

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



NUEVO ENFOQUE EN EL DISEÑO Y ENTRENAMIENTO DE
REDES NEURONALES PARA LA CLASIFICACION

POR

M.C. Francisco Román Angel Bello Acosta

TESIS

CON OPCION AL GRADO DE
DOCTOR EN INGENIERIA
CON ESPECIALIDAD EN INGENIERIA DE SISTEMAS

SAN NICOLAS DE LOS GARZA, N. L.

ENERO DE 2001

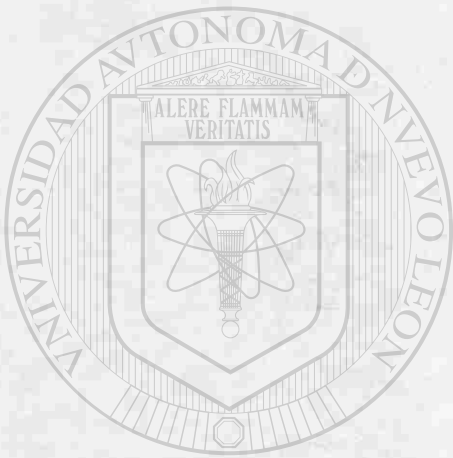
F.R.A.B.A.

MUJERVO EMPFOOQUE EN EL DISEÑO Y ENTRENAMIENTO DE
REDES NEURONALES PARA LA CLASIFICACION

ID
Z5853
.M2
FIME
2001
B4



1020145329

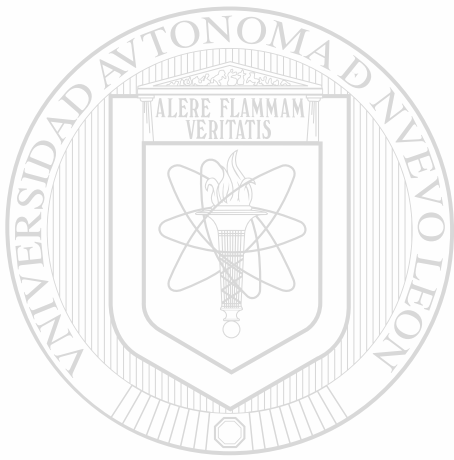


UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

®

DIRECCIÓN GENERAL DE BIBLIOTECAS



UANL

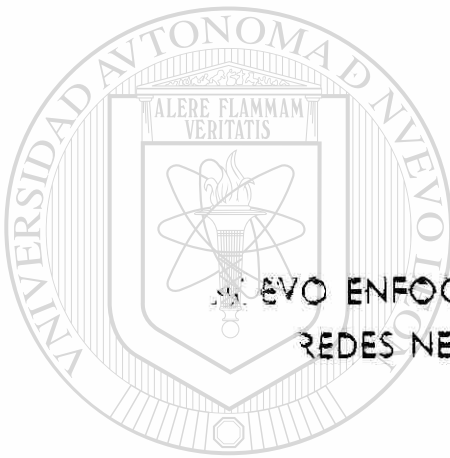
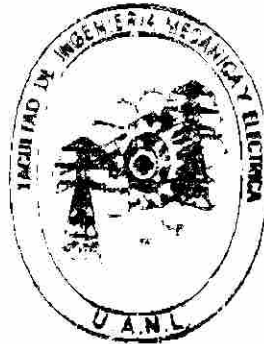
UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

®

DIRECCIÓN GENERAL DE BIBLIOTECAS

m

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



UN NUEVO ENFOQUE EN EL DISEÑO Y ENTRENAMIENTO DE
REDES NEURONALES PARA LA CLASIFICACION

POR

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

M.C. Francisco Román Angel Bello Acosta
DIRECCIÓN GENERAL DE BIBLIOTECAS



TESIS

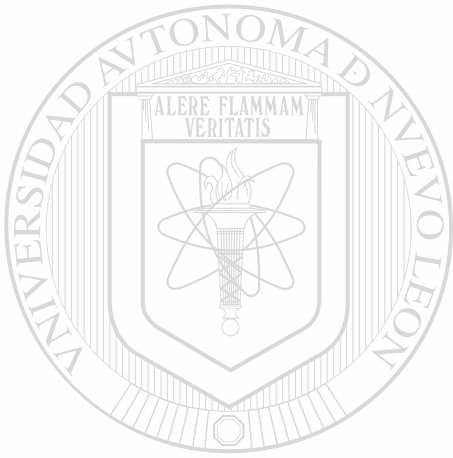
CON OPCION AL GRADO DE
DOCTOR EN INGENIERIA
CON ESPECIALIDAD EN INGENIERIA DE SISTEMAS

NICOLAS DE LOS GARZA, N. L.

ENERO DE 2001

0149-43260

TD
Z5853
•M2
FIME
2001
B4



UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



DIRECCIÓN GENERAL DE BIBLIOTECAS

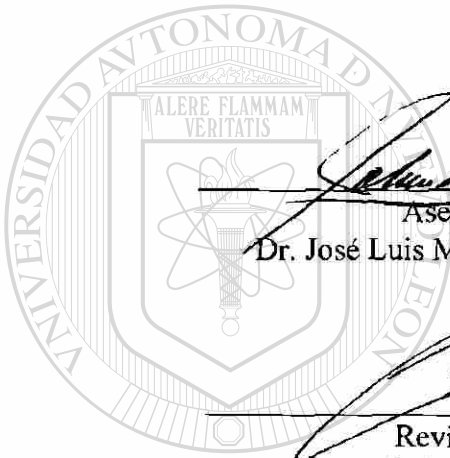


**FONDO
TESIS**

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


Los miembros del comité de tesis recomendamos que la tesis **“Nuevo enfoque en el diseño y entrenamiento de redes neuronales para la clasificación”** realizada por el M.C. Francisco Román Angel Bello Acosta sea aceptada para su defensa con opción al grado de Doctor en Ingeniería con especialidad en Ingeniería de Sistemas.

El Comité de Tesis

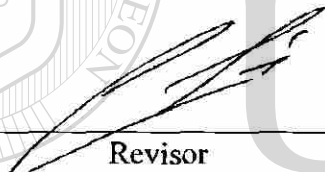




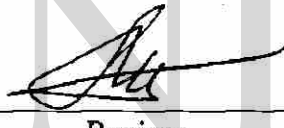
Asesor
Dr. José Luis Martínez Flores



Revisor
Dr. Oscar Leonel Chacón Mondragón

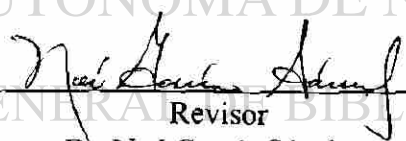


Revisor
Dr. Rafael Colás Ortiz




Revisor
Dr. Igor Litvinchev

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



Revisor
Dr. Noé García Sánchez



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

San Nicolás de los Garza; Nuevo León. Enero de 2001

AGRADECIMIENTOS

Deseo expresar mi sincero agradecimiento al Dr. José Luis Martínez Flores, asesor de esta investigación, por su valiosa ayuda y orientación para la culminación de este trabajo.

Agradezco a los Doctores Rafael Colás Ortiz, Igor Litvinchev y Noé García Sánchez, revisores de esta tesis, por sus acertadas recomendaciones y especialmente al Dr. Oscar Leonel Chacón Mondragón por su paciente revisión y el interés que puso en este trabajo y en mi persona.

Por último, a las autoridades de la Universidad Autónoma de Nuevo León, al Programa de Apoyo a la Investigación Científica y Tecnológica, a la Facultad de Ingeniería Mecánica y Eléctrica y muy particularmente a su División de Estudios de Posgrado, muchas gracias por el apoyo recibido a lo largo de mis estudios doctorales en esta institución.

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



DIRECCIÓN GENERAL DE BIBLIOTECAS

RESUMEN

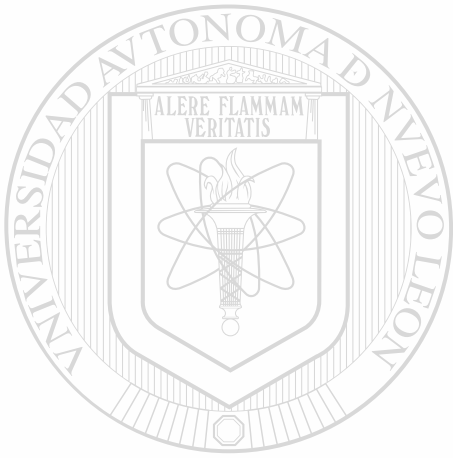
Los modelos de redes neuronales artificiales se están aplicando con gran éxito en la solución de una amplia variedad de problemas complejos, tales como, el reconocimiento de formas y de voz, el control de robot, el filtrado de señales, etc. A pesar de esto, existen una serie de problemas no resueltos en el aprendizaje de las redes neuronales que limitan, en cierto modo, una mayor aplicación de este tipo de metodología. En este trabajo de investigación se tratan tres de los problemas fundamentales en el aprendizaje de redes neuronales artificiales de propagación hacia adelante: 1) Determinar la cantidad de neuronas en la capa oculta para que una red neuronal pueda clasificar un conjunto de patrones, 2) Obtener un algoritmo de entrenamiento, con menor cantidad de operaciones de punto flotante que las diferentes variantes del algoritmo de retropropagación del error y 3) Desarrollar un algoritmo de entrenamiento para redes neuronales clasificadoras de memoria entera.

Para dar solución al primer problema, a partir del análisis de los espacios de patrones y de pesos, se diseñó un algoritmo que determina la cantidad de neuronas en la capa oculta para clasificar un conjunto no contradictorio de patrones en dos clases. Este algoritmo determina, en cada iteración, un hiperplano que separa la mayor cantidad de patrones de una misma clase.

Para resolver el segundo problema, se diseñó un algoritmo para entrenar una red con funciones continuas de activación con un número de operaciones muy inferior al de las diferentes variantes del algoritmo de retropropagación del error. Este algoritmo toma como memoria inicial, la que se obtuvo de la red del problema anterior y es aplicable al filtrado de señales, donde el objetivo es minimizar una función de error cuadrático.

Por último, se estudia el problema de entrenamiento de una red neuronal de memoria entera para clasificar un conjunto de patrones en dos clases. Aquí, utilizando

los procedimientos se desarrollaron para el diseño de la red, se obtienen diferentes hiperplanos separadores y después de realizar un proceso de discretización de los coeficientes, se selecciona el hiperplano que separa la mayor cantidad de patrones de una misma clase. La importancia de este algoritmo de entrenamiento está en que las redes de memoria entera son muy económicas en sus diferentes formas de implementación.



UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



DIRECCIÓN GENERAL DE BIBLIOTECAS

INDICE

Capítulo	Página
1. INTRODUCCION	1
1.1 Antecedentes	1
1.2 Justificación del trabajo de investigación	6
1.3 Objetivos de la investigación	7
1.4 Metodología de investigación	8
1.5 Estructura de la tesis	8
2. MARCO TEORICO	10
2.1 Introducción	10
2.2 Neuronas biológicas y sus modelos artificiales	11
2.3 Clasificación y reconocimiento de patrones	16
2.4 Aprendizaje en redes neuronales de propagación hacia adelante	23
2.4.1 Aprendizaje para clasificadores lineales	23
2.4.2 Aprendizaje como aproximación	24
2.4.3 Regla general de aprendizaje en redes de propagación hacia adelante	25
2.4.4 Reglas clásicas de aprendizaje en redes de propagación hacia adelante	26
2.4.5 Algoritmos constructivos de aprendizaje	37
2.5 Teorema de aproximación universal	40
2.6 Conclusiones	41
3. DISEÑO DE REDES NEURONALES DE PROPAGACION HACIA ADELANTE PARA LA CLASIFICACION	42
3.1 Introducción	42
3.2 Análisis del problema de clasificación	44
3.3 Cota superior del número de neuronas en la capa oculta	47
3.4 Algoritmo de solución	49
3.4.1 Transformación de los problemas planteados	49
3.4.2 Análisis para la selección del elemento pivote	53
3.4.3 Transformación de la matriz de dependencia lineal	57

Capítulo	Página
3.4.4 Obtención de los valores de las variables auxiliares, pesos y umbral	62
3.4.5 Resumen del algoritmo para determinar los hiperplanos separadores	63
3.4.6 Cálculo de los pesos y el umbral entre la capa oculta y la capa de salida	64
3.1 Conclusiones	66
4. ENTRENAMIENTO DE UNA RED NEURONAL PARA LA CLASIFICACION CON FUNCIONES CONTINUAS DE ACTIVACION	67
4.1 Introducción	67
4.2 Formulación del problema	68
4.3 Análisis del problema planteado	69
4.4 Descripción del algoritmo de solución	73
4.5 Resumen del algoritmo de solución	78
4.6 Conclusiones	81
5. DISEÑO Y ENTRENAMIENTO DE UNA RED NEURONAL PARA LA CLASIFICACION DE MEMORIA ENTERA	82
5.1 Introducción	82
5.2 Formulación del problema	84
5.3 Análisis del problema del diseño de la red	85
5.4 Descripción del algoritmo de solución para el problema de diseño	87
5.5 Resumen del algoritmo de solución para el problema de diseño	94
5.6 Conclusiones	97
6. CONCLUSIONES Y RECOMENDACIONES	98
BIBLIOGRAFIA	100
APENDICE A.- Procedimientos auxiliares para el diseño de la red de memoria real	109
APENDICE B.- Procedimientos auxiliares para el diseño de la red de memoria entera	126
LISTA DE FIGURAS	135
RESUMEN AUTOBIOGRAFICO	137

CAPITULO 1

INTRODUCCION

1.1 Antecedentes.

Debido a la complejidad de los problemas actuales de la ciencia y la técnica se están desarrollando de forma acelerada nuevos modelos matemáticos y métodos de las ciencias de la computación que se basan en el comportamiento de sistemas biológicos. Estos métodos son capaces de manejar incertidumbres que aparecen cuando nos enfrentamos a problemas reales y además pueden ofrecer soluciones robustas y de fácil implementación.

Los modelos de redes neuronales artificiales (RNA) están inspirados en el funcionamiento del cerebro humano y son capaces de dar solución a una extensa variedad de problemas complejos. Estos modelos están formados por una gran cantidad de unidades de procesamiento (neuronas), interconectadas entre sí y que operan masivamente en paralelo.

Las ideas primarias sobre los modelos de RNA aparecen en 1936 en los estudios de A. Turing, que es el primero en estudiar el cerebro como una forma de ver el mundo de la computación, aunque los primeros teóricos de la computación neuronal son considerados W. McCulloch y W. Pitts, quienes en 1943 desarrollan

una teoría sobre la forma de trabajar de las neuronas y modelan una red neuronal simple mediante circuitos eléctricos [107].

Los sucesos de mayor importancia en los primeros años de desarrollo de la teoría de las RNA se pueden resumir en los siguientes [104]:

- En 1958 F. Rosenblatt introduce un nuevo enfoque al problema de reconocimiento de patrones en sus trabajos sobre el *Perceptrón*, que es considerada como la red neuronal más antigua.
- En 1960 B. Widrow y M. Hoff introducen el algoritmo de los mínimos cuadrados y lo usan para formular *Adaline*, que es la primera red neuronal aplicada a un problema real. La diferencia fundamental entre el *Perceptrón* y *Adaline* radica en el procedimiento de aprendizaje.
- En 1967 S. Grossberg desarrolla una red neuronal, denominada *Avalancha*, que es utilizada para resolver actividades tales como reconocimiento continuo del habla y aprendizaje del movimiento de los brazos de un robot.

Esta primera etapa de desarrollo se caracteriza fundamentalmente por ver a las redes neuronales como unos modelos matemáticos “maravillosos” con los que se podía resolver una gran cantidad de problemas difíciles.

Por estos años comienzan a aparecer algunos problemas, tales como el problema de clasificación para el “or-exclusivo” y el problema de asignación de crédito, que no podían ser resueltos por los modelos existentes de RNA. Esto conlleva a la realización de un análisis más profundo sobre el funcionamiento de estos modelos, surgiendo así una serie de críticas que culminan en 1969 cuando M. Minsky y S. Papert [76] publican el libro “*Perceptrons*”, donde ofrecen un análisis matemático detallado del *Perceptrón*, criticándolo fuertemente y considerando que la extensión a perceptrones multicapa era completamente estéril.

A partir de la aparición de este libro disminuye considerablemente la producción científica sobre redes neuronales, la mayoría de los investigadores se orientan hacia la Inteligencia Artificial y solo continúa trabajando en el tema un pequeño grupo, que por lo general elude utilizar el término de red neuronal.

En 1982 aparecen dos trabajos científicos que hacen resurgir el interés por las redes neuronales.

J. Hopfield [32] presenta un trabajo en el que describe con claridad y rigor matemático una red que es una variación del asociador lineal. Además, mostró cómo pueden trabajar tales redes y qué se puede hacer con ellas.

T. Kohonen [51] publica un artículo sobre un modelo de red neuronal con capacidad de formar mapas de características de manera similar a como ocurre en el cerebro humano, basándose en el principio de formación de mapas topológicos para establecer características comunes entre la información de entrada a la red. Una explicación más detallada sobre este trabajo puede encontrarse en la referencia [50].

A pesar de que estos trabajos no son sobre redes neuronales de propagación hacia adelante, sí permiten que se vuelva a retomar este tipo de modelo, comenzando así una nueva etapa de desarrollo de la Teoría de Redes Neuronales Artificiales.

Desde los primeros años de desarrollo se había notado, que si un problema no podía ser resuelto por una red neuronal de propagación hacia adelante, entonces era necesario agregar capas de neuronas ocultas, pero no se tenía fundamentación teórica sobre esto y además no existían algoritmos de entrenamiento.

En ese momento, los problemas fundamentales a los que se enfrentan los investigadores son:

1. Diseñar algoritmos de entrenamiento para redes multicapas.
2. Determinar la cantidad de capas ocultas de neuronas en la red.
3. Determinar la cantidad de neuronas en cada capa oculta.

El primero de estos problemas es resuelto en 1986, cuando aparece publicado un artículo por Rumelhart, Hinton y Williams [82], donde reportan un algoritmo de aprendizaje para redes multicapas, conocido como retropropagación del error (*backpropagation*). Además también fue obtenido de forma independiente por otros dos investigadores al mismo tiempo, Parker [80] y LeCun [67].

Es necesario señalar [103] que en 1974 Werbos, en su tesis doctoral realizada en la Universidad de Harvard, describe este algoritmo para modelos de redes más generales, donde las redes neuronales pueden ser consideradas como un caso

particular. Desafortunadamente este trabajo de Werbos fue desconocido para la comunidad científica por más de una década.

El primero en dar una solución al segundo problema es Cybenko, quien demostró rigurosamente que una sola capa de neuronas ocultas es suficiente para poder aproximar cualquier función continua soportada sobre un hipercubo unitario. Este resultado se conoce en el entorno de las RNA como el *Teorema de Aproximación Universal* y fue publicado, por primera vez en 1988, como un reporte técnico de la Universidad de Illinois y un año después en la referencia [15].

En 1989 son reportado otros dos trabajos sobre las capacidades de aproximación de las redes de propagación hacia adelante: Funahashi [25] y Hornik, Stinchcombe y White [33].

El teorema de aproximación universal es un teorema de existencia y es considerado como el resultado teórico de mayor importancia para este tipo de redes.

No obstante, este teorema deja abiertos dos problemas de gran importancia para la implementación de sistemas de RNA de propagación hacia adelante :

1. Cantidad de neuronas en la capa oculta
2. Acotamiento de los pesos de la red.

En el problema de mantener la propiedad de aproximador universal con una cantidad de neuronas limitada y pesos acotados se está trabajando actualmente en diferentes institutos de investigación a nivel mundial y aunque se han obtenido algunos resultados particulares [3,21,35,38,39,55,59,60,61,95], el problema general aún no está resuelto.

En 1992 se reportan dos resultados que garantizan que la superficie de error no presente mínimos locales. Estos resultados representan condiciones suficientes para que el algoritmo de entrenamiento sea convergente.

El primero es presentado por X. Yu [105] y plantea que son suficientes $T-1$ neuronas en la capa oculta, pero este valor resulta demasiado grande en la solución de problemas prácticos, por lo que esto representa una cota superior para el número de neuronas en la capa oculta.

El otro resultado es dado por Gori y Tesi [27], pero las condiciones que plantean son muy difíciles de verificar en la práctica. Otra forma de abordar este problema ha sido mediante algoritmos constructivos [10,11,12,23,71,80], pero solo se pueden aplicar a redes con funciones discretas de activación.

Para determinar el número de neuronas en la capa oculta, aún se sigue utilizando un procedimiento de ensayo y error [44,29], es decir, se construye una red de gran dimensión y se van eliminando neuronas de la capa oculta mientras se mantenga la convergencia del algoritmo de entrenamiento, o bien se construye una red pequeña y se le van agregando neuronas hasta que se alcance la convergencia.

En el problema del acotamiento de los pesos de la red también se está trabajando actualmente y se han obtenido algunos resultados teóricos [35,38,53,59,62,97], pero los mayores esfuerzos están orientados hacia redes con memoria discreta y en particular las que utilizan memoria entera [22,46,71,72,77,84,86].

Cuando la memoria de la red está formada por valores enteros pequeños, estos modelos resultan muy atractivos, ya que sus diferentes formas de implementación son sumamente económicas.

El caso ideal es cuando los pesos toman solamente valores del conjunto $\{-1, 0, 1\}$, que son las llamadas redes libres de multiplicación [47], ya que para calcular la entrada neta a cada neurona, no es necesario realizar operaciones de multiplicación, solo sumas y restas. Además, en este caso los pesos de la red pueden ser almacenados mediante cadenas binarias de longitud 2.

Otro caso de gran interés es cuando los pesos están restringidos a tomar valores enteros del intervalo $[-3; 3]$, ya que se requieren solo de 3 bits para su almacenamiento y además proporcionan mayor diversificación para los hiperplanos separadores [48].

En estos dos tipos de redes los principales resultados han sido obtenidos por Khan [46] en su tesis doctoral presentada en 1996.

Otro problema que ha llamado la atención de los investigadores en RNA es que el algoritmo de retropropagación del error resulta demasiado costoso, desde el punto

de vista computacional, a pesar de todas las mejoras implementadas por diferentes autores [14,19,27,28,31,40,81,88,99,106].

El resultado más significativo en este sentido fue reportado por M. Biancini y M. Gori [7] en 1996. Ellos reducen el cálculo del gradiente del orden $O(M^2.T)$ al orden $O(M.T)$, donde M es el total de pesos de la red y T es la cantidad de patrones en el conjunto de aprendizaje.

1.2 Justificación del trabajo de investigación.

Debido a sus fundamentos, los modelos de RNA presentan varias características semejantes a las del cerebro. Esos modelos son capaces de aprender de la experiencia, de generalizar de casos anteriores a casos nuevos, de extraer características esenciales a partir de datos que contienen información irrelevante, etc. Esto hace que ofrezcan numerosas ventajas, por lo que este tipo de metodología se está aplicando en múltiples áreas.

A pesar de que los modelos de RNA se están aplicando con gran éxito en la solución de problemas reales de alta complejidad, todavía existen una serie de problemas teóricos y prácticos no resueltos.

Los tres problemas fundamentales en el aprendizaje de redes neuronales de propagación hacia adelante son:

1. Diseño de la red neuronal.
2. Desarrollo de algoritmos eficientes de entrenamiento desde el punto de vista computacional.
3. Diseño y entrenamiento de redes neuronales de memoria entera.

El problema del diseño de la red neuronal consiste en determinar la cantidad de neuronas en la capa oculta para que el algoritmo de entrenamiento sea convergente, ya que la cantidad de neuronas en las capas de entrada y de salida es fija para cada problema particular.

En el caso de redes neuronales para la clasificación, este problema está estrechamente relacionado con el problema de determinar el mayor subconjunto

linealmente separable de un conjunto de entrenamiento, el cual es de la clase NP-completo [90].

El segundo problema está relacionado con las deficiencias del algoritmo de retropropagación del error [17,18,3052,54,65,66,69,83,94], las cuales son :

- La arquitectura de la red neuronal es determinada por ensayo y error.
- Alto número de operaciones, tanto en su fase hacia adelante en el cálculo de las salidas, como en su fase hacia atrás en la modificación de los parámetros de la red.
- La cantidad de ciclos de entrenamiento depende de la memoria inicial, la cual se selecciona de forma aleatoria.

La importancia del problema de diseño y entrenamiento de redes neuronales de memoria entera se debe al bajo costo de producción de este tipo de tecnología. Tanto el problema de diseño, como el de entrenamiento de la red cuando la memoria es entera están catalogados como NP-completos [46].

A partir del análisis de estos problemas definimos la orientación de nuestro trabajo de investigación hacia el desarrollo de métodos y algoritmos de entrenamiento para modelos de RNA de propagación hacia adelante que determinen la cantidad de neuronas en la capa oculta, que sean eficientes computacionalmente y que obtengan una memoria con valores enteros pequeños.

1.3 Objetivos de la investigación.

Del análisis de los problemas fundamentales existentes en el aprendizaje de redes neuronales para la clasificación, planteamos los siguientes objetivos de investigación.

1. Desarrollar un algoritmo para determinar la cantidad de neuronas en la capa oculta de una red neuronal de propagación hacia adelante.
2. Diseñar una nueva forma de entrenar una red con menor número de operaciones que la mejor variante del algoritmo de aprendizaje existente (*backpropagation*).

3. Diseñar un método de entrenamiento para redes neuronales de propagación hacia adelante con memoria entera.

1.4 Metodología de investigación.

Para llevar a cabo el trabajo de investigación se propuso la siguiente metodología a seguir :

1. Investigación bibliográfica sobre los problemas del aprendizaje en redes neuronales de propagación hacia adelante.
2. Análisis de los problemas de clasificación de patrones, de separabilidad lineal y de transformación de conjuntos de patrones en conjuntos linealmente separables. Esto permitirá desarrollar un método que determine el número de neuronas en la capa oculta de redes neuronales de propagación hacia adelante.
3. Análisis del problema de aproximación de funciones y su aplicación en el aprendizaje en redes neuronales de propagación hacia adelante.
4. Estudio de los fundamentos teóricos y análisis riguroso del problema de aprendizaje en redes neuronales de propagación hacia adelante, así como de su algoritmo de entrenamiento. Esto permitirá proponer un método más eficiente en relación con la cantidad de operaciones a realizar en el proceso de aprendizaje.
5. Análisis de la propiedad de aproximación universal en redes neuronales con memoria entera y de métodos de programación entera para diseñar un algoritmo de entrenamiento para redes con memoria entera.
6. Conclusiones de la investigación y recomendaciones para futuros trabajos en esta línea de investigación.

1.5 Estructura de la tesis.

El trabajo de tesis está compuesto por 6 capítulos. El capítulo 2 es de marco teórico y se da una descripción de los conceptos, métodos y algoritmos que se referencian en este trabajo de tesis.

En el capítulo 3 se estudia el problema de diseño de redes neuronales para la clasificación. Aquí, a partir del estudio del problema de clasificación y reconocimiento de patrones y del análisis de la estructura de los espacios de patrones y de pesos, se desarrolla un algoritmo para determinar la cantidad de neuronas en la capa oculta.

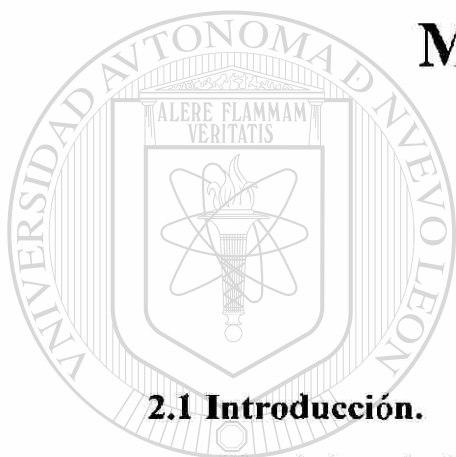
En el capítulo 4 se analiza el problema de la eficiencia del algoritmo de entrenamiento y se propone una nueva forma de entrenar redes neuronales para la clasificación que toma como memoria inicial la que se obtuvo de la solución del problema de diseño de la red. El funcionamiento de este algoritmo se basa en explotar las propiedades de las funciones continuas de activación. Con esta nueva forma de entrenamiento se logra reducir significativamente la cantidad de operaciones del proceso de aprendizaje de la red, así como la cantidad de parámetros a modificar. Además, la memoria inicial no se toma de forma aleatoria.

El capítulo 5 está dedicado a los problemas de diseño y entrenamiento de redes neuronales de memoria entera para la clasificación. En este capítulo los pesos de la red están restringidos a tomar valores enteros del intervalo $[-3, 3]$ y los métodos propuestos son modificaciones de los algoritmos desarrollados en los capítulos 3 y 4.

El capítulo 6 es el de conclusiones y recomendaciones. Aquí se resaltan los principales logros y limitaciones de los resultados obtenidos en este trabajo de investigación, así como se dan una serie de recomendaciones para trabajos futuros, en relación con la generalización de los resultados obtenidos.

CAPITULO 2

MARCO TEORICO



2.1 Introducción.

Uno de los principales objetivos y preocupaciones de los científicos a lo largo de la historia ha sido conseguir diseñar y construir máquinas capaces de realizar procesos con cierta inteligencia. De los intentos realizados en este sentido se han llegado a definir las líneas fundamentales para la obtención de máquinas inteligentes.

A pesar de disponer de herramientas y lenguajes de programación diseñados expresamente para el desarrollo de máquinas inteligentes, existe un problema de fondo que limita enormemente los resultados que se puedan obtener y es que estas máquinas se implementan sobre procesadores basados en una filosofía de funcionamiento, que se apoya en una descripción secuencial del proceso de tratamiento de la información. El desarrollo de estos procesadores no deja de seguir la línea de que, una máquina puramente mecánica es capaz de realizar tareas mecánicas, de forma increíblemente rápida, pero es incapaz de obtener resultados aceptables

cuando se trata de tareas sencillas para un ser humano como reconocimiento de formas, de voz, etc.

El desarrollo de la lógica formal permitió contar con una notación precisa para representar aseveraciones relacionadas con todo lo que existe en el mundo, así como sus relaciones mutuas.

Para crear sistemas inteligentes, la Inteligencia Artificial se esfuerza por elaborar programas que puedan describir un problema en notación lógica y encontrarle solución.

Actuar racionalmente implica actuar de manera que se logren los objetivos deseados, con base en cierto supuesto. Un agente es algo que es capaz de percibir y actuar. De acuerdo con este enfoque, la Inteligencia Artificial se considera como el estudio y construcción de agentes racionales.

Un agente es todo aquello que puede considerarse que percibe su ambiente mediante sensores y que responde o actúa en tal ambiente por medio de efectores. El objetivo fundamental es diseñar y construir agentes racionales, es decir, agentes que logren un buen desempeño en su ambiente.

Uno de los objetivos fundamentales de los Sistemas de Redes Neuronales Artificiales es también el diseño y construcción de agentes racionales, pero lo hacen de una forma completamente diferente. Las redes neuronales se pueden enmarcar dentro de las denominadas redes de autoproseso, que están formadas por nodos procesadores de información de cuyas interacciones locales depende el comportamiento global del sistema.

2.2 Neuronas biológicas y sus modelos artificiales.

La teoría y la modelación de redes neuronales artificiales está inspirada en la estructura y funcionamiento del sistema nervioso.

La *neurona* o célula nerviosa es la unidad funcional básica de los tejidos del sistema nervioso, incluido el cerebro. Las neuronas están formadas por el cuerpo de célula o *soma*, donde se aloja el núcleo de la célula. Del cuerpo de la célula salen

ramificaciones conocidas como *dendritas*, y sale también una más larga denominada *axón*. Como se muestra en la figura 1.

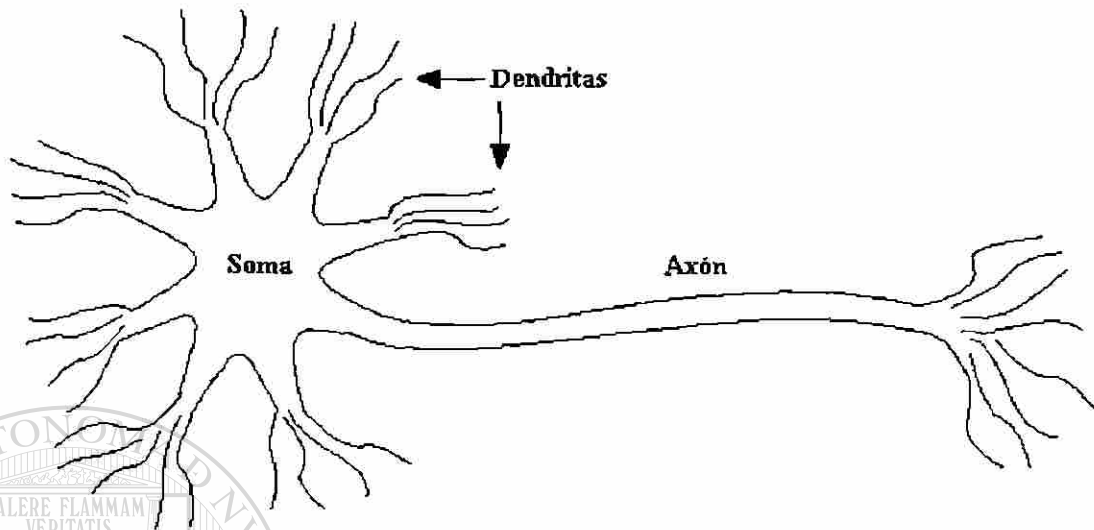


Figura 1 : Partes de la neurona.

