Como vemos, los sistemas expertos se acercarían más al razonamiento deductivo y las redes neuronales al inductivo. La gestión empresarial utiliza frecuentemente ambos esquemas de razonamiento, por lo que ambas técnicas tienen cabida. Además, ambos modelos son perfectamente compatibles, de forma que se pueden integrar en un único sistema, que se suele conocer como red experta. Probablemente este tipo de sistemas mixtos sí son capaces de recoger las ventajas de ambos modelos.

Los elementos básicos de neurocomputación son las neuronas artificiales. Como dijimos anteriormente estas se agrupan en capas, constituyendo una red neuronal. Una o varias redes, más las interfaces con el entorno, conforman el sistema global. Un conjunto de capas constituyen una red neuronal, aunque también existen estructuras de una única capa. Una red neuronal determinada está confeccionada y entrenada para llevar a cabo una labor específica. Existen diferentes modelos de conexiones entre capas, en general se suelen distinguir dos básicos: las arquitecturas hacia adelante o ***feedforward*** y las retroalimentadas o ***feedback***. En las arquitecturas *feedforward,* la información siempre se propaga hacia adelante. En las arquitecturas de retroalimentación, las señales pueden en ocasiones fluir hacia atrás a través de lazos de retroalimentación.

El procedimiento para operar con redes neuronales queda reflejado en la figura 5.

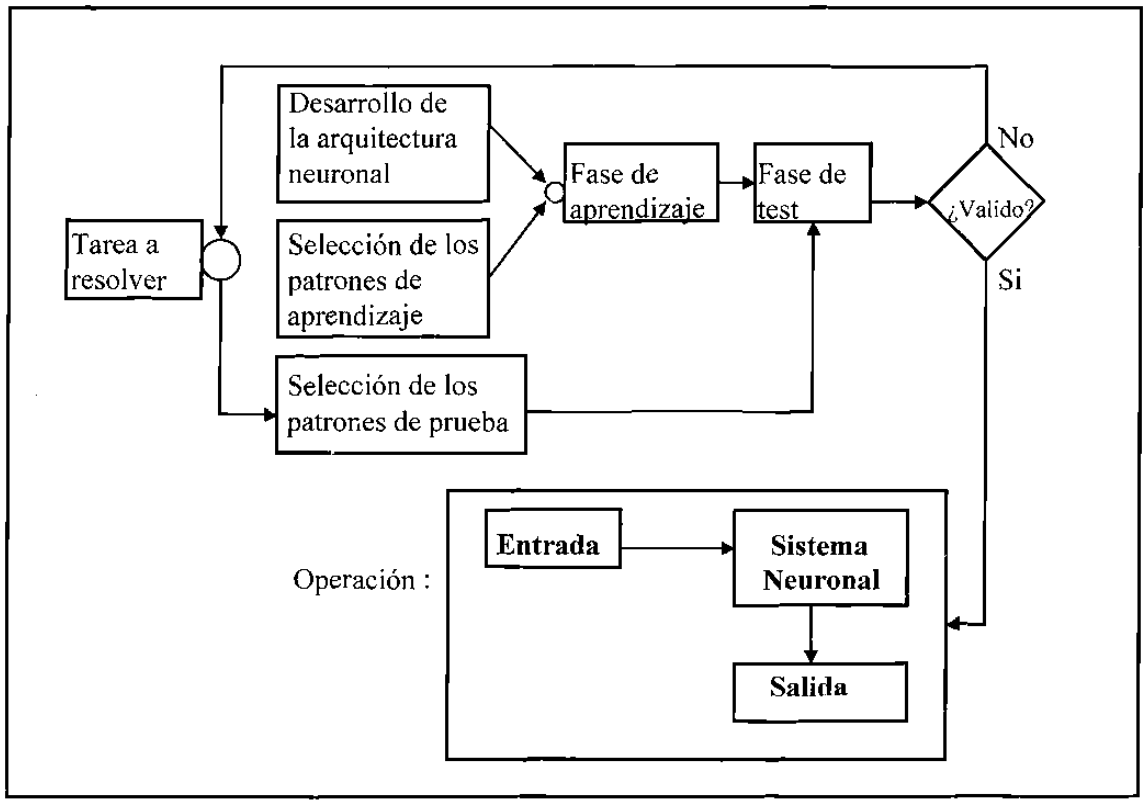


Figura 5. Procedimiento de operación aplicando Redes Neuronales

Originalmente la red neuronal no dispone de ningún tipo de conocimiento útil almacenado. Para que ejecute una tarea es preciso entrenar o enseñar a la red neuronal. El entrenamiento se realiza mediante ***patrones-ejemplo***. Existen dos tipos de aprendizaje (como mencionamos anteriormente): supervisado y no supervisado. Si la red utiliza un tipo de aprendizaje supervisado debemos proporcionarle parejas de patrones entrada-salida y la red neuronal aprende a asociarlos. En terminología estadística equivale a los modelos en los que hay vectores de variables independientes y dependientes. Si el entrenamiento es no supervisado, únicamente debemos suministrar a la red los datos de entrada para que extraiga los rasgos característicos esenciales [Serrano 96].

Durante la fase de aprendizaje, en la mayor parte de los modelos, se produce una variación de los pesos sinápticos, es decir, de la intensidad de interacción entre las neuronas, lo que en terminología estadística equivale a calcular los coeficientes de las funciones de ajuste.

Los diferentes modelos neuronales se diferencian en el modelo de neurona, su organización, forma de las conexiones y el algoritmo de aprendizaje que emplea. Existen multitud de modelos y variantes, como son el modelo Hopfield, la resonancia adaptiva o ART, etc. [Serrano 96]

2.5 Redes Neuronales con Conexión hacia Adelante

Este tipo de redes se caracterizan por su arquitectura en niveles y conexiones estrictamente hacia adelante entre las neuronas. Esta redes utilizan aprendizaje supervisado.

Este grupo incluye el perceptrón y la red backpropagation, en donde la red backpropagation es una de las más utilizadas hoy en día por sus características y el tipo de problemas que soporta.

Teniendo en cuenta las características del trabajo a desarrollar en esta tesis, nos centraremos en el análisis de redes que trabajen con el algoritmo backpropagation.

2.5.1 Red Backpropagation

En 1986, Rumelhart, Hinton y Williams, basándose en los trabajos de otros investigadores, formalizaron un método para que una red neuronal aprendiera la asociación que existe entre los patrones de entrada a la misma y las clases correspondientes de salida esperada, utilizando más niveles de neuronas que los que utilizó Rosenblatt para desarrollar el perceptron [Hayking 94],[Hilera 95]. Este método, conocido en general como backpropagation (propagación del error hacia atrás), está basado en la generalización de la regla delta (también conocida como regla del mínimo error cuadrado medio, ya que trata de minimizar la diferencia entre el valor observado y el deseado en la salida de la red). A pesar de sus propias limitaciones, ha ampliado de forma considerable el rango de aplicaciones de las redes neuronales.

El algoritmo de propagación hacia atrás, o retropropagación, es una regla de aprendizaje que se puede aplicar en modelos de redes con más de dos capas de neuronas. Una característica importante de este algoritmo es la representación interna del conocimiento que es capaz de organizar en la capa intermedia de las neuronas para conseguir cualquier correspondencia entre la entrada y la salida de la red. En algunas ocasiones, es imposible encontrar los pesos adecuados para establecer la correspondencia entre la entrada y la salida mediante una red sin capas intermedias. Con una capa de neuronas ocultas, sí es posible establecer dicha correspondencia.

De forma simplificada, el funcionamiento de una red backpropagation (backpropagation net, BPN) consiste en un aprendizaje de un conjunto predefinido de pares entradas - salida dados como ejemplo, empleando un ciclo propagación adaptación de dos fases: primero se aplica un patrón de entrada como estímulo para la primera capa de las neuronas de la red, se va propagando a través de todas las capas superiores hasta generar una salida, se compara el resultado obtenido en las neuronas de salida con la salida que se desea obtener y se calcula un valor del error

para cada neurona de salida. A continuación, estos errores se transmiten hacia atrás, partiendo de la capa de salida, hacia todas la neuronas de la capa intermedia que contribuyan directamente a la salida, recibiendo el porcentaje de error aproximado a la participación de la neurona intermedia en la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido un error que describa su aportación relativa al error total. Basándose en el valor del error recibido, se reajustan los pesos de conexión de cada neurona, de manera que en la siguiente vez que se presente el mismo patrón, la salida esté más cercana a la deseada; es decir, el error disminuya.

La importancia de la red backpropagation consiste en su capacidad de autoadaptar los pesos de las neuronas de la capas intermedias para aprender la relación que existe entre un conjunto de patrones dados como ejemplo y sus salidas correspondientes. Después del entrenamiento puede aplicar esa misma relación, a nuevos vectores de entrada con ruido o incompletas, dando una salida activa si la nueva entrada es parecida a las presentadas durante el aprendizaje. Esta característica importante, que se exige a los sistemas de aprendizaje, es la capacidad de generalización, entendida con la facilidad de entrenamiento. La red debe encontrar una representación interna que le permita generar las salidas deseadas cuando se le dan las entradas de entrenamiento, y que pueden aplicar, además, a entradas no presentadas durante la etapa de aprendizaje para clasificarlas según las características que compartan con los ejemplos de entrenamiento.

2.5.2 Estructura y Aprendizaje de la Red Backpropagation

En una red backpropagation existe una capa de entrada con **n** neuronas, una capa de salida con **m** neuronas y al menos una capa oculta de neuronas internas. Cada neurona de una capa (excepto las de entrada) recibe entradas de todas las neuronas de la capa

anterior y envía su salida a todas las neuronas de la capa posterior (excepto las de salida). No hay conexiones hacia atrás feedback ni laterales entre neuronas de la misma capa.

Como se comentó anteriormente, la aplicación del algoritmo backpropagation tiene dos fases, una hacia adelante y otra hacia atrás. Durante la primera fase el patrón de entrada es presentado a la red y propagado a través de las capas hasta llegar a la capa de salida. Obtenidos los valores de salida de la red, se inicia la segunda fase, comparándose estos valores con la salida esperada para obtener el error. Se ajustan los pesos de la última capa proporcionalmente al error. Se pasa a la capa anterior con una retropropagación del error (backpropagation), ajustando convenientemente los pesos y continuando con este proceso hasta llegar a la primera capa. De esta manera se han modificado los pesos de las conexiones de la red para cada ejemplo o patrón de aprendizaje del problema, del que conocíamos su valor de entrada y la salida deseada que debería generar la red ante dicho patrón.

A diferencia de la regla delta en el caso del perceptrón, la técnica de retropropagación o generalización de la regla delta, requiere el uso de neuronas cuya función de activación sea de derivada continua. Generalmente, la función utilizada será de tipo sigmoidal (figura 3).

2.5.3 Consideraciones sobre el Algoritmo Backpropagation

El algoritmo de backpropagation encuentra un valor mínimo de error (local o global) mediante la aplicación del método de máximo descenso. Cada punto de la superficie de la función error corresponde a un conjunto de valores de pesos de la red. Con él, siempre que se realiza un cambio en todos los pesos de la red, se asegura el descenso

por la superficie del error hasta encontrar el valle más cercano, lo que puede hacer que el proceso de aprendizaje se detenga en un mínimo local de error. Por tanto, uno de los problemas que presenta este algoritmo de entrenamiento de redes multicapa es que busca minimizar la función de error, pudiendo caer en un mínimo local o en algún punto estacionario, con lo cual no se llega a encontrar el mínimo global de la función del error. Sin embargo, ha de tenerse en cuenta que no tiene porqué alcanzarse el mínimo global en todas las aplicaciones, sino que puede ser suficiente con un error mínimo preestablecido.

2.5.4 Control de la Convergencia

En las técnicas de gradiente decreciente es conveniente avanzar por la superficie de error con incrementos pequeños de los pesos. Esto se debe a que tenemos una información local de la superficie y no se sabe lo lejos o lo cerca que se está del punto mínimo. Con incrementos grandes, se corre el riesgo de pasar por encima del punto mínimo sin conseguir estacionarse en él. Con incrementos pequeños, aunque se tarde más en llegar, se evita que ocurra esto.

El elegir un incremento de longitud paso adecuado influye en la velocidad con la que converge el algoritmo. Esta velocidad se controla a través de la constante de proporcionalidad o tasa de aprendizaje α. Normalmente, α debe ser un número pequeño (del orden de 0,05 a 0,25), para asegurar que la red llegue a asentarse en una solución. Un valor pequeño de α, significa que la red tendrá que hacer un gran número de iteraciones. Si esa constante es muy grande, los cambio de pesos son muy grandes, avanzando muy rápidamente por la superficie de error, con el riesgo de saltar el mínimo y estar oscilando alrededor de él, pero sin poder alcanzarlo.

2.5.5 Dimensionamiento de la Red. Número de Neuronas Ocultas

No se pueden dar reglas concretas para determinar el número de neuronas o el número de capas de una red para resolver un problema concreto. Lo mismo ocurre a la hora de seleccionar el conjunto de patrones de entrenamiento. En todos estos casos, lo único que se puede obtener son unas cuantas ideas generales deducidas de la experiencia de numerosos autores.

Respecto al número de capas de la red, en general tres capas son suficientes (entrada - oculta - salida), sin embargo, hay veces en que un problema es más fácil de resolver (la red aprende más rápidamente) con más de una capa oculta. El tamaño de las capas, tanto de entrada como de salida, suele venir determinado por la naturaleza de la aplicación. En cambio, decidir cuántas neuronas debe tener la capa oculta no suele ser tan evidente.