Las dendritas se ramifican tejiendo una tupida red alrededor de la célula, mientras que el axón tiene una longitud, por lo general, de 1 cm (100 veces el diámetro del cuerpo de la célula) y en casos extremos llega a medir hasta un metro. Finalmente, el axón también se ramifica en filamentos, mediante los cuales establece conexión con las dendritas y cuerpos de otras células. Esta unión o conexión se le conoce como *sinapsis*.

Cada neurona establece sinapsis con una cantidad variable de otras neuronas, que oscila desde una docena hasta ciento de miles.

Las señales se propagan de neurona a neurona mediante una complicada reacción electroquímica. Las sinapsis liberan sustancias químicas transmisoras y entran a la dendrita con la cual se eleva o se reduce el potencial eléctrico del cuerpo de la célula. Una vez que el potencial eléctrico rebasa cierto valor de umbral, se envía al axón un impulso eléctrico o potencial de acción. El impulso se difunde a través de las ramas del axón y finalmente llega a la sinapsis y libera transmisores en los cuerpos de otras células.

Las sinapsis que aumentan el potencial se conocen como excitadoras, y las que lo disminuyen se denominan inhibitoras. La característica más importante de las conexiones sinápticas es que muestran plasticidad, es decir, alteraciones a largo plazo de la intensidad de las conexiones como respuesta al patrón de estimulación. Las neuronas establecen nuevas conexiones con otras neuronas, y en ocasiones con grupos completos de neuronas capaces de migrar de un sitio a otro.

Se considera que los mecanismos anteriores constituyen el fundamento del aprendizaje en el cerebro.

Este modelo del sistema nervioso parte de que las neuronas se comunican entre sí por medio de impulsos eléctricos y que forman una red neuronal que tiene una estructura compleja de interconexiones. La entrada a la red proviene de receptores sensitivos que están en contacto con el mundo exterior. Estos sensores envían estímulos en forma de impulsos eléctricos que llevan la información a la red de neuronas. Como resultado del procesamiento de la información en el sistema nervioso central, los efectores controlan y dan respuesta en forma de diversas acciones.

De esta forma este sistema está formado por los receptores, la red neuronal y los efectores en el control del organismo y sus acciones, como se muestra en la figura 2.



Figura 2: Flujo de información en el sistema nervioso.

Las redes neuronales artificiales son modelos que intentan reproducir el comportamiento del cerebro humano. Cada modelo neuronal consiste de un elemento de procesamiento con conexiones de entrada sinápticas y una salida simple.

En la figura 3 se muestra un modelo general de un elemento de procesamiento (PE). La i -ésima señal de entrada que recibe este elemento de procesamiento se representa por x_i . El flujo de señal de una neurona de entrada es considerado unidireccional y está indicado por el sentido de la flecha. Cada conexión tiene asociada una magnitud llamada peso o intensidad de conexión y se denota por w_i .

Cada PE determina un valor de entrada neta basándose en todas las conexiones de entrada. Por lo general, se calcula el valor de entrada neta mediante la suma ponderada de las entradas por sus pesos correspondientes y se denota por *net* es decir,

$$net = \sum_{i=1}^n w_i x_i = W' X \quad (2.1)$$

donde $W = [w_1, w_2, \dots, w_n]'$ es el vector de los pesos, y $X = [x_1, x_2, \dots, x_n]'$ es el vector de entradas.

Nótese, además, que el umbral se puede considerar como un peso más para una señal de entrada igual a -1 y que por comodidad se asume que es igual a x_n .

Una vez que la entrada neta ha sido calculada, se puede calcular el valor de salida aplicando la función de activación $o = f(net)$

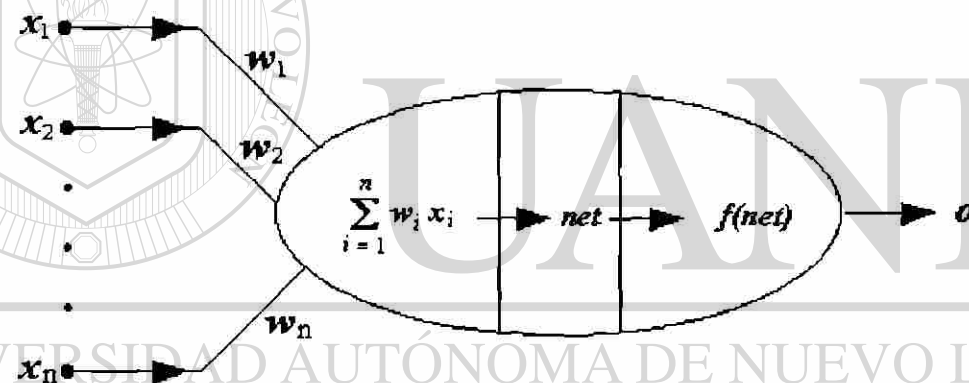


Figura 3 : Simbología para las neuronas artificiales

Las funciones de activación más usadas son

$$f(net) = \frac{2}{1 + e^{-\lambda net}} - 1 \quad , \quad (2.2)$$

$$f(net) = \begin{cases} 1, & net \geq 0 \\ -1, & net < 0 \end{cases} \quad (2.3)$$

Las funciones de activación (2.2) y (2.3) son conocidas como función continua bipolar (función sigmoide) y función binaria bipolar (función signo). Note que cuando $\lambda \rightarrow \pm\infty$, la función continua tiende a $\text{sgn}(net)$.

Otras funciones que también se usan como funciones de activación son

$$f(\text{net}) = \frac{1}{1 + e^{-\lambda \text{net}}} \quad (2.4)$$

$$f(\text{net}) = \begin{cases} 1, & \text{net} \geq 0 \\ 0, & \text{net} < 0 \end{cases} \quad (2.5)$$

que se denominan función continua unipolar y función binaria unipolar respectivamente.

La mayoría de los modelos neuronales emplean uno de estos tipos de funciones de activación

Dada una capa de m neuronas, sus valores de salida o_1, o_2, \dots, o_m se pueden agrupar en un vector de salida

$$O = [o_1, o_2, \dots, o_m]^t \quad (2.6)$$

donde o_i es la señal de salida de la neurona i .

El dominio del vector O está definido en un espacio m -dimensional por

$$(-1, 1)^m = \{O \in R^m, o_i \in (-1, 1)\}, i = 1, 2, \dots, m$$

para la función bipolar continua, o por

$$(0, 1)^m = \{O \in R^m, o_i \in (0, 1)\}, i = 1, 2, \dots, m$$

para la función unipolar continua. Es evidente que el dominio del vector O , en este caso, es el interior de un cubo m -dimensional.

Para valores de salida binarios o_i , el dominio de O en el espacio m -dimensional está dado por

$$\{-1, 1\}^m = \{O \in R^m, o_i \in \{-1, 1\}\}, i = 1, 2, \dots, m$$

para la función bipolar discreta, o por

$$\{0, 1\}^m = \{O \in R^m, o_i \in \{0, 1\}\}, i = 1, 2, \dots, m$$

para la función unipolar discreta.

En este caso el dominio del vector O está formado por los vértices de un cubo m -dimensional.

2.3 Clasificación y Reconocimiento de Patrones.

En la etapa de funcionamiento de redes neuronales, el proceso cálculo de una salida O para una entrada X dada, es conocido como *recordar* y el objetivo de esta fase es restaurar la información. *Recordar* corresponde a descifrar el contenido almacenado que puede haber sido introducido a la red previamente.

El término de *patrón* es utilizado para referirse a los elementos del conjunto de entradas que se le presentan a la red en la etapa de entrenamiento. Más aún, un patrón debe ser una descripción cuantitativa de un objeto, evento o fenómeno.

Supongamos que un conjunto de patrones puede ser almacenado en la red. Existen diferentes formas de asociación entre la entrada y la salida, como se muestra en la figura 4.

Autoasociación : A la red se le presenta una entrada y ella responde con miembro del conjunto almacenado que más se parece a esta entrada. Este proceso generalmente es usado para reconstruir una determinada información de entrada que se presenta incompleta o distorsionada.

Heteroasociación : es cuando la red almacena parejas de datos $(A_1, B_1); (A_2, B_2); \dots; (A_n, B_n)$ de forma tal que cuando se presente una entrada ella la asocia al patrón almacenado mas parecido A_i y responde con la salida correspondiente B_i .

Clasificación : Es cuando el conjunto de patrones de entrada es dividido en clases o categorías. En este caso al presentar una entrada, la red debe dar como respuesta a qué clase pertenece. Generalmente las clases son expresadas por vectores de salida de valores discretos y se usan funciones de activación binarias.

Reconocimiento : En este caso también el conjunto de patrones de entrada es dividido en clases o categorías, pero la nueva entrada que se le presenta a la red no es exactamente igual a ningún miembro del conjunto de patrones de entrada, por lo que la red lo asocia a una clase donde se encuentre el patrón que más se le asemeje.

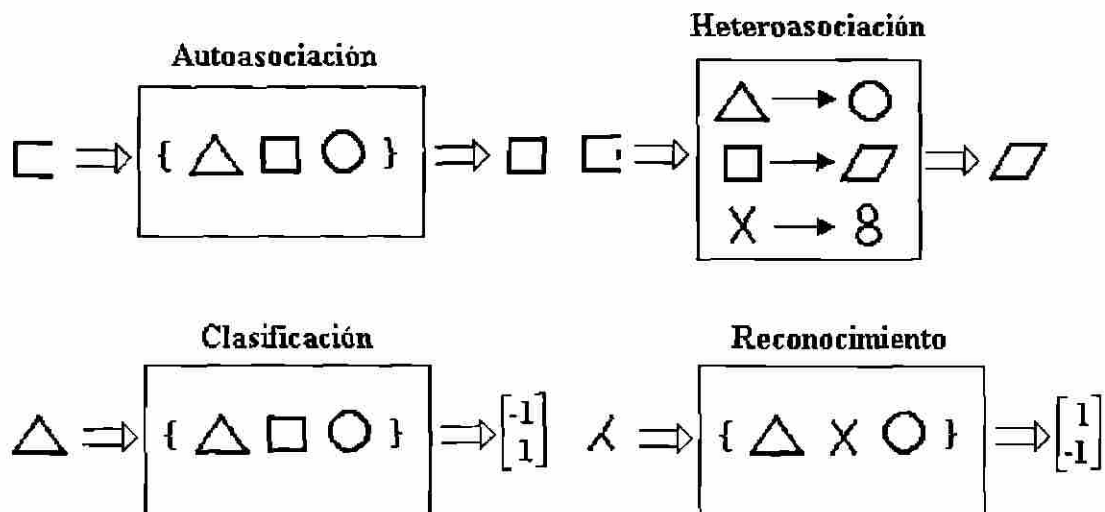


Figura 4 : Tipos de asociación.

La principal función de un sistema de clasificación es decidir a qué clase pertenece la entrada que se presente. Conceptualmente, el problema puede ser descrito como una transformación de conjuntos o funciones desde el espacio de entrada al espacio de salida, que es llamado espacio de clasificación.

El objetivo de la clasificación de patrones es asignar un objeto físico, evento o fenómeno a una de las clases o categorías preestablecidas. El problema de clasificación de patrones puede ser considerado como uno de discriminación de datos de entrada dentro de una población de objetos, mediante la búsqueda de atributos invariantes entre los miembros de la población.

El estudio extensivo de procesos de clasificación ha conducido al desarrollo de modelos matemáticos abstractos que proporcionan las bases para el diseño de clasificadores [8,24,68,92,98].

En la figura 5 se muestra un diagrama de bloques para sistemas de reconocimiento y clasificación.

Los sistemas de clasificación contienen un traductor de entradas que provee al extractor de características de patrones de entrada. Generalmente, las entradas al extractor de características son conjuntos de vectores de datos que pertenecen a cierta categoría. El extractor de características reduce la dimensión de los datos, por lo que el espacio de características es de menor dimensión que el espacio de patrones. El

vector de características contiene solamente las características esenciales del vector de patrones que le permita mantener la probabilidad de hacer una clasificación correcta.

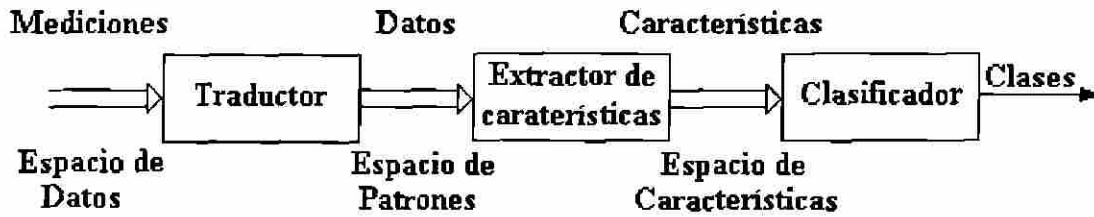


Figura 5 : Diagrama de bloques para sistemas de reconocimiento y clasificación.

La entrada es representada como un vector X y la clasificación a la salida del sistema es obtenida por un clasificador implementado por una función de decisión $i_0(X)$ que puede tomar uno de los valores discretos $1, 2, \dots, R$, donde la respuesta representa la categoría a la cual puede ser asignado el patrón, como se muestra en la figura 6. Es decir

$$i_0 = i_0(X) \quad (2.7)$$

donde $X = [x_1, x_2, \dots, x_n]'$

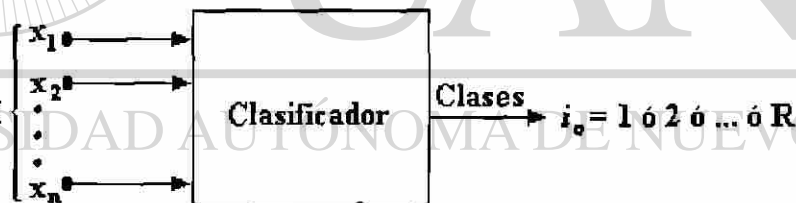


Figura 6 : Esquema de un clasificador multicategoría.

La función de clasificación (de decisión) de la ecuación (2.7) representa una transformación o aplicación de un vector n -dimensional X a una de las categorías $i_0(X)$.

La clasificación también puede ser descrita convencionalmente en forma geométrica. Un patrón puede ser representado por un punto en un espacio euclidiano n -dimensional \mathbf{R}^n , denominado espacio de patrones. Los puntos en este espacio corresponden a los elementos del conjunto de patrones que son vectores n -

dimensionales. Un clasificador de patrones aplica conjuntos de puntos del espacio \mathbf{R}^n en el espacio de uno de los números $i_0(X) = 1, 2, \dots, R$ como describe la función de decisión (2.7).

Las regiones denotadas por C_j son llamadas regiones de decisión y las fronteras que separan una región de las otras se denominan superficies de decisión. En un espacio \mathbf{R}^n las superficies de decisión son hipersuperficies de $n-1$ dimensiones.

Durante la etapa de clasificación, para determinar la pertenencia a una categoría, el clasificador necesita basarse en la comparación de los cálculos para el patrón de entrada X de R funciones de discriminación $g_1(X), g_2(X), \dots, g_R(X)$. Las funciones de discriminación toman valores escalares y un patrón pertenece a la i -ésima categoría si y solo si se cumple que

$$g_i(X) > g_j(X), \quad \forall i, j = 1, 2, \dots, R; \quad i \neq j \quad (2.8)$$

Esto significa que dentro de la región C_j la i -ésima función de discriminación toma el mayor valor. Esta propiedad de la función de discriminación $g_i(X)$ de tomar el valor máximo para un patrón que pertenezca a la clase i es fundamental y es usado para seleccionar formas específicas de las funciones $g_i(X)$.

Las funciones de discriminación $g_i(X)$ y $g_j(X)$ para regiones de decisión contiguas C_i y C_j definen las superficies de decisión entre patrones de las clases i y j en el espacio \mathbf{R}^n .

El diagrama de bloques para un clasificador básico lo podemos adoptar como el que se muestra en la figura 7.

Para un patrón dado el i -ésimo discriminador calcula el valor de la función $g_i(X)$ que se denomina simplemente discriminante. El selector del máximo implementa la condición (2.8) y selecciona la mayor de todas las entradas produciendo una respuesta igual al número de categoría $i_0(X)$.

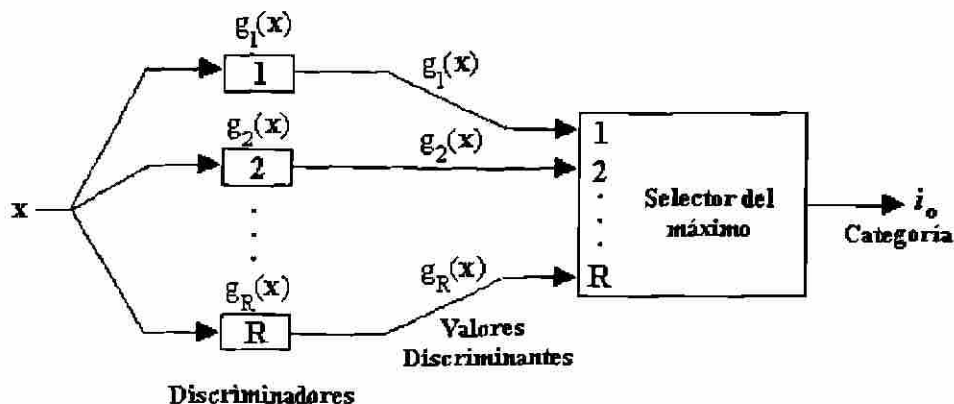


Figura 7: Diagrama de bloque de un clasificador básico

En el caso que $R = 2$, el clasificador es denominado dicotomizador y en este caso la condición (2.8) puede ser reducida a la inspección del signo de la siguiente función de discriminación

$$g(X) = g_1(X) - g_2(X) \quad (2.9)$$

Por lo que aquí la regla general (2.9) puede ser reescrita como

$$\begin{aligned} g(X) > 0, & \text{ si } X \in C_1 \\ g(X) < 0, & \text{ si } X \in C_2 \end{aligned} \quad (2.10)$$

La evaluación de esta condición es más fácil de implementar en la práctica que la condición del máximo. Para construir un dicotomizador simple puede ser usada una unidad lógica de umbral (TLU) simple como la que se muestra en la figura 8.

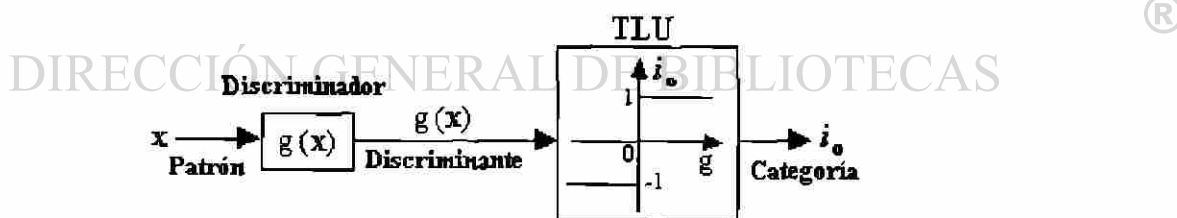


Figura 8 : Dicotomizador.

Una TLU puede ser considerada como una versión binaria de una neurona, en la cual los pesos son introducidos como en un perceptrón binario. Las respuestas 1, -1 de la TLU pueden ser interpretadas como indicaciones de las categorías 1 y 2 respectivamente. La TLU simple implementa la función signo definida como

$$i_0(X) = \text{sgn}(g(X)) = \begin{cases} 1, & g(X) > 0 \\ -1, & g(X) < 0 \end{cases}$$

El diseño de clasificadores se puede basar por completo en el cálculo de las fronteras de decisión que se derivan de los patrones y de su pertenencia a determinada clase.

Un clasificador eficiente, que tenga un diagrama de bloques como el que se mostró en la figura 7, puede ser descrito, en general, por funciones de discriminación que dependan de forma no lineal de las entradas x_1, x_2, \dots, x_n . Pero, el uso de funciones de discriminación no lineales puede ser eludido mediante el diseño de clasificadores de propagación hacia adelante que sean multicapas.

En el caso de la clasificación lineal, la superficie de decisión es un hiperplano. En la figura 9 se muestra una función discriminante lineal en el caso bidimensional.

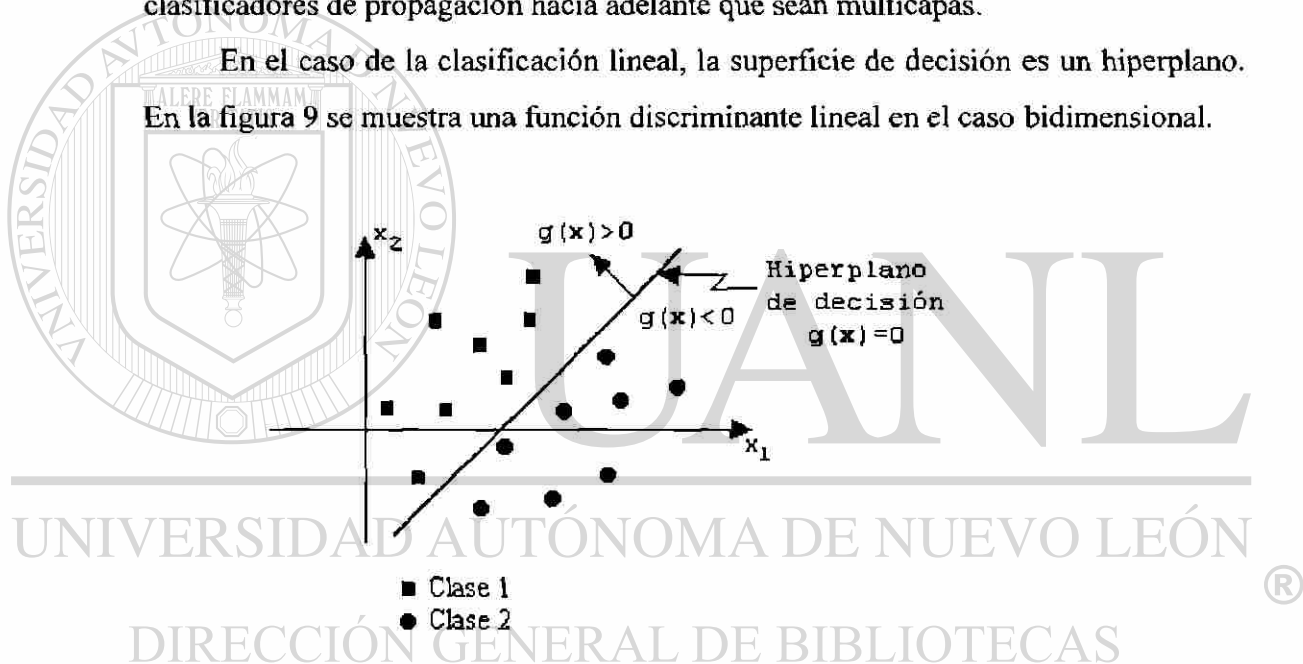


Figura 9 : Ilustración de una función discriminante lineal bidimensional.

La forma lineal de las funciones discriminantes también puede ser usada para la clasificación en más de dos categorías. En el caso de R categorías separables dos a dos existen a lo sumo $R(R-1)/2$ hiperplanos de decisión. Para un número de categorías elevado, algunas de las regiones de decisión C_i, C_j pueden no ser contiguas, por lo que se eliminan los hiperplanos de decisión.

El diagrama de bloques para un clasificador lineal se muestra en la figura 10 que se puede ver como un caso especial del diagrama de la figura 7.

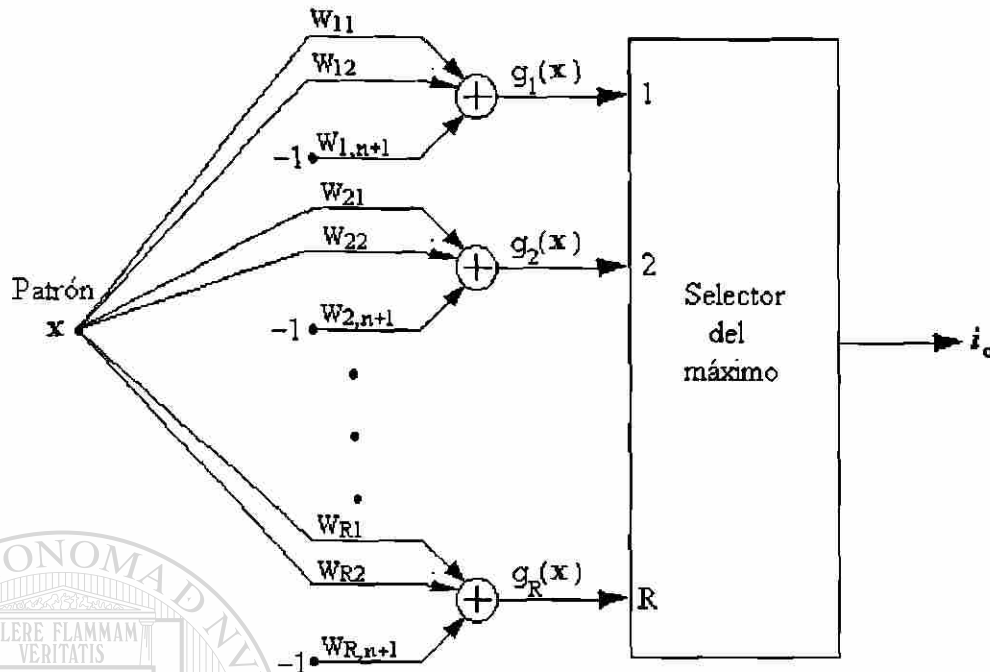


Figura 10 : Clasificador lineal.

El clasificador lineal está compuesta por R nodos de cálculo de producto escalar y un selector de máximo. Durante la clasificación, se calculan R funciones discriminantes $g_i(X)$ para el patrón de entrada X , estos valores se dan como entrada al selector del máximo que da como resultado el número de clase para el cual el determinante tuvo mayor valor.

Para introducir la noción de patrones linealmente separables, se asume que el conjunto de patrones X se puede dividir en los subconjuntos X_1, X_2, \dots, X_R respectivamente. Si un clasificador lineal puede clasificar los patrones de X_i como pertenecientes a la clase " i " para $i = 1, 2, \dots, R$, entonces el conjunto de patrones X es linealmente separable.

Usando las propiedades de las funciones discriminantes lineales, la separabilidad lineal se puede formular más formalmente. Si existen R funciones lineales de X tales que

$$g_i(X) > g_j(X), \quad \forall X \in X_j; i, j = 1, 2, \dots, R; i \neq j,$$

entonces los conjuntos de patrones X_j son linealmente separables.

2.4 Aprendizaje en redes neuronales de propagación hacia adelante.

2.4.1 Aprendizaje para clasificadores lineales.

Cuando se analiza el problema de clasificación de patrones, resulta interesante el estudio de aquellos clasificadores, que sus capacidades de decisión son generadas por patrones de entrenamiento mediante aprendizaje, entrenamiento o algoritmos iterativos.

La clasificación de un dato es aprendida gradualmente mediante la inspección repetida y clasificación de ejemplos.

Cuando el tipo de función discriminante ha sido seleccionado, el algoritmo de aprendizaje da como resultado la solución para los coeficiente, inicialmente desconocidos, de la función discriminante, que se obtiene a partir del conjunto de patrones de entrenamiento.

Para el estudio de clasificadores entrenables (adaptativos) se asume que :

1) El conjunto de patrones de entrenamiento es conocido, así como la clasificación de todos sus elementos, por lo que el entrenamiento es supervisado.

2) Las funciones discriminantes tienen una forma lineal y solo sus coeficientes [®] son ajustados en el proceso de entrenamiento.

Bajo estas suposiciones, un clasificador entrenable puede ser implementado por el aprendizaje mediante ejemplos. El interés, por lo tanto, está enfocado hacia vectores de datos de entrada para los cuales se conoce su clasificación correcta, y a los que se denominan prototipos de clase.

El problema de clasificación consistirá entonces en determinar las superficies de decisión en un espacio n-dimensional a partir de la correcta clasificación de los prototipos y que permita con un grado de confianza realizar correctamente el reconocimiento y la clasificación de patrones desconocidos que no hayan sido usados en el entrenamiento. La única limitación que se tiene para que los patrones

desconocidos sean reconocidos es que tengan el mismo formato que se usó en los patrones de entrenamiento.

2.4.2 Aprendizaje como aproximación.

En general, el aprendizaje es un cambio permanente y relativo en el comportamiento basado en la experiencia. En redes neuronales, el aprendizaje es un proceso más directo y se puede entender como una relación causa-efecto que puede ser vista como una relación que transforma las entradas en las salidas para un conjunto de ejemplos de pares entrada-salida.

Una fundamentación clásica a este problema proviene de la Teoría de Aproximación. Esta teoría consiste en aproximar una función continua multivariable $h(X)$ por otra función $H(W, X)$, donde $X = [x_1, x_2, \dots, x_n]'$ es el vector de las entradas y $W = [w_1, w_2, \dots, w_n]'$ es el vector de los parámetros (pesos). En este sentido a las redes neuronales se les puede ver como un sistema que puede aprender la aproximación de relaciones.

Con este enfoque, el aprendizaje consiste en hallar el vector W que de la mejor aproximación posible de $h(X)$ para un conjunto de ejemplos de entrenamiento \mathbf{X} . La selección de la función $H(W, X)$ para representar a $h(X)$ es conocido como un problema de representación. Una vez que haya sido seleccionada $H(W, X)$, el algoritmo de aprendizaje de la red es aplicado para encontrar los parámetros óptimos W^* .

Una formulación más precisa del problema de aprendizaje puede ser establecida como los cálculos que involucran a W^* de forma tal que

$$\rho[H(W^*, X), h(X)] \leq \rho[H(W, X), h(X)]$$

donde $\rho[H(W, X), h(X)]$ es una función distancia y representa una medida cualitativa de aproximación entre $H(W, X)$ y $h(X)$. Cuando el ajuste se determina sobre la base del cuadrado de las diferencias para un conjunto de ejemplos de entrenamiento \mathbf{X} , entonces la distancia tiene la forma de la suma de los errores cuadrados.

2.4.3 Regla general de aprendizaje en redes de propagación hacia adelante.

Una neurona es considerada como un elemento adaptativo. Sus pesos son modificables en dependencia de la señal de entrada que recibe, de su valor de salida y de la respuesta del supervisor. En el caso del aprendizaje no supervisado la modificación de los pesos se basa solamente en la señal de entrada y/o en los valores de salida.

En el proceso de aprendizaje se modifican las componentes w_{ij} , que están sobre las conexiones de la j -ésima entrada con la i -ésima neurona de cada vector de pesos W_i . En general las entradas pueden ser las salidas de otras neuronas o entradas externas.

En la figura 11 se muestra el entrenamiento de una neurona.