El número de neuronas ocultas interviene en la eficiencia de aprendizaje y de generalización de la red. No hay ninguna regla que indique el número óptimo, en cada problema se debe ensayar con distintos números de neuronas para organizar la representación interna y escoger la mejor. La idea más utilizada, sobre todo en los sistemas simulados, consiste en tener el menor número posible de neuronas en la capa oculta, porque cada una de ellas supone una mayor carga de procesamiento en el caso de una simulación software. En un sistema implementado en hardware, este problema no es crucial, sin embargo, sí habrá que tener presente el problema de comunicación entre los distintos elementos de proceso.

Es posible eliminar neuronas ocultas si la red logra obtener una convergencia sin problemas, determinando el número final en función del rendimiento global del sistema. Si por el contrario, la red no logra obtener una convergencia, es posible que sea necesario aumentar este número. Por otro lado, examinando los valores de los pesos de

las neuronas ocultas periódicamente en la fase de aprendizaje, se pueden detectar aquellas cuyos pesos cambian muy poco durante el aprendizaje respecto a sus valores iniciales, y reducir por tanto el número de neuronas que apenas participan en el proceso de aprendizaje.

2.6 Aplicaciones de Redes Neuronales

A continuación, se muestran algunas de las múltiples aplicaciones que pueden darse a las redes neuronales [Catalina 96].

Visión artificial: Se emplean modelos de redes neuronales que son capaces de emular características del funcionamiento visual humano permitiendo, por ejemplo, el reconocimiento de imágenes textuales en color, el aprendizaje para determinar posiciones a partir de la información proveniente de dos cámaras, y representación de la visión binocular.

Reconocimiento y categorización de patrones: Estas redes emplean las arquitecturas de la teoría de Resonancia Adaptiva o ART. Entre otras aplicaciones se encuentran el reconocimiento de caracteres manuscritos, autorización de descubiertos bancarios y clasificación de cromosomas.

Procesos Químicos: Dos aplicaciones posibles son: el control de la temperatura en un reactor químico y el control de procesos químico-orgánicos no lineales.

Control Motor: Permiten resolver el problema cinemático inverso en manipuladores y robótica móvil, consistente en determinar la secuencia de movimientos que deben realizar las distintas partes del robot para alcanzar una posición deseada. También permiten el aprendizaje de la dinámica del manipulador, es decir, de la generación de las fuerzas y pares que hay que aplicar para producir un movimiento determinado.

Otros Campos: como la predicción económica y problemas de gestión, aprendizaje preventivo, etc.

2.7 Perspectivas Futuras

El interés de los sistemas neuronales artificiales está basado en las perspectivas científicas y económicas. Podemos asegurar que las redes neuronales artificiales no van a reemplazar a las computadoras convencionales ya que estas son ahora muy baratas, extremadamente rápidas y concretas para la ejecución de subrutinas matemáticas, procesamiento de texto y muchas otras tareas.

Una vez que exista la posibilidad de utilizar sistemas neuronales artificiales, se podrían utilizar para simular sistemas físicos que son generalmente expresados por redes paralelas. Aunque las mejores aplicaciones de la redes neuronales podrían ser las involucradas en la clasificación, asociación y razonamiento.

La mejor esperanza para el uso de un sistema neuronal artificial, sería en la aplicación de las áreas que no han sido atacadas por computadoras convencionales, como lo son: en los requerimientos del gusto humano, tales como la inferencia y percepción del habla y visión, los cuales son los mejores candidatos de esta aplicación. Estos sucesos podrían traer mayor y mejores aplicaciones en control de tiempo real en sistemas complejos.

Las neurocomputadoras de hoy son también computadoras convencionales programables que corren software de simulación neuronal. Tal simulación proviene de arquitecturas apropiadas, o tableros neurocomputarizados, que hacen que la simulación de la red neuronal opere rápidamente y con más precisión comparada con la arquitectura estándar.

Afortunadamente, la fabricación de sistemas neuronales artificiales sigue los paradigmas inventados en la computación. Desde 1989, cuando AT&T fabricó el primer circuito neuronal integrado de memoria, se pudo observar la proliferación de microelectrónica en chips en redes neuronales.

Podemos expresar que los sistemas neuronales artificiales pueden ser usados en aplicaciones implicando visión, lenguaje, toma de decisiones y razonamiento; pero también en procesos de señales tal como filtros, detectores y sistemas de control de calidad. También, las redes neuronales pueden ofrecer soluciones en casos en que procesamientos algorítmicos o soluciones analíticas son difíciles de encontrar.

La tecnología de redes neuronales artificiales es aún muy nueva, y es desarrollada rápidamente. Somos testigos de la rápida expansión de las redes neuronales basadas en máquinas inteligentes. Con esperanza, esta expansión podría mejorar la calidad de nuestras vidas y hacer las tareas difíciles, fáciles de realizar.

2.8 Conclusiones del Capítulo

Las redes neuronales son otra forma de emular las características propias de los humanos: la capacidad de memorizar y asociar hechos. Si examinamos con atención aquellos problemas que no pueden expresarse a través de un algoritmo nos daremos cuenta que todos ellos tienen una característica común: la experiencia. El hombre es capaz de resolver estas situaciones acudiendo a la experiencia acumulada. Así, parece claro que una forma de aproximarse al problema consista en la construcción de sistemas que sean capaces de reproducir esta característica humana. En conclusión, las redes neuronales son un modelo artificial y simplificado del cerebro humano, que es l ejemplo más perfecto de sistema que es capaz de adquirir conocimiento a través de la experiencia.

Las redes neuronales alcanzan cada vez mayor auge, teniendo multitud de aplicaciones en campos diversos y dando soluciones sencillas a problemas cuya resolución resulta complicada cuando se emplean máquinas algorítmicas.

CAPITULO 3

LAS REDES NEURONALES Y LOS PROBLEMAS DE OPTIMIZACION

3.1 Introducción

En este capítulo se analizarán aspectos importantes acerca de la solución a problemas de optimización y en particular de programación lineal. Podemos definir problema de optimización como aquel problema que tenga como objetivo el mejoramiento del sistema que se estudie, siendo obvio que para mejorar este sistema es necesario que exista por lo menos una solución para el mismo. Un problema de programación lineal es aquel en el que tanto la función objetivo como sus restricciones son lineales. Matemáticamente tiene la forma:

$$\text{Opt.} \qquad c^T x$$

Sujeto a:

$$Ax \leq b$$

Donde :

- c es un vector de $\mathbb{R}^n$,
- A una matriz m x n, y
- b un vector de $\mathbb{R}^m$

En los últimos cuarenta años, los investigadores han propuesto diversos sistemas dinámicos para resolver problemas de programación lineal. Esta metodología fue propuesta por primera vez por Pyne y más tarde se reportaron estudios realizados por Rybashov, Karpinskaya [Zak 95], entre otros. Los sistemas dinámicos para problemas de optimización son especialmente usados en aplicaciones de tiempo real con funciones de costo dependientes de tiempo, optimización en línea para robots o dirección satelital, etc. Sin embargo, las características de estos sistemas difieren por ejemplo en velocidad de convergencia, estabilidad, complejidad y arquitectura.

La programación lineal ha sido ampliamente usada en aplicaciones en las áreas de producción, economía, ciencias sociales y planeación gubernamental, por lo que hace muy interesante su estudio y sus aplicaciones con nuevas tecnologías tales como las redes neuronales.

Los algoritmos basados en redes neuronales para resolver problemas de programación lineal pueden tener apariencia de algoritmos infinitos. Los algoritmos infinitos pueden algunas veces ser mejores que los finitos sobre problemas finitos [Zak 95]. Por eso la importancia de investigar el aprovechamiento de las redes neuronales para la solución de problemas de programación lineal como una clase alternativa de algoritmos matemáticamente infinitos.

En este capítulo se analizarán tres clases de modelos de redes neuronales para resolver problemas de programación lineal, se compararán cada una de las clases en términos de complejidad del modelo y precisión de solución. Además hablaremos de la importancia de las redes neuronales en el problema del agente viajero, así como en problemas de optimización combinatoria y la red Hopfield.

3.2 Análisis de algunos modelos de redes neuronales para resolver problemas de programación lineal

La red neuronal de Tank y Hopfield [Adenso 96] para la resolución de problemas de programación lineal fue una de las causas que revivieron el interés en aplicaciones de circuitos analógicos para resolver problemas de optimización. Su trabajo fue la inspiración de investigadores para estudiar otros modelos de redes neuronales para solucionar problemas de programación lineal y no lineal. Consecuentemente, surgen suficientes modelos de redes neuronales para resolver problemas de programación lineal.

Aquí comentaremos aspectos importantes de tres diferentes clases de modelos de redes neuronales, el modelo Kennedy y Chua, el modelo Rodríguez-Vázquez y el modelo descrito por Stanislaw H. Zak [Zak 95] propuesto recientemente.

Tank y Hopfield y Kennedy y Chua [Adeson 96] usan el concepto de función energía en el análisis y síntesis de sus modelos propuestos. El mínimo de la función energía debe coincidir con la solución óptima del problema de programación lineal. La función energía puede ser vista como función penalidad. Los modelos propuestos por Rodríguez-Vázquez usan el método de penalidad directamente, el cual convierte problemas de optimización con restricciones en problemas de optimización sin restricciones. Debemos hacer notar que los dos primeros modelos requieren que todas las restricciones sean representadas por desigualdades, mientras que el descrito por Stanislaw Zak [Zak 95] usa dos funciones penalidad, una para las restricciones de igualdad y otra para la desigualdad.

El modelo de redes neuronales propuesto por Kennedy y Chua y Rodríguez-Vázquez puede resolver problemas de programación lineal de la forma:

$$
\begin{aligned}
(3.1) \qquad & \text{minimizar} \quad c^T x \\
& \text{sujeto a} \\
& A_1 x = e_1 \\
& A_2 x \leq e_2 \\
& x \geq 0
\end{aligned}
$$

donde $A_1 \in \mathbb{R}^{m_E \times n}$, $A_2 \in \mathbb{R}^{m_I \times n}$, $e_1 \in \mathbb{R}^{m_E}$, $e_2 \in \mathbb{R}^{m_I}$.

Sin embargo, la implementación del problema de programación lineal (3.1) usa dos redes neuronales, requiriendo que el problema (3.1) sea representado como:

$$
\begin{aligned}
(3.2) \qquad & \text{minimizar} \quad c^T x \\
& \text{sujeto a} \\
& g(x) = \begin{pmatrix} g_1(x) \\ . \\ . \\ g_{2\,m_E + m_{I+n}}(x) \end{pmatrix} = \hat{A}x - \hat{e} \geq 0
\end{aligned}
$$

donde :

$$
\hat{A} = \begin{pmatrix} A_1 \\ -A_1 \\ -A_2 \\ -I_n \end{pmatrix} \quad , \quad \hat{e} = \begin{pmatrix} e_1 \\ -e_1 \\ -e_2 \\ 0 \end{pmatrix}
$$

$\mathbf{I_n}$ es la matriz identidad n x n, $\mathbf{O}$ es un vector columna con elementos cero, y $\mathbf{c}$, $x \in \mathbb{R}^m$. Note que se tiene representada una restricción de igualdad como dos restricciones de desigualdad, $g_i(x) = 0$ es representado como $g_i(x) \geq 0$ y $-g_i(x) \geq 0$. Alternativamente, como fue sugerido por Barther [Zak 93], pudiéramos representar una restricción de igualdad por una restricción desigualdad, de forma que, $g_i(x) = 0$ puede ser representada como $-(g_i(x))^2 \geq 0$.

Para simplificar nuestra notación, podemos definir:

(3.3) $\nabla g(x) = \hat{A}^T$,

$g_i(x) = -\min(g_i(x), 0)$

(3.4) $$g_i^-(x) = \begin{cases} 0 & \text{si } g_i(x) \geq 0 \\ -g_i(x) & \text{en otro caso} \end{cases}$$

y $g^-(x) = [g_1^-(x), \ldots, g_{2^{mE+mI+n}}^-(x)]^T$

- Modelo Kennedy and Chua

El modelo propuesto por Kennedy and Chua proporciona un circuito de programación canónico dinámico no lineal como se muestra en la figura 6.

Note que en esa figura, $g_j(x)$, $c^T = [c_1, \ldots, c_n]$ y x, son definidos en (3.2), y $C_1, \ldots C_n$ son las capacitancias; definiendo la ocurrencia, i_j, $j = 1, \ldots, 2m_E + m_{I+n}$, en cada lazo sobre el lado izquierdo de la figura 6 como:

(3.5) $i_j = \phi_j(g_j(x))$

Aplicando la ley de Kirchhoff para el circuito del lado derecho de la figura 6, tenemos:

$$\bar{C}_k + \sum_{j=1}^{2m_E+m_{I+n}} i_j \frac{\partial g_j(x)}{\partial x_k} + C_k \frac{dx_k}{dt} = 0, \quad k = 1, \ldots, n$$

Resolviendo para dx_k / dt obtenemos :

(3.6) $$\frac{dx_k}{dt} = -\frac{1}{C_k}\left[c_k + \sum_{j=1}^{2m_E+m_{I+n}} i_j \frac{\partial g_j(x)}{\partial x_k}\right], \quad k = 1, \ldots, n$$

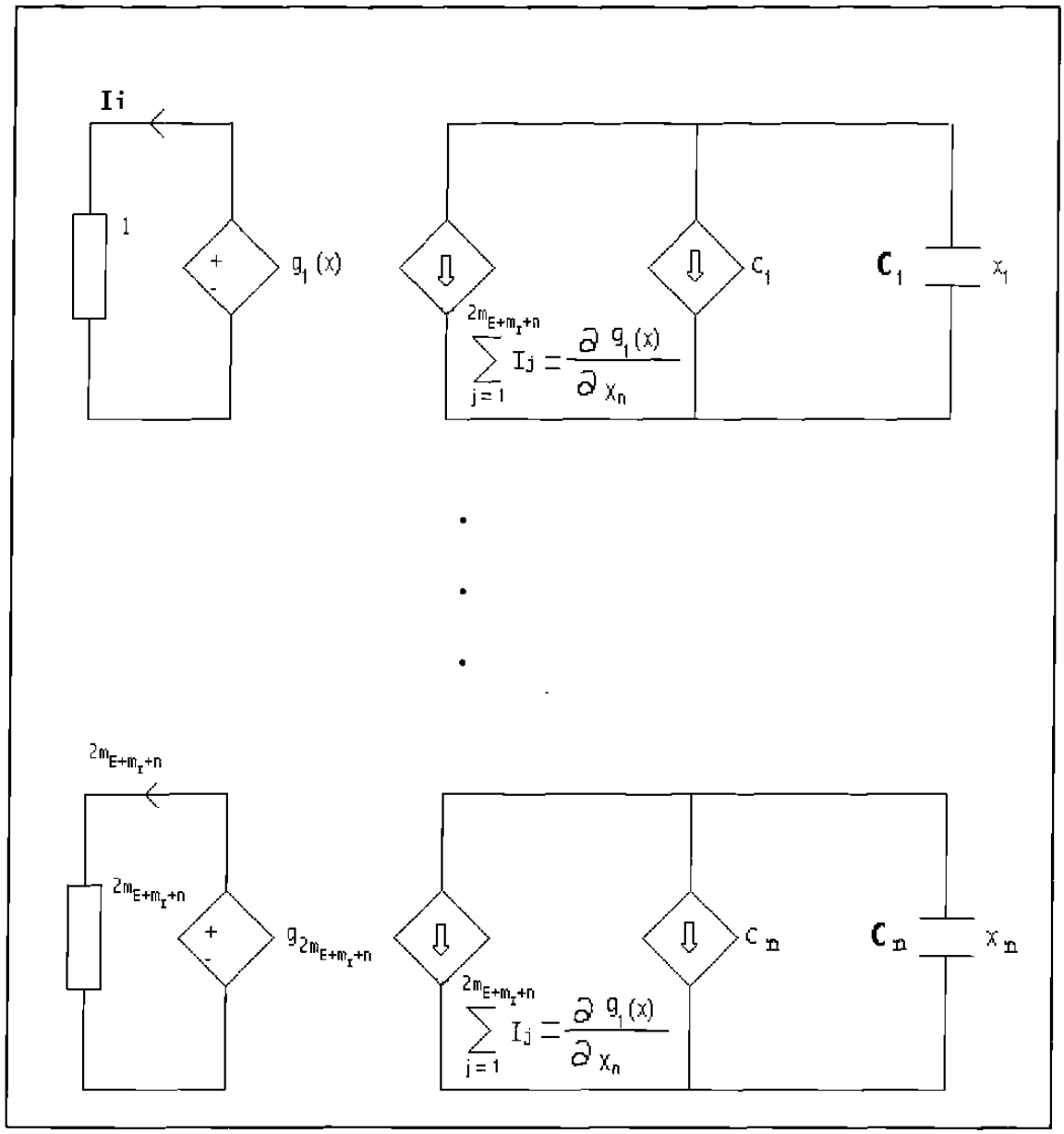


Figura 6. Modelo de red de Kennedy-Chua para resolver problemas de PL

Ahora, consideremos la siguiente función, la cual es referenciada en la literatura como la función energía computacional del circuito:

$$E = E(x(t)) = c^T x + \sum_{j=1}^{2m_E+m_{I+n}} \int_0^{g_j(x)} \phi_j(s)ds$$

Tomando la derivada de E con respecto al tiempo y teniendo en cuenta (3.5) y (3.6) tenemos:

$$(3.7)\quad \frac{dE}{dt} = \sum_{k=1}^{n} \frac{\partial E}{\partial x_k} \frac{dx_k}{dt}$$

$$= \sum_{k=1}^{n} \left(c_k + \sum_{j=1}^{2m_E+m_{I+n}} \phi_j (g_j (x)) \frac{\partial g_j (x)}{\partial x_k} \right) \frac{dx_k}{dt}$$

$$= \sum_{k=1}^{n} \left(c_k + \sum_{j=1}^{2m_E+m_{I+n}} i_j \frac{\partial g_j (x)}{\partial x_k} \right) \frac{dx_k}{dt}$$

$$= -\sum_{k=1}^{n} \left(C_k + \frac{dx_k}{dt} \right) \frac{dx_k}{dt}$$

$$= -\sum_{k=1}^{n} C_k \left(\frac{dx_k}{dt} \right)^2 \leq 0 \qquad \text{para todo } \mathbf{t}$$

Kenedy and Chua usan (3.7) y la hipótesis de que E está acotada inferiormente para concluir que la trayectoria del sistema puede alcanzar un estado estable. Este modelo puede ser presentado por:

$$\dot{x} = C^{-1}(-c - s \nabla g(x)\, g^{-}(x)), \text{ donde}$$

$$C = \begin{pmatrix} C_1 & & 0 \\ & \ddots & \\ 0 & & C_n \end{pmatrix}, \quad \text{y} \quad s > 0 \text{ es el parámetro penalidad.}$$

Así, si tenemos $C = I_n$ entonces la función energía correspondiente al circuito está dada por:

$$E(x) = c^T x + (s/2) \sum_{j=1}^{2m_E+m_{I+n}} (g^-_j(x))^2$$

La función energía puede ser vista como una función penalidad "inexacta". Por esta razón sus trayectorias convergerán a una solución del correspondiente problema de programación lineal cuando $s \rightarrow \infty$. Sin embargo, la opción $s = \infty$ es impráctica y la opción **s** muy grande es indeseable.

- Modelo Rodríguez – Vázquez

El modelo expuesto por Rodríguez – Vázquez aligera el problema de selección del parámetro **s**, proponiendo el siguiente modelo de red neuronal para solucionar problemas de programación lineal.

$$\dot{x} = -u(x)\, c - s\,(1- u(x))\nabla g(x)\, v(x),$$

donde :

$$v(x) = [v_1, \ldots, v_{2mE+mI+n}]^T$$

$$u(x) = \begin{cases} 1 & \text{si } g_j \geq 0 \text{ para toda } j = 1, \ldots, 2m_E+m_{I+n} \\ 0 & \text{en otro caso} \end{cases}$$

$$v_j(x) = \begin{cases} 1 & \text{si } g_j(x) \geq 0 \\ 0 & \text{en otro caso} \end{cases}$$

y $s > 0$. El diagrama de bloque de este modelo es mostrado en la figura 7.

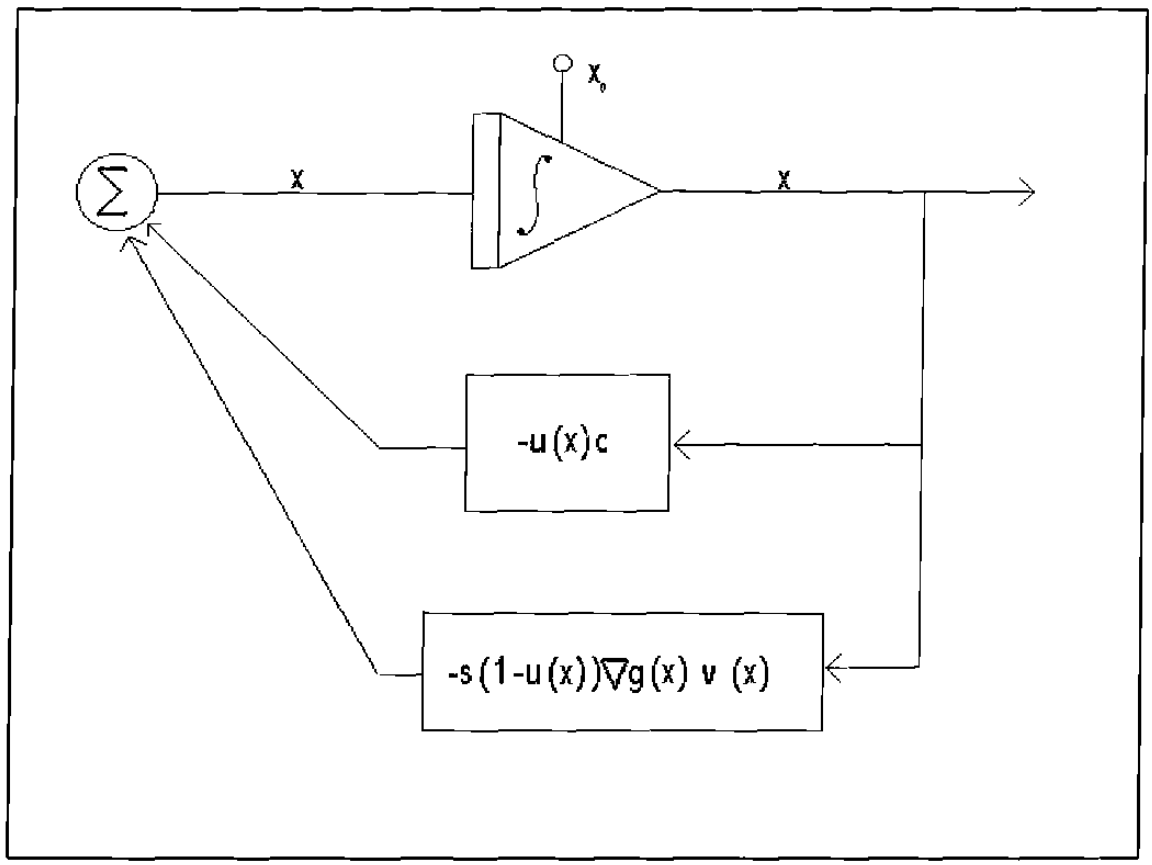


Figura 7. Modelo de red de Rodríguez-Vázquez para resolver problemas de PL

En la región $\{ x \mid g_j(x) > 0, j= 1, \ldots, 2m_E+m_{I+n} \}$, las trayectorias se mueven a lo largo de $-(c)$, mientras que en cualquier otro lugar se mueven a lo largo de combinaciones de los gradientes negativos de las restricciones violadas y posiblemente del gradiente negativo de la función objetivo. Así, iniciando en cualquier punto fuera de la región factible, las trayectorias de la red serán dirigidas hacia $\{ x \mid g_j(x) \geq 0, j= 1, \ldots, 2m_E+m_{I+n} \}$. Una vez que la región factible es alcanzada, la trayectoria se moverá en dirección de minimizar la función objetivo del problema de programación lineal.

- Modelo de Zak

El modelo de Zak requiere que el problema de programación lineal sea representado en la forma estándar mostrada a continuación:

Minimizar $\tilde{c}^T$

Sujeto a :

$$\tilde{A}\,\tilde{x} = \tilde{b}$$
$$\tilde{x} \geq 0$$

donde :

$$\tilde{A} = \begin{pmatrix} A_1 & 0 \\ A_2 & I_{m_I} \end{pmatrix} \in \mathbb{R}^{(m_E + m_{I+n})^* (n + m_I)}$$

$$\tilde{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \in \mathbb{R}^{m_E + m_I}$$

$$\tilde{c} = [c_1, \ldots, c_n, 0, \ldots, 0]^T \in \mathbb{R}^{n+m_I}$$

$$\tilde{x} = [x_1, \ldots, x_{n+m_I}]^T \in \mathbb{R}^{n+m_I}$$

Posteriormente el problema de programación lineal en forma estándar es convertido en el siguiente problema de minimización sin restricciones equivalente;

$$\min E = \min (\tilde{c}^T \tilde{x} + \rho \| \tilde{A}\,\tilde{x} - \tilde{b} \|_p + \gamma \sum_{i=1}^{n} \tilde{x}^-_i), \text{ donde}$$

$$\rho > 0, \ \gamma > 0, \ 1 \leq p \leq \infty, \ \text{y} \ \tilde{x}^-_i \begin{cases} -\tilde{x}_i & \text{si} \quad \tilde{x}_i < 0 \\ 0 & \text{si} \quad \tilde{x}_i \geq 0 \end{cases}$$

Este problema es resuelto a través del siguiente sistema dinámico:

$$\dot{\tilde{x}} = \frac{d\tilde{x}}{dt} = - \nabla E_{p,\rho,\gamma} (\tilde{x}), \ \tilde{x}_0 = \tilde{x}(0)$$