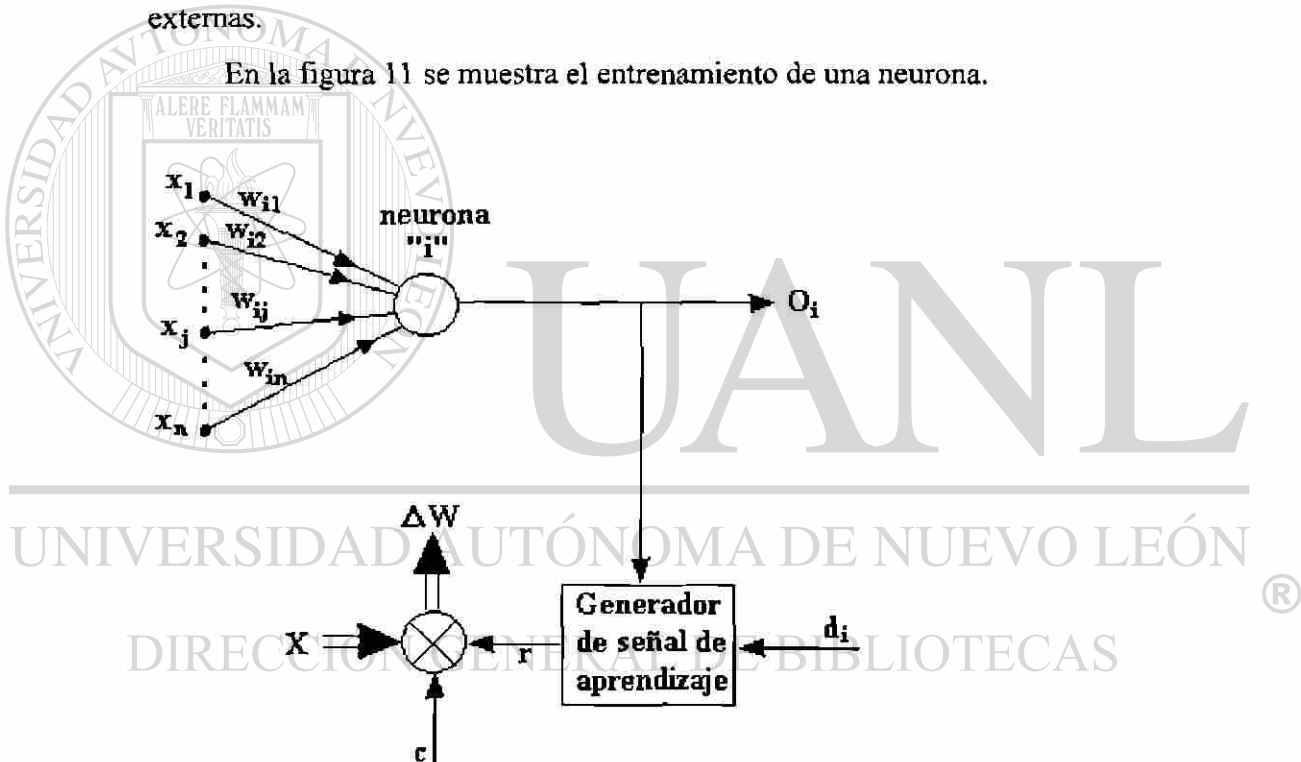


Figura 11: Esquema del aprendizaje supervisado en una neurona.

Para el estudio de redes neuronales se toma la siguiente regla general de aprendizaje :

“ El vector de pesos $W_i = [w_{i1}, w_{i2}, \dots, w_{in}]'$ se incrementa en proporción al producto de la entrada X y de la señal de aprendizaje r ”.

Esta señal de aprendizaje es en general una función de W_i , de X y en algunos casos de la señal del supervisor d_i .

$$r = r(W_i, X_i, d_i)$$

El incremento del vector de pesos W_i es producido por el paso de aprendizaje en el instante de tiempo t de acuerdo a la regla de aprendizaje

$$\Delta W_i(t) = c r[W_i(t), X(t), d_i(t)] X(t) \quad (2.11)$$

donde c es un número positivo denominado constante de aprendizaje y determina la velocidad del aprendizaje.

De (2.11) se tiene que

$$W_i(t+1) = W_i(t) + c r[W_i(t), X(t), d_i(t)] X(t)$$

Para sistemas discretos en el tiempo se utilizan superíndices para especificar el paso de aprendizaje, es decir, para el k -ésimo paso se tiene

$$W_i^{k+1} = W_i^k + c r[W_i^k, X^k, d_i^k] X^k$$

por lo que el aprendizaje toma la forma de una secuencia discreta de modificaciones de los pesos.

2.4.4 Reglas clásicas de aprendizaje en redes de propagación hacia adelante.

Las reglas de aprendizaje más comunes [2,5,29,107] para las redes neuronales de propagación hacia adelante son:

- **Regla de aprendizaje del perceptrón.**

El análisis del diseño de clasificadores de patrones está basado en el cálculo de las superficies de decisión a partir de los prototipos o grupos de patrones muestrales. Los coeficientes de las funciones discriminantes lineales son los componentes del vector de pesos y pueden ser determinados a partir de la información que se tiene del conjunto de patrones sobre la pertenencia de sus elemento a clases o categorías.

Los vectores de patrones de muestrales X_1, X_2, \dots, X_T forman una secuencia de entrenamiento y se les presentan al clasificador conjuntamente con la respuesta de

salida correcta. El clasificador modifica los parámetros mediante un aprendizaje iterativo supervisado.

El aprendizaje del clasificador se basa en la experiencia lograda por la comparación de la respuesta correcta con la respuesta obtenida y su estructura usualmente es ajustada después de cada respuesta incorrecta. El ajuste se realiza sobre la base del valor del error generado.

El dicotomizador consta de $n+1$ pesos y de una TLU como elemento de decisión binaria. La entrada a este TLU es la suma ponderada de los componentes del vector de entrada. Ver figura 12.

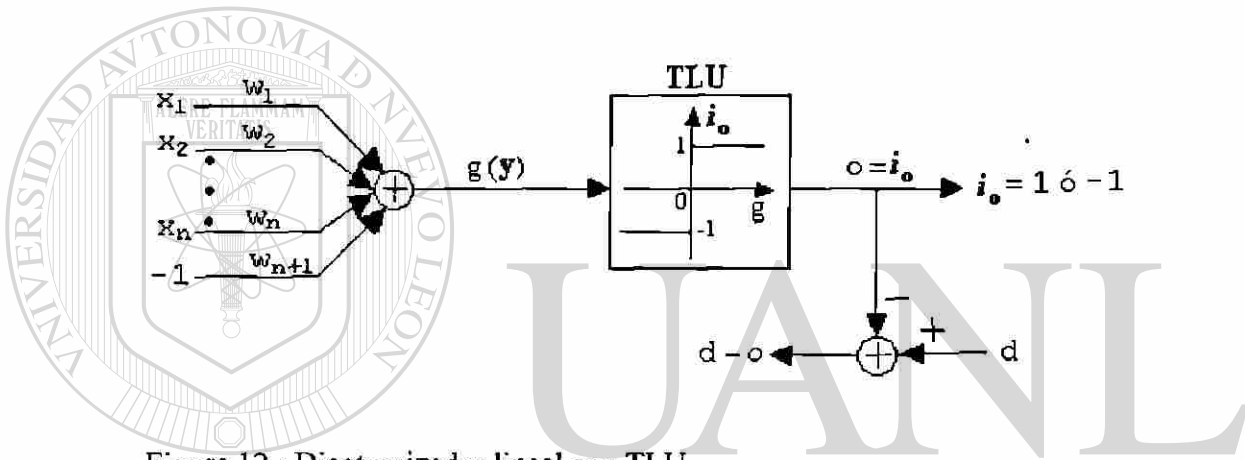


Figura 12 : Dicotomizador lineal con TLU.

Al presentarle al dicotomizador un patrón, la información sobre el error puede ser usada para adaptar los pesos.

Por lo que puede verse que esto es lo mismo que la regla de aprendizaje para el perceptrón discreto.

Para esta regla la señal de aprendizaje es la diferencia entre la salida deseada y la salida obtenida de la neurona al aplicarle una entrada X . Además, aquí el aprendizaje es supervisado. Ver figura 13.

En este caso la señal de aprendizaje tiene la forma

$$r = d_i - o_i$$

donde $o_i = \text{sgn}(W_i' \cdot X)$ y d_i es la respuesta deseada.

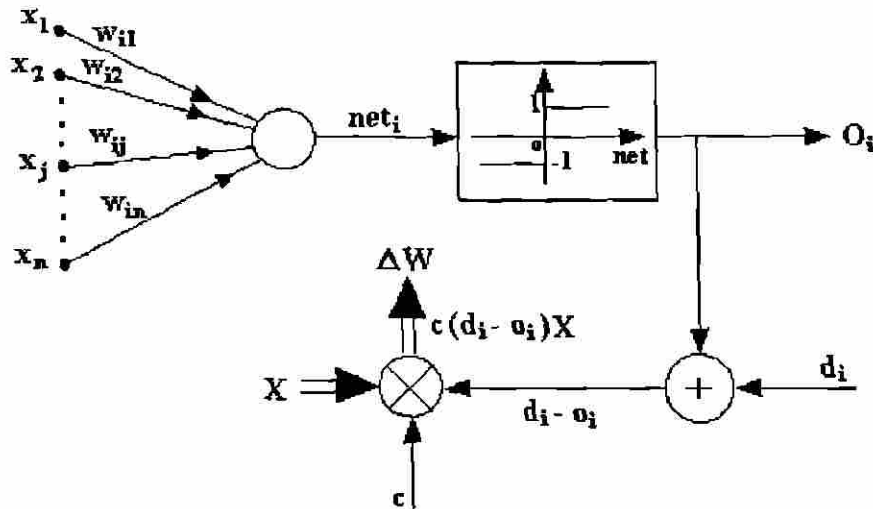


Figura 13 : Aprendizaje en un perceptrón discreto.

El ajuste de los pesos en esta regla de aprendizaje se realiza mediante la expresión

$$\Delta W_i = c [d_i - \text{sgn}(W_i' X)] X$$

y cada elemento del vector de pesos se ajusta por

$$\Delta w_{ij} = c [d_i - \text{sgn}(W_i' X)] x_j \quad (2.12)$$

Esta regla es aplicable solo para neuronas de respuesta binaria y las relaciones

(2.12) expresan la regla para el caso bipolar binario. Bajo esta regla los pesos son ajustados solo en caso que o_i sea incorrecta. Obviamente, como la respuesta deseada es 1 ó -1, el ajuste de los pesos se reduce a

$$\Delta W_i = \pm 2cX$$

donde el signo “+” es aplicable cuando $d_i = 1$ y $o_i = -1$ y el signo “-” cuando $d_i = -1$ y $o_i = 1$.

- **Regla delta de aprendizaje.**

Esta regla es aplicable solamente para las funciones de activación continuas bipolar y unipolar definidas anteriormente y para un aprendizaje supervisado. En esta regla la señal de aprendizaje se denomina delta y está definida por :

$$r = [d_i - f(W_i' X)] f'(W_i' X)$$

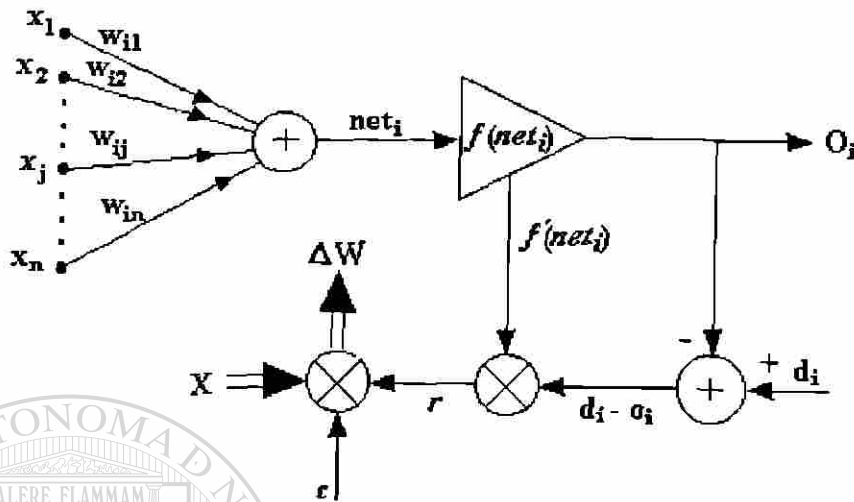


Figura 14 : Aprendizaje en un Perceptrón continuo.

El término $f'(W_i' X)$ es la derivada de la función de activación, calculada para $net = W_i' X$.

Esta regla de aprendizaje puede ser obtenida directamente de la condición del error mínimo cuadrático entre d_i y o_i .

Calculando el vector gradiente respecto a W_i del error cuadrático definido por

$$E = \frac{1}{2} (d_i - o_i)^2$$

o a su expresión equivalente

$$E = \frac{1}{2} [d_i - f(W_i' X)]^2$$

obtenemos el vector gradiente del error.

$$\nabla E = -[d_i - o_i] f'(W_i' X) X \quad (2.13)$$

Los componentes del vector gradiente serán

$$\frac{\partial E}{\partial w_{ij}} = -[d_i - o_i] f'(W_i' X) x_j$$

Como se requiere la minimización del error cuadrático, para realizar el cambio de los pesos, se toma la dirección negativa del gradiente, es decir,

$$\Delta W_i = -\eta \nabla E \quad (2.14)$$

donde η es una constante positiva.

Entonces de (2.13) y (2.14) se obtiene

$$\Delta W_i = \eta [d_i - o_i] f'(W_i X) X \quad (2.15)$$

y para cada componente del vector de pesos

$$\Delta w_{ij} = \eta [d_i - o_i] f'(W_i X) x_j, j = 1, 2, \dots, n$$

Aplicando a este caso la fórmula general de aprendizaje definida por (2.11) se tiene que el ajuste de los pesos se realiza mediante

$$\Delta W_i = c [d_i - o_i] f'(W_i X) X$$

y comparándola con la expresión (2.15) se ve que son idénticas ya que c y η son constantes arbitrarias.

Para el entrenamiento de este tipo de redes es necesario inicializar el vector de los pesos.

Esta regla requiere del cálculo de $f'(net)$ en cada paso. Para este propósito se usa el resultado que $f'(net) = \frac{1}{2}(1 - o_i^2)$, que es válido para la función de activación continua bipolar.

Este resultado es de gran utilidad ya que expresa la pendiente de la función de activación en términos de la señal de salida de la neurona.

Como este método está basado en el movimiento del vector de pesos sobre la superficie de los pesos en la dirección negativa del gradiente de error, requiere de pequeños valores de c , el cual es seleccionado de forma empírica y ajustado en el proceso de aprendizaje.

- **Regla de aprendizaje del perceptrón discreto multicategoría.**

Para poder entrenar un clasificador multicategoría es necesario asumir que cada clase es linealmente separable respecto a las restantes clases, esto es equivalente a que existan R funciones discriminantes lineales tales que

$$g_i(X) > g_j(X), \quad \forall X \in \mathbf{X}_i; i, j = 1, 2, \dots, R; i \neq j$$

Por comodidad, el umbral de la i -ésima neurona es denotado por $w_{i,n+1}$ y es agregado a los vectores de pesos, también a los patrones de entrada se le agrega una componente $x_{i,n+1} = -1$, obteniéndose los vectores ampliados

$$\bar{W}_i = [w_{i1}, w_{i2}, \dots, w_{in}, w_{i,n+1}]'$$

$$Y_i = [x_{i1}, x_{i2}, \dots, x_{in}, -1]'$$

En el perceptrón discreto multicategoría, el selector del máximo puede ser sustituido por R unidades lógicas de umbral, que son más fáciles de implementar. En la figura 15 se muestra la estructura de un perceptrón discreto multicategoría.

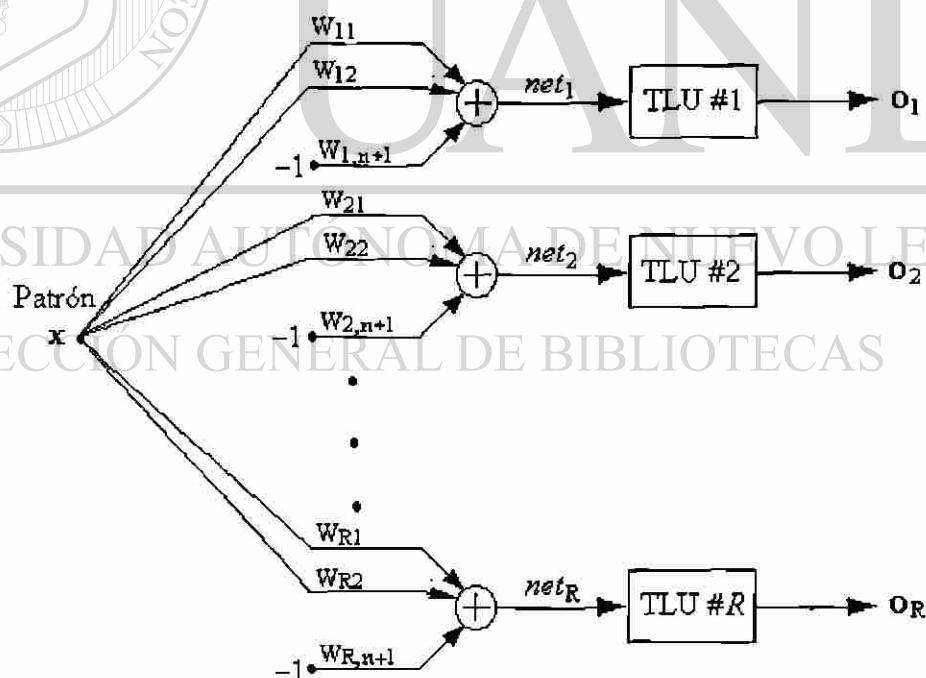


Figura15 : Estructura de un perceptrón discreto multicategoría.

En este caso la salida deseada correspondiente al patrón X_i , será un vector $\mathbf{d}_i = [d_{i1}, d_{i2}, \dots, d_{iR}]'$, donde las componentes d_{ij} cumplen que

$$d_{ij} = \begin{cases} 1, & \text{si } \mathbf{X}_i \in C_k, j = k \\ -1, & \text{si } \mathbf{X}_i \in C_k, j \neq k \end{cases}$$

En esta regla de entrenamiento, el objetivo es que el vector de salida de la red $\mathbf{o}_i = [o_{i1}, o_{i2}, \dots, o_{iR}]'$, para cada patrón de entrenamiento X_i , coincida con el vector de salida deseada correspondiente, por tanto, la red se considera entrenada si se cumple que

$$o_{ij} = \begin{cases} 1, & \text{si } \mathbf{X}_i \in C_k, j = k \\ -1, & \text{si } \mathbf{X}_i \in C_k, j \neq k \end{cases}$$

lo que significa que $\mathbf{d}_i - \mathbf{o}_i = [0, 0, \dots, 0]' \in \mathbf{R}^R$.

Aquí, el problema de aprendizaje puede ser formulado como:

Dado un el conjunto de entrenamiento $\{(\mathbf{X}_1, \mathbf{d}_1), (\mathbf{X}_2, \mathbf{d}_2), \dots, (\mathbf{X}_T, \mathbf{d}_T)\}$, donde

$\mathbf{X}_i = (x_{i1}, x_{i2}, \dots, x_{in}) \in \mathbf{R}^n$, $i = 1, \dots, T$ representan a los patrones de entrenamiento y

$\mathbf{d}_i = (d_{i1}, d_{i2}, \dots, d_{iR})$, $i = 1, \dots, T$ es el vector de salida deseada correspondiente al patrón X_i , siendo

$$d_{ij} = \begin{cases} 1, & \text{si } \mathbf{X}_i \in C_k, j = k \\ -1, & \text{si } \mathbf{X}_i \in C_k, j \neq k \end{cases}$$

determinar una matriz de pesos

$$\overline{\mathbf{W}} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} & w_{1,n+1} \\ w_{21} & w_{22} & \dots & w_{2n} & w_{2,n+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{R1} & w_{R2} & \dots & w_{Rn} & w_{R,n+1} \end{bmatrix}$$

de forma tal que $E = \frac{1}{2} \sum_{i=1}^T \sum_{j=1}^R (d_{ij} - o_{ij})^2 = 0$,

donde $o_{ij} = \text{sgn}(\text{net}_{ij})$, $i = 1, \dots, T$; $j = 1, \dots, R$

Si el patrón está mal clasificado, la modificación de los pesos se realiza según la expresión

$$\bar{W}_j^{(k+1)} = \bar{W}_j^{(k)} + \frac{c}{2} (d_{ij} - o_{ij}) Y_i, \quad i = 1, \dots, T; j = 1, \dots, R$$

- **Regla de aprendizaje del perceptrón continuo multicategoría.**

Este tipo de redes difiere del caso discreto en que utiliza funciones continuas de activación, por lo que las componentes del vector de salida, en el caso de la función bipolar, cumplen $|o_{ij}| < 1$. Ver figura 16.

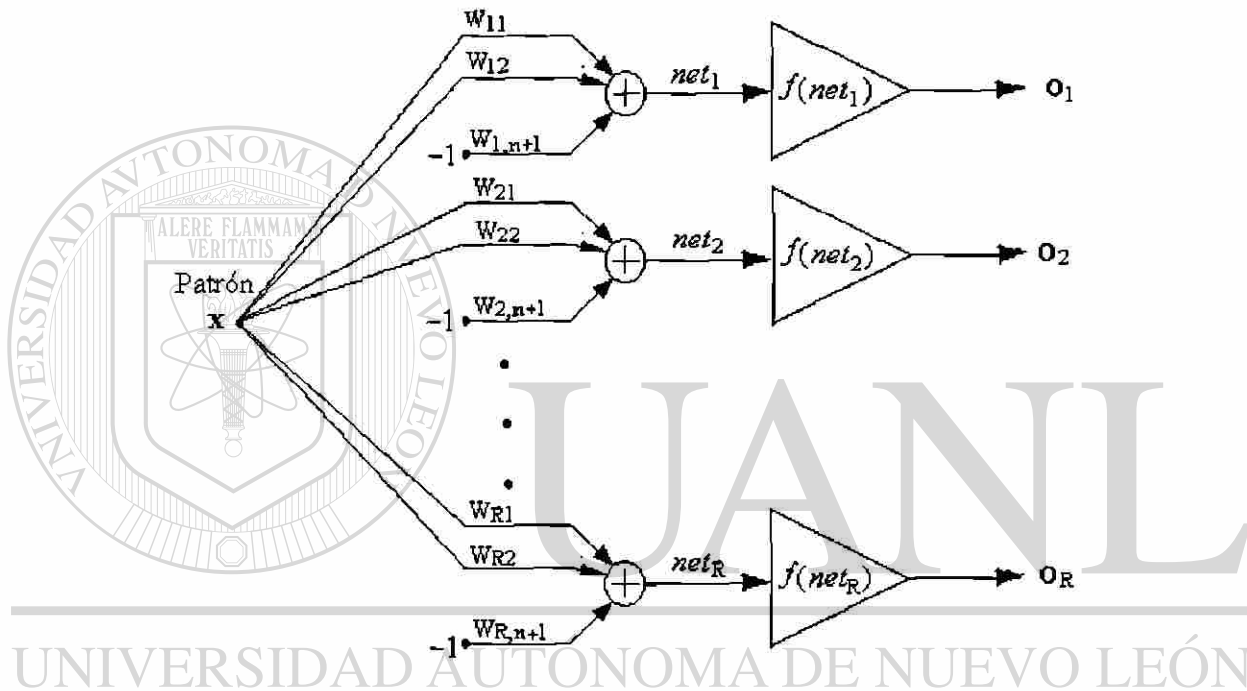


Figura 16 : Estructura de un perceptrón continuo multicategoría.

Aquí, la función de error tiene la misma forma que en el caso discreto, es decir

$$E = \frac{1}{2} \sum_{i=1}^T \sum_{j=1}^R (d_{ij} - o_{ij})^2,$$

pero $o_{ij} = \frac{2}{1 + e^{-\lambda net_{ij}}} - 1$, $i = 1, \dots, T; j = 1, \dots, R$, en el caso bipolar y

$o_{ij} = \frac{1}{1 + e^{-\lambda net_{ij}}}$, $i = 1, \dots, T; j = 1, \dots, R$, en el caso unipolar.

En esta regla de aprendizaje, los pesos se modifican según el método del descenso acelerado, después que se le presenta un patrón a la red.

Aquí, el problema de aprendizaje puede ser formulado como:

Dado un el conjunto de entrenamiento $\{(\mathbf{X}_1, d_1), (\mathbf{X}_2, d_2), \dots, (\mathbf{X}_T, d_T)\}$, donde $\mathbf{X}_i = (x_{i1}, x_{i2}, \dots, x_{in}) \in \mathbf{R}^n$, $i = 1, \dots, T$ representan a los patrones de entrenamiento y $d_i = (d_{i1}, d_{i2}, \dots, d_{iR})$, $i = 1, \dots, T$ es el vector de salida deseada correspondiente al patrón X_i , siendo

$$d_{ij} = \begin{cases} 1, & \text{si } \mathbf{X}_i \in C_k, j = k \\ -1, & \text{si } \mathbf{X}_i \in C_k, j \neq k \end{cases}$$

determinar una matriz de pesos

$$\bar{W} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} & w_{1,n+1} \\ w_{21} & w_{22} & \dots & w_{2n} & w_{2,n+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{R1} & w_{R2} & \dots & w_{Rn} & w_{R,n+1} \end{bmatrix}$$

de forma tal que $E = \frac{1}{2} \sum_{i=1}^T \sum_{j=1}^R (d_{ij} - o_{ij})^2 < E_{max}$,

donde $o_{ij} = \frac{2}{1 + e^{-\lambda n e_{ij}}} - 1$, $i = 1, \dots, T$; $j = 1, \dots, R$ y E_{max} es el error de aproximación.

Los componente de la matriz de pesos se modifica según la expresión

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

Como los pesos se modifican al presentar a la red cada patrón, entonces la

función de error puede ser escrita como $E = \sum_{i=1}^T E_i$, donde $E_i = \frac{1}{2} \sum_{j=1}^R (d_{ij} - o_{ij})^2$ es el error asociado al patrón X_i .

En este caso los pesos se modifican según la expresión

$$\Delta \bar{W}_j = c (d_{ij} - o_{ij}) f'(\bar{W}_j' Y_i) Y_i$$

donde

$$\bar{W}_j = [w_{j1}, w_{j2}, \dots, w_{jn}, w_{j,n+1}]'$$

$$Y_i = [x_{i1}, x_{i2}, \dots, x_{in}, -1]^t$$

- **Regla de aprendizaje para redes multicapas.**

Cuando el conjunto de patrones no es linealmente separable, una red neuronal con una sola capa de neuronas no es capaz de realizar correctamente la tarea de clasificación. En estos casos son necesarias superficies de decisión no lineales para separar el conjunto de patrones, por lo que se implementan redes con capas de neuronas ocultas.

En la figura 17 se muestra una red neuronal multicapa con dos capas ocultas de neuronas.

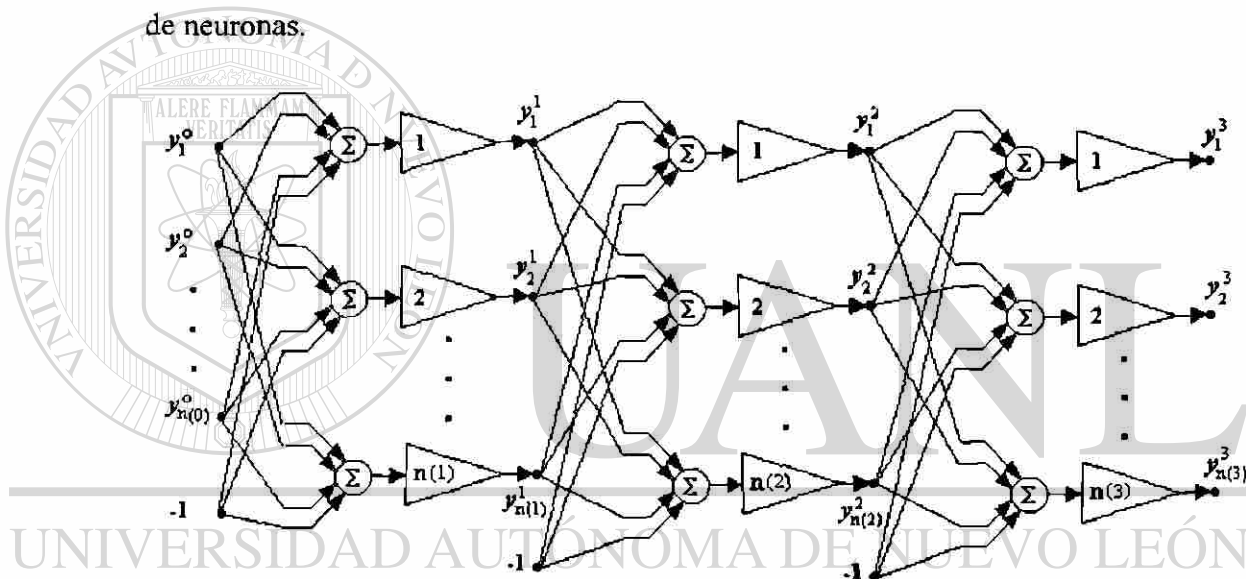


Figura 17 : Red Neuronal con dos capas ocultas.

Hasta el momento no se ha encontrado ninguna generalización de la regla de aprendizaje del perceptrón para redes multicapas, solo existen algunos algoritmos constructivos de aprendizaje que van agregando neuronas durante el proceso de entrenamiento, que están basados en la regla de aprendizaje del perceptrón discreto o en alguna de sus modificaciones.

Para el entrenamiento de redes multicapas de propagación hacia adelante se utiliza un algoritmo, conocido como el algoritmo de retropropagación del error (*backpropagation*) y es una generalización de la regla delta de aprendizaje.

El algoritmo de retropropagación del error es un algoritmo iterativo, basado en la técnica del descenso acelerado y su objetivo de entrenamiento consiste en minimizar determinada función de error.

Supongamos que se tiene un conjunto de aprendizaje no contradictorio de la forma $\{(X_1, \mathbf{d}_1), (X_2, \mathbf{d}_2), \dots, (X_T, \mathbf{d}_T)\}$,

donde $X_i = (x_{i1}, x_{i2}, \dots, x_{im})' \in R^n$, ($i = 1, \dots, T$) representan los de patrones de entrenamiento y $\mathbf{d}_i \in R^L$ es la salida deseada correspondiente al patrón X_i , siendo además

$$d_{ij} = \begin{cases} 1, & \text{si } \mathbf{X}_i \in C_k, j = k \\ -1, & \text{si } \mathbf{X}_i \in C_k, j \neq k \end{cases}$$

Para analizar el funcionamiento del algoritmo de retropropagación del error en el entrenamiento de una red con $L-1$ capas de neuronas ocultas ($L \geq 2$), se introducen las siguientes notaciones:

- $n(l)$ a la cantidad de neuronas en la capa l , ($0 \leq l \leq L$).
- y_j^l a la salida de la neurona j en la capa l para el vector X_i .
- w_{kj}^l al peso de la conexión entre la neurona k de la capa l y la neurona j de la capa $l-1$

- net_j^l a la entrada a la neurona j en la capa l , ($1 \leq l \leq L$) para el vector X_i .
- e_j al error generado por el vector X_i en la neurona j de la capa de salida.

Según la notación utilizada, entonces

$$n(0) = n$$

$$y_{ij}^0 = x_{ij}, (j = 1, \dots, n(0))$$

$$y_{i, n(0)+1}^0 = -1$$

Cada iteración del algoritmo consta de dos fases, una hacia adelante y otra hacia atrás.

En la fase hacia adelante se le presenta a la red un patrón y esta calcula la salida y el error correspondientes.

En esta primera fase se calcula sucesivamente para $l = 1, \dots, L$

$$net_{ij}^l = \sum_{k=1}^{n(l-1)} w_{jk}^l y_{ik}^{l-1}, \quad (i = 1, \dots, n(l-1) + 1; j = 1, \dots, n(l))$$

$y_{ij}^l = f(net_{ij}^l)$, donde f es la función de activación continua bipolar.

Al llegar a la capa de salida se determina el error asociado al patrón X_i como

$$E_i = \sum_{j=1}^{n(L)} (e_{ij})^2$$

donde $e_{ij} = d_{ij} - y_{ij}^L$

La función del error total de un ciclo de entrenamiento se puede expresar como el cuadrado de la norma euclidiana de la diferencia entre la salida deseada y la salida calculada por la red, es decir,

$$E = \sum_{i=1}^T E_i = \sum_{i=1}^T \sum_{j=1}^{n(L)} (e_{ij})^2$$

En la segunda fase el error es propagado hacia atrás, a través de la red y los pesos son modificados según la dirección negativa del gradiente de la función de error.

En un proceso de propagación hacia atrás, para $l = L, L-1, \dots, 1$, se calcula

$$\delta_{ij}^l = \begin{cases} e_{ij} f'(net_{ij}^L), & l = L \\ f'(net_{ij}^l) \sum_{k=1}^{n(l+1)} \delta_{ik}^{l+1} w_{kj}^{l+1}, & l = L-1, \dots, 1 \end{cases}$$

y los pesos se modifican según la regla delta generalizada

$$\Delta w_{kj}^l = c \delta_{ij}^l y_{ij}^{l-1}$$

La presentación completa del conjunto de entrenamiento es conocida como ciclo de entrenamiento y el algoritmo termina cuando al concluir un ciclo de entrenamiento, el error total del ciclo es menor que un error prefijado con antelación.

2.4.5 Algoritmos constructivos de aprendizaje.

Para el entrenamiento de redes multicapas, el algoritmo de aprendizaje utilizado tradicionalmente es el algoritmo de retropropagación del error. Este algoritmo

presenta la desventaja de que presupone conocida la arquitectura de la red, es decir, el número de capas y la cantidad de neuronas por capas.