El gradiente de la función objetivo, $\nabla E_{p,\rho,\gamma}(\underline{\tilde{x}})$, del problema de optimización sin restricciones existe excepto en los puntos del conjunto $\{\tilde{x} \mid \tilde{x}_i = 0,$ para algunos $1 \leq i \leq n+m_I\} \cup \{\tilde{x} \mid \tilde{A}\tilde{x} = \tilde{b}\}$. Para $j = 1,\ldots, m_E + m_I$, sea $\tilde{a}^T_j$ es el j-th renglón de $\tilde{A}$. El gradiente es definido como:

$$\nabla \underline{E}_{p,\rho,\gamma}(\tilde{x}) = \tilde{c} + \rho \nabla \| \tilde{A}\tilde{x} - \tilde{b} \|_p + \gamma \nabla \left(\sum_{i=1}^{n+m_I} \tilde{x}_i \right),$$

donde

$$\nabla \| \tilde{A}\tilde{x} - \tilde{b} \|_p = \frac{\tilde{A}^T}{\| \tilde{A}\tilde{x} - \tilde{b} \|_p^{p-1}} \times \begin{pmatrix} \operatorname{sgn}(\tilde{a}^T_1 \tilde{x} - \tilde{b}_1) \, | \tilde{a}^T_1 \tilde{x} - \tilde{b}_1 |^{p-1} \\ \vdots \\ \operatorname{sgn}(\tilde{a}^T_{mE+mI} \tilde{x} - \tilde{b}_{mE+mI}) \, | \tilde{a}^T_{mE+mI} \tilde{x} - \tilde{b}_{mE+mI} |^{p-1} \end{pmatrix},$$

$$\nabla \left(\sum_{i=1}^{n+m_I} \tilde{x}_i \right) = \begin{pmatrix} \frac{\partial \tilde{x}_1}{\partial \tilde{x}_1} \\ \vdots \\ \frac{\partial \tilde{x}_{n+mI}}{\partial \tilde{x}_{n+mI}} \end{pmatrix}$$

Note que $$\frac{\partial \tilde{x}_i}{\partial \tilde{x}_i} = \begin{cases} -1 \text{ si } \tilde{x}_i < 0 \\ 0 \text{ si } x_i > 0 \end{cases}$$

Para sintetizar, Zak [Zak 93], extiende la definición de $\nabla E_{p,\rho,\gamma}(\tilde{x})$ para $\mathbb{R}^{n+m_I}$ definiendo:

$$\frac{\partial \tilde{x}_i}{\partial \tilde{x}_i} = 0 \text{ si } \tilde{x}_i = 0 \text{ y } \nabla \| \tilde{A}\tilde{x} - \tilde{b} \|_p = 0 \text{ si } \tilde{A}\tilde{x} - \tilde{b} = 0$$

El diagrama de bloque del modelo de RN de Zak es mostrado en la figura 8. Zak [Zak 93] muestra que las trayectorias del sistema convergerá a la solución correspondiente problema de programación lineal, si

$$\rho > \| c^T \tilde{A}^T (\tilde{A}\tilde{A}^T)^{-1} \|_q + \gamma (n+m_I)^{1/q} \| \tilde{A}^T (\tilde{A}\tilde{A}^T)^{-1} \|_q), \text{ y}$$

$$\gamma = \frac{\| \tilde{c} \|_2}{\left\| P \nabla \left(\sum_{i=1}^{n+m_I} \tilde{x}_i \right) \right\|_2^2},$$

donde:

$$\frac{1}{q} + \frac{1}{p} = 1, \quad 1 \leq q \leq \infty, \qquad P = I_{n+m_I} - \tilde{A}^T (\tilde{A}\, \tilde{A}^T)^{-1} \tilde{A}$$

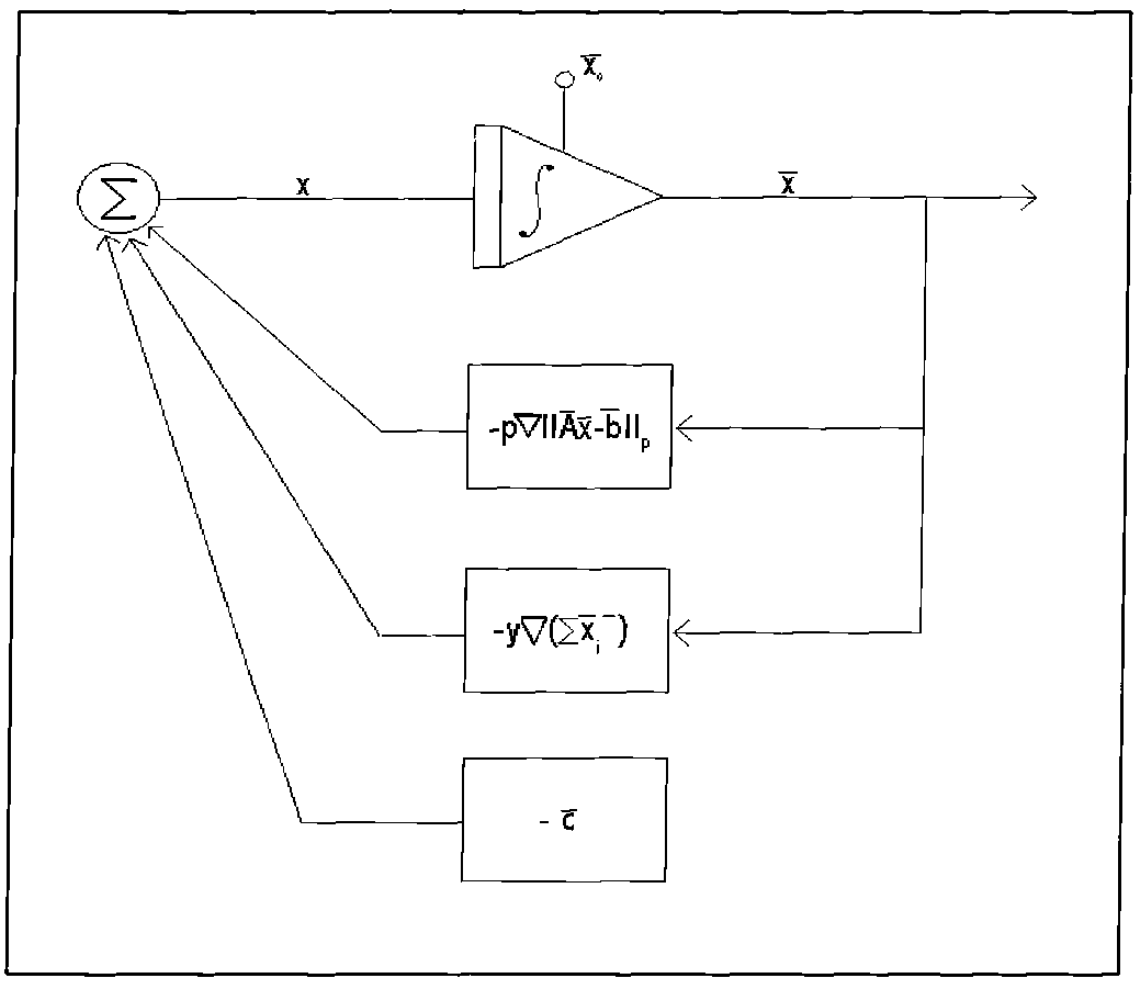


Figura 8. Programa de bloque del Modelo de red de Zak

3.3 Problemas de Optimización Combinatoria y la Red Hopfield

Una de las aplicaciones de las redes neuronales en general y del modelo de Hopfield en particular, es la resolución de problemas de optimización complejos, aquellos cuyo tiempo de resolución mediante algoritmos convencionales crece exponencialmente (k^N) o factorialmente (N !) con el aumento del tamaño (N) del problema. En realidad, cuando las redes se aplican para tal función, lo que se está haciendo es intentar resolver el problema mediante un algoritmo paralelo formulado basándose en una red neuronal, que no es sino un sistema de computación paralela, al estar trabajando multitud de diferentes elementos del proceso (neuronas) simultáneamente. Ejemplos de tales problemas son los problemas de optimización combinatoria.

El problema de optimización combinatoria genérico [Nemhawser 88] se puede definir como sigue: sea $N = \{ 1,2,...,n\}$ un conjunto finito y sea $c = \{ c_1,c_2,...,c_n\}$ un n-vector. Para $F \subseteq N$ defínase

$$c(F) = \sum_{j \in F} c_j$$

Supóngase que se tiene una colección δ de subconjuntos de N. Se desea encontrar $\max\{c(F), F \in \delta\}$

Ejemplos clásicos son el problema del agente viajero y el problema de la mochila [Nemhawser 88]. La mayoría de los problemas de optimización combinatoria son difíciles de resolver, pues el tiempo de ejecución de los algoritmos conocidos crece exponencialmente con el crecimiento del tamaño del problema [Papadimitriou 82],[Gary 78].

La utilización del modelo Hopfield para la resolución de problemas de optimización se basa en la idea de intentar fijar el objetivo del problema mediante una expresión matemática denominada función de costo o función objetivo, que haya que minimizar. A continuación se compara con la expresión general de la función energía de una red Hopfield, determinándose los valores de los pesos (w_{ij}) y de los umbrales (θ_i) en términos de parámetros de la función objetivo para que ambas expresiones sean equivalentes. De esta forma, cuando se pone en funcionamiento una red Hopfield continua con los valores calculados, ésta itera hasta alcanzar un mínimo de la función energía, que en este caso continuará con un mínimo de la función objetivo, encontrando así una posible solución de mínimo costo del problema de optimización. Generalmente no es fácil encontrar un mínimo global, debido a la presencia de mínimos locales de los que habrá que escapar realizando un complicado ajuste de parámetros constantes de la función de costo y de las ganancias (pendientes) de las funciones de activación de las neuronas de la red.

3.3.1 Características de la Red Hopfield

El modelo Hopfield consiste en una red mono capa con **N** neuronas cuyos valores de salida son binarios. En la versión original del modelo [Hilera 95], las funciones de activación de las neuronas eran de tipo escalón (figura 4). Se trata, por lo tanto, de una red discreta con entradas y salida binarias, sin embargo, posteriormente se desarrolló una versión continua con entradas y salidas analógicas utilizando neuronas con funciones de activación tipo sigmoidal (figura 3). Cada neurona de la red se encuentra conectada a todas las demás (conexiones laterales), pero no consigo misma. Además, los pesos asociados en las conexiones entre pares de neuronas son simétricos, esto significa que el peso de la conexión de una neurona i con una j es de igual valor que el de la conexión de la neurona j con la i ($w_{ij} = w_{ji}$).

La versión discreta de esta red fue ideada para trabajar con valores binarios -1 y +1 (aunque mediante un ajuste en los pesos pueden utilizarse en su lugar los valores 1 y 0). Por tanto, la función de activación de cada neurona (i) de la red (f(x)) es de tipo escalón (figura 4).

$$f(x) = \begin{cases} +1 \quad \text{para } x > \theta_i \\ \\ -1 \quad \text{para } x < \theta_i \end{cases}$$

Cuando el valor x coincide exactamente con θ_i , la salida de la neurona i permanece con su valor anterior. θ_i es el umbral de disparo de la neurona i, que representa el desplazamiento de la función de transferencia a lo largo del eje de ordenadas (x). En el módulo Hopfield discreto suele adoptarse un valor proporcional a la suma de los pesos de las conexiones de cada neurona con el resto:

$$\theta_i = k \sum_{j=1}^{N} w_{ij}$$

Si se trabajó con los valores binarios -1 y +1, suele considerarse el valor nulo para θ_i. Si los valores binarios son 0 y 1, se toma un valor de ½ para k.

En el caso de las redes Hopfield continuas, se trabaja con valores reales en los rangos [-1, 1] o [0, 1]. En ambos casos, la función de activación de las neuronas es de tipo sigmoidal.

Una de las características de este modelo es que se trata de una red autoasociativa. Así, la información (patrones) diferente puede ser almacenada en la red, como si tratase de una memoria durante la etapa de aprendizaje. Posteriormente, si se presenta a la entrada alguna de las informaciones almacenadas, la red evoluciona hasta estabilizarse, ofreciendo entonces en la salida, la información almacenada que coincide con la presentada en la entrada. Si por el contrario, la información de entrada no coincide con

ninguna de las almacenadas, por estar distorsionada o incompleta, la red evoluciona generando como salida la más parecida.

3.3.2 Lo Neuronal en Problemas Combinatorios

Uno de los problemas típicos de optimización combinatoria es el problema del agente viajero. Este problema consiste en, dadas N ciudades que tiene que visitar un vendedor, encontrar el camino más corto para, partiendo de una de ellas, visitarlas todas sin pasar más de una vez por cada una y volviendo finalmente a la ciudad de partida. La complejidad reside en la enorme cantidad de posibles caminos, muchos de ellos de longitud similar. Para N ciudades, el número de rutas alternativas es de N! / 2N. De esta forma, para 5 ciudades existen 12 caminos posibles, para 10, el número de rutas es de 181,400 y para 50, del orden 3×10^{62}.

Las primeras pruebas aplicando redes neuronales al problema del viajante se realizaron con problemas de no más de 30 ciudades. Además, aunque la red podría encontrar soluciones óptimas, ella daba solamente el 80% de recorridos válidos. Esto es, aún para problemas pequeños el sistema podía no garantizar la factibilidad. Los investigadores Sihirazi y Yith [Buker 92] descubrieron insuficiencias fundamentales de la red Hopfield en aplicaciones a problemas NP-hard.

Ramanujam y Sasayppan [Buker 92] trabajaron con diferentes problemas combinatorios y de teoría de las gráficas en redes Hopfield. Su principal contribución fue la propuesta de un nuevo conjunto de función energía.

Para los problemas de cubrimiento mínimo de vértices, máximo conjunto independiente, acoplamiento máximo y particionamiento de gráficas [Fould 92] se derivaron casos especiales de la función energía de Hopfield y Tank y se determinaron

los pesos apropiados. El trabajo ayudó a establecer la aplicabilidad de las redes neuronales a estos problemas e ilustró la manera imprecisa y artificiosa en que deben construirse las funciones de energía.

La función energía es semejante a la función penalidad en optimización. Tal función incluye el término "función objetivo" y términos que aseguran la factibilidad. La red cambia de estado con el objetivo de encontrar un mínimo de la función; por tanto, para describir una restricción se requiere encontrar un término cuyos bajos valores correspondan a la satisfacción de la restricción. Por ejemplo, para un problema de un conjunto independiente máximo sobre un grafo G = (V,E), Ramanujam y Sadayappan [Buker 92] dan la función energía:

$$E = - A \Sigma_i \Sigma_j \; v_i \; v_j + B / 2 \; \Sigma_i \Sigma_j \; v_i \; v_j \; a_{ij} \qquad (3.8)$$

donde :

v_i = estado del nodo i = { 1, si el nodo i está en el conjunto; 0 si no lo está }

a_{ij} = { 1 si existe una arista entre los vértices i y j en G, 0 si no }

A,B = constantes

Tal que el valor de E en (3.8) pueda ser minimizado.

El primer término da una estimación de neuronas activas. El segundo término asegura que nodos adyacentes no pueden ambos pertenecer al conjunto, satisfaciendo la definición de un conjunto independiente de vértices. Los pesos pueden ser derivados de la función de energía genérica de Hopfield. Estos pesos pueden ser calculados por:

$$w_{ij} = 2A - Ba_{ij} \qquad (3.9)$$

La función energía (3.8) satisface los siguientes criterios:

1) Soluciones factibles y buenas producirán bajos valores de energía.
2) Se incluyen términos cuadráticos así que pueden determinarse los pesos

Estos criterios pueden absolutamente producir un número considerable de funciones diferentes.

Ahora bien, cómo construir una función energía efectiva sigue siendo un problema. De hecho puede que, únicamente a través de pruebas empíricas, podamos conocer si la función escogida producirá buenos resultados. Además de la dificultad en la determinación de los parámetros, la metodología de Hopfield se ha deteriorado en aplicaciones a problemas de gran tamaño. Como se mencionó antes, la mayoría de los primeros trabajos en redes neuronales para el problema del agente viajero nunca excedieron las 30 ciudades.

3.3.3 Avances en la metodología de redes neuronales de optimización

Los avances en Hopfield y redes dinámicas relacionadas han transformado a estas en más atractivas. El aspecto más crítico es la determinación de los parámetros, por lo tanto, autores tales como Van den Bout y Miller [Buker 92] plantearon una función de energía más simple para la red Hopfield, la cual requiere un parámetro fácil de determinar. Nuevamente, sin embargo, sus experimentos se enfocaron en problemas pequeños del problema del agente viajeroe (30 ciudades), pero pudieron al menos asegurar la factibilidad.

Al mismo tiempo surgió una metodología diferente de optimización, en forma de redes neuronales adaptativas. Estos sistemas parecen ser particularmente útiles en problemas de Teoría de Gráficas que incluyan distancia y por consiguiente han sido aplicados al problema del agente viajero. Durbin y Wilshaw [Buker 92] diseñaron un sistema el cual va evolucionando continuamente hacia un tour. Se ha obtenido mayor

éxito con las redes adaptativas que con las de Hopfield con respecto al problema del agente viajero y en general con problemas geométricos de optimización combinatoria.

Las redes Hopfield [Buker 92] son esencialmente no adaptativas, ellas no "aprenden". Las redes adaptativas adaptan sus parámetros, los cuales son pesos en las conexiones, en respuesta a la información externa. Las metodologías adaptativas aplicadas en particular al problema del agente viajero, generalmente se enfocan en desarrollar las posiciones de los nodos hasta que ellos coinciden con la posición de las ciudades, mientras intentan minimizar localmente la distancia a los vecinos en el tour. Aunque la red requiere cientos de miles de presentaciones del conjunto de entrenamiento (el cual es el conjunto de ciudades a ser visitadas) el tiempo de procesamiento es extremadamente rápido y con una representación razonable.

3.4 Conclusiones del Capítulo

En este capítulo hemos discutido algunos modelos de Redes Neuronales para resolver problemas de programación lineal, en donde la red neuronal de Tank y Hopfield fue una de las causas para revivir el interés de circuitos análogos para resolver problemas de optimización y la inspiración de investigadores para profundizar en otros modelos de redes neuronales para la solucionar problemas de programación lineal y no lineal.

Los modelos analizados y discutidos en este capítulo son el modelo de Kennedy y Chua, el modelo de Rodríguez-Vázquez y el descrito por Sctanislaw H. Zak, los cuales se comparan en términos de complejidad del modelo, complejidad de neuronas individuales y la precisión de solución. De igual manera se destaca la importancia de las redes neuronales en Investigación de Operaciones basadas en el problema del agente viajero, así como en problemas de optimización combinatoria y la red Hopfield.

CAPITULO 4

IMPLEMENTACION DE UNA RED NEURONAL ARTIFICIAL DE DISTRIBUCION

4.1 Introducción

El propósito de esta investigación es diseñar e implementar una red neuronal artificial que permita encontrar un patrón de distribución de flujo que minimice el costo total que implicaría abastecer las demandas de un conjunto de sumideros a partir de las ofertas de varias fuentes. Para lograr lo anterior, se aplicaron técnicas de Programación Lineal (especialmente aquellas que son utilizadas en los métodos de transporte), redes neuronales artificiales de retroalimentación, así como algunos aspectos importantes de métodos de optimización.

Los problemas de distribución de flujo son encontrados en muchas áreas de aplicación, tales como, el abastecimiento de combustible, agua y en la programación de la fuerza eléctrica, por citar algunos.

Nuestra investigación está centrada en aquellos problemas de distribución que tengan las siguientes características: **n** centros de abastecimientos cada uno de los cuales tiene una oferta de o_i unidades para i = 1,2, ..., n y **m** centros de recepción, cada uno con una demanda de d_j unidades para j = 1,2,...,m. Estas ofertas y demandas deben satisfacer la ecuación de balance, esto es :

$$\sum_{i=1}^{n} o_i = \sum_{j=1}^{m} d_j .$$

Se desea encontrar el patrón de flujos x_{ij} que satisfaga las demandas de los centros de recepción a través de enlaces existentes desde cada centro de abastecimiento hacia todos los centros de recepción. Estos enlaces (i,j) tienen cierta capacidad c_{ij} que no debe excederse, así como un costo unitario asociado a_{ij} . Por consiguiente deben satisfacerse las siguientes ecuaciones:

$$\sum_{j=1}^{m} x_{ij} = o_i \quad \forall\, i = \overline{1, n} \tag{1}$$

$$\sum_{i=1}^{n} x_{ij} = d_j \quad \forall\, j = \overline{1, m} \tag{2}$$

$$0 \leq x_{ij} \leq c_{ij} \quad \forall\, i,j \tag{3}$$

El objetivo primordial de esta investigación es el de encontrar un patrón de distribución de flujo que permita minimizar el costo total del envío, esto es:

$$\text{mín} \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij}\, x_{ij} \tag{4}$$

Existen otras consideraciones importantes que hay que tomar en cuenta en la implementación de esta red, las cuales serán tratadas a lo largo de este capítulo, tales como la arquitectura, características del patrón de entrenamiento, adaptaciones en ecuaciones de programación lineal y optimización, etc.

4.2 Antecedentes en la Solución de Problemas de Flujo

Los problemas de flujo pueden ser resueltos por técnicas clásicas de programación lineal, sin embargo, en aplicaciones de tiempo real, sería interesante adaptar patrones de flujo, variaciones en la capacidad o bien fallas en las ramas, en un tiempo de respuesta corto. En estos casos, las redes neuronales artificiales son mucho más atractivas ya que permiten realizar este tipo de adaptaciones.

Las redes neuronales son utilizadas para resolver problemas de flujo que se presentan en diversas áreas de aplicación, tales como el abastecimiento de combustible, agua, programación de fuerza eléctrica [citado en Perfetti 95] etc.

Renzo Perfetti [Perfetti 95] desarrolló una red neuronal que permite resolver problemas de flujo, la cual constituyó el punto de partida para nuestra investigación. Las características de esta red es que consta de dos capas, una capa oculta y una capa de salida, en donde las unidades ocultas corresponden a los nodos del flujo o corriente gráfica, y las unidades de salida representan el flujo a través de las ramas. Esta red puede resolver problemas de optimización en tiempos de ejecución, por otra parte, se puede garantizar que la solución encontrada por la red esté cercana a una solución verdadera del problema de optimización, mientras no exista el riesgo de un mínimo local.

La red neuronal descrita por Renzo Perfetti, resuelve problemas de flujo que puedan ser expresados como se muestra a continuación:

$$\text{minimizar} \quad \phi(x) = a^T x \tag{5}$$

sujeto a

$$A(x) = b \tag{6}$$

$$0 \leq x_i \leq c_i \quad \text{para } i = 1,2,\ldots, B \tag{7}$$

en donde

B : es el número de ramas en la red

$a = [a_1, a_2,\ldots, a_B]^T$: representa el vector costo, a_i es el costo asociado a cada rama i

$x = [x_1, x_2,\ldots, x_B]^T$: vector variable, x_i representa el flujo en la rama i

$b = [b_1, b_2,\ldots, b_N]^T$: b_i representa el flujo inyectado al nodo i

N : es el número de nodos que forma la red

A : es la matriz de incidencia nodo-arco de orden NxB definida como sigue :

$A_{ij} = +1$ si la rama j entra al nodo i

$A_{ij} = -1$ si la rama j sale del nodo i

$A_{ij} = 0$ si la rama j no está conectada al nodo i

$c = [c_1, c_2,\ldots, c_B]^T$: vector capacidad, c_i representa la capacidad de la rama i.

Como un ejemplo ilustrativo, la figura 9 muestra una red en donde se involucran siete nodos, los cuales están interconectados por un total de 13 ramas, cada rama tiene asociada a ella un valor que indica la capacidad máxima aceptada del flujo que podrá fluir por ella. El problema consiste en enviar el mayor flujo posible desde el nodo 1 llamado fuente, hasta el nodo 7 llamado sumidero.

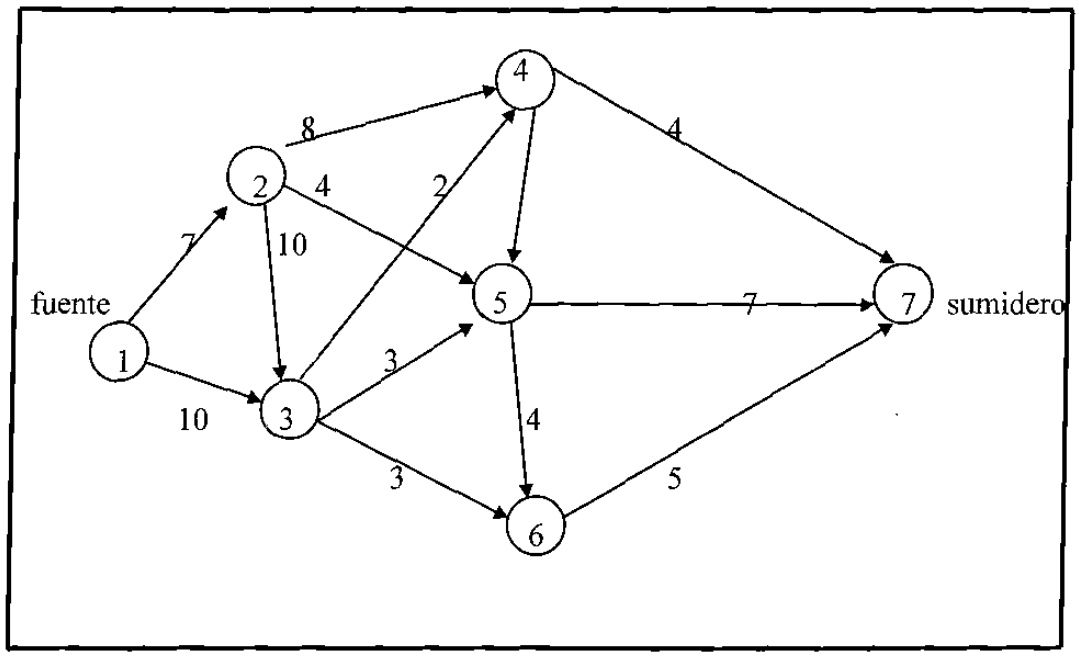


Figura 9 Gráfica de un problema de flujo

El problema de flujo (5-7), es considerado como un problema de programación lineal que puede ser formulado como un problema de optimización sin restricciones usando el método de función penalidad, esto es:

$$\text{minimizar} \quad E(x) = \phi(x) + \mu P(x) \tag{8}$$

donde $\phi(x)$ es la función objetivo original del problema y $P(x)$ es la función penalidad diferenciable cuya minimización ocasiona la satisfacción de las restricciones (6) y (7); $\mu > 0$ es llamado multiplicador penalidad.

Para el caso de problema del tipo (5)-(7), es conveniente la siguiente función penalidad:

$$P(x) = (\tfrac{1}{2})(Ax - b)^T(Ax-b) + (\tfrac{1}{2})(g(c-x))^T g(c-x) + (\tfrac{1}{2})(g(x))^T g(x) \tag{9}$$

$$g(x) = [g(x_1), g(x_2), \ldots, g(x_B)]^T \tag{10}$$

donde $g(\xi) = 0$ para $\xi > 0$ y $g(\xi) = \xi$ para $\xi \leq 0$.

La red neuronal puede ser implementada entonces de la siguiente manera.