Los algoritmos constructivos de aprendizaje [10,11,79,100,102] son procedimientos que se usan para diseñar y entrenar redes neuronales multicapas para la clasificación. Estos algoritmos obtienen redes neuronales de arquitectura sub-optimal en el sentido del número de neuronas en las capas ocultas.

En la mayoría de estos algoritmos, el entrenamiento está basado en alguna variante de la regla de aprendizaje del perceptrón discreto y su funcionamiento consiste en ir agregando neuronas a la red hasta que se logre que la igualdad a cero de la función de error para todos los patrones del conjunto de entrenamiento.

Es conocido que cuando el conjunto de patrones no es linealmente separable, es imposible realizar la clasificación correcta del conjunto de entrenamiento sin agregar capas de neuronas ocultas.

El principio general de funcionamiento de los algoritmos constructivos de aprendizaje es determinar, en cada iteración del algoritmo, un vector de pesos y un valor de umbral que proporcione el valor mínimo de la función de error, el cual es igual a cero si el conjunto de entrenamiento es linealmente separable.

Entre los algoritmos constructivos, los más eficientes son [10,11]:

- Algoritmo de bolsa con mecanismo de reten (*Pocket algorithm with ratchet modification*).
- Algoritmo del perceptrón térmico (*Thermal perceptron algorithm*).
- Procedimiento de corrección baricéntrica (*Barycentric correction procedure*).

Esta comprobado [90] que el problema de determinar el mayor subconjunto linealmente separable de un conjunto de entrenamiento es NP-completo. De aquí que todos estos algoritmos constructivos son procedimientos heurísticos que en cada iteración tratan de determinar, con una complejidad polinomial, el mayor subconjunto linealmente separable.

El algoritmo de bolsa con mecanismo de reten utiliza la regla del perceptrón para la modificación de los pesos y guarda en un vector W_{pocket} el vector de pesos que proporciona el menor valor de la función de error. En cada iteración se compara el

valor de la función de error para el vector de pesos W calculado con el valor para W_{pocket} y si este valor es menor, se reemplaza W_{pocket} por W . Está comprobado [26] que este algoritmo converge al menor valor de la función de error.

El algoritmo del perceptrón térmico es utilizado para controlar la modificación de los pesos durante el proceso de entrenamiento. En el algoritmo clásico de entrenamiento del perceptrón, cuando el conjunto de patrones no es linealmente separable, pueden ocurrir cambios bruscos en los pesos, que producen fluctuaciones severas en la función de error y entorpecen el proceso de clasificación.

Para estabilizar el proceso de aprendizaje se introduce el siguiente factor amortiguador en la ecuación de modificación de los pesos

$$W \leftarrow W + c [d_i - o_i] X_i \exp\left(-\frac{|W' \cdot X_i|}{Q}\right)$$

El valor de Q se le da un valor de Q_0 al comienzo del entrenamiento y gradualmente se aproxima a cero a medida que progresa el aprendizaje.

Este factor amortiguador introducido en la regla de modificación de los pesos no permite cambios bruscos en los pesos al final del entrenamiento.

El procedimiento de corrección baricéntrica es un algoritmo eficiente para entrenar una unidad lógica de umbral. En este procedimiento los patrones son separados en dos subconjuntos S^+ y S^- . El baricentro de cada subconjunto se define como la media ponderada de los patrones multiplicados por su correspondiente coeficiente de peso. El vector de pesos $W = (w_1, w_2, \dots, w_n)'$ es determinado como la diferencia entre los baricentros de los dos subconjunto de patrones y el valor de umbral w_{n+1} es seleccionado de forma tal que minimice la función de error. Inicialmente a cada patrón se le asocia un coeficiente de peso igual 1.

Si el conjunto de patrones es linealmente separable, este procedimiento determina, de forma mas eficiente que los dos algoritmos anteriores, el hiperplano que separa al conjunto de patrones en las dos clases o categorías.

2.5 Teorema de Aproximación Universal.

Después de encontrar un algoritmo de entrenamiento para redes multicapas, el problema fundamental que enfrentaron los investigadores en redes neuronales fue determinar la menor cantidad de capas de neuronas ocultas para que este algoritmo fuera convergente.

En 1989 se da solución a este problema mediante la demostración de un teorema que es conocido como el *Teorema de Aproximación Universal*. Este teorema está considerado como el resultado teórico de mayor importancia para redes neuronales de propagación hacia adelante y fue reportado en tres trabajos diferentes: Cybenko [15], Funahashi [25] y Hornik, Stinchcombe y White [33].

El teorema puede ser formulado como:

Teorema: Sea $\varphi(\cdot)$ una función continua, monótona creciente y acotada. Denotemos por \mathbf{I}_n el hipercubo unitario n-dimensional $[0, 1]^n$ y por $C(\mathbf{I}_n)$ el espacio de las funciones continuas sobre \mathbf{I}_n . Entonces, dados una función $f \in C(\mathbf{I}_n)$ y un $\varepsilon > 0$, existen un entero m y conjuntos de constantes reales α_i, θ_i y w_{ij} , donde $i = 1, \dots, m$; $j = 1, \dots, n$ tales que se puede definir

$$F(x_1, \dots, x_n) = \sum_{i=1}^m \alpha_i \varphi \left(\sum_{j=1}^n w_{ij} x_j - \theta_i \right)$$

como una aproximación de la función f , que cumple

$$|F(x_1, \dots, x_n) - f(x_1, \dots, x_n)| < \varepsilon \quad \text{para todo } (x_1, \dots, x_n) \in \mathbf{I}_n.$$

El teorema de aproximación universal es un teorema de existencia y establece que una sola capa oculta es suficiente para que una red neuronal multicapa calcule una aproximación uniforme para un conjunto de entrenamiento dado, representado por el conjunto de entradas (x_1, \dots, x_n) y una salida deseada $f(x_1, \dots, x_n)$.

La demostración de este teorema está fundamentada en el teorema de aproximación por superposición de Kolmogorov. En el entorno de las redes neuronales, estos dos resultados han sido ampliamente discutidos y analizados por

diferentes autores [34,36,37,41,42,43,45,56,57,58,74,75,85,96], además, han sido aplicados a redes neuronales con otros tipos de funciones de activación [20,31,78].

El teorema de aproximación universal no impone ninguna restricción sobre la cantidad de neuronas en la capa oculta ni sobre la magnitud de los valores de los pesos de la red. De aquí que deja abiertos los problemas de si se mantiene la propiedad de aproximación universal, si se imponen restricciones a la cantidad de neuronas en la capa oculta y/o a los pesos de la red.

2.6 Conclusiones.

En este capítulo se da una breve introducción a la teoría de las redes neuronales para la clasificación.

El objetivo de este capítulo es describir y explicar los conceptos, métodos y procedimientos que, a nuestro juicio, son necesarios para una mejor comprensión de este trabajo de tesis.

Comenzamos con la explicación de la teoría del funcionamiento del cerebro en que se fundamentan los modelos de redes neuronales artificiales y de como esta teoría es simplificada para conformar los modelos artificiales.

Luego tratamos el problema de clasificación de patrones y como este puede ser abordado mediante los modelos de redes neuronales de propagación hacia adelante.

Más adelante, tratamos el problema del aprendizaje de las redes neuronales de propagación hacia adelante, describiendo las diferentes reglas clásicas de aprendizaje, así como los algoritmos constructivos de aprendizaje de mayor eficiencia computacional, reportados en la literatura.

Al final del capítulo incluimos el teorema de aproximación universal, debido a su importancia dentro de la teoría de las redes neuronales de propagación hacia adelante, ya que garantiza la existencia de la solución del problema de entrenamiento.

CAPITULO 3

DISEÑO DE REDES NEURONALES DE PROPAGACION HACIA ADELANTE PARA LA CLASIFICACION

3.1 Introducción.

Uno de los problemas fundamentales en las aplicaciones de los modelos de redes neuronales multicapas de propagación hacia adelante es el problema del diseño de la red, es decir, la cantidad de neuronas en cada capa.

La cantidad de neuronas en las capas de entrada y de salida está determinada por la naturaleza del problema a resolver, por lo que el problema de diseño de la red se reduce a determinar cuántas neuronas deben tener las capas ocultas.

El teorema de aproximación universal [15,25,33] garantiza que una sola capa oculta es suficiente para que los modelos de redes neuronales de propagación hacia adelante puedan aproximar uniformemente cualquier función continua soportada sobre un hipercubo unitario, pero deja abierto el problema de la cantidad de neuronas necesarias en esta capa oculta para que se alcance esta propiedad.

Este teorema, también es aplicable a redes neuronales con funciones discretas de activación en las neuronas, que se utilizan para la clasificación y/o reconocimiento de patrones [7,27,105].

En otros trabajos posteriores [27,105] aplicados al problema de clasificación de patrones, se demuestra que con $(T - 1)$ neuronas en la capa oculta, donde T es la cardinalidad del conjunto de entrenamiento, la superficie de error no presenta mínimos locales, por lo que el algoritmo de aprendizaje es convergente. Pero en la práctica este valor $(T - 1)$ resulta demasiado grande y además, en la resolución de diferentes problemas prácticos, se ha comprobado que el algoritmo de entrenamiento es convergente con menor número de neuronas en la capa oculta.

Los métodos más usuales, que aparecen en la literatura para determinar la cantidad de neuronas en la capa oculta, son los llamados procedimientos de depuración y crecimiento [29,44], que operan por ensayo y error. Esto conlleva a que el proceso de entrenamiento de la red sea muy costoso computacionalmente.

En los procedimientos de depuración se construye una red de gran tamaño y se eliminan neuronas mientras se mantenga la convergencia y en los de crecimiento se construye una red pequeña y se agregan neuronas hasta que se alcance la convergencia del algoritmo de entrenamiento. Además, estos métodos utilizan como

algoritmo de entrenamiento al algoritmo de retropropagación del error, el cual es criticado en la literatura por su alto costo computacional.

Otra forma de abordar el problema en redes neuronales para la clasificación, ha sido mediante algoritmos constructivos [10,11,12,23,71,80,89,103]. Estos algoritmos tratan de encontrar el mayor subconjunto linealmente separable de un conjunto de entrenamiento dado y su funcionamiento se basa en diferentes modificaciones del algoritmo de aprendizaje del perceptrón discreto.

En este capítulo se trata el problema de diseño de una red neuronal para la clasificación. Este problema se puede enunciar como: *Determinar el número de neuronas en la capa oculta para que una red neuronal, con funciones discretas de activación en las neuronas, clasifique correctamente a un conjunto de patrones de entrenamiento.*

3.2 Análisis del problema de clasificación.

Analicemos el problema de clasificación de un conjunto de patrones en dos clases o categorías C_1 y C_2 .

Denotemos por $\{(X_1, d_1), (X_2, d_2), \dots, (X_T, d_T)\}$ el conjunto de aprendizaje, donde $X_i = (x_{i1}, x_{i2}, \dots, x_{in})^t \in R^n$, ($i = 1, \dots, T$) representan los patrones de entrenamiento y d_i es la salida deseada correspondiente al patrón X_i , siendo además

$$d_i = \begin{cases} 1, & \text{si } X_i \in C_1, \\ -1, & \text{si } X_i \in C_2 \end{cases}$$

Cuando un conjunto de patrones es linealmente separable, entonces existen un vector de pesos $W = (w_1, w_2, \dots, w_n)^t \in R^n$ y un valor de umbral $w_{n+1} \in R$, tales que para cada patrón de entrenamiento X_i se cumple que

$$\begin{cases} \sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1} > 0, & \text{si } X_i \in C_1 \\ \sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1} < 0, & \text{si } X_i \in C_2 \end{cases} \quad (3.1)$$

Esto significa que existe un hiperplano de la forma $X^t \cdot W - w_{n+1} = 0$ que separa el conjunto de patrones en las dos clases C_1 y C_2 .

Si se multiplica cada una de las desigualdades en (3.1) por la salida deseada del patrón correspondiente, obtenemos una condición equivalente

$$d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1} \right) > 0, \quad i = 1, \dots, T \quad (3.2)$$

Cada una de estas desigualdades en (3.2) representa un semiespacio abierto en el espacio de pesos y si el conjunto de patrones es linealmente separable, este sistema de T desigualdades representa el interior de un cono poliédrico, que es a su vez, el conjunto de soluciones factibles de este sistema de desigualdades.

En el caso en que el conjunto de patrones no sea linealmente separable, el conjunto de soluciones factibles de este sistema de desigualdades es vacío, lo que

significa que para cualesquiera vector de pesos W y valor de umbral w_{n+1} , siempre existe un grupo de desigualdades en (3.2) que se incumplen.

En este caso, se determinará un hiperplano $X'.W - w_{n+1} = 0$ que separe correctamente la mayor cantidad de patrones, por lo que es necesario determinar un vector W y un umbral w_{n+1} para los cuales se cumplan la mayor cantidad de desigualdades en (3.2).

Este hiperplano divide el conjunto de patrones en dos subconjuntos, los cuales pueden contener patrones de las dos clases, por lo que para cada uno de estos subconjuntos se buscará un hiperplano que dé la mayor cantidad de patrones bien clasificados y se procederá así sucesivamente hasta que sean separados todos los patrones.

En el principio de determinar el hiperplano que separe la mayor cantidad de patrones se basan todos los procedimientos de diseño de redes para la clasificación que aparecen reportados en la literatura. Estos procedimientos tienen la dificultad de no separar ningún patrón, cuando la menor cantidad de patrones mal clasificados coincide con todos los patrones de una misma clase.

Para eliminar esa dificultad, desde el principio, buscará un hiperplano que clasificando correctamente todos los patrones de una misma clase, garantice la mayor cantidad de patrones bien clasificados de la otra clase.

Para determinar este hiperplano, en el sistema de desigualdades (3.2) se introducen T variables auxiliares z_1, z_2, \dots, z_T , de la siguiente forma

$$d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1} \right) - z_i = 0, \quad (i = 1, \dots, T) \quad (3.3)$$

Si el conjunto de patrones es linealmente separable, entonces existen un vector de pesos $W = (w_1, w_2, \dots, w_n)'$ y un valor de umbral w_{n+1} , para los cuales el vector de las variables auxiliares $Z = (z_1, z_2, \dots, z_T)$ tiene todas sus componentes positivas.

En caso en que el conjunto de patrones no sea linealmente separable, en el sistema (3.2) existen desigualdades que se incumplen y por tanto, algunas de las variables auxiliares toman obligatoriamente valores menores o iguales que cero.

Con esta forma de introducir las variables auxiliares se logra una doble correspondencia entre patrón bien clasificado y variable auxiliar positiva.

Ahora, el problema se ha transformado en determinar las soluciones $(w_1, w_2, \dots, w_n, w_{n+1}, z_1, z_2, \dots, z_T)$ del sistema de ecuaciones (3.3) que contengan la mayor cantidad de valores positivos correspondientes a las variables auxiliares z_1, z_2, \dots, z_T y que entre las positivas, estén todas las que pertenecen a una misma clase.

El sistema de ecuaciones (3.3) puede ser escrito como

$$d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1} \right) = z_i, \quad (i = 1, \dots, T), \quad (3.4)$$

o en forma matricial,

$$\begin{bmatrix} d_1 x_{11} & d_1 x_{12} & \dots & d_1 x_{1n} & -d_1 \\ d_2 x_{21} & d_2 x_{22} & \dots & d_2 x_{2n} & -d_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ d_T x_{T1} & d_T x_{T2} & \dots & d_T x_{Tn} & -d_T \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_T \end{bmatrix},$$

teniendo por matriz asociada, la matriz

$$\begin{bmatrix} d_1 x_{11} & d_1 x_{12} & \dots & d_1 x_{1n} & -d_1 & 1 & 0 & \dots & 0 \\ d_2 x_{21} & d_2 x_{22} & \dots & d_2 x_{2n} & -d_2 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ d_T x_{T1} & d_T x_{T2} & \dots & d_T x_{Tn} & -d_T & 0 & 0 & \dots & 1 \end{bmatrix}$$

Supóngase que el conjunto de entrenamiento contiene T_1 patrones de la clase 1 y $T - T_1$ patrones de la clase 2, además que está ordenado de forma tal que primero se encuentran los T_1 patrones de la clase 1.

Entonces para determinar el hiperplano separador hay que resolver uno de los siguientes problemas:

$$\left\{ \begin{array}{l} \max \sum_{j=T_1+1}^T \text{sgn}(z_j) \\ \text{s. a } d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1} \right) = z_i, \quad i = 1, \dots, T \\ z_j > 0, \quad j = 1, \dots, T_1 \end{array} \right. \quad (\text{P3.1})$$

$$\left\{ \begin{array}{l} \max \sum_{j=1}^{T_1} \text{sgn}(z_j) \\ \text{s. a } d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1} \right) = z_i, \quad i = 1, \dots, T \\ z_j > 0, \quad j = T_1 + 1, \dots, T \end{array} \right. \quad (\text{P3.2})$$

Después de obtener un hiperplano separador, el conjunto de patrones se divide en dos subconjuntos. Si el conjunto de patrones no es linealmente separable, entonces uno de estos subconjuntos contiene patrones de las dos clases, por lo que a este subconjunto se le aplica de nuevo el procedimiento descrito. Este proceso continúa, hasta que se encuentre un subconjunto linealmente separable.

Cada uno de estos hiperplanos separadores representa una neurona de la capa oculta de la red y las componentes del vector W y el valor de w_{n+1} , asociados a cada hiperplano, representan los pesos y el umbral respectivamente, entre la capa de entrada y cada neurona de la capa oculta de la red.

3.3 Cota superior para el número de neuronas en la capa oculta.

El procedimiento de separación de patrones que se propone, se basa en dividir el conjunto de patrones en dos subconjuntos de forma tal que: la intersección de sus envolturas convexas sea vacía, uno de estos subconjuntos contenga patrones de una sola clase y la cardinalidad del subconjunto que contiene patrones de una sola clase sea máxima.

Esto permite determinar un hiperplano separador que divide al conjunto de patrones en dos subconjuntos, donde en uno de ellos solo hay patrones de una misma clase y además la cardinalidad de este subconjunto es máxima.

En el proceso de separación de los patrones se obtiene un árbol como el que se muestra en la figura 18.

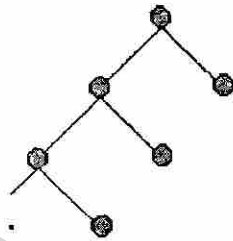


Figura 18: Árbol asociado al proceso de división.

El nodo raíz de este árbol representa al conjunto de patrones de entrenamiento y los restantes nodos representan los subconjuntos en que se va dividiendo este conjunto de patrones. Los nodos hojas del árbol representan los subconjuntos en que se divide al final el conjunto original de patrones y la cantidad de estos nodos hojas, disminuida en una unidad, representa la cantidad de hiperplanos que se utilizaron para separar el conjunto de patrones.

En el peor de los casos, en cada división se logra separar un solo patrón y el último subconjunto a dividir estará formado por dos patrones de diferentes clases, siendo en este caso la cantidad de nodos hoja será igual a T y la cantidad de hiperplanos igual a $T - 1$. Este último valor coincide con el valor conocido de la cota superior del número de neuronas en la capa oculta para que la superficie de error no presente mínimos locales.

3.4 Algoritmo de solución.

3.4.1 Transformación de los problemas planteados.

Para facilitar la resolución de los problemas (P3.1) y (P3.2), se realiza un intercambio entre variables libres y variables dependientes del sistema, mediante un proceso de eliminación gaussiana, de forma tal que queden como variables dependientes las componentes del vector de pesos y el umbral.

Nótese que el rango del sistema de ecuaciones homogéneas (3.3) es igual a T por lo que siempre tiene soluciones diferente de la solución trivial, además el sistema fundamental de soluciones consta de $n+1$ soluciones. Esto significa que hay $n+1$ variables libres y T variables dependientes.

Si el rango de la matriz

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} & -1 \\ x_{21} & x_{22} & \cdots & x_{2n} & -1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{T1} & x_{T2} & \cdots & x_{Tn} & -1 \end{bmatrix},$$

formada con los patrones ampliados, es igual a $n+1$, entonces se pueden intercambiar las variables que representan los pesos y el umbral con $n+1$ de las variables auxiliares, obteniendo una matriz de la siguiente estructura

$$\begin{bmatrix} 10\cdots0 & a_{11} & a_{12} & \cdots & a_{1,n+1} & 0\cdots0 \\ 01\cdots0 & a_{21} & a_{22} & \cdots & a_{2,n+1} & 0\cdots0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 00\cdots1 & a_{n+1,1} & a_{n+1,2} & \cdots & a_{n+1,n+1} & 0\cdots0 \\ 00\cdots0 & a_{n+2,1} & a_{n+2,2} & \cdots & a_{n+2,n+1} & 1\cdots0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 00\cdots0 & a_{T,1} & a_{T,2} & \cdots & a_{T,n+1} & 0\cdots1 \end{bmatrix}$$

Si $\text{rank}(X) = n_1 < n+1$, entonces al realizar el proceso de intercambio entre variables libres y variables dependientes, solo n_1 de las componentes del vector $\bar{W} = (w_1, w_2, \dots, w_n, w_{n+1})'$ pasan a ser variables dependientes.

En este caso la matriz del sistema (3.4) se transforma a una matriz, donde entre las columnas asociadas a las variables $w_1, w_2, \dots, w_n, w_{n+1}$, solo existen n_1 columnas formadas por vectores canónicos y las $n+1-n_1$ columnas restantes contienen, en general, n_1 componentes distintas de cero.

Por ejemplo, se obtiene una matriz de la siguiente estructura, aunque el orden de las columnas no es necesariamente el que se presenta.

$$\begin{bmatrix} 1 & 0 \cdots 0 & \tilde{x}_{1,n_1+1} & \cdots & \tilde{x}_{1,n+1} & a_{11} & a_{12} & \cdots & a_{1n_1} & 0 \cdots 0 \\ 0 & 1 \cdots 0 & \tilde{x}_{2,n_1+1} & \cdots & \tilde{x}_{2,n+1} & a_{21} & a_{22} & \cdots & a_{2n_1} & 0 \cdots 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 \cdots 1 & \tilde{x}_{n_1,n_1+1} & \cdots & \tilde{x}_{n_1,n+1} & a_{n_1,1} & a_{n_1,2} & \cdots & a_{n_1,n_1} & 0 \cdots 0 \\ 0 & 0 \cdots 0 & 0 & \cdots & 0 & a_{n_1+1,1} & a_{n_1+1,2} & \cdots & a_{n_1+1,n_1} & 1 \cdots 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 \cdots 0 & 0 & \cdots & 0 & a_{T1} & a_{T2} & \cdots & a_{Tn_1} & 0 \cdots 1 \end{bmatrix}$$

Como el objetivo es determinar un vector $\bar{W} = (w_1, w_2, \dots, w_n, w_{n+1})' \neq \bar{0}$, que le correspondan valores no nulos de las variables auxiliares, entonces se les pueden asignar valores iguales a cero a todas las componentes del vector \bar{W} que quedaron como variables libres. Esto significa que existen $n+1-n_1$ componentes en los patrones ampliados que no tienen influencia sobre la determinación del hiperplano separador y por tanto en este caso se puede reducir a n_1 la dimensión de los patrones ampliados.

De todo esto se puede concluir que cuando $\text{rank}(X) = n_1 < n+1$, el vector \bar{W} tiene $n+1-n_1$ componentes iguales a cero y para tener uniformidad en el desarrollo del procedimiento, se realizará la asignación $n \leftarrow n_1 - 1$.

Supóngase ahora, por comodidad de notación, que las primeras $n+1$ variables auxiliares z_1, z_2, \dots, z_{n+1} son ahora las variables libres, entonces el sistema (3.4) se transforma en

$$\begin{cases} w_i = \sum_{j=1}^{n+1} a_{ij} z_j, & i = 1, \dots, n+1 \\ \sum_{j=1}^{n+1} (a_{ij} z_j) + z_i = 0, & i = n+2, \dots, T \end{cases} \quad (3.5)$$

La primera expresión en (3.5) proporciona las dependencias de los pesos y el umbral de las nuevas variables libres z_1, z_2, \dots, z_{n+1} . La segunda expresión en (3.5) proporcionan las relaciones de dependencia lineal entre las variables libres z_1, z_2, \dots, z_{n+1} y las variables dependientes $z_{n+2}, z_{n+3}, \dots, z_T$.

De aquí al asignarle valores a las variables libres, de la primera expresión de (3.5) se pueden determinar los valores de los pesos y el umbral entre la capa de entrada y una neurona de la capa oculta, es decir, la ecuación de un hiperplano separador.

Como el problema radica en determinar entre las soluciones del sistema (3.3), una que tenga la mayor cantidad de componentes positivos para las variables auxiliares, se trabajará con las últimas $T - (n+1)$ filas de la matriz asociada al sistema (3.5), es decir, con el sistema

$$\sum_{j=1}^{n+1} (a_{ij} z_j) + z_i = 0, \quad i = n+2, \dots, T \quad (3.6)$$

La matriz de este sistema de ecuaciones tiene la forma

$$\begin{bmatrix} a_{n+2,1} & a_{n+2,2} & \dots & a_{n+2,n+1} & 1 & 0 & \dots & 0 \\ a_{n+3,1} & a_{n+3,2} & \dots & a_{n+3,n+1} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{T,1} & a_{T,2} & \dots & a_{T,n+1} & 0 & 0 & \dots & 1 \end{bmatrix} \quad (3.7)$$

y se denominará matriz de dependencia lineal.

Entonces los problemas (P3.1) y (P3.2) se transforman respectivamente en

$$\begin{cases} \max \sum_{i=n+2}^T \text{sgn}(z_i) \\ \text{s.a.} \sum_{j=1}^{n+1} (a_{ij} z_j) + z_i = 0, & i = n+2, \dots, T \\ z_i > 0, & i = 1, \dots, T_1 \end{cases} \quad (P3.3)$$

y

$$\left\{ \begin{array}{l} \max \sum_{i=1}^{T_1} \text{sgn}(z_i) \\ \text{s.a.} \sum_{j=1}^{n+1} (a_{ij}z_j) + z_i = 0, \quad i = n+2, \dots, T \\ z_i > 0, \quad i = T_1 + 1, \dots, T \end{array} \right. \quad (\text{P3.4})$$

El principio de solución de estos problemas será: realizar movimientos en el espacio de las variables auxiliares de forma tal que manteniendo con valores positivos los valores de las variables auxiliares correspondientes a una clase podamos aumentar el número de variables auxiliares positivas de la otra clase.

Para lograr que todas las variables auxiliares correspondientes a una misma clase sean positivas, se realizará un intercambio entre variables libres y variables dependientes mediante transformaciones elementales en la matriz (3.7).

El objetivo de este intercambio es obtener una columna, asociada a una variable libre, que tenga la mayor cantidad de elementos negativos, ya que al asignarle valores positivos a todas las variables libres, excepto a la variable libre correspondiente a esta columna, entonces existe un número positivo q , tal que si se le asigna a esta variable valores mayores que q , todas las variables dependientes con coeficiente negativo en esta columna toman valores positivos.

Para tener un control sobre las variables libres y variables dependientes, se define un vector de índices $\mathbf{U} = (u(1), u(2), \dots, u(T))$, donde las primeras $n+1$ componentes están asociadas a los subíndices de las variables libres y las restantes a los subíndices de las variables dependientes, además la posición de cada componente asociada a una variable dependiente, determina la fila que le corresponde a esta variable en la matriz de dependencia lineal según la siguiente regla:

$$\left\{ \begin{array}{l} z_{u(n+2)} \rightarrow \text{fila 1} \\ z_{u(n+3)} \rightarrow \text{fila 2} \\ z_{u(T)} \rightarrow \text{fila } T - (n+1) \end{array} \right\}$$

Supóngase que se ha seleccionado la columna $u(s)$, asociada a una variable

libre $z_{u(s)}$, ($1 \leq s \leq n+1$), para aumentar en ella la cantidad de elementos negativos.

Para realizar las transformaciones en la matriz de dependencia lineal se buscará, entre las otras columnas asociadas a las restantes variables libres, elementos pivotes que aumenten la cantidad de elementos negativos en la columna $u(s)$.

3.4.2 Análisis para la selección del elemento pivote.

Supóngase que el elemento pivote lo se está buscando en una columna $u(r)$, ($r \neq s$) y se denotará por $a_{k,u(r)}$.

Para la selección de los pivotes, es necesario analizar los cuatro casos siguientes :

1. $a_{k,u(s)} < 0$ y $a_{k,u(r)} > 0$
2. $a_{k,u(s)} > 0$ y $a_{k,u(r)} < 0$
3. $a_{k,u(s)} < 0$ y $a_{k,u(r)} < 0$
4. $a_{k,u(s)} > 0$ y $a_{k,u(r)} > 0$

Caso 1: $a_{k,u(s)} < 0$ y $a_{k,u(r)} > 0$

Se mantienen con valores negativos :

- El nuevo valor $\tilde{a}_{k,u(s)}$.
- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} < 0$ y $a_{i,u(r)} < 0$.
- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} < 0$, $a_{i,u(r)} > 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| > \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|$$

Se transforman a negativos :

- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} \geq 0$, $a_{i,u(r)} < 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| < \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|.$$

Se mantienen positivos :

- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} \geq 0$ y $a_{i,u(r)} > 0$
- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} > 0$, $a_{i,u(r)} < 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| \geq \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|$$

Se transforman a positivos :

- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} < 0$, $a_{i,u(r)} > 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| \leq \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|$$

Caso 2: $a_{k,u(s)} > 0$ y $a_{k,u(r)} < 0$

Se mantienen con valores negativos :

- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} < 0$ y $a_{i,u(r)} < 0$.
- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} < 0$, $a_{i,u(r)} > 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| > \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|$$

Se transforman a negativos :

- El nuevo valor $\tilde{a}_{k,u(s)}$.
- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} \geq 0$, $a_{i,u(r)} < 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| < \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|$$

Se mantienen positivos :

- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} \geq 0$ y $a_{i,u(r)} > 0$
- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} > 0$, $a_{i,u(r)} < 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| \geq \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|$$

Se transforman a positivos :

- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} < 0$, $a_{i,u(r)} > 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| \leq \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|$$

Caso 3: $a_{k,u(s)} < 0$ y $a_{k,u(r)} < 0$

Se mantienen con valores negativos :

- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} < 0$ y $a_{i,u(r)} \geq 0$.
- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} < 0$, $a_{i,u(r)} < 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| > \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|$$

Se transforman a negativos :

- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} \geq 0$, $a_{i,u(r)} > 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| < \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|$$

Se mantienen positivos :

- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} \geq 0$ y $a_{i,u(r)} < 0$
- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} > 0$, $a_{i,u(r)} > 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| > \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|$$

Se transforman a positivos :

- El nuevo valor $\tilde{a}_{k,u(s)}$.
- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} < 0$, $a_{i,u(r)} < 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| \leq \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|$$

Caso 4: $a_{k,u(s)} > 0$ y $a_{k,u(r)} > 0$

Se mantienen con valores negativos :

- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} < 0$ y $a_{i,u(r)} \geq 0$.

- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} < 0$, $a_{i,u(r)} < 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| > \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|$$

Se transforman a negativos :

- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} \geq 0$, $a_{i,u(r)} > 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| < \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|.$$

Se mantienen positivos :

- El nuevo valor $\tilde{a}_{k,u(s)}$.
- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} \geq 0$ y $a_{i,u(r)} < 0$
- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} > 0$, $a_{i,u(r)} > 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| > \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|.$$

Se transforman a positivos :

- Todos los nuevos valores $\tilde{a}_{i,u(s)}$ que cumplan $a_{i,u(s)} < 0$, $a_{i,u(r)} < 0$ y

$$\left| \frac{a_{iu(s)}}{a_{iu(r)}} \right| \leq \left| \frac{a_{ku(s)}}{a_{ku(r)}} \right|$$

Como se puede apreciar, cuando se utilizan pivotes de los casos 1 y 2, en la columna $u(s)$ mantienen su signo todos aquellos elementos que tenían el signo igual al de su compañero en la columna $u(r)$, por lo que solamente interesan las variaciones que sufran los elementos de la columna $u(s)$ que tienen signo contrario al de su compañero en la columna del elemento pivote.

Para los casos 3 y 4, ocurre lo contrario, es decir, se mantienen del mismo tipo en la columna $u(s)$, todos aquellos elementos que tenían el signo contrario al de su compañero en la columna $u(r)$. Aquí interesan los cambios que sufran los elementos en la columna $u(s)$, que tienen signo contrario al de su compañero en la columna $u(r)$.