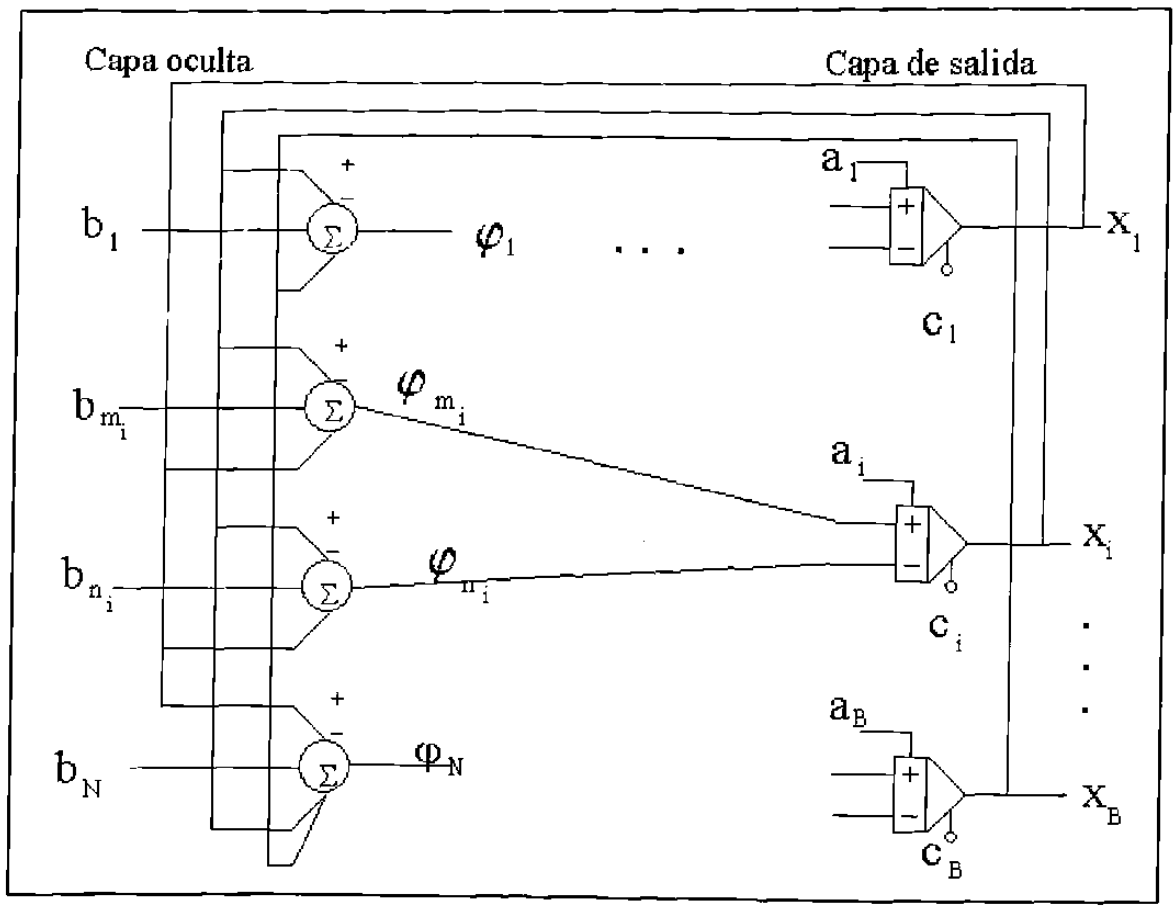


Figura 10. Red neuronal para resolver problemas de flujo

La solución de la red neuronal mostrada en la figura 10 puede ser obtenida por la construcción del siguiente sistema dinámico de gradiente, donde la función potencial es E(x).

$$x = -\nabla E(x) = -a - \mu A^T Ax + \mu A^T b + \mu g(x) + \mu g(c-x) \qquad (11)$$

pudiendo reescribir (11) como:

$$x = -a - \mu A^T (Ax - b) - \mu g(x) + \mu g(c-x) \qquad (12)$$

y en forma escalar

$$x_i = -a_i - \mu (\varphi m_i - \varphi n_i) - \mu g(x_i) + \mu g(c_i - x_i) \qquad (13)$$

para $i = 1,2,..., B$

donde se asume que la rama i conecta el nodo m_i con el nodo n_i y

$$\varphi_k = \sum_{j \varepsilon S_k} A_{kj} x_j - b_k \qquad \text{para } k = 1,2,..., N$$

en donde S_k denota el conjunto de ramas conectadas al nodo k.

4.3 Implementación de la Red Neuronal Artificial de Distribución

La red neuronal propuesta en nuestro proyecto da solución a problemas de distribución con las siguientes características: supóngase que se desea distribuir cierta cantidad de artículos de uno o varios centros productores hacia uno ó varios almacenes. Cada centro productor está enlazado con todos los almacenes, pero no hay comunicación entre los centros productores ni entre los almacenes. Además se cuenta con la siguiente información:

- Centros que cuentan con la mercancía (nodos fuente)
- Centros que demandan la mercancía (nodos sumidero)
- Cantidad que oferta cada nodo fuente
- Cantidad demandada por cada nodo sumidero
- Capacidad máxima que puede circular del producto por cada enlace fuente-sumidero
- Costo asociado al traslado de una unidad de flujo por cada enlace.

Con esta información debemos encontrar el patrón de distribución que nos permita minimizar el costo total que implicaría distribuir la oferta de los nodos fuente hacia los nodos sumideros, sin exceder la capacidad de ningún enlace.

Este tipo de situación puede modelarse matemáticamente a través de las expresiones (1-4).

Para resolver el programa anterior se ha diseñado un sistema de simulación que utiliza una red nueronal artificial que permita dar solución a problemas con las características antes mencionadas, dando como resultado el patrón de distribución más apropiado.

Este sistema simulador monitorea el comportamiento de la red después de alimentarla con la información correspondiente al problema en cuestión. El tiempo de la ejecución para lograr la solución deseada es relativamente corto, esto gracias a la constante penalidad utilizada (μ), la cual inicia con un valor de 200 y va incrementando su valor cada cierto rango de valores generados, permitiendo acercarnos más rápidamente a la convergencia del patrón.

A continuación mostraremos los aspectos más relevantes de nuestra investigación, la cual resulta ser una aportación más al sistema de redes neuronales de distribución.

4.3.1 Características de la Red

Las principales características de nuestra red es que su arquitectura es creada a través de dos capas, una capa de entrada (compuesta por nodos fuente) y una capa de salida (compuesta por nodos sumidero). Cada uno de los nodos fuente está conectado a cada uno de los nodos sumideros, pero no hay conexión entre los nodos fuente ni entre los nodos sumidero.

Para simplificar notación enumeraremos en forma corrida los n+m nodos de la red y denotaremos sus valores de oferta o demanda (según corresponda) mediante un único símbolo:

$$b_i = \begin{cases} -o_i & \text{para } i = 1, \ldots, n \\ d_{i-n} & \text{para } i = n+1, \ldots n+m \end{cases}$$

donde:

n: representa la cantidad de nodos fuente, y

m: la cantidad de nodos sumidero.

Cada o_i representa la cantidad que sale del nodo fuente i (oferta) y cada d_{i-n} la cantidad esperada en cada nodo sumidero n+i (demanda). La distribución de cantidades se realiza a través de las ramas que conectan los nodos.

Algo importante a resaltar es que el total de las ofertas debe ser siempre igual al total de las demandas. En caso de que la oferta total exceda a la demanda total, será necesario agregar un nodo sumidero artificial con demanda igual a la diferencia entre esos valores, así como las conexiones desde cada nodo fuente hacia este nodo artificial.

Para poder ilustrar el esquema de la arquitectura de nuestra red, analicemos el siguiente ejemplo: suponga que tenemos la necesidad de distribuir la cantidad de 60 paquetes, los cuales están concentrados en tres almacenes (35 en el primero, 21 en el segundo y 7 en el tercero) y deben llegar a tres nuevas casas de venta (23 a la primera, 20 a la segunda y 20 a la tercera). De esta manera obtenemos una red con las siguientes características : una capa de entrada compuesta con tres nodos fuente (n = 3), una capa de salida formada por tres nodos sumidero (m = 3) y un total de nueve ramas existentes en la red (B = 9 = n*m).

Ahora bien, otras consideraciones importantes en nuestra red es que debemos conocer la máxima capacidad $\mathbf{c_i}$ para $i = 1,2,..., B$ que podrá fluir por cada una de las ramas.

De igual manera, a cada rama **i** irá asociada un valor $\mathbf{a_i}$ para $i = 1,2,..., B$, el cual representa el costo que implicará la circulación de una unidad de flujo $\mathbf{x_i}$ por esa rama. Por lo que nuestra red quedaría de la siguiente manera, considerando el ejemplo anterior: supongamos una capacidad máxima por rama de valor 12, y un vector costo de: a =[10, 12, 15, 10, 12, 15, 10,12,15]

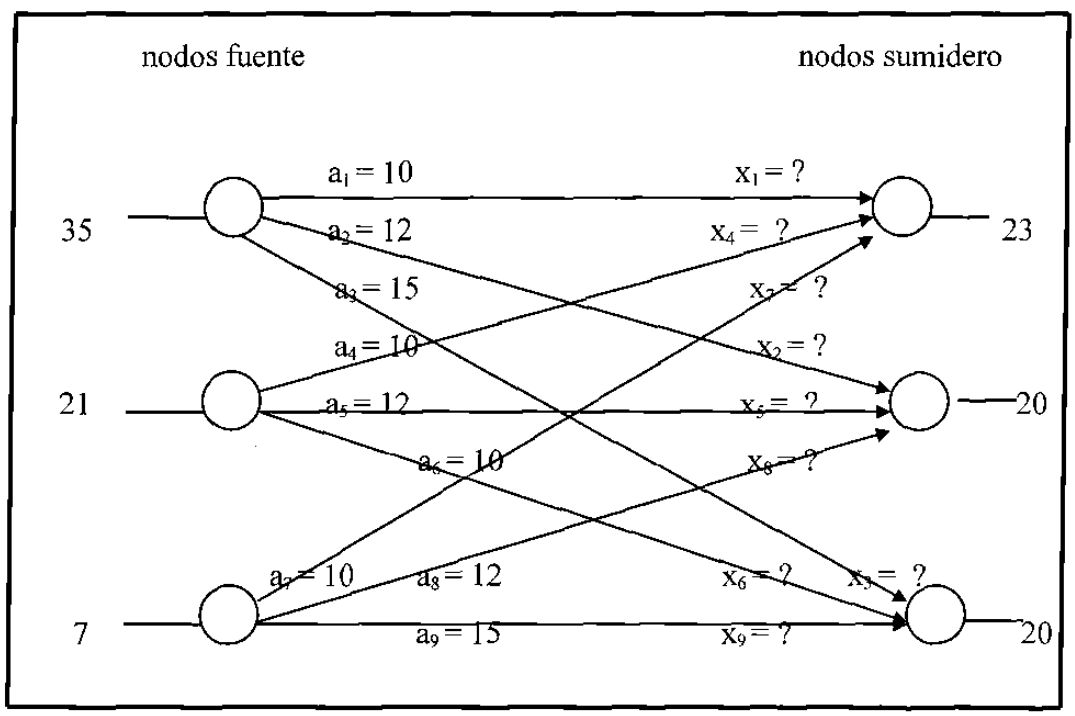


Figura 11 Arquitectura de una Red de Distribución

Se desea encontrar el vector **x** que satisfaga las demandas expuestas en los nodos sumidero, sin olvidar que la máxima capacidad (en este ejemplo en particular) es de 12 y el costo por rama está indicado en el vector **a**.

Esta es la manera en que a partir de problemas de distribución, nuestro sistema simulador formula la arquitectura de la red de distribución. A continuación, se expondrá el comportamiento de la red para lograr obtener la solución del problema.

4.3.2 Comportamiento de la Red

Los problemas de distribución a resolver con nuestra red deberán poder ser expresados como se muestra a continuación:

minimizar $\phi(x) = a^T x$

sujeto a:

$A(x) = b$, donde A es la matriz de incidencia nodo-arco

$0 <= x_i <= c_i$ para $i = 1,2,..., B$

Nuestro sistema simulador, pretende obtener el patrón de distribución (x) que permita minimizar el costo total que implicará abastecer las necesidades de demanda expuestas.

En nuestro programa simulador se implantó el siguiente sistema de gradiente, el cual está basado en el propuesto por Perfetti [Perfetti 95].

$$x = (-a - \mu A^T (Ax - b) - \mu g(x) + \mu g(c-x)) * h + x \qquad (14)$$

Con esta ecuación se pretende predecir el nuevo valor del vector variable **x**, haciendo uso de la pendiente para explorar linealmente sobre el tamaño de paso **h** y el valor anterior del vector **x**. Esta ecuación se aplicará paso a paso para calcular la solución futura y así lograr obtener la solución esperada del problema expuesto.

El comportamiento que sigue nuestra red en el sistema generado es el que a continuación se indica.

Nuestro sistema requiere de los siguientes datos de entrada :

m : cantidad de nodos fuente

n : cantidad de nodos sumidero

o_i : cantidad a ofertar por cada nodo fuente (para i = 1,2, ... , n)

d_{i-n} : cantidad demandada por cada nodo sumidero (para i =n+1, ... , n+m)

h : longitud de paso aceptable (la cual debe ser pequeña por ejemplo menor o igual a 0.0001)

μ : multiplicador penalidad (considerada de valor relativamente grande $\sim$ 100 o mayor)

c : vector capacidad (máxima capacidad aceptable por cada rama)

a : vector costo

Emax : Error máximo aceptado para considerar la convergencia de **x** durante el proceso

Nuestro sistema, al contar con toda esta información inicial, inicializa el vector variable **x** en cero, y genera la matriz de incidencia nodo-arco correspondiente **(A)**. Posterior a esto, se inicia el proceso de retroalimentación, el cual se efectuará hasta lograr la convergencia del vector **x**, asegurando así que se logren satisfacer las restricciones expuestas por el problema. Para lograr acelerar la convergencia en el proceso, fue necesario hacer incrementos fuertes al multiplicador penalidad μ, el cual es utilizado en (11). Consiguiendo la convergencia de **x** y verificando cumplir con el conjunto de restricciones, podemos considerar que los valores obtenidos en **x** son los que dan solución a nuestro problema, logrando cumplir nuestro objetivo.

4.4 Conclusiones de Capítulo

En este capítulo analizamos los antecedentes en la solución de problemas de flujo utilizando redes neuronales, los cuales representaron la base de nuestra investigación. También se describieron las características del sistema simulador implementado para dar solución al problema formulado utilizando redes neuronales artificiales.

El sistema generado cumple con los requerimientos expuestos y logra cumplir con el objetivo de nuestra investigación "minimizar el costo total de la distribución oferta/demanda de nodos fuente a nodos sumidero".

CAPITULO 5

APLICACION E IMPLEMENTACION

5.1 Introducción

La implementación computacional de nuestro programa fue desarrollada en el software de aplicación computacional ***MatLab.*** El programa no genera una solución exacta pero sí una aproximación a la solución optima del problema de distribución expuesto.

A continuación se exponen las características de la aplicación e implementación del programa desarrollado.

5.2 Programación

El lenguaje en el que originalmente se penso implementar el programa simulador de nuestra red fue el lenguaje *Pascal,* por su capacidad en el manejo de implementaciones matemáticas y por su facilidad de adaptar procedimientos y funciones.

Posteriormente se realizaron pruebas aplicando otros lenguajes y aplicaciones, buscando con esto seleccionar el más apropiado para nuestra implementación (tales lenguajes fueron el C y el MatLab).

Al observar las ventajas que cada uno de ellos ofrecía, se optó por programar en *MatLab* por la facilidad que este lenguaje ofrece al manejar vectores y matrices. Podremos observar la codificación de nuestro programa en el apéndice A.

5.3 Limitaciones

Entre las limitaciones del programa computacional implementado encontramos :

1) El tiempo de convergencia del vector solución **x** dependerá del incremento proporcional que se le aplique al multiplicador penalidad μ (μ representa la constante penalidad en nuestro sistema, la cual juega un papel muy importante ya que nos ayuda a lograr obtener menor tiempo la convergencia del vector de salida x), el cual se realiza cada 100 iteraciones.

   Nuestro sistema permite aplicar un incremento (el cual se pide como dato de entrada) proporcional a μ, para obtener nuestros resultados.

2) La velocidad del programa para obtener el vector de salida (**x**) óptimo, también dependerá de la velocidad de procesamiento de la computadora donde se corra el programa.

5.4 Características de Implementación

1) La longitud de paso (h) se recomienda sea pequeña (ej. h = 0.00001) para evitar saltos descontrolados durante la ejecución del programa y la pérdida del control.

2) El vector resultante **x** no debe tener ningún elemento menor que cero (negativo) por lo que fue necesario ajustar una condición en el programa (esta fue, siempre que x[i] < 0 se ajustará a x[i] = 0).

Es importante resaltar que los tiempos de solución dependerán de la computadora en donde se corra el programa y del valor inicial que se le otorgue al multiplicador penalidad (μ). De igual forma es importante el incremento que se le aplique a μ durante el proceso, el cual permitirá acelerar la convergencia del vector solución **x.**

Para determinar el valor de incremento que se asignaría al multiplicador penalidad fue necesario realizar un cierto número de ejemplos. Posterior a esto, llegamos a la conclusión de recomendar aplicar incrementos de μ = 300 unidades cada 100 iteraciones durante la ejecución del programa.

Nuestro programa permite realizar un nuevo proceso con los mismos datos de entrada pero ajustando el valor capacidad (una unidad menos en cada proceso, cuando esto sea posible) en la primera rama por la cual se haya distribuido un valor de x[i] igual al valor de **c** (máxima capacidad por rama), lo cual permite observar las cualidades del programa en la adaptación de los ejemplos. El indicará cuando no sea posible seguir adaptando el ejemplo a una nueva simulación, esto lo hará al detectar que si se sigue decrementando la capacidad de esa rama sería imposible lograr distribuir las cantidades de los nodos fuente a los nodos sumidero.

Los resultados que aquí presentamos comprenden números enteros y reales, los números reales se representan con la parte entera y cuatro dígitos en la parte fraccionaria (por lo que **no** nos permite obtener la solución óptima pero sí la más aproximada a ella) estos valores corresponden a los arrojados por el programa corrido en una máquina 586 de 133 Mhz y 16 MB en RAM, partiendo de un valor inicial de $\mu = 200$.

5.5 Conclusiones del Capítulo

El programa simulador de nuestra red de distribución fue implementado computacionalmente en el software de aplicación *MatLab,* ya que éste posee herramientas simples y sencillas para la programación además de proporcionar un buen manejo de matrices.

Consideramos que el programa generado proporciona resultados que permiten cumplir con el objetivo planteado en esta investigación.

CAPITULO 6

PRESENTACION DE RESULTADOS

6.1 Introducción

El programa implementado para simular nuestra red neuronal de distribución fue desarrollado en el software de aplicación *MatLab* y fue probado aproximadamente con 50 problemas de distribución. Los tiempos en los que se obtuvo solución a los problemas expuestos están entre 10 minutos para problemas simples y aproximadamente 1 hora con 30 minutos para problemas con características más complejas. El programa se corrió en una máquina 586 de 133 Mhz y 16 Mb en RAM.

6.2 Ejemplos

El programa simulador nos mostrará los resultados generados de la siguiente manera:

Nodos Sumidero / Nodos Fuente	NS1 Cantidad demandada	NS2 Cantidad demandada	NS3 Cantidad demandada
NF1 Cantidad ofertada	x[1,1]	x[1,2]	x[1,3]
NF2 Cantidad ofertada	x[2,1]	x[2,2]	x[2,3]
NF3 Cantidad ofertada	x[3,1]	x[3,2]	x[3,3]

Figura 12. Tabla de Resultados

A continuación presentaremos los resultados arrojados por nuestro programa aplicando los siguientes ejemplos:

6.3.1 Ejemplo 1

Proporcionar la cantidad de nodos fuente : 3

Proporcionar la cantidad de nodos sumidero : 3

Cantidad a ofertar por el nodo fuente 1 : 35

Cantidad a ofertar por el nodo fuente 2 : 21

Cantidad a ofertar por el nodo fuente 3 : 7

Cantidad demandada por el nodo sumidero 1 : 23

Cantidad demandada por el nodo sumidero 2 : 20

Cantidad demandada por el nodo sumidero 3 : 20

Error máximo aceptado para convergencia : 0.0001

Longitud de paso aceptable : 0.00001

Valor inicial del multiplicador penalidad (μ): 200

Incremento que ajustará a μ en el proceso : 300

Máxima capacidad aceptable por rama : 12

Costo por rama [r1,r2,r3,r4,r5,r6,r7,r8,r9] : [3,2,4,5,4,6,3,4,2]

Resultados

Primer proceso:

Nodos Sumidero / Nodos Fuente		NS1 **23**	NS2 **20**	NS3 **20**
NF1	**35**	12.0000	11.5008	11.4991
NF2	**21**	7.8309	6.5853	6.5837
NF3	**7**	3.1690	1.9138	1.9170

Figura 13. Tabla de Resultados del ejemplo 1 (Proceso 1)

Segundo proceso:

{máxima capacidad aceptada en la rama 1 igual a 11 (c[1] = 11), conservando las ramas restantes el valor de capacidad iniciales (c[2]=12, c[3]=12, ... , etc.) }

Nodos Sumidero / Nodos Fuente		NS1 **23**	NS2 **20**	NS3 **20**
NF1	**35**	10.999	12.0000	12.0000
NF2	**21**	8.3253	6.3414	6.3333
NF3	**7**	3.6748	1.6585	1.6666

Figura 14. Tabla de Resultados del ejemplo 1 (Proceso 2)

* Se detecta que no es posible seguir con otro proceso ya que se llegó al límite.

6.3.2 Ejemplo 2

Proporcionar la cantidad de nodos fuente : 3
Proporcionar la cantidad de nodos sumidero : 4

Cantidad a ofertar por el nodo fuente 1 : 30
Cantidad a ofertar por el nodo fuente 2 : 25
Cantidad a ofertar por el nodo fuente 3 : 12

Cantidad demandada por el nodo sumidero 1 : 25

Cantidad demandada por el nodo sumidero 2 : 20

Cantidad demandada por el nodo sumidero 3 : 16

Cantidad demandada por el nodo sumidero 4 : 6

Error máximo aceptado para convergencia : 0.0001

Longitud de paso aceptable : 0.00001

Valor inicial del multiplicador penalidad (μ): 200

Incremento que ajustará a μ en el proceso : 300

Máxima capacidad aceptable por rama : 9

Costo por rama [r1,r2,r3,r4,r5,r6,r7,r8,r9,r10.r11.r12]:

[10,10,10,10,10,10,10,10,10,10,10,10]

Resultados

Primer proceso:

Nodos Sumidero / Nodos Fuente		NS1 **25**	NS2 **20**	NS3 **16**	NS4 **6**
NF1	**30**	9.0000	9.0000	8.0666	3.9332
NF2	**25**	9.0000	7.7332	6.1999	2.0666
NF3	**12**	6.9998	3.2666	1.7333	0.0000

Figura 15. Tabla de Resultados del ejemplo 2 (Proceso 1)

Segundo proceso:

{máxima capacidad aceptada en la rama 1 igual a 8 (c[1] = 8), conservando las ramas restantes el valor de capacidad iniciales (c[2]=9, c[3]=9, ... , etc.) }

Nodos Sumidero / Nodos Fuente		NS1 **25**	NS2 **20**	NS3 **16**	NS4 **6**
NF1	**30**	8.0000	9.0000	8.7332	4.2666
NF2	**25**	9.0000	8.0666	6.1999	1.7333
NF3	**12**	7.9998	2.9333	1.0667	0.0000

Figura 16. Tabla de Resultados del ejemplo 2 (Proceso 2)

Tercer proceso:

{máxima capacidad aceptada en la rama 1 igual a 7 (c[1] = 7), conservando las ramas restantes el valor de capacidad iniciales (c[2]=9, c[3]=9, ... , etc.) }

Nodos Sumidero / Nodos Fuente		NS1 **25**	NS2 **20**	NS3 **16**	NS4 **6**
NF1	**30**	7.0000	9.0000	9.0000	4.9997
NF2	**25**	9.0000	8.4998	6.4998	1.0001
NF3	**12**	8.9998	2.5000	0.5000	0.0000

Figura 17. Tabla de Resultados del ejemplo 2 (Proceso 3)

* Se detecta que no es posible seguir con otro proceso ya que se llegó al límite.

6.3.3 Ejemplo 3

Proporcionar la cantidad de nodos fuente : 4
Proporcionar la cantidad de nodos sumidero : 6

Cantidad a ofertar por el nodo fuente 1 : 60
Cantidad a ofertar por el nodo fuente 2 : 25
Cantidad a ofertar por el nodo fuente 3 : 12
Cantidad a ofertar por el nodo fuente 4 : 10

Cantidad demandada por el nodo sumidero 1 : 30
Cantidad demandada por el nodo sumidero 2 : 25
Cantidad demandada por el nodo sumidero 3 : 22
Cantidad demandada por el nodo sumidero 4 : 16
Cantidad demandada por el nodo sumidero 5 : 10
Cantidad demandada por el nodo sumidero 4 : 4

Error máximo aceptado para convergencia : 0.0001
Longitud de paso aceptable : 0.00001
Valor inicial del multiplicador penalidad (μ) :200
Incremento que ajustará a μ en el proceso : 300
Máxima capacidad aceptable por rama : 12
Costo por rama [r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,
r13,r14,r15,r16,r17,r18,r19,r20,r21,r22,r23,r24]:
[10,12,10,12,10,12,10,12,10,12,10,12,
10,12,10,12,10,12,10,12,10,12,10,12]

Resultados

Primer proceso:

Nodos / Sumidero / Nodos Fuente		NS1 **30**	NS2 **25**	NS3 **22**	NS4 **16**	NS5 **10**	NS6 **4**
NF1	**60**	12.0000	12.0000	12.0000	11.2381	8.7618	3.9997
NF2	**25**	8.1269	6.4602	5.4603	3.7142	1.2380	0.0000
NF3	**12**	5.1864	3.5197	2.5198	0.7737	0.0000	0.0000
NF4	**10**	4.6864	3.0197	2.0198	0.2737	0.0000	0.0000

Figura 18. Tabla de Resultados del ejemplo 3 (Proceso 1)

Segundo proceso:

{máxima capacidad aceptada en la rama 1 igual a 11 (c[1] = 11), conservando las ramas restantes el valor de capacidad iniciales (c[2]=12, c[3]=12, ... , etc.) }

Nodos / Sumidero / Nodos Fuente		NS1 **30**	NS2 **25**	NS3 **22**	NS4 **16**	NS5 **10**	NS6 **4**
NF1	**60**	11.0000	12.0000	12.0000	11.8571	9.1428	3.9997
NF2	**25**	8.5238	6.5237	5.5238	3.5714	0.8570	0.0000
NF3	**12**	5.4880	3.4880	2.4880	0.5356	0.0000	0.0000
NF4	**10**	4.9880	2.9880	1.9880	0.0356	0.0000	0.0000

Figura 19. Tabla de Resultados del ejemplo 3 (Proceso 2)

Tercer proceso:

{máxima capacidad aceptada en la rama 1 igual a 10 (c[1] = 10), conservando las ramas restantes el valor de capacidad iniciales (c[2]=12, c[3]=12, ..., etc.)}

Nodos Sumidero / Nodos Fuente		NS1 **30**	NS2 **25**	NS3 **22**	NS4 **16**	NS5 **10**	NS6 **4**
NF1	**60**	10.0000	12.0000	12.0000	12.0000	9.9997	3.9997
NF2	**25**	9.0138	6.6804	5.6804	3.6248	0.0001	0.0000
NF3	**12**	5.7638	3.43044	2.4305	0.3748	0.0000	0.0000
NF4	**10**	5.2221	2.8887	1.8888	0.0000	0.0000	0.0000

Figura 20. Tabla de Resultados del ejemplo 3 (Proceso 3)

* Se detecta que no es posible seguir con otro proceso ya que se llegó al límite.