3.4.3 Transformación de la matriz de dependencia lineal.

Analicemos como resolver el problema (P3.3). El objetivo es transformar la matriz (3.7) de forma tal que se obtenga una columna, asociada a una variable libre correspondiente a la clase 1, que tenga la menor cantidad posible de coeficientes no negativos y además, que todos estos coeficientes estén en las filas de las variables dependientes correspondientes a patrones de la clase 2.

Primero, se selecciona una columna $u(s)$ de la matriz de dependencia lineal, asociada a una variable libre de la clase 1, que tenga la mayor cantidad de elementos negativos en las filas correspondientes a variables dependientes de la clase 1.

Si en la columna $u(s)$ existen elementos, asociados a las variables dependientes de la clase 1, con valores mayores o iguales a cero, significa que hay patrones mal clasificados de esta clase y por tanto, primero se debe transformar a negativos esos elementos, para luego tratar de aumentar la cantidad de patrones bien clasificados de la clase 2.

De aquí que el procedimiento se divide en dos fases. En la primera fase se convierten a negativos todos los elementos de la columna $u(s)$, asociados a variables dependientes de la clase 1 y en la segunda fase, manteniendo negativos estos elementos, se trata de aumentar la cantidad de negativos asociados a variables dependientes de la clase 2.

Después de haber seleccionado la columna $u(s)$, se determinan las cantidades de componentes mayores o iguales a cero correspondientes a las variables dependientes de ambas clases. Estos valores los denotamos por k_1 y k_2 respectivamente.

Nótese que si k_1 y k_2 son iguales a cero, el conjunto de patrones es linealmente separable.

Si $k_1 > 0$ se realizan intercambios entre las variables libres y las dependientes de forma tal que este valor se convierta en cero.

Analicemos primero el caso, cuando buscamos elementos pivotes de los tipos 1 y 2 para convertir k_1 en cero. Este algoritmo está desarrollado en el procedimiento

3.1.1 del apéndice A.

Como se vio anteriormente, aquí solo es necesario enfocarse hacia aquellos elementos que tengan el signo contrario al de su compañero en la columna $u(s)$.

Para facilitar este proceso, se determina un vector \mathbf{B} con componentes

$$b_i = \begin{cases} 1, & \text{si } a_{i+n+1,u(r)} > 0 \text{ y } a_{i+n+1,u(s)} < 0 \\ -1, & \text{si } a_{i+n+1,u(r)} < 0 \text{ y } a_{i+n+1,u(s)} \geq 0 \\ 0, & \text{en caso contrario} \end{cases},$$

$$(i = 1, \dots, T - n - 1)$$

y para todo $b_i \neq 0$, se calcula $c_i = \left| \frac{a_{i+n+1,u(s)}}{a_{i+n+1,u(r)}} \right|$.

Para que un pivote seleccionado aumente la cantidad de elementos negativos en la columna $u(s)$, asociados a variables de la clase 1, debe cumplirse que :

$$\bullet c_k < c_i, \forall i : b_i = 1, u(i+n+1) \leq T_1 \quad (3.8)$$

$$\bullet \exists j : c_k > c_j, b_j = -1, u(j+n+1) \leq T_1 \quad (3.9)$$

Además, en el caso de pivotes del tipo 1, puede ser seleccionado un pivote asociado a una variable de la clase 1, si el valor de c_k es un mínimo único entre los valores de c_i de las restantes variables de esta clase, para las cuales $b_i = 1$.

En el caso de pivotes del tipo 2, además, hay que analizar el caso, cuando se selecciona un pivote asociado a una variable de la clase 1, para el cual se cumple solamente la condición (3.8), ya que en este caso su compañero en la columna $u(s)$ se transforma en negativo.

De aquí se puede concluir que :

1. Los pivotes del tipo 1 pueden ser :

- a) Un elemento $a_{k,u(r)}$ correspondiente a una variable dependiente de la clase 1, para el cual, el valor de c_k es un mínimo único entre los restante valores de c_i de las otras variables de esta clase, que tienen asociado un valor de $b_i = 1$.
- b) Un elemento $a_{k,u(r)}$ correspondiente a una variable dependiente de la clase 2, para el cual, el valor de c_k es menor que todos los valores de c_i para las variables de la clase 1, que tienen asociado un valor de $b_i = 1$.

2. Los pivotes del tipo 2 pueden ser :

- Un elemento $a_{k,u(r)}$ correspondiente a una variable dependiente de la clase 1, que cumpla solamente con la condición (3.8).
- Un elemento $a_{k,u(r)}$ que cumpla con las dos condiciones.

Analicemos ahora, el caso cuando se buscan elementos pivotes de los casos 3 y 4. Aquí solo es necesario enfocarse hacia aquellos elementos que tengan el mismo signo que su compañero en la columna $u(s)$. El algoritmo para este caso se encuentra desarrollado en el procedimiento 3.1.2 del apéndice A.

En estos casos, se determina un vector **B** con componentes

$$b_i = \begin{cases} 1, & \text{si } a_{i+n+1,u(r)} > 0 \text{ y } a_{i+n+1,u(s)} \geq 0 \\ -1, & \text{si } a_{i+n+1,u(r)} < 0 \text{ y } a_{i+n+1,u(s)} < 0 \\ 0, & \text{en caso contrario} \end{cases},$$

$$(i = 1, \dots, T - n - 1)$$

y para todo $b_i \neq 0$, se calcula $c_i = \left| \frac{a_{i+n+1,u(s)}}{a_{i+n+1,u(r)}} \right|$.

Para que el pivote seleccionado aumente la cantidad de elementos negativos en la columna $u(s)$, asociados a variables de la clase 1, debe cumplirse que :

$$\bullet \quad c_k < c_i, \quad \forall i : b_i = -1, \quad u(i+n+1) \leq T_1 \quad (3.10)$$

$$\bullet \quad \exists j : c_k > c_j, \quad b_j = 1, \quad u(j+n+1) \leq T_1 \quad (3.11)$$

Nótese además, que no puede ser seleccionado un pivote asociado a una variable de la clase 1 para el caso de pivotes del tipo 3.

Entre los elementos candidatos a pivote se seleccionará aquel, para el cual se obtenga el mayor valor de l_1 , donde l_1 es la cantidad de elementos de la columna $u(s)$, asociados a variables de la clase 1, que se transforman en negativos.

Cuando ya se ha seleccionado el elemento pivote $a_{k,u(r)}$, para realizar el intercambio entre una variable libre y una variable dependiente en la matriz de dependencia lineal, se divide la fila k por $a_{k,u(r)}$ y se convierten en cero todos los restantes elementos de la columna $u(r)$. Además, se realiza un intercambio entre las componentes $u(r)$ y $u(k+n+1)$ del vector **U** y se actualiza el valor de $k_1 \leftarrow k_1 - l_1$.

Para el proceso de intercambiar variables se utiliza el procedimiento 3.3 del apéndice A.

Todo este proceso se repite hasta que $k_1 = 0$ y está desarrollado en el procedimiento 3.4.1 del apéndice A.

Cuando $k_1 = 0$, se pasa a verificar si se puede aumentar la cantidad de elementos negativos asociados a variables dependientes de la clase 2.

Aquí, el objetivo es aumentar la cantidad de elementos negativos asociados a variables de la clase 2, teniendo como restricción que deben mantenerse negativos, los elementos asociados a variables de la clase 1.

Esto lleva a que es necesario realizar algunas modificaciones a los procedimientos de búsqueda de los elementos pivotes.

El principio de funcionamiento de estos procedimientos será buscar pivotes que mantengan bien clasificadas a las variables de la clase 1 y para las variables de la clase 2, sea positiva la diferencia entre la cantidad de elementos que se transforman a negativos y la cantidad de elementos que se transforman a positivos.

Entre los elementos candidatos a pivote se seleccionará aquel, para el cual se obtenga el mayor valor de la diferencia $l = l_1 - l_2$ para variables asociadas a la clase 2, donde l_1 es la cantidad de elementos de $u(s)$ que se transforman a negativos y l_2 es la cantidad de elementos de $u(s)$ que se transforman a positivos.

En los pivotes de los casos 1 y 2, para garantizar que se mantengan negativos los elementos asociados a variables de la clase 1, primero se determina $c_{m1} = \min\{c_i\}$ entre los valores de i que cumplen $u(i+n+1) \leq T_1$, $b_i = 1$.

En estos casos los candidatos a pivote serán :

1. $a_{m1+n+1, u(r)}$, si c_{m1} es un mínimo único.
2. $a_{i+n+1, u(r)}$, si $c_i < c_{m1}$.

Nótese que, como la condición 2 solo se cumple para elementos asociados a variables de la clase 2 y que tengan el signo contrario al de su compañero en la columna $u(s)$, se reduce considerablemente el espacio de búsqueda. El algoritmo para la selección del elemento pivote está desarrollado en el procedimiento 3.5.1 del

apéndice A.

En la selección del pivote de los casos 3 y 4, para garantizar que se mantengan negativos los elementos asociados a variables de la clase 1, primero se determina $c_{m1} = \min\{c_i\}$ entre los valores de i que cumplen $u(i+n+1) \leq T_1$, $b_i = -1$.

En estos casos los candidatos a pivote serán todos aquellos elementos $a_{i+n+1,u(r)}$ para los cuales se cumple que $c_i < c_{m1}$.

Como esta condición solo se cumple para elementos asociados a variables de la clase 2 y que tengan el signo igual al de su compañero en la columna $u(s)$, se reduce considerablemente el espacio de búsqueda. El algoritmo para la selección del elemento pivote está desarrollado en el procedimiento 3.5.2 del apéndice A.

Cuando ya se ha seleccionado el elemento pivote $a_{k,u(r)}$, se realizan los intercambios entre una variable libre $z_{u(r)}$ y una variable dependiente $z_{u(k+n+1)}$ en la matriz de dependencia lineal y entre las componentes $u(r)$ y $u(k+n+1)$ del vector U . Además, se actualiza el valor de $k_2 \leftarrow k_2 - 1$.

El proceso de disminución de k_2 termina cuando ya no existen pivotes que mejoren la solución o cuando $k_2 = 0$. El algoritmo para disminuir el valor de k_2 está desarrollado en el procedimiento 3.7.1 del apéndice A.

Si como resultado de estas transformaciones, se encuentra una columna que tenga todos sus coeficientes negativos, es decir, $k_1 = 0$ y $k_2 = 0$, entonces el conjunto de patrones es linealmente separable y no tiene sentido tratar de resolver el problema (P3.4).

Si $k_2 > 0$, se pasa a resolver el problema (P3.4). En este caso se sigue el mismo procedimiento, con la diferencia de que la columna $u(s)$, que es seleccionada para una variable de la clase 2 y las restricciones de que sean todos negativos se les imponen a los coeficientes de la columna $u(s)$, asociados a las variables dependientes de esta clase.

Los algoritmos para transformar las matrices de dependencia lineal de los problemas (P3.3) y (P3.4) están desarrollados en los procedimientos 3.8.1 y 3.8.2 respectivamente, que se encuentran en el apéndice A.

Entre las soluciones de los problemas (P3.3) y (P3.4), se selecciona la que separe mayor cantidad de patrones.

En las dos matrices finales, los valores de k_2 representan la cantidad de patrones mal clasificados. Para determinar, cuantos patrones de la clase 2 se lograron separar mediante la solución de (P3.3) y cuantos de la clase 1 se lograron separar mediante la solución de (P3.4), se calculan $p_1 = T - T_1 - k_2$ y $p_2 = T_1 - k_2$ respectivamente. De estos dos valores se determina $\max\{p_1, p_2\}$ y se selecciona la matriz de dependencia lineal del problema para el cual se obtuvo el valor máximo.

3.4.4 Obtención de los valores de las variables auxiliares, pesos y umbral.

Ahora es necesario asignarle valores a las variables auxiliares, para luego determinar los valores de los pesos y el umbral, asociados al hiperplano separador.

Aquí, se seguirá el siguiente procedimiento para la asignación de valores a las variables auxiliares.

Primeramente, se calcularán los valores de las variables dependientes mediante las relaciones de dependencia lineal

$$z_{u(i)} = -\sum_{j=1}^{n+1} \tilde{a}_{i,u(j)} z_{u(j)}, \quad (i = n+2, \dots, T),$$

asignando a las variables libres $z_{u(s)} = 0$, $z_{u(j)} = 1$, ($j = 1, \dots, n+1; j \neq s$), luego

para todas las variables dependientes con valores negativos y que tienen asociado un

coeficiente negativo en la columna $u(s)$, se determina $q = \max_i \left\{ \frac{z_{u(i)}}{a_{i,u(s)}} \right\}$ y se le

asigna a

$$z_{u(s)} = \begin{cases} \lceil q \rceil, & \text{si } q \notin \mathbb{N} \\ \lceil q \rceil + 1, & \text{si } q \in \mathbb{N} \end{cases}$$

Finalmente, se le asignan a las variables dependientes, los valores

$$z_{u(i)} \leftarrow z_{u(i)} - \tilde{a}_{i,u(s)} z_{u(s)}, \quad (i = n+2, \dots, T)$$

De esta forma se logra que todas las variables libres y las variables dependientes que tienen asociado un coeficiente negativo en la columna $u(s)$ de la matriz de dependencia lineal, tomen valores positivos. Este algoritmo se encuentra desarrollado en el procedimiento 3.9 del apéndice A.

Para calcular los pesos y el umbral, se sustituyen los valores calculados de las z_i en la primera expresión del sistema de ecuaciones (3.5). Esto nos proporciona la ecuación de un hiperplano separador.

Si el valor de k_1 es mayor que cero, entonces existen patrones mal clasificados, por lo que uno de los dos subconjunto contiene patrones de las dos clases. A este subconjunto se le aplica el procedimiento descrito.

El algoritmo termina cuando, en el proceso de división se obtiene un subconjunto de patrones linealmente separable.

3.4.5 Resumen del algoritmo para determinar los hiperplanos separadores.

El algoritmo descrito se puede resumir en el siguiente procedimiento, que determina los valores de la matriz de pesos y umbrales entre la capa de entrada y la capa oculta.

Procedimiento 3.1: *Cálculo de hiperplanos separadores.*

Paso 1: Ordenar el conjunto de entrenamiento de forma tal que primero se encuentren todos los patrones de la clase 1.

Paso 2: Determinar la matriz asociada al sistema de ecuaciones (3.4).

Paso 3: Transformar esta matriz a la matriz asociada al sistema de ecuaciones (3.5).

Paso 4: Extraer la matriz (3.7) de relaciones de dependencia lineal entre las variables auxiliares.

Paso 5: Aplicar el procedimiento 3.8.1. Si $k_2 > 0$, ir al paso 6, en caso contrario, ir al paso 8.

Paso 6: Aplicar el procedimiento 3.8.2.

Paso 7: Seleccionar $p = \max\{p_1, p_2\}$. Si $p = p_1$, seleccionar la matriz de dependencia lineal del paso 5, en caso contrario, seleccionar la del paso 6.

Paso 8: Aplicar procedimiento 3.9

Paso 9: Determinar los valores de los pesos y el umbral a partir de la primera expresión del sistema de ecuaciones (3.5).

Paso 10: Determinar si un subconjunto contiene patrones de las dos clases.

- Si $p = p_1$ y $k_2 > 0$
 - Eliminar los patrones X_i , para los cuales se cumple que $\tilde{a}_{i,u(s)} < 0$, $u(i) > T_1$
 - Actualizar $T \leftarrow T - p_1$
- Si $p = p_2$
 - Eliminar los patrones X_i , para los cuales se cumple que $\tilde{a}_{i,u(s)} < 0$, $u(i) \leq T_1$
 - actualizar $T \leftarrow T - p_2$, $T_1 \leftarrow T_1 - p_2$
 - Si $k_2 > 0$ regresar al paso 1, en caso contrario parar.

Al concluir este procedimiento, se tienen todas las neuronas de la capa oculta con sus parámetros correspondientes.

3.4.6 Cálculo de los pesos y el umbral entre la capa oculta y la capa de salida.

Si ahora, se le presenta a una red, como la que se muestra en la figura 19, todos los patrones de entrenamiento $X_i = (x_{i1}, x_{i2}, \dots, x_{in})^t \in R^n$, ($i = 1, \dots, T$), se obtiene que con las imágenes $Y_i = (y_{i1}, y_{i2}, \dots, y_{im})^t \in R^m$, ($i = 1, \dots, T$) se puede formar un conjunto de entrenamiento $\{(Y_1, d_1), (Y_2, d_2), \dots, (Y_T, d_T)\}$ que es linealmente separable.

Si a este nuevo conjunto de entrenamiento formado por las imágenes de los patrones originales, se le aplica el procedimiento 3.1 descrito anteriormente, al resolver el problema (P3.3) se obtiene un hiperplano separador de la forma $Y^t \cdot V - v_{m+1} = 0$, que divide a este conjunto en las dos clases C_1 y C_2 .

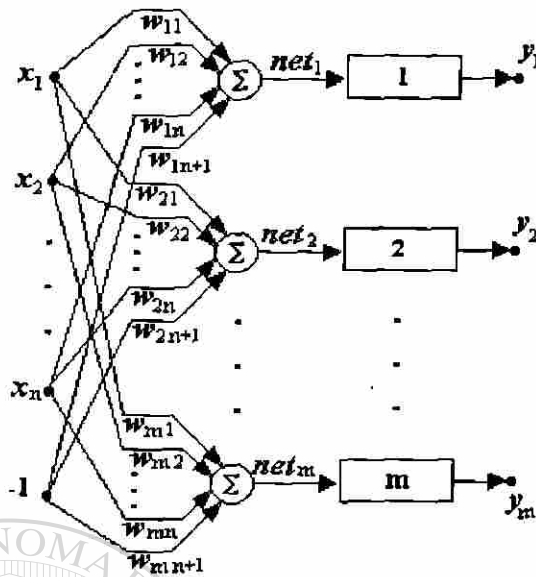


Figura 19 : Red neuronal transformadora a un conjunto linealmente separable.

De aquí se tiene, que es posible completar la red, como la que se muestra en la figura 20, la cual clasifica correctamente al conjunto original de patrones.

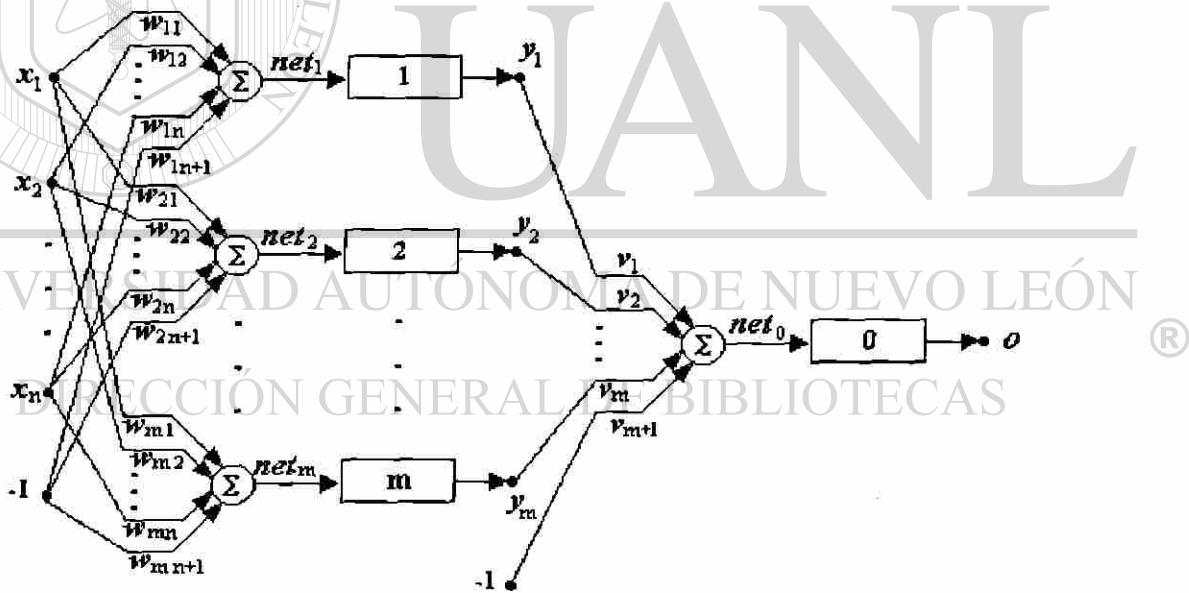


Figura 20 : Red neuronal clasificadora con funciones discretas de activación.

El proceso para determinar los valores de los pesos y el umbral entre las neuronas de la capa oculta y la neurona de la capa de salida se resume en el siguiente procedimiento.

Procedimiento 3.2: *Cálculo de los pesos y el umbral entre la capa oculta y la capa de salida.*

Paso 1: Formar el conjunto de entrenamiento $\{(Y_1, d_1), (Y_2, d_2), \dots, (Y_T, d_T)\}$, donde

$$Y_i = (y_{i1}, y_{i2}, \dots, y_{im})^t, \quad (i = 1, \dots, T),$$

$$y_{ij} = \operatorname{sgn}\left(\sum_{k=1}^n x_{ik} w_{jk} - w_{j,n+1}\right), \quad (j = 1, \dots, m; i = 1, \dots, T).$$

Paso 2: Aplicar el procedimiento 3.1 al conjunto $\{(Y_1, d_1), (Y_2, d_2), \dots, (Y_T, d_T)\}$.

3.5 Conclusiones.

En este capítulo, a partir de un análisis de la estructura de los espacios de patrones y de pesos, se desarrolló un algoritmo que nos permite obtener una red neuronal de tres capas con funciones discretas de activación en las neuronas, la cual clasifica correctamente un conjunto de patrones en dos clase o categorías.

Esto representa un nuevo método de diseño de redes neuronales para la clasificación, ya que los otros métodos reportados en la literatura se basan en el método clásico de aprendizaje del perceptrón discreto o en alguna de sus modificaciones.

En la primera fase del algoritmo presentado, en cada iteración se obtiene una neurona de la capa oculta con sus parámetros correspondientes. Al finalizar esta fase se obtiene una matriz de pesos entre la capa de entrada y la capa oculta.

En la segunda fase, se forma un conjunto de aprendizaje con las imágenes de los patrones originales al pasar por la capa oculta y se le aplica una sola vez el mismo procedimiento que al conjunto de aprendizaje original. Esto proporciona un vector de pesos entre la capa oculta y la capa de salida.

Se ha comprobado, además, que en el peor de los casos, el número de neuronas que se obtiene por este algoritmo es igual a $(T-1)$, donde T es la cardinalidad del conjunto de entrenamiento, el cual coincide con el valor de la cota superior dado en [3,4].

CAPITULO 4

ENTRENAMIENTO DE UNA RED

NEURONAL PARA LA

CLASIFICACION CON FUNCIONES

CONTINUAS DE ACTIVACION

4.1 Introducción.

El algoritmo de retropropagación del error (*backpropagation*) [103,104] está considerado como uno de los logros de mayor importancia en la Teoría de las Redes Neuronales de propagación hacia adelante.

Este algoritmo está diseñado para entrenar redes con varias capas de neuronas ocultas, pero está demostrado que una sola capa oculta es suficiente para que una red neuronal de propagación hacia adelante pueda aproximar de forma uniforme cualquier función continua soportada sobre un hipercubo unitario [15,33].

Es conocido que este algoritmo de entrenamiento presenta algunas deficiencias y esto ha sido el centro de atención de las críticas de diferentes autores.

Entre las deficiencias fundamentales que presenta, están:

1. Es desconocido el número de neuronas que se necesitan en la capa oculta para que el algoritmo de aprendizaje sea convergente.

2. La cantidad de ciclos de entrenamiento depende de los valores iniciales de los pesos, los cuales son seleccionados de forma aleatoria.
3. Alto número de operaciones en el proceso de entrenamiento de la red neuronal, tanto en su fase hacia adelante en el cálculo de las salidas, como en su fase hacia atrás en la modificación de los pesos.

Entre las deficiencias, la tercera es la que ha sido tratada más ampliamente en la literatura [7,19,81,99]. En la mayoría de los trabajos se pretende disminuir el número de operaciones en la fase de entrenamiento, pero todos se basan en realizar modificaciones al algoritmo de retropropagación del error.

En el presente capítulo, a partir de la distribución de pesos obtenida en el capítulo anterior, se desarrollará un algoritmo que aborda el problema de entrenamiento de manera diferente a como se ha hecho anteriormente.

La ventaja fundamental de este algoritmo es que reduce considerablemente el número de operaciones del proceso de entrenamiento, en comparación con las diferentes variantes del algoritmo de retropropagación del error.

4.2 Formulación del problema.

En este capítulo se estudia el problema de clasificación de un conjunto de patrones en dos clases, o categorías, C_1 y C_2 por una red neuronal con funciones continuas de activación del tipo

$$f(net) = \frac{2}{1 + e^{-\lambda net}} - 1 \quad (4.1)$$

Denotemos por

$$\{(X_1, d_1), (X_2, d_2), \dots, (X_T, d_T)\}$$

el conjunto de aprendizaje, donde $X_i = (x_{i1}, x_{i2}, \dots, x_{im}) \in \mathbf{R}^n$, ($i = 1, \dots, T$) representan los patrones de entrenamiento y d_i es la salida deseada del patrón X_i ,

siendo, además $d_i = \begin{cases} 1, & \text{si } X_i \in C_1 \\ -1, & \text{si } X_i \in C_2 \end{cases}$,

El problema de entrenamiento se puede formular como:

Determinar una matriz de pesos

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} & w_{1,n+1} \\ w_{21} & w_{22} & \cdots & w_{2n} & w_{2,n+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} & w_{m,n+1} \end{bmatrix}$$

entre la capa de entrada y la capa oculta, y un vector de pesos

$$V = [v_1, v_2, \dots, v_m, v_{m+1}]$$

entre la capa oculta y la capa de salida, de forma tal que

$$E = \frac{1}{2} \sum_{i=1}^T (d_i - o_i)^2 < E_{max} \quad ,$$

donde

$$o_i = f \left[\sum_{j=1}^m v_j f \left(\sum_{k=1}^n x_{ik} w_{jk} - w_{j,n+1} \right) - v_{m+1} \right]$$

$$f(net) = \frac{2}{1 + e^{-\lambda net}} - 1$$

4.3 Análisis del problema planteado.

El primer problema que se presenta en el entrenamiento de esta red, es que se desconoce la cantidad de neuronas de la capa oculta, pero en [93] está demostrado que, si una red con funciones discretas de activación en las neuronas puede clasificar correctamente un conjunto de patrones, entonces una red que tenga la misma estructura, pero que utilice funciones continuas de activación en las neuronas, también puede hacerlo, usando el algoritmo de retropropagación del error como algoritmo de aprendizaje.

Esto significa que para determinar la cantidad de neuronas en la capa oculta de esta red con funciones continuas de activación, se puede utilizar el algoritmo desarrollado en el capítulo anterior para clasificar correctamente el conjunto de patrones, cuando las funciones de activación en las neuronas son discretas.

Este algoritmo, además de la cantidad de neuronas en la capa oculta, proporciona una matriz de pesos entre la capa de entrada y la capa oculta

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} & w_{1,n+1} \\ w_{21} & w_{22} & \cdots & w_{2n} & w_{2,n+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} & w_{m,n+1} \end{bmatrix}$$

y un vector de pesos entre la capa oculta y la capa de salida

$$V = [v_1, v_2, \dots, v_m, v_{m+1}]$$

que pueden ser usados como una memoria inicial para el problema de entrenamiento con funciones continuas de activación. Esto salva la dificultad de la inicialización aleatoria de los pesos.

Este algoritmo proporciona una red entrenada, como se muestra en la figura 21, que clasifica correctamente el conjunto de patrones, utilizando en las neuronas funciones de activación bipolares discretas, es decir, funciones del tipo

$$f(\text{net}) = \begin{cases} 1, & \text{si } \text{net} > 0 \\ -1, & \text{si } \text{net} \leq 0 \end{cases}$$

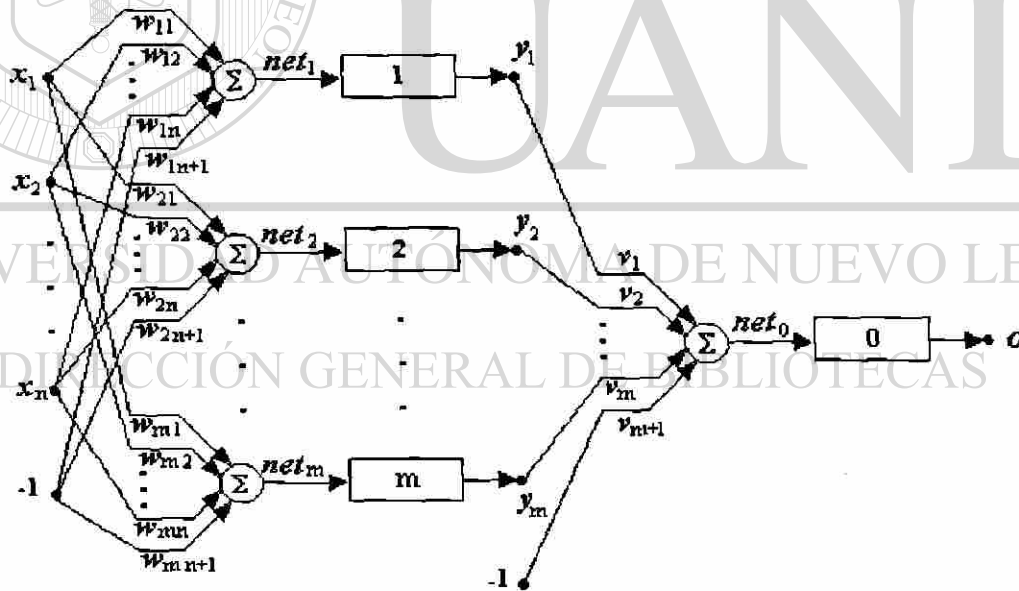


Figura 21 : Red neuronal con funciones discretas de activación.

En esta red, las salidas de la capa oculta para cada patrón de entrenamiento forman un vector con componentes del conjunto $\{-1, 1\}$, por lo que cada uno de estos vectores es un vértice del hipercubo $[-1, 1]^m$, es decir, se obtiene una matriz

$$Y = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1m} \\ y_{21} & y_{22} & \cdots & y_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ y_{T1} & y_{T2} & \cdots & y_{Tm} \end{bmatrix},$$

donde cada fila $Y_i = (y_{i1}, y_{i2}, \dots, y_{im})$, ($i = 1, \dots, T$) representa la imagen de cada patrón de entrenamiento X_i , al pasar por la capa oculta. Además, como la red clasifica correctamente al conjunto de patrones, entonces se cumple que

$$E = \frac{1}{2} \sum_{i=1}^T (d_i - o_i)^2 = 0,$$

donde

$$o_i = \operatorname{sgn} \left[\sum_{j=1}^m v_j \operatorname{sgn} \left(\sum_{k=1}^n x_{ik} w_{jk} - w_{j,n+1} \right) - v_{m+1} \right],$$

lo que significa que estas imágenes son linealmente separables por el hiperplano

$$v_1 y_1 + v_2 y_2 + \cdots + v_m y_m - v_{m+1} = 0. \quad (4.2)$$

Para resolver el problema de entrenamiento planteado, se cambiarán las funciones discretas de activación en la red de la figura 21, por funciones continuas del tipo (4.1), obteniéndose una red como la que se muestra en la figura 22.

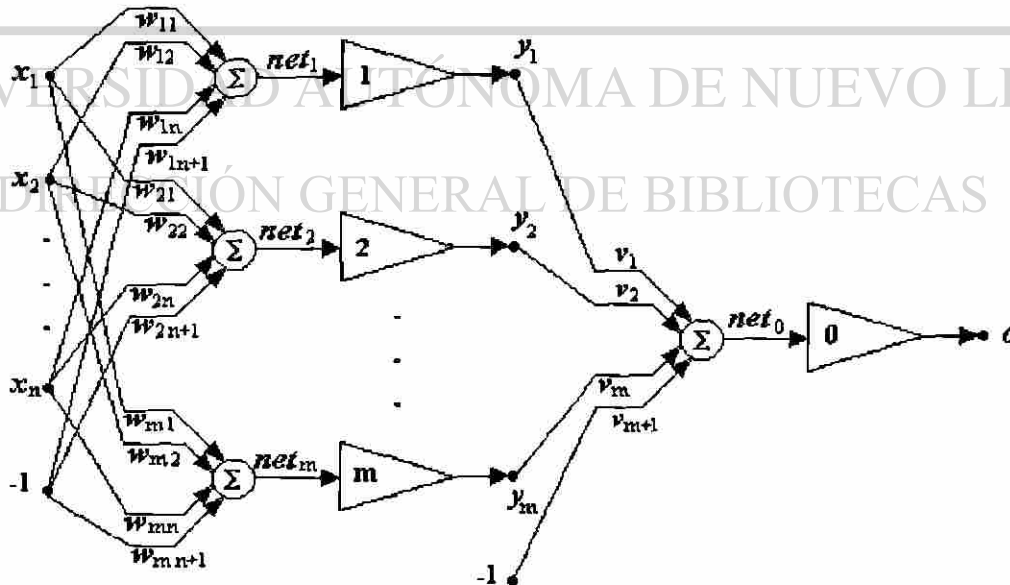


Figura 22 : Red neuronal con funciones continuas de activación.