6.3 Conclusiones del Capítulo

Observamos que el sistema muestra las soluciones más cercanas a las óptimas de cada ejemplo, habiéndose probado con un número considerable de ejemplos (el apéndice B muestra las soluciones óptimas correspondientes a los ejemplos expuestos en esta investigación). En este capítulo sugerimos valores importantes que deben ser considerados para lograr los propósitos del programa y de esta manera cumplir nuestro objetivo.

CAPITULO 7

CONCLUSIONES Y RECOMENDACIONES

7.1 Conclusiones Generales

Este estudio se realiza con el propósito de aplicar las herramientas de redes neuronales artificiales a problemas de distribución, logrando con esto eficientar el proceso de distribución.

Nuestra red neuronal de distribución adopta el comportamiento de una red neuronal artificial de retroalimentación aplicando ecuaciones de optimización, las cuales nos permiten obtener buenos resultados de nuestro sistema.

El objetivo de esta tesis es adaptar un patrón de distribución de flujo que permita minimizar el costo total que implicará abastecer las demandas de varios centros, a partir de las ofertas de otros. Cada centro ofertante está enlazado con todos los centros de demanda, pero no hay comunicación entre los centros que ofertan ni entre los que demandan.

En la implementación de la red neuronal se tuvieron en cuenta trabajos realizados anteriormente, pero fue necesario adaptarlos a las características de nuestro problema.

7.2 Recomendaciones Futuras

Los problemas tanto de flujo como de distribución representan un campo muy amplio de estudio. En esta investigación hemos abordado el cómo optimizar un proceso de distribución aplicando redes neuronales artificiales y partiendo de investigaciones realizadas basadas en términos de programación lineal

Algo importante para investigaciones futuras aplicando redes neuronales artificiales en problemas de distribición sería la ***vulnerabilidad*** de la red, la cual representa la suseptibilidad de la red al perder algún lazo entre nodos fuente-sumidero, esta pérdida provocaría sierta reacción en la red neuronal para lograr identificar si el lazo destruido afecta seriamente a la red para lograr su proceso de distribución entre nodos fuente-sumidero, o bien identificar si se trata de un lazo de poca prioridad para la red.

REFERENCIAS

[Adenson 96] Adenso Díaz Bernardino, Optimización, Heurísticas y Redes Neuronales. Editorial Paraninfo (Julio 1996).

[Buker 92] Buker Laura I, Neuronal Network and Operations Research: and overview, Computers Ops Res. Vol. 19 (1992).

[Catalina 96] Catalina Gallego Alfredo. Introducción a las Redes Neuronales Artificiales. Artículo Red Internet, www.gui.uva.es/ login/13/redesn.html (1996).

[Hayking 94] Hayking Simón, Neuronal Network. A Comprehensive Foundation 1994 by Macmillan College Publishing Compay.

[Hilera 95] Hilera José Ramón, Redes Neuronales Artificiales, Fundamentos, Modelos y Aplicaciones. Editorial Ra-Ma (1995).

[Foulds 92] Foulds L.R., Graph Theory Aplications. Springer Verlag. New York, (1992).

[Gary 78] Gary M. and Johnson D., Computers and Intractability. Freeman and Company, (1978).

[Goldberg] Goldberg, D. Algorithms in Search, Optimization and Machine Learning Editorial Adison Wesley.

[Nemhawser 88] Nemhawser George and Wolsey Laurence, Integer and Combinatorial Optimization. John Wiley and Sons (1988).

[Papadimitriou 82] Papadimitriou Cristo H., Combinatorial Optimization : Algorithms and complexity. Editorial Prentice Hall (1982).

[Perfetti 95] Perfetti Renzo, Optimization Neuronal Network for Solving Flow Problems, IEEE Transactions on Neuronal Network, Vol. 6 No. 5 (Septiembre 1995).

[Serrano 96] Serrano Cinca Carlos, Gallizo josé. Las Redes Neuronales en el Tratamiento de la Información Financiera, Red Internet //ciberconta.unizar.es/biblioteca/0004/SerGall96.html (Marzo 1996).

[Sierra 95] Sierra Guillermo J., Sistemas Expertos en Contabilidad y Administración de Empresas Editorial Ra-Ma, (1995).

[Zak 95] Zak Stanislaw H., Solving Linear Programing Problems with Neuronal Network : A Comparative Study, IEEE Transactions on Neuronal Networks , Vol. 6, No. 1 (Enero1995).

[Zurada 92] Zurada Jacek M., Introduction to Artificial Neuronal System, (1992).

APENDICE A

Codificación del Sistema de Distribución Aplicando RNA

```
% Programa Simulador
%       Proceso de Distribución Aplicando Redes Neuronales con Supervisión

clc
% Solicitando datos iniciales para el proceso

Nf = input('Cantidad de Nodos Fuente ? ');
Ns = input('Cantidad de Nodos Demanda/Salida ?');
M  = input(' Valor de la constante penalidad ? ');
Emax = input(' Error máximo aceptado en convergencia : ');
paso = input('Longitud de paso : ');
veces = input('Veces que deseas incrementar constante
                        penalidad: ');
incremento = input(' Que incremento se le dará a la Cte. penalidad ? :');
N = Nf+Ns;
B = Nf*Ns;
Max = 0;
x(B) = 0;
b(N) = 0;
llave = 1;
while llave == 1
   se = 0;
   clc
   for i=1 : Nf
     b(i) = input('Valor de entrada al nodo fuente :');
     se = se + b(i);
     if b(i) > Max
           Max = b(i);
     end
     b(i) = b(i) * (-1);
   end
   i = i + 1;
   sd = 0;
   for j=1 : Ns
     b(i) = input('Valor de demanda del nodo :');
     sd = sd + b(i);
     i = i + 1;
   end
   if se ~= sd
         disp(' Error en valores de oferta/demanda...
                  Intentar de nuevo ')
    pause
```

```
      else
        llave = 0;
    end
end  %while llave
cap = input('capacidad por rama : ');

% se asigna la misma capacidad a todas las ramas
for k=1 : B
    a(k) = input('costo causado en la rama ');
    c(k) = cap;
end

% Generando la matriz identidad (A)
sw = 1;
R = 0;
A = zeros(N,B);
for i = 1 : N
   if i <= Nf
          R = R + Ns;
          cont = 0;
          for j = 1 : B
           if j <= R & sw == 1
                 A(i,j) = -1;
            else
                 A(i,j) = 0;
                 sw = 2;
                 cont = cont + 1;
                 if cont == (Ns * (i-1))
                   sw = 1;
                 end
           end
          end
          cont = 0;
    else
          cont = cont + 1;
          sw = cont;
          for j = 1 : B
           if j == sw
                 A(i,j) = 1;
                 sw = sw + Ns;
            else
                 A(i,j) = 0;
           end
          end
   end                %if
```

```
 end                    %for
clc
disp('matris A... ')
disp(A)

% retroalimentacion a través del vector X
clc
primero = 0;
yes = 1;
while yes == 1
 clc;
 disp(' Proceso de retroalimentacion')
 disp(' Calculando el vector  X ... ')
 conta = 0;
 Mu = M;
 for k = 1 : B
    x(k) = 0;
    xant(k) = 0;
    pxant(k) = 0;
    qxant(k) = 0;
 end
 i = 1;
 corr = 0;
 llave = 1;
 termina = 0;
 while llave == 1
        clc
        disp(' muestra valores: ')
        for j = 1 : B
          if xant(j) > 0
            pxant(j) = 0;
          end
          if (c(j) - xant(j)) > 0
             qxant(j) = 0;
           else
                 qxant(j) = c(j) - xant(j);
          end
        end
        conta = conta + 1;
        if conta == 100 & corr <= veces
          Mu = Mu + incremento;
          conta = 0;
          corr = corr + 1;
        end
```

```
        if corr > veces
          termina = termina + 1;
    end

        % Calculando vector resultante (x)
        x = (-a' - (Mu * A')*((A * xant')-b')-Mu*(pxant')
                +Mu*(qxant'))*paso +xant'
        i = i + 1;
        vi = 0;
        for k = 1 : B
          xnew = (abs(x(k)-xant(k)));
          if x(k) < 0
            x(k) = 0;
          end
          if (xnew < Emax) & (x(k) <= c(k)) & (x(k) >= 0)
                vi = vi + 1;
          end
        end
        if corr > veces & termina > veces
          nuevo = input('Deseas parar el proceso ? (1:Si /
                                        0:No) : ');
          if nuevo == 1
            llave = 0;
          end
          Mu = Mu + 50;
          termina = 0;
        end
        disp('valores que converge')
        disp(vi)
        corr
        if vi == B
          disp(' Todos Los valores convergen ');
          pause
          llave = 0;
         else
          xant = x';
        end
 end    %while de convergencia

% Verificando Igualdades
 clc
 v1 = 0;
 disp('verificando igualdades ');
 igual(N) = 0;
```

```
for i = 1 : N
 sum = 0;
 for j = 1 : B
   if A(i,j) ~= 0
       sum = sum + (A(i,j)*x(j));
     end
end
 if round(sum) == b(i)
       igual(i) = sum;
       v1 = v1 + 1;
 else
       igual(i) = sum;
 end
end % for
 if v1 == N
       llave = 0;
       disp(' Se logro cumplir con el conjunto de
                  igualdades ');
       disp(igual)
       disp('Valores del vector X resultante ');
       disp(x);
      else
   disp('No se logro cumplir el conjunto de
                  igualdades ');
       disp(igual)
       pause
 end

%  Permite realizar otro proceso decrementando una unidad
%  una de las ramas que alcanzo cubrir la máxima capacidad

 nuevo = input('Deseas probar de nuevo
                    (1 - Si/0 - No):');
if nuevo == 1 & primero == 0
 % busqueda del flujo igual a capacidad
     for k = 1 : B
       if round(x(k)) == c(k)
          disp(' se localizo un flujo igual a capacidad en
                  la rama');
        disp(k);
        primero = k;
        c(k) = c(k) - 1
        pause
        veces = veces - 2;
```

```
                    break;
                end
            end
    else
            if primero > 0
                c(primero) = c(primero) - 1;
                disp(c)
                veces = veces - 2;
                pause
            end
    end
    if nuevo == 0
            yes = 0;
    end
end %while de veces

disp('fin del proceso ');
disp(' Vector resultante X ... ');
disp(x)
clear all
end
```

APENDICE B

Solución Optima a Problemas de Distribución

APENDICE B

A continuación, mostraremos en este apéndice una de las soluciones óptimas y exactas de los ejemplos presentados en él capitulo 6 (donde se muestran los resultados arrojados por nuestro programa).

Las soluciones aquí presentadas corresponden a las generadas de la herramienta TORA versión 1.03 la cual consiste en dar solución a problemas de programación lineal.

Resultados correspondientes al ejemplo 1:

Nodos Sumidero / Nodos Fuente		NS1 **23**	NS2 **20**	NS3 **20**
NF1	**35**	12	11	12
NF2	**21**	7	7	7
NF3	**7**	4	2	1

Resultados correspondientes al ejemplo 2:

Nodos / Sumidero / Nodos Fuente		NS1	NS2	NS3	NS4
		25	**20**	**16**	**6**
NF1	**30**	8	8	8	6
NF2	**25**	9	8	8	0
NF3	**12**	8	4	0	0

Resultados correspondientes al ejemplo 3:

Nodos / Sumidero / Nodos Fuente		NS1	NS2	NS3	NS4	NS5	NS6
		30	**25**	**22**	**16**	**10**	**4**
NF1	**60**	12	12	11	11	10	4
NF2	**25**	7	7	6	5	0	0
NF3	**12**	4	4	4	0	0	0
NF4	**10**	7	2	1	0	0	0

RESUMEN AUTOBIOGRAFICO

Araceli Campos Ortiz

Candidato para el grado de

Maestro en Ciencias de la Administración con Especialidad en Sistemas

Tesis:

Proceso de Distribución aplicando Redes Neuronales Artificiales con Supervisión.

Campo de Estudio:

Redes Neuronales Artificiales

Biografía:

Nacida en Monclova Coah., el 14 de agosto de 1970; hija de Irma Ortiz Barrera y Doroteo Campos Moreno

Educación:

Egresada del Instituto Tecnológico de Saltillo, grado obtenido de Ingeniero en Sistemas Computacionales en Programación.

Experiencia Profesional:

Participé durante cinco años tanto en el área administrativa como apoyo en la docencia en el Instituto Tecnológico de Estudios Superiores de la Región Carbonífera en la cd. de Agujita, Coah., (1992 – 1997). Actualmente colaboro en el Instituto Tecnológico de Saltillo en la Jefatura del Centro de Cómputo, apoyando a las carreras de Sistemas Computacionales y Licenciatura en Informática como docente.