Estas funciones tienen la propiedad de que $|f(\text{net})| < 1$, y además cuando su argumento tiende a $\pm \infty$, sus valores se aproximan a ± 1 .

Por tanto, al utilizar este tipo de función en las neuronas de la capa oculta, los valores de y_{ij} cumplen que $|y_{ij}| < 1$, $i = 1, \dots, T$, $j = 1, \dots, m$, y en consecuencia los vectores

$$Y_i = (y_{i1}, y_{i2}, \dots, y_{im}) \quad , \quad i = 1, \dots, T \quad (4.3)$$

serán puntos interiores del hipercubo $[-1, 1]^m$ y a pesar de que pueden ser linealmente separables, generalmente no lo son por el hiperplano (4.2). Esto significa que para patrones mal clasificados, la diferencia $(d_i - o_i)$ en valor absoluto, siempre será mayor que 1, es decir, $|d_i - o_i| > 1$.

El problema planteado puede ser resuelto usando el algoritmo de retropropagación del error, pero al existir patrones mal clasificados, se cumple que $|d_i - o_i| > 1$ y esto ocasiona un incremento en la cantidad de ciclos de entrenamiento. De aquí se tiene, que el proceso de modificación de los pesos, para lograr que el error sea menor que un valor prefijado con antelación, resulta demasiado costoso desde el punto de vista computacional.

Debido a esto se buscará otra forma de realizar el entrenamiento de la red, que no sea mediante el algoritmo de retropropagación del error.

De todo lo antes expuesto se puede resumir que el problema de entrenamiento planteado consiste en, a partir de la distribución de pesos que se obtuvo para la red de la figura 21, entrenar una red, como la que se muestra en la figura 22, que mantenga la misma estructura, pero que utilice en las neuronas funciones de activación bipolares continuas del tipo (4.1).

El algoritmo de entrenamiento que se propone, disminuye de forma considerable tanto la cantidad de operaciones en un ciclo, así como la cantidad de ciclos a realizar. Además no presenta la dificultad de que los pesos sean inicializados de forma aleatoria.

4.4 Descripción del algoritmo de solución.

Resolver el problema de entrenamiento planteado, significa determinar un punto (o_1, o_2, \dots, o_T) sobre la superficie de error que sea interior a una bola en \mathbf{R}^T con centro en el punto (d_1, d_2, \dots, d_T) y de radio igual a $\sqrt{2E_{max}}$.

Para buscar este punto, el algoritmo de retropropagación del error modifica la posición de los hiperplanos separadores.

Para cambiar la posición de los hiperplanos separadores, es necesario modificar sus parámetros, es decir, los pesos de la red. Esto nos lleva a que en el proceso de entrenamiento es necesario modificar un total de $[m(n+1) + m + 1]$ parámetros y esta modificación se debe realizar un gran número de veces para llegar a la solución.

En el algoritmo de solución propuesto, el principio de funcionamiento es completamente diferente. En lugar de modificar las posiciones de los hiperplanos separadores, lo que se modifica son las posiciones de las imágenes de los patrones al pasar por la capa oculta.

En las funciones continuas de activación aparece un parámetro λ que generalmente se le asigna un valor unitario, pero si se aumenta el valor de este parámetro en las neuronas de la capa oculta se pueden aproximar las imágenes de los patrones de entrenamiento a los vértices del hipercono $[-1, 1]^m$ y convertir este conjunto de imágenes $Y_i = (y_{i1}, y_{i2}, \dots, y_{im})$, $i = 1, \dots, T$ en un conjunto linealmente separable por el hiperplano (4.2).

El algoritmo de solución propuesto, se basa en las propiedades, antes mencionadas, de las funciones de activación bipolares continuas y en considerar en estas funciones a λ como variable y no a la entrada neta, es decir,

$$f(\lambda) = \frac{2}{1 + e^{-\lambda net}} - 1.$$

Por lo tanto, en lugar de ajustar todos los pesos de la red, lo que se hace es modificar un parámetro λ en cada neurona.

Ahora, se considera que cada neurona j de la red tiene asociada una variable λ_j . Al aumentar los valores de λ_j en cada neurona de la capa oculta, las imágenes

de los patrones de entrenamiento se aproximan a los vértices del hipercubo $[-1, 1]^m$ y estos valores de λ_j se incrementarán hasta que el conjunto de las imágenes (4.3) sea separable por el hiperplano (4.2).

Analicemos ahora, como realizar el incremento de los valores de λ_j en las neuronas de la capa oculta para que el conjunto de las imágenes sea linealmente separable por el hiperplano (4.2).

El objetivo del procedimiento es mover las imágenes de los patrones mal clasificados hacia el otro semiespacio definido por el hiperplano separador, por lo que para incrementar los valores de las λ_j se utilizarán solamente los patrones mal clasificados. Además, como es deseable obtener valores pequeños de λ_j , el movimiento se realizará en la dirección del vector normal al hiperplano (4.2) y la magnitud del movimiento se determinará a partir de la imagen más alejada del hiperplano.

En primer lugar, se construye un conjunto de entrenamiento con las imágenes $Y_i = (y_{i1}, y_{i2}, \dots, y_{im})$, $(i = 1, \dots, T)$ de la forma $\{(Y_1, d_1), (Y_2, d_2), \dots, (Y_T, d_T)\}$. Entonces, para los patrones mal clasificados se cumple que

$$d_i(v_1 y_{i1} + v_2 y_{i2} + \dots + v_m y_{im} - v_{m+1}) \leq 0.$$

Como la distancia de los puntos $Y_i = (y_{i1}, y_{i2}, \dots, y_{im})$, $(i = 1, \dots, T)$ al hiperplano (4.2), se determina por la expresión

$$h_i = \frac{|v_1 y_{i1} + v_2 y_{i2} + \dots + v_m y_{im} - v_{m+1}|}{\sqrt{v_1^2 + v_2^2 + \dots + v_m^2}},$$

para determinar el patrón mal clasificado que está más alejado del hiperplano, entre los valores de i que cumplen

$$d_i(v_1 y_{i1} + v_2 y_{i2} + \dots + v_m y_{im} - v_{m+1}) \leq 0,$$

se calcula

$$\min_i \{d_i(v_1 y_{i1} + v_2 y_{i2} + \dots + v_m y_{im} - v_{m+1})\}.$$

Supóngase que el valor del mínimo se alcanzó para el patrón $Y_k = (y_{k1}, y_{k2}, \dots, y_{km})$. Como el movimiento se realiza en la dirección del vector normal al hiperplano, se deben incrementar las componentes del vector Y_k según la fórmula

$$y_{kj}^1 = y_{kj}^0 + \alpha v_j, \quad (4.4)$$

donde el superíndice 0 está asociado a los valores iniciales y el superíndice 1 a los valores modificados.

Denotemos por $Y_k^1 = (y_{k1}^1, y_{k2}^1, \dots, y_{km}^1)$ a la proyección de Y_k^0 sobre el hiperplano. Para determinar el valor de α se utilizarán las coordenadas de este punto proyección.

Como este punto está sobre el hiperplano, entonces satisface su ecuación, es decir,

$$v_1 y_{k1}^1 + v_2 y_{k2}^1 + \dots + v_m y_{km}^1 - v_{m+1} \equiv 0.$$

De aquí, si se sustituyen los valores de y_{kj}^1 por su expresión en (4.4), se obtiene

$$v_1 (y_{k1}^0 + \alpha v_1) + v_2 (y_{k2}^0 + \alpha v_2) + \dots + v_m (y_{km}^0 + \alpha v_m) - v_{m+1} \equiv 0,$$

de donde es posible determinar el valor de α por la fórmula

$$\alpha = -\frac{v_1 y_{k1}^0 + v_2 y_{k2}^0 + \dots + v_m y_{km}^0 - v_{m+1}}{v_1^2 + v_2^2 + \dots + v_m^2}. \quad (4.5)$$

Ahora, si con este valor de α se calculan

$$y_{ij}^1 = \begin{cases} y_{ij}^0 + |\alpha| v_j, & \text{si } d_i = 1 \\ y_{ij}^0 - |\alpha| v_j, & \text{si } d_i = -1 \end{cases}, \quad (i = 1, \dots, T; j = 1, \dots, m),$$

entonces todos los patrones $Y_i^1 = (y_{i1}^1, y_{i2}^1, \dots, y_{im}^1)$, ($i = 1, \dots, T$) estarán bien clasificados por el hiperplano (4.2), menos el patrón Y_k^1 , que estará exactamente sobre el hiperplano.

Ahora es necesario determinar los valores de los parámetros λ_j que le corresponden a este valor de α .

Los valores de y_{ij} en la red de la figura 22, se determinan por la expresión

$$y_{ij} = \frac{2}{1 + \exp(-net_{ij}\lambda_j)} - 1, \text{ de donde es posible determinar qué valores deben tomar}$$

los parámetros λ_j para que las imágenes iniciales Y_i^0 , ($i = 1, \dots, T$) se transformen en Y_i^1 , ($i = 1, \dots, T$).

Según esta fórmula, los valores de y_{kj}^1 deben cumplir

$$y_{kj}^1 = \frac{2}{1 + \exp(-net_{kj}\lambda_j^1)} - 1, \quad (4.6)$$

por lo que se puede despejar el valor de λ_j^1 de la forma

$$\lambda_j^1 = -\frac{1}{net_{kj}} \ln \left[\frac{2}{y_{kj}^1 + 1} - 1 \right] \quad (4.7)$$

Ahora para que todos los patrones estén bien clasificados, es suficiente asignar

$$\lambda_j \leftarrow \lceil \lambda_j^1 \rceil + 1 \quad (4.8)$$

Si en las funciones de activación de las neuronas de la capa de la red de la figura 22 se le asignan a los parámetros λ_j los valores calculados por la expresión

(4.8), se logra que $|d_i - o_i| < 1$. Si ahora se aumenta el valor de λ_0 en la neurona de

salida, los valores de o_i se aproximan a los d_i y el valor del error se puede hacer tan pequeño como se quiera.

Una primera aproximación del valor de λ_0 se determina a partir del patrón Y_k , que es el más próximo al hiperplano y por lo tanto es el que aporta una mayor contribución al valor del error, es decir, determinamos cuánto debe valer λ_0 para que

$$\frac{1}{2}(d_k - o_k)^2 = E_{max}.$$

A partir de esta relación se obtiene que

$$o_k = d_k \left(1 - \sqrt{2E_{max}} \right)$$

y por otro lado, según la expresión para el cálculo de o_k , se tiene que

$$o_k = \frac{2}{1 + \exp(-neto_k \lambda_0)} - 1$$

De la igualdad de estas dos expresiones para o_k , es posible despejar λ_0 y se obtiene

$$\lambda_0 = -\frac{1}{neto_k} \ln \left[\frac{2}{d_k (1 - \sqrt{2E_{max}}) + 1} - 1 \right] \quad (4.9)$$

Ahora, realizando la asignación

$$\lambda_0 \leftarrow \lceil \lambda_0 \rceil + 1 \quad (4.10)$$

se garantiza que la contribución al error total de cada patrón por separado, sea menor que E_{max} , es decir,

$$\frac{1}{2} (d_i - o_i)^2 < E_{max}, \text{ para } i = 1, \dots, T.$$

Este valor de λ_0 no garantiza que se cumpla la condición de error total

$$\frac{1}{2} \sum_{i=1}^T (d_i - o_i)^2 < E_{max}, \quad (4.11)$$

pero en la solución de problemas específicos, cuando este valor no es el óptimo, se encuentra muy cercano a él.

Después de calcular el valor de λ_0 por la expresión (4.9) y ajustarlo por (4.10), se pasa a verificar si se cumple la condición de error total (4.11). Para ello se calculan

$$o_i = \frac{2}{1 + \exp(\lambda_0 neto_i)} - 1, \quad (i = 1, \dots, T) \quad \text{y} \quad E = \frac{1}{2} \sum_{i=1}^T (d_i - o_i)^2. \quad \text{Si } E \geq E_{max},$$

entonces se incrementa $\lambda_0^{k+1} = \lambda_0^k + 1$ y de nuevo se calculan las salidas o_i , ($i = 1, \dots, T$) y el valor del error total. Este proceso se repite hasta que se cumpla la condición de error (4.11).

De todo lo expuesto, se puede resumir que algoritmo propuesto consta de dos fases o etapas de entrenamiento. En una primera fase se modifican los valores de los parámetros λ_j en las neuronas de la capa oculta, para que el conjunto de imágenes de los patrones de entrenamiento sea linealmente separable por el hiperplano (4.2) y se mantiene fijo el valor de λ_0 en la neurona de la capa de salida.

En la segunda etapa se incrementa el valor de λ_0 hasta que se alcance un valor del error menor que un valor prefijado con antelación, manteniendo fijos los valores de las λ_j en las neuronas de la capa oculta. Inicialmente, el incremento de λ_0 se realiza a partir del patrón que aporta el sumando de mayor valor al error total y si con este valor de λ_0 , aún no se cumple la condición de error, se le dan incrementos unitarios, según la expresión $\lambda_0^{k+1} = \lambda_0^k + 1$.

Este algoritmo recibe como datos :

a) La matriz de patrones de entrenamiento

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{T1} & x_{T2} & \cdots & x_{Tn} \end{bmatrix},$$

donde cada fila representa un elemento del conjunto de patrones.

b) El vector de salidas deseadas para los patrones de entrenamiento

$$\mathbf{d} = [d_1, d_2, \dots, d_T]^t$$

c) La matriz de pesos entre la capa de entrada y la capa oculta

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} & w_{1,n+1} \\ w_{21} & w_{22} & \cdots & w_{2n} & w_{2,n+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} & w_{m,n+1} \end{bmatrix}$$

y el vector de pesos entre la capa oculta y la capa de salida

$$\mathbf{V} = [v_1, v_2, \dots, v_m, v_{m+1}]$$

que se obtienen del algoritmo desarrollado en el capítulo 3.

4.5 Resumen del algoritmo.

Procedimiento 4.1: *Entrenamiento con funciones continuas de activación.*

Paso 1 : Inicializar los valores de $\lambda_j = 1$, $j = 0, \dots, m$, el valor de $k = 0$ y el valor de E_{max} .

Paso 2 : Calcular la matriz de entrada a las neuronas de la capa oculta,

$$net = \begin{bmatrix} net_{11} & net_{12} & \cdots & net_{1m} \\ net_{21} & net_{22} & \cdots & net_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ net_{T1} & net_{T2} & \cdots & net_{Tm} \end{bmatrix}$$

donde cada fila $net_i = [net_{i1}, net_{i2}, \dots, net_{im}]$ representa el vector de entradas a la capa oculta para el patrón de entrenamiento X_i y sus componentes se determinan de la forma

$$net_{ij} = \sum_{k=1}^n x_{ik} w_{jk} - w_{j,n+1}, \quad i = 1, \dots, T; \quad j = 1, \dots, m$$

Paso 3 : Calcular la matriz de salida de la capa oculta

$$Y = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1m} \\ y_{21} & y_{22} & \cdots & y_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ y_{T1} & y_{T2} & \cdots & y_{Tm} \end{bmatrix},$$

donde cada fila $Y_i = (y_{i1}, y_{i2}, \dots, y_{im})$, ($i = 1, \dots, T$) representa el vector de salidas de la capa oculta para el patrón de entrenamiento X_i y sus componentes se determinan por la expresión

$$y_{ij} = \frac{2}{1 + \exp(\lambda_j net_{ij})} - 1.$$

Paso 4 : Entre los valores de i para los que se cumple

$$d_i(v_1 y_{i1} + v_2 y_{i2} + \cdots + v_m y_{im} - v_{m+1}) \leq 0,$$

determinar el índice k , para el cual se alcanza

$$\min_i \{d_i(v_1 y_{i1} + v_2 y_{i2} + \cdots + v_m y_{im} - v_{m+1})\}$$

Paso 5 : Si $k \neq 0$, calcular

- $\alpha = -\frac{v_1 y_{k1}^0 + v_2 y_{k2}^0 + \cdots + v_m y_{km}^0 - v_{m+1}}{v_1^2 + v_2^2 + \cdots + v_m^2}$
- $y_{kj}^1 = y_{kj}^0 + \alpha v_j, \quad (j = 1, \dots, m)$

- $\lambda_j^1 = -\frac{1}{net_{kj}} \ln \left[\frac{2}{y_{kj}^1 + 1} - 1 \right]$

- $\lambda_j \leftarrow \lceil \lambda_j^1 \rceil + 1$

Paso 6 : Calcular

- $y_{ij} = \frac{2}{1 + \exp(\lambda_j net_{ij})} - 1$, $i = 1, \dots, T$; $j = 1, \dots, m$.

- $neto_i = \sum_{j=1}^m (v_j y_{ij}) - v_{m+1}$, $i = 1, \dots, T$.

Paso 7 : Si $k = 0$

- Calcular $o_i = \frac{2}{1 + \exp(\lambda_0 neto_i)} - 1$, ($i = 1, \dots, T$)

- Entre todos los valores de i , determinar el índice k para el cual se alcanza

$\min_i |o_i|$
en caso contrario

- Calcular $o_k = \frac{2}{1 + \exp(\lambda_0 neto_k)} - 1$

Paso 8 : Si $|d_k - o_k| > \sqrt{2E_{max}}$, entonces $\lambda_0 = -\frac{1}{neto_k} \ln \left[\frac{2}{d_k (1 - \sqrt{2E_{max}}) + 1} - 1 \right]$

Paso 9 : Asignar $\lambda_0 \leftarrow \lceil \lambda_0 \rceil + 1$

Paso 10 : Calcular

- $o_i = \frac{2}{1 + \exp(\lambda_0 neto_i)} - 1$, ($i = 1, \dots, T$)

- $E = \frac{1}{2} \sum_{i=1}^T (d_i - o_i)^2$

Paso 11 : Si $E > E_{max}$, asignar $\lambda_0 \leftarrow \lambda_0 + 1$, ir al paso 10 , en caso contrario parar.

4.6 Conclusiones.

En el presente capítulo se desarrolló una nueva forma de entrenar una red neuronal de propagación hacia adelante con funciones continuas de activación en las neuronas, a partir de una distribución inicial de pesos que se obtiene de una red que tiene la misma estructura, pero con funciones discretas de activación, la cual clasifica correctamente un conjunto de patrones de entrenamiento, en dos clase o categorías.

Con el procedimiento presentado se logra reducir considerablemente la cantidad de operaciones a realizar en el proceso de aprendizaje de una red neuronal en comparación con las diferentes variantes del algoritmo de retropropagación del error.

Con el algoritmo descrito, se reduce de forma considerable la cantidad de operaciones en un ciclo de entrenamiento, la cantidad de ciclos a realizar y la cantidad de parámetros a modificar en el proceso de aprendizaje.

En este procedimiento los valores de las entradas netas net_{ij} a las neuronas de la capa oculta son calculados una sola vez, mientras que el algoritmo de retropropagación del error tiene que hacerlo, como mínimo, en cada ciclo de entrenamiento.

La cantidad total de operaciones de multiplicación de este proceso es del orden de $[n.m.T]$, donde n es la dimensión de los patrones de entrenamiento, m es la cantidad de neuronas en la capa oculta y T es la cardinalidad del conjunto de aprendizaje.

Además, el algoritmo de retropropagación del error tiene que modificar, en su forma más eficiente, un total de $[m.(n+1) + m + 1]$ parámetros de la red en cada ciclo de aprendizaje, mientras que el algoritmo propuesto tiene que modificar $[m+1]$ parámetros en el primer ciclo de entrenamiento y en los ciclos restantes, si son necesarios, solamente se modifica un parámetro mediante un incremento unitario.

CAPITULO 5

DISEÑO Y ENTRENAMIENTO DE UNA RED NEURONAL PARA LA CLASIFICACION DE MEMORIA ENTERA

5.1 Introducción.

Entre las aplicaciones fundamentales de los modelos de redes neuronales de propagación hacia adelante se encuentran la clasificación y el reconocimiento de patrones. Cuando la memoria de la red está formada por valores enteros pequeños, estos modelos resultan muy atractivos desde el punto de vista práctico, ya que sus diferentes formas de implementación son sumamente económicas. Por ejemplo, cuando los pesos están restringidos a valores enteros del intervalo $[-3, 3]$, se requieren solamente de 3 bits para almacenar los pesos de la red.

El caso ideal es cuando los pesos toman solamente valores del conjunto $\{-1, 0, 1\}$, que son las llamadas redes libres de multiplicación [47], ya que para calcular la entrada neta a cada neurona no es necesario realizar operaciones de multiplicación. Además, en este caso los pesos de la red pueden ser almacenados mediante cadenas binarias de longitud 2.

Cuando el entrenamiento de la red se realiza para la clasificación de patrones en dos clases o categorías, si el conjunto de patrones es linealmente separable, entonces existe un hiperplano que separa el conjunto de patrones, y aquí se puede implementar un perceptrón (discreto o continuo) que resuelve correctamente el problema de clasificación.

Si el conjunto de patrones no es linealmente separable, el problema de aprendizaje puede ser resuelto, si se le agrega a la red una capa oculta de neuronas [15,33], pero al considerar pesos enteros el problema de entrenamiento está considerado como NP-completo [46].

Por otra parte, en el proceso de entrenamiento de una red neuronal con capas ocultas, uno de los problemas más difíciles, aún cuando los pesos pueden tomar valores reales, es el diseño de la red, es decir, determinar la cantidad de neuronas en la capas ocultas para que el algoritmo de entrenamiento sea convergente. Este problema también está considerado como un problema NP-completo [90].

Diferentes autores [16,71,72,77,84,86] han tratado el problema de entrenamiento de redes con pesos discretos. Por ejemplo, en [22] se describen tres técnicas de discretización de los pesos para redes ópticas que parten del algoritmo de retropropagación del error y en [101] se dan comparaciones empíricas de diferentes técnicas de discretización.

Los resultados más relevantes, relacionados con la capacidad de aproximación de las redes con memoria entera, son obtenidos en [47,48,49], los cuales están contenidos también en la tesis doctoral [46].

En [55,60,62] se presentan resultados de carácter teórico sobre la relación entre el número de neuronas en la capa oculta y el acotamiento de los pesos de la red.

Una mejora computacional del algoritmo de discretización descrito en [46] es presentada en [1] para valores de pesos del conjunto $\{-3, -2, -1, 0, 1, 2, 3\}$.

En ninguno de estos trabajos se trata el problema del diseño de la red, es decir, se supone conocida la cantidad de neuronas en la capa oculta. En la literatura revisada encontramos solamente un trabajo [71], donde se trata el problema del diseño de la

red para pesos binarios, pero el problema es tratado de forma diferente a como es resuelto en este capítulo.

En el presente capítulo, se estudia el problema de entrenamiento de una red con una capa oculta para clasificar un conjunto de patrones en dos clases, que no son linealmente separable. Además los pesos están restringidos a valores del conjunto $\{-3, -2, -1, 0, 1, 2, 3\}$.

El objetivo de este trabajo es desarrollar un algoritmo eficiente que resuelva, tanto el problema de diseño de la red, así como de su entrenamiento.

Aquí, el proceso de entrenamiento está dividido en dos fases. En la primera fase del algoritmo propuesto, se obtiene una red entrenada de memoria entera con funciones discretas de activación. En la segunda fase se implementan funciones continuas de activación en las neuronas de la red y mediante un proceso de modificación de un parámetro en cada función de activación se logra completar el proceso de entrenamiento.

5.2 Formulación del problema.

El problema de entrenamiento se puede formular como :

Dado el $\{(X_1, d_1), (X_2, d_2), \dots, (X_T, d_T)\}$ el conjunto de aprendizaje, donde $X_i = (x_{i1}, x_{i2}, \dots, x_{im}) \in \mathbb{R}^n$, $(i = 1, \dots, T)$ representan los patrones de entrenamiento y

d_i es la salida deseada del patrón X_i , siendo, además $d_i = \begin{cases} 1, & \text{si } X_i \in C_1 \\ -1, & \text{si } X_i \in C_2 \end{cases}$,

determinar una matriz de pesos

$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} & w_{1,n+1} \\ w_{21} & w_{22} & \dots & w_{2n} & w_{2,n+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} & w_{m,n+1} \end{bmatrix}$$

entre la capa de entrada y la capa oculta, y un vector de pesos

$$V = [v_1, v_2, \dots, v_m, v_{m+1}]$$

entre la capa oculta y la capa de salida, donde $w_{ij}, v_i \in \mathbb{Z}$, $i = 1, \dots, m$; $j = 1, \dots, n$, de forma tal que

$$E = \frac{1}{2} \sum_{i=1}^T (d_i - o_i)^2 < E_{max} \quad ,$$

donde

$$o_i = f \left[\sum_{j=1}^m v_j f \left(\sum_{k=1}^n x_{ik} w_{jk} - w_{j,n+1} \right) - v_{m+1} \right]$$

$$f(net) = \frac{2}{1 + e^{-\lambda net}} - 1 \quad (5.1)$$

Nótese que los umbrales de las neuronas no están restringidos a tomar valores enteros.

La primera dificultad que aparece al tratar de resolver este problema es que se desconoce el valor de m , es decir la cantidad de neuronas en la capa oculta, por lo que primero se tratará el problema del diseño de la red y luego el problema de su entrenamiento con una memoria entera.

5.3 Análisis del problema del diseño de la red.

Para determinar el valor de m , se usará un procedimiento similar al descrito en el capítulo 3, con algunas modificaciones.

En este procedimiento se parte de que, cuando el conjunto de patrones no es linealmente separable, entonces existen un grupo de desigualdades que se incumplen en el sistema

$$d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1} \right) > 0, \quad i = 1, \dots, T \quad (5.2)$$

y al introducir T variables auxiliares z_1, z_2, \dots, z_T , de la forma

$$d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1} \right) - z_i = 0, \quad (5.3)$$

$$i = 1, \dots, T$$

se logra una doble correspondencia entre patrón bien clasificado y variable auxiliar positiva.

Con la introducción de estas variables auxiliares, el problema se transforma en determinar las soluciones $(w_1, w_2, \dots, w_n, w_{n+1}, z_1, z_2, \dots, z_T)$ del sistema de ecuaciones (5.3) que contengan valores enteros para las variables w_j , $(j = 1, \dots, n)$ y la mayor cantidad de valores positivos para las variables auxiliares correspondientes a patrones de una misma clase.

Supóngase que en el conjunto de entrenamiento se tienen T_1 patrones de la clase 1 y $T_2 = T - T_1$ patrones de la clase 2 y además, que están ordenados de forma tal que primero se encuentran los T_1 patrones de la clase 1.

El sistema de ecuaciones (5.3) puede ser escrito como

$$d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1} \right) = z_i, \quad (i = 1, \dots, T) \quad (5.4)$$

y para determinar el hiperplano que separe la mayor cantidad de patrones de una misma clase, es necesario resolver los siguientes problemas

$$\begin{cases} \max \sum_{i=M+1}^T \text{sgn}(z_i) \\ \text{s. a } d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1} \right) = z_i, \quad i = 1, \dots, T \end{cases} \quad (P5.1)$$

$$\begin{cases} z_i > 0, \quad i = 1, \dots, M \\ w_j \in \mathbb{Z}, \quad j = 1, \dots, n \end{cases}$$

$$\begin{cases} \max \sum_{i=1}^M \text{sgn}(z_i) \\ \text{s. a } d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1} \right) = z_i, \quad i = 1, \dots, T \\ z_i > 0, \quad i = M+1, \dots, T \\ w_j \in \mathbb{Z}, \quad j = 1, \dots, n \end{cases} \quad (P5.2)$$

Estos problemas, por su forma, son similares a los problemas (P3.1) y (P3.2) respectivamente, pero al restringir los pesos a que puedan tomar solamente valores enteros, se convierten en problemas de una complejidad aún mayor.

5.4 Descripción del algoritmo de solución para el problema de diseño.

Para resolver estos problemas, se hará primeramente una relajación, considerando que los pesos pueden tomar valores reales, para posteriormente aplicarle los procedimientos del capítulo 3 descritos en el apéndice A.

En estos procedimientos, se realiza un intercambio entre variables libres y variables dependientes del sistema (5.4), mediante un proceso de eliminación gaussiana, de forma tal que queden como variables dependientes las componentes del vector de pesos y el umbral y pasen a ser variables libres $n + 1$ de las T componentes del vector de variables auxiliares, obteniendo

$$\begin{cases} w_i = \sum_{j=1}^{n+1} a_{ij} z_j, & i = 1, \dots, n+1 \\ \sum_{j=1}^{n+1} (a_{ij} z_j) + z_i = 0, & i = n+2, \dots, T \end{cases} \quad (5.5)$$

Por comodidad de notación, en este sistema se hizo la suposición de que las primeras $n + 1$ variables auxiliares z_1, z_2, \dots, z_{n+1} son ahora las variables libres

La primera expresión en (5.5) proporciona las dependencias de los pesos y el umbral de las nuevas variables libres z_1, z_2, \dots, z_{n+1} . La segunda expresión en (5.5)

proporcionan las relaciones de dependencia lineal entre las variables libres z_1, z_2, \dots, z_{n+1} y las variables dependientes $z_{n+2}, z_{n+3}, \dots, z_T$.

Como la resolución del problema relajado consiste en determinar entre las soluciones del sistema (5.4), una que tenga la mayor cantidad de componentes positivos para las variables auxiliares, se puede trabajar solamente con la segunda de las expresiones en (5.5), es decir, con el sistema

$$\sum_{j=1}^{n+1} (a_{ij} z_j) + z_i = 0, \quad i = n+2, \dots, T \quad (5.6)$$

La matriz de este sistema de ecuaciones tiene la forma

$$\begin{bmatrix} a_{n+2,1} & a_{n+2,2} & \cdots & a_{n+2,n+1} & 1 & 0 \cdots 0 \\ a_{n+3,1} & a_{n+3,2} & \cdots & a_{n+3,n+1} & 0 & 1 \cdots 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots \vdots \\ a_{T,1} & a_{T,2} & \cdots & a_{T,n+1} & 0 & 0 \cdots 1 \end{bmatrix} \quad (5.7)$$

Para determinar el hiperplano separador, se tiene que resolver uno de los siguientes problemas, denotados por (P5.3) y (P5.4) respectivamente.

$$\begin{cases} \max \sum_{i=M+1}^T \text{sgn}(z_i) \\ \text{s.a.} \sum_{j=1}^{n+1} (a_{ij} z_j) + z_i = 0, \quad i = n+2, \dots, T \\ z_i > 0, \quad i = 1, \dots, M \end{cases} \quad (P5.3)$$

$$\begin{cases} \max \sum_{i=1}^M \text{sgn}(z_i) \\ \text{s.a.} \sum_{j=1}^{n+1} (a_{ij} z_j) + z_i = 0, \quad i = n+2, \dots, T \\ z_i > 0, \quad i = M+1, \dots, T \end{cases} \quad (P5.4)$$

Como se puede apreciar, en estos problemas no están incluidas las variables w_j , ($j = 1, \dots, n+1$), pero sus valores se pueden determinar de la primera expresión en (5.5).

Si a estos problemas se les aplican los procedimientos del capítulo 3, se obtiene una matriz de pesos entre la capa de entrada y la capa oculta, pero sus elementos tendrán valores reales. Para obtener una memoria con valores enteros se realizarán algunas modificaciones a estos procedimientos.

Debido a que en [6] se muestra que cuando los pesos toman valores del conjunto $\{-3, -2, -1, 0, 1, 2, 3\}$, se puede obtener una gran diversificación de hiperplanos separadores, en el algoritmo que se propone, se considera este conjunto para los valores de los pesos.

En el proceso de discretización de los pesos se explotan los siguientes resultados:

1. Si se multiplican los valores de los pesos y el umbral por una constante positiva, el hiperplano conserva la propiedad de ser separador.
2. La ecuación del hiperplano separador no es única, es decir, se pueden determinar diferentes ecuaciones de hiperplanos separadores para obtener los mismos subconjuntos de patrones.

Utilizando el primer resultado se puede realizar un escalamiento de los pesos a que tomen valores del intervalo $[-3, 3]$, seleccionando $w = \max_{1 \leq j \leq n} |w_j|$ y multiplicando los pesos y el umbral por $\frac{3}{w}$.

El segundo resultado permite tener varias variantes de discretización, por lo que se puede seleccionar entre ellas la del hiperplano que separe mayor cantidad de patrones.

Analicemos en detalle este segundo resultado. Tanto en la resolución del problema (P5.3), como en la del (P5.4), se realizan intercambios entre las variables libres y las variables dependientes del sistema (5.6) para obtener todas las variables de una clase y la mayor cantidad de variables de la otra clase con valores positivos. Esto lleva a una matriz de dependencia lineal entre las variables auxiliares, que por comodidad de notación, se supone que tiene la misma forma que (5.7).

$$\begin{bmatrix} a_{n+2,1} & a_{n+2,2} & \cdots & a_{n+2,n+1} & 1 & 0 & \cdots & 0 \\ a_{n+3,1} & a_{n+3,2} & \cdots & a_{n+3,n+1} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{T,1} & a_{T,2} & \cdots & a_{T,n+1} & 0 & 0 & \cdots & 1 \end{bmatrix} \quad (5.8)$$

En esta matriz, cada fila está asociada a una variable dependiente y la asociación se determina a partir de las componentes $u(i)$, ($i = n + 2, \dots, T$) del vector de índices $U = (u(1), u(2), \dots, u(T))$, definido en el capítulo 3.

Analicemos primero, el problema (P5.3). Cuando se le aplica a este problema el procedimiento 3.8.1 del apéndice A, se obtiene una columna $u(s)$ en la matriz (5.8) correspondiente a una variable libre de la clase 1, que tiene coeficientes negativos en las posiciones asociadas a todas las variables dependientes de la clase 1 y en la mayor cantidad de posiciones asociadas a variables dependientes de la clase 2.

Si ahora, se le asignan valores a las variables libres, de la forma descrita en el procedimiento 3.9 del apéndice A, se obtiene un hiperplano separador.

Para obtener otras ecuaciones diferentes de hiperplanos separadores que dividan al conjunto de patrones en los mismos subconjuntos, se puede variar la forma de asignar valores a las variables libres.

Como el objetivo es que se mantengan bien clasificados los mismos patrones, se determinarán los posibles intercambios entre variables libres y variables dependientes que mantengan los mismos coeficientes negativos en la columna asociada a $z_{u(s)}$. Para ello se modifica en los procedimientos 3.5.1 y 3.6.1 la condición { si $l > \max$, entonces $\max \leftarrow l$, $k \leftarrow i$ } por la condición { si $l \geq \max$, entonces $\max \leftarrow l$, $k \leftarrow i$ }. Estos procedimientos modificados se encuentran desarrollados en el Apéndice B.

Ahora, si se le aplican a la matriz (5.8) estos procedimientos modificados, es posible encontrar otras combinaciones para las variables libres que mantienen bien clasificados a los mismos patrones.

En el caso del problema (P5.4), se procede de manera similar, con la diferencia de que las condiciones se modifican en los procedimientos 3.5.1 y 3.6.1.

Estos procedimientos proporciona diferentes matrices de dependencia lineal y si se le aplica el procedimiento 3.9 para el cálculo de los valores de las variables auxiliares, descrito en el apéndice A, se obtienen diferentes hiperplanos separadores, que dividen al conjunto de patrones en los mismos subconjuntos.

Ahora para el proceso de discretización de los pesos, primero, se realiza un escalamiento de los pesos de forma tal que $|w_j| \leq 3$. Esto se hace en cada hiperplano

separador, seleccionando $w = \max_{1 \leq j \leq n} |w_j|$ y multiplicando los pesos y el umbral por $\frac{3}{w}$.

Luego se aproximan los pesos al valor entero más cercano.

Al discretizar los pesos, se está cambiando la posición del hiperplano y esto puede llevar a un empeoramiento de la solución, es decir, que patrones que estaban bien clasificados, ya no lo estén.

Aquí pueden darse las siguientes dos situaciones:

1. Que en los dos subconjuntos existan patrones de las dos clases.
2. Que disminuyó la cantidad de elementos en el subconjunto que contenía patrones de una sola clase.

En la primera situación, hay que mover el hiperplano separador hasta que se logre que en uno de los subconjuntos solo existan patrones de una misma clase.

En el caso, cuando el hiperplano se obtuvo mediante la discretización de una solución del problema (P5.3), se calculan, a partir del sistema de ecuaciones (5.4), los valores de las variables auxiliares correspondientes a los patrones de la clase 1, se selecciona el menor valor negativo

$$z_k = \min\{z_i : z_i \leq 0, i = 1, \dots, T_1\},$$

y se le asigna al umbral el valor de

$$w_{n+1}^k = w_1 x_{k1} + w_2 x_{k2} + \dots + w_n x_{kn}.$$

Con este nuevo valor de umbral se calculan los valores de las variables auxiliares correspondientes a patrones de la clase 2, se selecciona el menor valor positivo

$$z_l = \min\{z_i : z_i > 0, i = M+1, \dots, T\}$$

y se le asigna al umbral el valor de

$$w_{n+1}^l = w_1 x_{l1} + w_2 x_{l2} + \dots + w_n x_{ln}.$$

Ahora, se determina

$$w_{n+1}^* = \frac{w_{n+1}^k + w_{n+1}^l}{2}$$

Cuando el hiperplano separador se obtiene de una discretización de una solución del problema (P5.4), se realiza el mismo procedimiento, con la única diferencia que se empieza por los patrones mal clasificados de la clase 2.

Con esta forma de determinar el umbral, se está desplazando el hiperplano separador en la dirección de su vector normal, de forma tal que en un semiespacio estén todos los patrones de una clase y en el otro semiespacio se encuentre la mayor cantidad de patrones de la otra clase.

En la segunda situación, se debe verificar si se puede aumentar la cantidad de elementos en el subconjunto que contiene patrones de una sola clase. Para ello, se determina en el conjunto que contiene patrones de las dos clases el menor valor positivo de las variables auxiliares z_i . Supóngase que este valor se alcanza para la variable z_k , entonces si existen variables con valores negativos que cumplan que $z_i > -z_k$, se puede aumentar la cantidad de elementos en el conjunto que contiene patrones de una sola clase.

Para agregar los patrones correspondientes a estas variables auxiliares, primero, se determina

$$z_i = \min\{z_i : -z_k < z_i \leq 0\}$$

y se calcula

$$w_{n+1}^l = w_1 x_{l1} + w_2 x_{l2} + \dots + w_n x_{ln}.$$

$$w_{n+1}^k = w_1 x_{k1} + w_2 x_{k2} + \dots + w_n x_{kn}.$$

Luego, se determina

$$w_{n+1}^* = \frac{w_{n+1}^k + w_{n+1}^l}{2}.$$

Con este valor de umbral, se logra pasar todos los patrones mal clasificados,

que cumplieran con la condición $-z_k < z_i \leq 0$, al conjunto que contiene patrones de una sola clase.

Todo este proceso de discretización de los pesos y del ajuste del valor de umbral, se resumen en los procedimientos 5.3.1 y 5.3.2, desarrollados en el apéndice B.

Con estos procedimientos de discretización, es posible desarrollar un procedimiento que determine, en cada iteración, un hiperplano separador con pesos enteros que clasifica correctamente la mayor cantidad de patrones.

Este hiperplano separa al conjunto de patrones en dos subconjuntos, uno contiene patrones de una sola clase y el otro patrones de las dos clases, por lo que a este último se le aplica el procedimiento descrito. El procedimiento termina cuando se encuentre un subconjunto linealmente separable.

Al terminar el procedimiento se obtiene un conjunto de m hiperplanos separadores, con los cuales podemos formar una red neuronal, como la que se muestra en la figura 23.

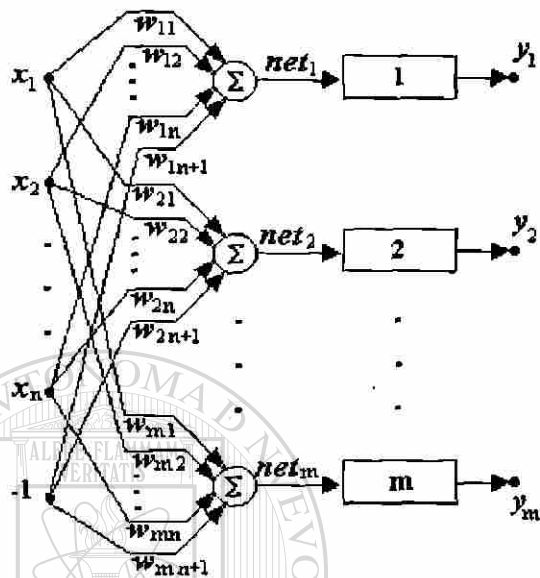


Figura 23 : Red neuronal transformadora a un conjunto linealmente separable.

Si ahora, se le presentan a esta red todos los patrones de entrenamiento

$X_i = (x_{i1}, x_{i2}, \dots, x_{in})' \in R^n$, $(i = 1, \dots, T)$, se obtiene que con las imágenes

$Y_i = (y_{i1}, y_{i2}, \dots, y_{im})' \in R^m$, $(i = 1, \dots, T)$ se puede formar un conjunto de entrenamiento $\{(Y_1, d_1), (Y_2, d_2), \dots, (Y_T, d_T)\}$ que es linealmente separable.

Si a este nuevo conjunto se le aplica el procedimiento descrito anteriormente, se obtiene un hiperplano separador con pesos enteros. Con este último hiperplano se puede completar la red neuronal con pesos enteros, como la que se muestra en la figura 24, que clasifica correctamente al conjunto original de patrones en las dos clases o categorías.

Para resolver el problema de entrenamiento con funciones continuas de activación en las neuronas de la red, se utiliza el procedimiento 4.1, desarrollado en el capítulo 4, para redes con pesos reales.

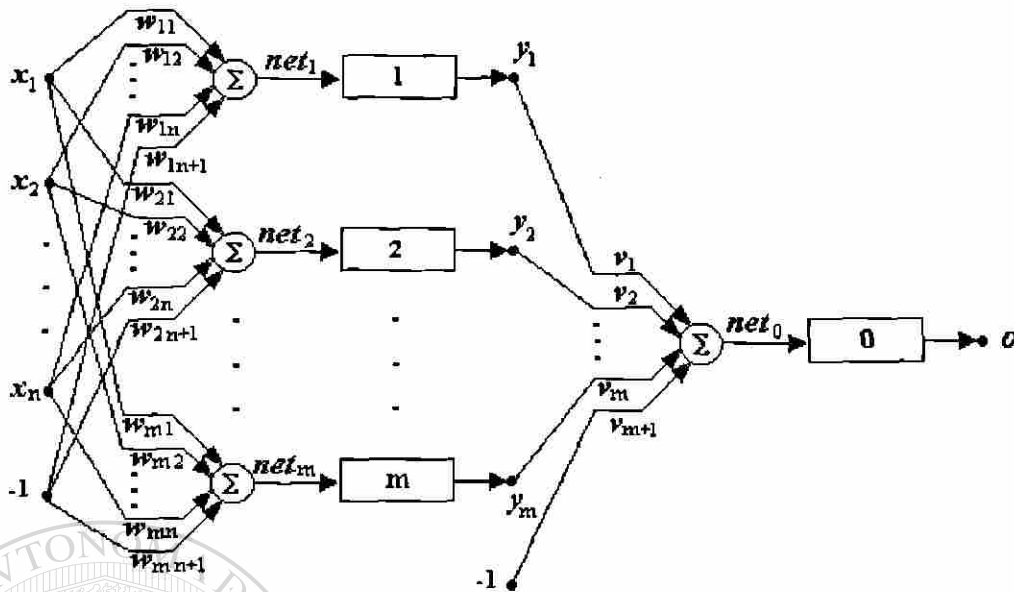


Figura 24 : Red neuronal de memoria entera con funciones discretas de activación.

5.5 Resumen del algoritmo de solución para el problema de diseño.

Procedimiento 5.4: Obtención de una memoria entera entre la capa de entrada y la capa oculta.

Paso 0 : Inicializar $m = 0$.

Paso 1 : En el conjunto de entrenamiento, colocar primero patrones de la clase 1.

Paso 2 : Formar el sistema de ecuaciones (5.4).

Paso 3 : Obtener el sistema de ecuaciones (5.5).

Paso 4 : Extraer la matriz de dependencia lineal (5.7).

Paso 5 : Aplicar procedimiento 3.8.1.

Paso 6 : Inicializar $r = 1$, $p = 0$.

Paso 7 : Aplicar procedimiento 3.9

Paso 8 : Calcular w_j^m , ($j = 1, \dots, n+1$), partir de la primera expresión del sistema de ecuaciones (5.5)

Paso 9 : Aplicar procedimiento 5.3.1.

Si $p_3 = p_1$,

- Asignar $p \leftarrow p_3$

- Asignar $\hat{W}^m \leftarrow W^m$
- ir al paso 12.

En caso contrario, si $p_3 > p$

- asignar $p \leftarrow p_3$
- asignar $\hat{W}^m \leftarrow W^m$
- ir al paso 10.

Paso 10 : Si $r = s$, asignar $r \leftarrow r + 1$.

Paso 11 : Si $r \leq n + 1$,

- Asignar $k \leftarrow 0$
- Aplicar procedimiento 5.1.1

• Si $k \neq 0$, entonces

- Aplicar procedimiento 3.3
- $r \leftarrow r + 1$
- Ir al paso 7

en caso contrario ($k = 0$),

- Aplicar procedimiento 5.2.1
- Si $k \neq 0$, entonces

- Aplicar procedimiento 3.3

- $r \leftarrow r + 1$

- Ir al paso 7

en caso contrario ($k = 0$),

- $r \leftarrow r + 1$

- Ir al paso 10

en caso contrario ($r > n + 1$), ir al paso 12.

Paso 12 : Aplicar procedimiento 3.8.2. Si $p \geq p_2$, ir al paso 19.

Paso 13 : Inicializar $r = 1$.

Paso 14 : Aplicar procedimiento 3.9

Paso 15 : Calcular $w_j^m, (j = 1, \dots, n + 1)$, partir de la primera expresión del sistema de ecuaciones (5.5)

Paso 16 : Aplicar procedimiento 5.3.2.

Si $p_4 > p$,

- Asignar $p \leftarrow p_4$
- Asignar $\hat{W}^m \leftarrow W^m$
- ir al paso 17.

Paso 17 : Si $r = s$, asignar $r \leftarrow r + 1$.

Paso 18 : Si $r \leq n + 1$,

- Asignar $k \leftarrow 0$
- Aplicar procedimiento 5.1.2
- Si $k \neq 0$, entonces
 - Aplicar procedimiento 3.3
 - $r \leftarrow r + 1$
 - Ir al paso 14

en caso contrario ($k = 0$),

- Aplicar procedimiento 5.2.2
- Si $k \neq 0$, entonces
 - Aplicar procedimiento 3.3

• $r \leftarrow r + 1$

• Ir al paso 14

en caso contrario ($k = 0$),

• $r \leftarrow r + 1$

• Ir al paso 17

en caso contrario ($r > n + 1$), ir al paso 19.

Paso 19 : Determinar si un subconjunto contiene patrones de las dos clases.

- Inicializar $k_2 = 0$.
- Si $p = p_3$,
 - Para cada valor de i que cumpla $\tilde{\alpha}_{i,u(s)} < 0$, $u(i) > T_1$
 - Incrementar $k_2 \leftarrow k_2 + 1$
 - Eliminar el patrón X_i .

- Actualizar $T \leftarrow T - p_3$
- Si $p = p_4$
 - Para cada valor de i que cumpla $\tilde{a}_{i,u(s)} < 0$, $u(i) \leq T_1$
 - Incrementar $k_2 \leftarrow k_2 + 1$
 - Eliminar el patrón X_i .
 - Actualizar $T \leftarrow T - p_4$, $T_1 \leftarrow T_1 - p_4$
- Si $k_2 = 0$, entonces parar, en caso contrario,
 - Incrementar $m \leftarrow m + 1$
 - Regresar al paso 1

5.6 Conclusiones.

Con el algoritmo desarrollado en este capítulo son resueltos dos problemas: el diseño y el entrenamiento de una red neuronal con memoria entera.

En la primera fase del algoritmo propuesto, a partir de un análisis de la estructura de los espacios de patrones y de pesos, se desarrolla un procedimiento que permite calcular el número de neuronas en la capa oculta, determinando en cada iteración una neurona con sus parámetros correspondientes. Además, si el conjunto de patrones es linealmente separable el algoritmo para en la primera iteración.

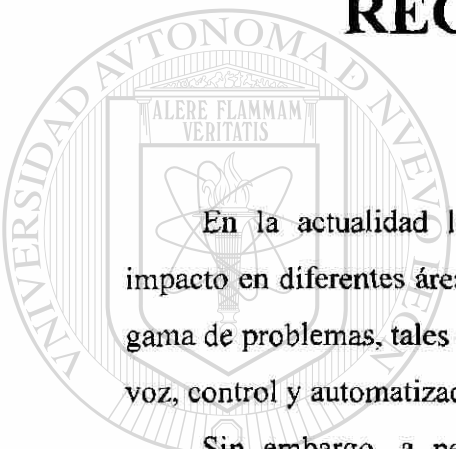
Este procedimiento, además proporciona una memoria de pesos enteros, que corresponde a la solución del problema de clasificación de patrones por una red con funciones discretas de activación en las neuronas.

En la segunda fase del algoritmo, esta memoria es utilizada como la memoria inicial para el entrenamiento de una red que tiene la misma estructura, pero con funciones continuas de activación.

Aquí, en la segunda fase, en lugar de ajustar todos los pesos de la red, se modifica un solo parámetro en cada neurona, lográndose reducir considerablemente la cantidad de operaciones en el proceso de entrenamiento, en comparación con otros algoritmos que se basan en el algoritmo de retropropagación del error.

CAPITULO 6

CONCLUSIONES Y RECOMENDACIONES



En la actualidad los modelos de redes neuronales artificiales tienen gran impacto en diferentes áreas, debido a que su aplicación permite resolver una variada gama de problemas, tales como clasificación de patrones, reconocimiento de formas y voz, control y automatización de procesos, filtrado de señales, etc.

Sin embargo, a pesar de los logros obtenidos, aun existen problemas no resueltos, relacionados con el diseño, aprendizaje y funcionamiento de las redes neuronales.

Este trabajo resuelve algunos problemas que surgen en el diseño y aprendizaje de redes neuronales para la clasificación.

Cuando se tratan de implementar modelos de redes neuronales para la clasificación, generalmente el conjunto de entrenamiento no es linealmente separable, por lo que es necesario utilizar una capa oculta de neuronas. Determinar la cantidad de neuronas en esta capa oculta es un problema muy complejo y en la mayoría de los casos se determina por ensayo y error. De aquí, la importancia de desarrollar algoritmos que determinen la cantidad de neuronas en la capa oculta.

Otro problema que presentan los modelos de redes neuronales es que el proceso de entrenamiento resulta muy costoso desde el punto de vista computacional. Es

conocido que, en el caso del problema de filtrado de señales, se realiza una discretización de las señales, transformando el problema a uno de clasificación, pero es necesario utilizar funciones continuas de activación en las neuronas de la red. De aquí, la importancia del algoritmo desarrollado para entrenar una red con funciones continuas de activación, que realiza un número de operaciones muy inferior al de las diferentes variantes del algoritmo de retropropagación del error.

Cuando se utilizan modelos de redes neuronales para resolver determinado problema, es deseable que la memoria de la red este formada por valores enteros pequeños, ya que las diferentes formas de implementación resultan muy económicas. En esto radica la importancia de desarrollar algoritmos de entrenamiento que permitan obtener una red con memoria entera.

En la realización de este trabajo de investigación, aportamos algunos resultados a la Teoría de las Redes Neuronales, pero aún existen muchos problemas por resolver en este ámbito, como por ejemplo, generalizar los resultados obtenidos para redes clasificadoras en varias categorías, desarrollar procedimientos de post-procesamiento que disminuyan la cantidad de neuronas en la capa oculta y aplicar los algoritmos desarrollados a la solución de problemas reales. Todo esto puede ser objeto de estudio para futuros trabajos de investigación.

BIBLIOGRAFIA

1. Aguilar, M., Martínez, J. y Alvarez, A. Un Algoritmo de entrenamiento para Redes Neuronales con pesos enteros. Universidad Autónoma de Nuevo León, México, (1997).
2. Anderson, J. Introduction to Practical Neural Modeling. Cambridge, MA: MIT Press, (1994).
3. Attali, J. y Pages, G. Approximations of functions by a multilayer perceptron - a new approach. Neural Networks vol. 10 No. 6: 1069-1081, (1997).
4. Baldi, P. Gradient Descent Learning Algorithm Overview: A General Dynamical Systems Perspective. IEEE Transactions on Neural Networks, vol. 6, No. 1, (1995).
5. Baldi, P. y Hornik, K. Learning in linear neural networks: A survey. IEEE Transactions on Neural Networks, NN-6(4):837-858, (1995).
6. Barlett, E. Dynamic Node Architecture Learning: An Information Theoretic Approach. Neural Network, Vol. 7, No. 1: 129-140, (1994).
7. Bianchini, M. Gori y M. Optimal Learning in Artificial Neural Networks: a Theoretical View. Dipartimento di Sistemi e Informatica, Università di Firenze, (1996).
8. Bianchini, M., Frasconi, P. y Gori, M. Learning in Multilayer Networks used as Autoassociators. IEEE Transactions on Neural Networks, vol. 6, No.1, (1995).
9. Brandt, R. y Lin, F. Supervised Learning in Neural Networks without Feedback Networks. Department of Electrical and Computer Engineering, Wayne State University, Detroit, (1997).
10. Campbell, C. Constructive Learning Techniques for Designing Neural Network Systems. Advanced Computing Research Centre, University of Bristol, England, (1997).

11. Campbell, C. y Pérez, C. Constructing Feedforward Neural Networks for Binary Classification Tasks. In European Symposium on Artificial Neural Networks, D. Facto Publications, Brussels: 241-246, (1995).
12. Campbell, C. y Coombes, S. Determining the optimal number of hidden nodes in a Feedforward Neural Network. Department of Engineering Mathematics, Bristol University, (1996).
13. Costa, R. y Larzabal, P. Inicialization of Supervised Training for Parametric Estimation. Neural Processing Letters vol. 9: 53-61, (1999).
14. Crespin, D. Generalized Backpropagation. Facultad de Ciencias, Universidad Central de Venezuela, (1995).
15. Cybenko, G. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems, 2: 303-314, (1989).
16. Chieueh, T. y Goodman, R. Learnig Algorithms for Neural Networks with Ternary Weights. Neural network, Vol, 1: 166-172, (1988).
17. Dasgupta, B. y Sontag, E. Sample Complexity for Learning Recurrent Perceptron Mappings. DOMACS Technical Report 95-17, (1995).
18. Dasgupta, B., Siegelma, H. y Sontag, E. On the complexity of training Neural Networks with continuous activation functions. IEEE Transactions on Neural Networks, vol. 6: 1490-1504, (1995).
19. Ergezinger, S. An Accelerated Learning Algorithm for Multilayer Perceptrons: Optimization Layer by Layer. IEEE Tansactions on Neural Networks, vol. 6, No. 1, (1995).
20. Vecci, L., Piazza, F. y Uncini, A. Learning and approximation capabilities of adaptive spline activation function neural networks. Neural Networks vol. 11 No. 2: 259-270, (1998).
21. Fiesler, E. Connectivity Maximization of Layered Neural Networks for Supervised learning. Proceeding of the International Conference on Artificial Neural Networks: 514, (1993).
22. Fiesler, E., Choudry, A. y Caulfield, H. A Weight Discretization Paradigm for Optical Neural Networks. Proceedings of the International Congress on Optical Science and Engineering, vol. SPIE-1281: 164-173, (1990).
23. Fujita, O. Statistical estimation of the number of hidden units for feedforward neural networks. Neural Networks vol. 11 No. 5: 851-859, (1998).

24. Fukushima, K. Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics* 36:193-202, (1980).
 25. Funahashi, K. On the approximate realization of continuous mapping by neural networks. *Neural Networks* 2: 183-192, (1989).
 26. Gallant, S. Perceptron based learning algorithms. *IEEE Transactions on Neural Networks*, 1 (2): 179-191, (1990).
 27. Gori, M. y Tesi, A. On the problem of local minima in backpropagation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **PAMI-14**: 76-86, (1992).
 28. Gori, M. y Maggini, M. Optimal Convergence of on-line Backpropagation. Dipartimento di Sistemi e Informatica, Università di Firenze, (1994).
 29. Haykin, S. *Neural Networks: A Comprehensive Foundation*. Prentice Hall. 2nd Edition. (1998).
 30. Hlavackova, K. Dependence of the rate of approximation in a feedforward network on its activation function. *Neural Network World*, 2: 163-171, (1996).
 31. Hlaváčková, K. Y Verleysen, M. Synthesis of Feedforward Neural Networks using Splines for Approximation of Functions. Institute of Computer Science, Academy of Science of Czech Republic, Technical Report No. 651, (1995).
-
32. Hopfield, J. Neural Networks and Physical Systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the U.S.A.* 79: 2554-2558, (1982).
 33. Hornik, K., Stinchcombe, M. y White, H. Multilayer feedforward network are universal approximators. *Neural Networks*, 2: 359-366, (1989).
 34. Hornik, K., Stinchcombe, M., White, H. y Auer, P. Degree of Approximation Results for Feedforward Networks Approximating Unknown Mappings and Their Derivatives. *Neural Computation*, vol. 6: 1262-1275, (1994).
 35. Hornik, K. Some new results on neural network approximation. *Neural Networks*, vol. 6: 1073-1087, (1993).
 36. Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, vol. 4: 251-257, (1991).

37. Hornik, K. Functional learning and approximation in artificial neural networks. *Neural Network World*, vol. 1: 257-266, (1991).
 38. Hornik, K. Some new results on neural network approximation. *Neural Networks*, vol. 6: 1069-1072, (1993).
 39. Huang, Sh. y Huang, Y. Bounds on the number of hidden neurons in multilayer perceptrons. *IEEE Transactions on Neural Networks*, vol. 2, No.1, pp. 47-55, (1991).
 40. Humpert, B. Improving backpropagation with a new error function. *Neural Networks* vol. 7 No. 8: 1191-1192, (1994).
 41. Ito, Y. Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory. *Neural Networks*, vol. 4: 385-394, (1991).
 42. Ito, Y. Approximation Capability of Layered Neural Networks with Sigmoid Units on Two Layers. *Neural Computation*, vol. 6: 1233-1243, (1994).
 43. Kainen, P., Kurková, V. y Vogt, A. Approximation by Neural Networks is not Continuous. *Neurocomputing*, 29: 45-56, (1999).
 44. Karnin, E. Simple procedure for pruning Backpropagation Trained Neural Networks. *IEEE Tansaction on Neural Networks*, vol. 1, No. 2 , (1990).
 45. Katsuura, H. y Sprecher, D. Comutational Aspects of Kolmogorov's Superposition Theorem. *Neural Networks*, vol. 7: 455-461, (1994).
-
46. Khan, A. Feedforward Neural Networks with Constrained Weights. Thesis for the degree of doctor of Philosophy, University of Warwick, Department of Engineering, (1996).
 47. Khan, A. Multiplier-free Feedforward Neural Network is a Universal Approximator in $C(R)$. *Proceedings of the 2nd IEEE National Multi-topic Conference (INMIC'97)*, Islamabad, Pakistan: 33-36, (1997).
 48. Khan, A. y Hines, E. Integer-Weight Neural Net. *Electron. Letters*, vol. 30, No.15: 1237-1238, (1994).
 49. Khan, A. y Wilson, R. Integer-Weight Approximation of Continuous-Weight Multilayer Feedforward Net. *Proceedings of the IEEE International Conference on Neural Networks*, Washington, DC, (1996).
 50. Kohonen, T. *Self-Organization and Associative Memory*. New York: Springer-Verlag. (1988).

51. Kohonen, T. Self-organized formation of topologically correct feature maps. *Biological Cybernetics* 43: 59-69, (1982).
 52. Koiran, P. On the complexity of approximating mapping using feedforward networks. *Neural Networks*, vol. 6: 649-653, (1993).
 53. Kurkova, V. y Kainen, P. Equivalent weight vectors in perceptron type networks. *Neural Network World*, vol. 6: 685-692, (1992).
 54. Kurkova, V., Kainen, P. y Kreinovich, V. Dimension-independent rates of approximation by Neural Networks and variation with respect to half-spaces. In *WC Neural Networks*, vol. 1: 54-57, (1995).
 55. Kurkova, V. Kainen, P. y Kreinovich, V. Estimates of the number of hidden units and variation with respect to half-spaces. *Neural Networks* vol. 10, No. 6: 1061- 1068, (1997).
 56. Kurkova, V. Kolmogorov's theorem is relevant. *Neural Computation*, vol. 3: 617-622, (1991).
 57. Kurkova, V. Are sigmoidals the best activation functions in multilayer feedforward networks?. *Neural Network World*, vol. 1: 27-34, (1992).
 58. Kurkova, V. Kolmogorov's theorem and multilayer Neural Networks. *Neural Networks*, vol. 5: 501-506, (1992).
 59. Kurkova, V. Optimal weight spaces for feedforward neural network. *IEEE Workshop CMP: 10-14*, (1994).
-
60. Kurkova, V. Trade-off between the size of parameters and the number of units in one-hidden-layer networks. *Neural Network World*, vol. 2: 191-200, (1996).
 61. Kurkova, V. Aproximation of functions by perceptron networks with bounded number of hidden units. *Neural Networks*, vol. 8: 745-750, (1995).
 62. Kurkova, V. Trade-off between the size of the weights and the number of hidden units in feedforward networks. Institute of Computer Science, Academy of Science of Czech Republic, Technical Report No. V-965, (1996).
 63. Kwok, T. y Yeung, D. Efficient Cross-Validation for Feedforward Neural Networks. Department of Computer Science, Hong Kong University of Science and Technology, (1995).

64. Lange, F. Fast and accurate training of multilayer Perceptrons using extended Kalman Filtr. Institute for Robotics and Systems Dynamics, DLR, Germany, (1995).
65. Lawrence, S. y Chung Tsoi, A. A Back Function Approximation with Neural Networks and Local Methods: Bias, Variance and Smmothness. Australian Conference on Neural Networks, ACNN'96: 16-21, (1996).
66. Lawrence, S., Lee Giles, C. y Chung Tsoi, A. Lessons in Neural Network Training : Overfitting may be hader than expected. Proceedings of the XIV National Conference on Artificial Intelligence, California: 540-545, (1997).
67. LeCun , Y. Une procedure d'apprentissage pour reseau a seuil assymetrique. *Cognitiva* 85: 599-604, (1985).
68. Leisch, F. y Hornik, K. Error-Dependent Resampling for Artificial Neural Network Classifiers. In Bernd Kröplin, editor, *Neuronale Nerze in den Ingenieurwissenschaften*: 1-14, (1997).
69. Manry, M., Apollo, S. y Yu, Q. Minimum Mean Square Stimation and Neural Networks. *Neurocomputing*, vol 13: 59-74, (1996).
70. Mayoraz, E. On the Complexity of Recognizing Regions Computable by two-Layered Perceptrons . *Annals of Mathematics and Artificial Intelligence* 24: 129-153, (1998).
71. Mayoraz, E. y Aviolat, F. Constructive Training Methods for Feedforward Neural Networks with Binary Weights. *International Journal of Neural Systems*, vol. 7, No. 2, (1996).
72. Mayoraz, E. y Robert, V. Maximizing the Robustness of a Linear Threshold Classifier with Discrete Weights. *Network: Computation in Neural Systems*, vol. 5: 299-315, (1994).
73. Mehrotra, K., Mohan, C. y Ranka, S. Bounds on the number of samples needed for neural learning. *IEEE Transactions on Neural Networks*, 2(6): 548-558, (1991).
74. Mhaskar, H. y Micchelli, C. Degree of approximation by neural and translation networks with a single hidden layer. *Advances in Applied Mathematics*, vol. 16: 151-183, (1995).
75. Mhaskar, H. Neural Networks for optimal approximation of smooth and analytic functions. *Neural Computation*, vol. 8: 164-177, (1995).

76. Minsky, M. y Papert, S. Perceptrons. Expanded Edition, Cambridge, MA: MIT Press, (1988).
77. Moerland, P., Fiesler, E. y Saxena, I. Discrete All-Positive Multilayer Perceptrons for Optical Implementation. Optical Engineering, v10.37, No. 4, (1998).
78. Neruda, R. y Stedry, A. Cascade Networks: Another Approach to Function Approximation. Institute of Computer Science, Academy of Science of Czech Republic, Technical Report No. V-697, (1997).
79. Parekh, R., Yang, J. y Honavar, V. Constructive neural network learning algorithms for multi-category pattern classification. Technical Report TR95-15, Artificial Research Group, Department of Computer Science, Iowa State University, U.S.A., (1995).
80. Parker, D. Learning-logic: Casting the cortex of the human brain in silicon. Technical Report TR-47. Center for Computational Research in Economics and Management Science, MIT, Cambridge, MA, (1985).
81. Peantionis, S. y Karras, D. Analysis of Backpropagation Algorithm with momentum. IEEE Transaction on Neural Networks, Vol. 5, No. 3: 505-506, (1994).
82. Rumelhart, D., Hinton, G. y Williams, R. Learning Representations by Back-propagating errors. Nature 323: 533-536, (1986).
83. Saarinen, S., Bramley, R. y Cybenko, G. Ill-conditioning in neural network training problems. SIAM Journal on Scientific Computing, vol. 14, No. 3: 693-714, (1993).
84. Saxena, I., Fiesler, E. y Moerland, P. A Method for All-Positive Optical Multilayer Perceptrons. In Proceeding of the Third IEEE International Conference on Electronics, Circuits and Systems (ICECS'96), vol. 1, IEEE, Piscataway, NJ:448-451, (1996).
85. Scarselli, F. y Tsoi, A. Universal approximation using feedforward neural networks - a survey of some existing methods, and some new results. Neural Networks vol. 11 No. 1: 15-37, (1998).
86. Schiffmann, W. y Westphal, G. Training Multilayer Feedforward Networks with Limited Numerical Precision and Integer Arithmetic. Institute of Physics, University of Koblenz, (1999).

87. Setiono, R. y Liu, H. Understanding Neural Networks via Rule Extraction. Department of System and Computer Science, National University of Singapore, (1996).
88. Setiono, R. y Chi Kwong Hui, L. Use of Quasi-Newton Method in a Feedforward Neural Network Contruccion Algorithm. IEEE Tansactions on Neural Networks, vol. 6, No. 1 , (1995).
89. Síma, J. Back-propagation is not efficient. Neural Networks, vol. 9, No. 6: 1017-1023, (1996).
90. Siu, K-Y., Roychowdhury, V. y Kailath, T. Discrete Neural Computation: A Theoretical Foundation. Prentice-Hall, Englewood Cliffs, NJ, (1995).
91. Sontag, E. Feedbake stabilization using two-hidden-layer nets. IEEE Tans. on Neural Networks, vol. 3: 981-990, (1993).
92. Sontag, E. Feedforward nets for interpolation and classification. Comp. Syst. Sci. Vol. 45: 20-48, (1992).
93. Sontag, E. y Sussmann, H. Backpropagation separates where perceptrons do. Neural Networks, vol. 4: 243-249, (1991).
94. Sontag, E., DasGupta, B. y Siegelmann, H. On the complexity of training neural networks with continuous activation functions. IEEE Trans. Neural Networks vol. 6: 1490-1504, (1995).
95. Sontag, E. y Sussmann, H. Backpropagation can give rise to spurious local minima even for networks without hidden layers. Complex Systems 3: 91-106, (1989).
96. Sprecher, D. A universal mapping for Kolmogorov's superposition theorem. Neural Networks, vol. 6:1089-1094, (1993).
97. Stinchcombe, M. y White, H. Approximating and Learning unknown mappings using multilayers feedforward networks with bounded weights. In Proceedings of IJCNN, IEEE Press, Vol. III, New York: 7-16, (1990).
98. Tamburini, F. y Davoli, R. An algorithmic method to build good training sets for neural network classifiers. Laboratory for Computer Science, University of Bologna, Technical Report UBLCS-94-18, (1994).
99. Topchy, A., Lebedko, O. y Miagkikh, V. Fast Learning in Multilayered Neural Networks by means of Hybrid Evolutionary and Gradient Algorithms. Research Institute for Multiprocessor Computer Systems, Taganrog, Russia, (1995).

100. Torres, J., Gordon, M. Efficient adaptative learning for classification tasks with binary units. Département de Recherche Fondamentale sur la Matière Condensée, France, (1997).
101. Ventura, D. y Martínez, T. An Empirical Comparison of Discretization Methods. Proceedings of the Tenth International Symposium on Computer and Information Science: 443-450, (1995).
102. Wang, Z., Dimassimo, C., Tham, M. y Morris, A. A procedure for determining the topology of multilayer feedforward neural networks. Neural Networks vol. 7 No. 2: 291-300, (1994).
103. Werbos, P. Backpropagation: Past and Future. Proceedings of IEEE International Conference on Neural Networks, IEEE Press, Vol. I, New York: 343-353, (1988).
104. Widrow, B. y Lehr, M. 30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation. Proceedings of IEEE, vol. 76, No. 9: 1415-1442, (1990).
105. Yu, X. Can backpropagation error surface not have local minima?. IEEE Trans. on Neural Networks, vol. 3, No. 6:1019-1021, (1992).
106. Yu, X. y Chen, G. Efficient backpropagation learning using optimal learning rate and momentum. Neural Networks vol. 10 No. 3: 517-527, (1997).
107. Zurada, J. Introduction to Artificial Neural Systems. PWS Publishing Company, (1992).

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

DIRECCIÓN GENERAL DE BIBLIOTECAS



APENDICE A

PROCEDIMIENTOS AUXILIARES PARA EL DISEÑO DE LA RED DE MEMORIA REAL

Los procedimientos 3.1.1, 3.1.2, 3.2.1 y 3.2.2 son para seleccionar elementos pivotes que disminuyen el valor de k_1 .

Estos procedimientos buscan en una columna $u(r)$, ($r \neq s$), el elemento pivote (si es que existe), que transforma a negativos, la mayor cantidad de elementos de la columna $u(s)$. En los procedimientos 3.1.1 y 3.2.1 la columna $u(s)$ está asociada a una variable libre de la clase 1 y en los procedimientos 3.1.2 y 3.2.2 la columna $u(s)$ está asociada a una variable libre de la clase 2.

Procedimiento 3.1.1: *Buscar Pivote 1* (Para el problema P3.3).

Paso 1: Determinar

$$b_i = \begin{cases} 1, & \text{si } a_{i+n+1,u(r)} > 0 \text{ y } a_{i+n+1,u(s)} < 0 \\ -1, & \text{si } a_{i+n+1,u(r)} < 0 \text{ y } a_{i+n+1,u(s)} \geq 0 \\ 0, & \text{en caso contrario} \end{cases}, \quad (i = 1, \dots, T - n - 1)$$

Paso 2: $\forall b_i \neq 0$, calcular

$$c_i = \left| \frac{a_{i+n+1,u(s)}}{a_{i+n+1,u(r)}} \right|.$$

Paso 3 : Inicializar $l_1 \leftarrow 0$.

Paso 4 : Entre los valores de i que cumplen $u(i+n+1) \leq T_1$, $b_i = 1$, determinar

$$c_{m1} = \min\{c_i\}.$$

Paso 5 : Si $c_{m1} < c_i$, $\forall i \neq m1 : u(i+n+1) \leq T_1$, $b_i = 1$, entonces

- Para los valores de j que cumplen $u(j+n+1) \leq T_1$, $c_j < c_{m1}$, $b_j = -1$,

$$\text{determinar } l_2 = \sum_j |b_j|$$

- Si $l_2 > l_1$, entonces $l_1 \leftarrow l_2$, $k \leftarrow m1 + n + 1$

Paso 6 : Para cada valor de i que cumpla $u(i+n+1) > T_1$, $b_i = 1$, $c_i < c_{m1}$, hacer

- Para los valores de j que cumplen $u(j+n+1) \leq T_1$, $c_j < c_i$, $b_j = -1$,

$$\text{determinar } l_2 = \sum_j |b_j|$$

- Si $l_2 > l_1$, entonces $l_1 \leftarrow l_2$, $k \leftarrow i + n + 1$

Paso 7 : Para cada valor de i que cumpla $b_i = -1$, $c_i < c_{m1}$, hacer

- Si $u(i+n+1) \leq T_1$, entonces para los valores de j que cumplen

$$u(j+n+1) \leq T_1, c_j < c_i, b_j = -1, \text{ determinar } l_2 = \sum_j |b_j| + 1$$

- Si $u(i+n+1) > T_1$, entonces para los valores de j que cumplen

$$u(j+n+1) \leq T_1, c_j < c_i, b_j = -1, \text{ determinar } l_2 = \sum_j |b_j|$$

- Si $l_2 > l_1$, entonces $l_1 \leftarrow l_2$, $k \leftarrow i + n + 1$

Procedimiento 3.1.2: *Buscar Pivote 1* (Para el problema P3.4).

Paso 1 : Determinar

$$b_i = \begin{cases} 1, & \text{si } a_{i+n+1,u(r)} > 0 \text{ y } a_{i+n+1,u(s)} < 0 \\ -1, & \text{si } a_{i+n+1,u(r)} < 0 \text{ y } a_{i+n+1,u(s)} \geq 0 \\ 0, & \text{en caso contrario} \end{cases}, \quad (j = 1, \dots, T - n - 1)$$

Paso 2 : $\forall b_i \neq 0$, calcular

$$c_i = \left| \frac{a_{i+n+1, u(s)}}{a_{i+n+1, u(r)}} \right|.$$

Paso 3 : Inicializar $l_1 \leftarrow 0$.

Paso 4 : Entre los valores de i que cumplen $u(i+n+1) > T_1$, $b_i = 1$, determinar

$$c_{m1} = \min\{c_i\}.$$

Paso 5 : Si $c_{m1} < c_i$, $\forall i \neq m1 : u(i+n+1) > T_1$, $b_i = 1$, entonces

- Para los valores de j que cumplen $u(j+n+1) > T_1$, $c_j < c_{m1}$, $b_j = -1$,

$$\text{determinar } l_2 = \sum_j |b_j|$$

- Si $l_2 > l_1$, entonces $l_1 \leftarrow l_2$, $k \leftarrow m1 + n + 1$

Paso 6 : Para cada valor de i que cumpla $u(i+n+1) \leq T_1$, $b_i = 1$, $c_i < c_{m1}$, hacer

- Para los valores de j que cumplen $u(j+n+1) > T_1$, $c_j < c_i$, $b_j = -1$,

$$\text{determinar } l_2 = \sum_j |b_j|$$

- Si $l_2 > l_1$, entonces $l_1 \leftarrow l_2$, $k \leftarrow i + n + 1$

Paso 7 : Para cada valor de i que cumpla $b_i = -1$, $c_i < c_{m1}$, hacer

- Si $u(i+n+1) > T_1$, entonces para los valores de j que cumplen

$$u(j+n+1) > T_1, c_j < c_i, b_j = -1, \text{ determinar } l_2 = \sum_j |b_j| + 1$$

- Si $u(i+n+1) \leq T_1$, entonces para los valores de j que cumplen

$$u(j+n+1) > T_1, c_j < c_i, b_j = -1, \text{ determinar } l_2 = \sum_j |b_j|$$

- Si $l_2 > l_1$, entonces $l_1 \leftarrow l_2$, $k \leftarrow i + n + 1$

Procedimiento 3.2.1: Buscar Pivote 2 (Para el problema P3.3).**Paso 1** : Determinar

$$b_i = \begin{cases} 1, & \text{si } a_{i+n+1,u(r)} > 0 \text{ y } a_{i+n+1,u(s)} \geq 0 \\ -1, & \text{si } a_{i+n+1,u(r)} < 0 \text{ y } a_{i+n+1,u(s)} < 0, \quad (i = 1, \dots, T-n-1) \\ 0, & \text{en caso contrario} \end{cases}$$

Paso 2 : $\forall b_i \neq 0$, calcular

$$c_i = \frac{|a_{i+n+1,u(s)}|}{|a_{i+n+1,u(r)}|}.$$

Paso 3 : Inicializar $l_1 \leftarrow 0$.**Paso 4** : Entre los valores de i que cumplen $u(i+n+1) \leq T_1$, $b_i = -1$, determinar

$$c_m = \min\{c_i\}.$$

Paso 5 : Para cada valor de i que cumpla $b_i = 1$, $c_i < c_m$, hacer

- Para los valores de j que cumplen $u(j+n+1) \leq T_1$, $c_j < c_i$, $b_j = 1$,

$$\text{determinar } l_2 = \sum_j |b_j|$$

- Si $l_2 > l_1$, entonces $l_1 \leftarrow l_2$, $k \leftarrow i+n+1$

Paso 6 : Para cada valor de i que cumpla $u(i+n+1) > T_1$, $b_i = -1$, $c_i < c_m$, hacer

- Para los valores de j que cumplen $u(j+n+1) \leq T_1$, $c_j < c_i$, $b_j = 1$,

$$\text{determinar } l_2 = \sum |b_j|$$

- Si $l_2 > l_1$, entonces $l_1 \leftarrow l_2$, $k \leftarrow i+n+1$

Procedimiento 3.2.2: Buscar Pivote 2 (Para el problema P3.4).**Paso 1** : Determinar

$$b_i = \begin{cases} 1, & \text{si } a_{i+n+1,u(r)} > 0 \text{ y } a_{i+n+1,u(s)} \geq 0 \\ -1, & \text{si } a_{i+n+1,u(r)} < 0 \text{ y } a_{i+n+1,u(s)} < 0, \quad (i = 1, \dots, T-n-1) \\ 0, & \text{en caso contrario} \end{cases}$$

Paso 2: $\forall b_i \neq 0$, calcular

$$c_i = \frac{|a_{i+n+1,u(s)}|}{|a_{i+n+1,u(r)}|}.$$

Paso 3: Inicializar $l_1 \leftarrow 0$.

Paso 4: Entre los valores de i que cumplen $u(i+n+1) > T_1$, $b_i = -1$, determinar

$$c_{m1} = \min\{c_i\}.$$

Paso 5: Para cada valor de i que cumpla $b_i = 1$, $c_i < c_{m1}$, hacer

- Para los valores de j que cumplen $u(j+n+1) > T_1$, $c_j < c_i$, $b_j = 1$,

determinar $l_2 = \sum_j |b_j|$

- Si $l_2 > l_1$, entonces $l_1 \leftarrow l_2$, $k \leftarrow i+n+1$

Paso 6: Para cada valor de i que cumpla $u(i+n+1) \leq T_1$, $b_i = -1$, $c_i < c_{m1}$, hacer

- Para los valores de j que cumplen $u(j+n+1) > T_1$, $c_j < c_i$, $b_j = 1$,

determinar $l_2 = \sum_j |b_j|$

- Si $l_2 > l_1$, entonces $l_1 \leftarrow l_2$, $k \leftarrow i+n+1$

El procedimiento 3.3 permite realizar el intercambio entre una variable libre $z_{u(r)}$ y una variable dependiente $z_{u(k+n+1)}$.

DIRECCIÓN GENERAL DE BIBLIOTECAS

Procedimiento 3.3: *Pivotear.*

Paso 1: Asignar

- $a_1 \leftarrow a_{k,u(r)}$
- $a_{k,j} \leftarrow \frac{a_{k,j}}{a_1}$, $j = 1, \dots, T$

Paso 2: $\forall i \neq k$, asignar

- $a_2 \leftarrow a_{i,u(r)}$
- $a_{i,j} \leftarrow a_{i,j} - a_2 \cdot a_{k,j}$; $j = 1, \dots, T$

Paso 3 : Intercambiar $u(r)$ con $u(k)$.

El proceso de transformación de la columna $u(s)$, para que contenga elementos negativos, asociados a todas las variables dependientes de la clase 1, se puede resumir en el procedimiento 3.4.1 y para que contenga elementos negativos, asociados a todas las variables dependientes de la clase 2, se puede resumir en el procedimiento 3.4.2.

Procedimiento 3.4.1: *Disminuir k_1* (Para el problema P3.3).

Paso 1 : Inicializar $r = 0$.

Paso 2 : Repetir, hasta que $r = n + 1$

- $k = 0$
- $r \leftarrow r + 1$
- Si $r \neq s$, aplicar procedimiento 3.1.1
- Si $k \neq 0$, entonces
 - Aplicar procedimiento 3.3
 - Actualizar $k_1 \leftarrow k_1 - l_1$
 - Si $k_1 = 0$, entonces $r = n + 1$

Paso 3 : Si $k_1 \neq 0$, entonces

- Inicializar $r = 0$
- Repetir, hasta que $r = n + 1$
 - $k = 0$
 - $r \leftarrow r + 1$
 - Si $r \neq s$, entonces aplicar procedimiento 3.2.1
 - Si $k \neq 0$, entonces
 - Aplicar procedimiento 3.3
 - Actualizar $k_1 \leftarrow k_1 - l_1$
 - Si $k_1 = 0$, entonces $r = n + 1$

Paso 4 : Si $k_1 \neq 0$, entonces ir al paso 1, en caso contrario parar.

Procedimiento 3.4.2: *Disminuir k_1* (Para el problema P3.4).

Paso 1 : Inicializar $r = 0$.

Paso 2 : Repetir, hasta que $r = n + 1$

- $k = 0$
- $r \leftarrow r + 1$
- Si $r \neq s$, aplicar procedimiento 3.1.2
- Si $k \neq 0$, entonces
 - Aplicar procedimiento 3.3
 - Actualizar $k_1 \leftarrow k_1 - l_1$
 - Si $k_1 = 0$, entonces $r = n + 1$

Paso 3 : Si $k_1 \neq 0$, entonces

- Inicializar $r = 0$
- Repetir, hasta que $r = n + 1$
 - $k = 0$
 - $r \leftarrow r + 1$
 - Si $r \neq s$, entonces aplicar procedimiento 3.2.2
 - Si $k \neq 0$, entonces

- Aplicar procedimiento 3.3

- Actualizar $k_1 \leftarrow k_1 - l_1$
- Si $k_1 = 0$, entonces $r = n + 1$

Paso 4 : Si $k_1 \neq 0$, entonces ir al paso 1, en caso contrario parar.

Los procedimientos 3.5.1, 3.5.2, 3.6.1 y 3.6.2 son modificaciones de los procedimientos 3.1.1, 3.1.2, 3.2.1 y 3.2.2, que se usan para seleccionar elementos pivotes que disminuyen el valor de k_2 . Los procedimientos 3.5.1 y 3.6.1 son para el problema P3.3 y los procedimientos 3.5.2 y 3.6.2 son para el problema P3.4.

Procedimiento 3.5.1: *Buscar Pivote 3* (Para el problema P3.3).

Paso 1 : Determinar

$$b_i = \begin{cases} 1, & \text{si } a_{i+n+1,u(r)} > 0 \text{ y } a_{i+n+1,u(s)} < 0 \\ -1, & \text{si } a_{i+n+1,u(r)} < 0 \text{ y } a_{i+n+1,u(s)} \geq 0, \quad (i = 1, \dots, T-n-1) \\ 0, & \text{en caso contrario} \end{cases}$$

Paso 2 : $\forall b_i \neq 0$, calcular

$$c_i = \left| \frac{a_{i+n+1,u(s)}}{a_{i+n+1,u(r)}} \right|.$$

Paso 3 : Inicializar $max = 0$.

Paso 4 : Entre los valores de i que cumplen $u(i+n+1) \leq T_1$, $b_i = 1$, determinar

$$c_{m1} = \min\{c_i\}.$$

Paso 5 : Si $c_{m1} < c_i$, $\forall i \neq m1 : u(i+n+1) \leq T_1$, $b_i = 1$, entonces

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_{m1}$, $b_j = -1$, determinar

$$l_1 = \sum_j |b_j|$$

- Para los valores de j que cumplen $c_j \leq c_{m1}$, $b_j = 1$, determinar

$$l_2 = \sum_j b_j$$

- Calcular $l = l_1 - l_2$
- Si $l > max$, entonces $max \leftarrow l$, $k \leftarrow m1 + n + 1$

Paso 6 : Para cada valor de i que cumpla $u(i+n+1) > T_1$, $b_i = 1$, $c_i < c_{m1}$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_i$, $b_j = -1$, determinar

$$l_1 = \sum_j |b_j|$$

- Para los valores de j que cumplen $c_j \leq c_i$, $b_j = 1$, determinar

$$l_2 = \sum_j b_j$$

- Calcular $l = l_1 - l_2$
- Si $l > max$, entonces $max \leftarrow l$, $k \leftarrow i + n + 1$

Paso 7 : Para cada valor de i que cumpla $b_i = -1$, $c_i < c_{m1}$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Entre los valores de j que cumplen $c_j < c_i$, $b_j = -1$, determinar

$$l_1 = \sum_j |b_j| + 1.$$

- Entre los valores de j que cumplen $c_j \leq c_i$, $b_j = 1$, determinar

$$l_2 = \sum_j b_j$$

- Calcular $l = l_1 - l_2$
- Si $l > max$, entonces $max \leftarrow l$, $k \leftarrow i + n + 1$

Procedimiento 3.5.2: *Buscar Pivote 3* (Para el problema P3.4).

Paso 1 : Determinar

$$b_i = \begin{cases} 1, & \text{si } a_{i+n+1,u(r)} > 0 \text{ y } a_{i+n+1,u(s)} < 0 \\ -1, & \text{si } a_{i+n+1,u(r)} < 0 \text{ y } a_{i+n+1,u(s)} \geq 0 \\ 0, & \text{en caso contrario} \end{cases}, \quad (i = 1, \dots, T - n - 1)$$

Paso 2 : $\forall b_i \neq 0$, calcular

$$c_i = \frac{a_{i+n+1,u(s)}}{a_{i+n+1,u(r)}}$$

Paso 3 : Inicializar $max = 0$.

Paso 4 : Entre los valores de i que cumplen $u(i + n + 1) > T_1$, $b_i = 1$, determinar

$$c_{m1} = \min\{c_i\}.$$

Paso 5 : Si $c_{m1} < c_i$, $\forall i \neq m1 : u(i + n + 1) > T_1$, $b_i = 1$, entonces

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_{m1}$, $b_j = -1$, determinar

$$l_1 = \sum_j |b_j|$$

- Para los valores de j que cumplen $c_j \leq c_{m1}$, $b_j = 1$, determinar

$$l_2 = \sum_j b_j$$

- Calcular $l = l_1 - l_2$
- Si $l > max$, entonces $max \leftarrow l$, $k \leftarrow ml + n + 1$

Paso 6: Para cada valor de i que cumpla $u(i + n + 1) \leq T_1$, $b_i = 1$, $c_i < c_{m1}$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_i$, $b_j = -1$, determinar

$$l_1 = \sum_j |b_j|$$

- Para los valores de j que cumplen $c_j \leq c_i$, $b_j = 1$, determinar

$$l_2 = \sum_j b_j$$

- Calcular $l = l_1 - l_2$
- Si $l > max$, entonces $max \leftarrow l$, $k \leftarrow i + n + 1$

Paso 7: Para cada valor de i que cumpla $b_i = -1$, $c_i < c_{m1}$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Entre los valores de j que cumplen $c_j < c_i$, $b_j = -1$, determinar

$$l_1 = \sum_j |b_j| + 1.$$

- Entre los valores de j que cumplen $c_j \leq c_i$, $b_j = 1$, determinar

$$l_2 = \sum_j b_j$$

- Calcular $l = l_1 - l_2$
- Si $l > max$, entonces $max \leftarrow l$, $k \leftarrow i + n + 1$

Procedimiento 3.6.1: *Buscar Pivote 4* (Para el problema P3.3).

Paso 1: Determinar

$$b_i = \begin{cases} 1, & \text{si } a_{i+n+1,u(r)} > 0 \text{ y } a_{i+n+1,u(s)} \geq 0 \\ -1, & \text{si } a_{i+n+1,u(r)} < 0 \text{ y } a_{i+n+1,u(s)} < 0, \quad (i = 1, \dots, T-n-1) \\ 0, & \text{en caso contrario} \end{cases}$$

Paso 2: $\forall b_i \neq 0$, calcular

$$c_i = \frac{|a_{i+n+1,u(s)}|}{|a_{i+n+1,u(r)}|}.$$

Paso 3: Inicializar $max = 0$.

Paso 4: Entre los valores de i que cumplen $u(i+n+1) \leq T_1$, $b_i = -1$, determinar

$$c_{m1} = \min\{c_i\}.$$

Paso 5: Para cada valor de i que cumpla $u(i+n+1) > T_1$, $b_i = 1$, $c_i < c_{m1}$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_i$, $b_j = 1$, determinar

$$l_1 = \sum_j b_j$$

- Para los valores de j que cumplen $c_j \leq c_i$, $b_j = -1$, determinar

$$l_2 = \sum_j |b_j|$$

- Calcular $l = l_1 - l_2$
- Si $l > max$, entonces $max \leftarrow l$, $k \leftarrow i + n + 1$

Paso 6: Para cada valor de i que cumpla $b_i = -1$, $c_i < c_{m1}$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_i$, $b_j = 1$, determinar

$$l_1 = \sum_j b_j.$$

- Para los valores de j que cumplen $c_j \leq c_i$, $b_j = -1$, determinar

$$l_2 = \sum_j |b_j| + 1$$

- Calcular $l = l_1 - l_2$
- Si $l > \max$, entonces $\max \leftarrow l$, $k \leftarrow i + n + 1$

Procedimiento 3.6.2: *Buscar Pivote 4* (Para el problema P3.4).

Paso 1: Determinar

$$b_i = \begin{cases} 1, & \text{si } a_{i+n+1,u(r)} > 0 \text{ y } a_{i+n+1,u(s)} \geq 0 \\ -1, & \text{si } a_{i+n+1,u(r)} < 0 \text{ y } a_{i+n+1,u(s)} < 0, \quad (i = 1, \dots, T - n - 1) \\ 0, & \text{en caso contrario} \end{cases}$$

Paso 2: $\forall b_i \neq 0$, calcular

$$c_i = \left| \frac{a_{i+n+1,u(s)}}{a_{i+n+1,u(r)}} \right|.$$

Paso 3: Inicializar $\max = 0$.

Paso 4: Entre los valores de i que cumplen $u(i + n + 1) > T_1$, $b_i = -1$, determinar

$$c_{m_1} = \min\{c_i\}.$$

Paso 5: Para cada valor de i que cumpla $u(i + n + 1) \leq T_1$, $b_i = 1$, $c_i < c_{m_1}$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_i$, $b_j = 1$, determinar

$$l_1 = \sum_j b_j$$

- Para los valores de j que cumplen $c_j \leq c_i$, $b_j = -1$, determinar

$$l_2 = \sum_j |b_j|$$

- Calcular $l = l_1 - l_2$
- Si $l > \max$, entonces $\max \leftarrow l$, $k \leftarrow i + n + 1$

Paso 6: Para cada valor de i que cumpla $b_i = -1$, $c_i < c_{m_1}$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_i$, $b_j = 1$, determinar

$$l_1 = \sum_j b_j.$$

- Para los valores de j que cumplen $c_j \leq c_i$, $b_j = -1$, determinar

$$l_2 = \sum_j |b_j| + 1$$

- Calcular $l = l_1 - l_2$
- Si $l > \max$, entonces $\max \leftarrow l$, $k \leftarrow i + n + 1$

Para tratar de disminuir, en la columna $u(s)$, la cantidad de elementos no negativos, en el problema P3.3, asociados a variables dependientes de la clase 2, se aplica el procedimiento 3.7.1 y en el problema P3.4, asociados a variables dependientes de la clase 1, se aplica el procedimiento 3.7.2

En estos procedimientos, el criterio de parada esta dado por la condición $\text{contador} = 0$, que se obtiene cuando no existe ningún elemento pivote que disminuya el valor de k_2 .

Procedimiento 3.7.1: Disminuir k_2 (Para el problema P3.3).

Paso 1: Inicializar $r = 0$, $\text{contador} = 0$.

Paso 2: Repetir, hasta que $r = n + 1$

- $k = 0$
- $r \leftarrow r + 1$
- Si $r \neq s$, aplicar procedimiento 3.5.1
- Si $k \neq 0$, entonces
 - Aplicar procedimiento 3.3
 - Actualizar $k_2 \leftarrow k_2 - \max$
 - $\text{contador} \leftarrow \text{contador} + 1$

Paso 3: Inicializar $r = 0$.

Paso 4: Repetir, hasta que $r = n + 1$

- $k = 0$
- $r \leftarrow r + 1$

- Si $r \neq s$, aplicar procedimiento 3.6.1
- Si $k \neq 0$, entonces
 - Aplicar procedimiento 3.3
 - Actualizar $k_2 \leftarrow k_2 - \max$
 - $\text{contador} \leftarrow \text{contador} + 1$

Paso 5: Si $\text{contador} \neq 0$, entonces ir al paso 1, en caso contrario parar.

Procedimiento 3.7.2: Disminuir k_2 (Para el problema P3.4).

Paso 1: Inicializar $r = 0$, $\text{contador} = 0$.

Paso 2: Repetir, hasta que $r = n + 1$

- $k = 0$
- $r \leftarrow r + 1$
- Si $r \neq s$, aplicar procedimiento 3.5.2
- Si $k \neq 0$, entonces
 - Aplicar procedimiento 3.3
 - Actualizar $k_2 \leftarrow k_2 - \max$
 - $\text{contador} \leftarrow \text{contador} + 1$

Paso 3: Inicializar $r = 0$.

Paso 4: Repetir, hasta que $r = n + 1$

- $k = 0$
- $r \leftarrow r + 1$
- Si $r \neq s$, aplicar procedimiento 3.6.2
- Si $k \neq 0$, entonces
 - Aplicar procedimiento 3.3
 - Actualizar $k_2 \leftarrow k_2 - \max$
 - $\text{contador} \leftarrow \text{contador} + 1$

Paso 5: Si $\text{contador} \neq 0$, entonces ir al paso 1, en caso contrario parar.

Utilizando los procedimientos 3.4.1 y 3.7.1 se desarrolla el procedimiento 3.8.1 que selecciona una columna $u(s)$ de la matriz de dependencia lineal y la convierte mediante transformaciones elementales en una columna que contiene todos los elementos negativos para las variables de la clase 1 y la mayor cantidad de coeficientes negativos para las variables de la clase 2.

De forma similar, pero utilizando los procedimientos 3.4.2 y 3.7.2 se desarrolla el procedimiento 3.8.2 que selecciona una columna $u(s)$ de la matriz de dependencia lineal y la convierte mediante transformaciones elementales en una columna que contiene todos los elementos negativos para las variables de la clase 2 y la mayor cantidad de coeficientes negativos para las variables de la clase 1.

Procedimiento 3.8.1: Transformación de la matriz de dependencia lineal

(Para el problema P3.3)

Paso 1: Seleccionar la columna $u(s)$ en la matriz de dependencia lineal

- Entre los valores de j que cumplen $1 \leq j \leq n+1$, $u(j) \leq T_1$ y para todos los valores de i que cumplen $n+2 \leq i \leq T$, $u(i) \leq T_1$, $a_{i,u(j)} < 0$, determinar el valor de s como el índice j para el cual se alcanza

$$\min_j \sum_i \text{sgn}(a_{i,u(j)})$$

Paso 2: Calcular k_1 y k_2

- Inicializar $k_1 = 0$, $k_2 = 0$.
- Para cada valor de i que cumpla $u(i) \leq T_1$, $a(i, u(s)) \geq 0$, asignar $k_1 \leftarrow k_1 + 1$
- Para cada valor de i que cumpla $u(i) > T_1$, $a(i, u(s)) \geq 0$, asignar $k_2 \leftarrow k_2 + 1$

Paso 3: Si $k_1 > 0$, aplicar procedimiento 3.4.1

Paso 4: Si $k_2 > 0$, aplicar procedimiento 3.7.1

Paso 5: Calcular $p_1 = T - T_1 - k_2$

Procedimiento 3.8.2: Transformación de la matriz de dependencia lineal

(Para el problema P3.4)

Paso 1 : Seleccionar la columna $u(s)$ en la matriz de dependencia linealSi $\forall j$, tal que $1 \leq j \leq n+1$ se cumple que $u(j) \leq T_1$, entonces

- Seleccionar en la columna $u(s)$ un elemento $a_{i,u(s)} > 0$ para $u(i) > T_1$
- Asignar $k \leftarrow i$, $r \leftarrow s$
- Aplicar procedimiento 3.3
- Asignar $s \leftarrow k$

en caso contrario

- Entre los valores de j que cumplen $1 \leq j \leq n+1$, $u(j) > T_1$ y para todos los valores de i que cumplen $n+2 \leq i \leq T$, $u(i) > T_1$, $a_{i,u(j)} < 0$, determinar el valor de s como el índice j para el cual se alcanza

$$\min_j \sum_i \text{sgn}(a_{i,u(j)})$$

Paso 2 : Calcular k_1 y k_2

- Inicializar $k_1 = 0$, $k_2 = 0$.

- Para cada valor de i que cumpla $u(i) > T_1$, $a_{i,u(s)} \geq 0$, asignar $k_1 \leftarrow k_1 + 1$

- Para cada valor de i que cumpla $u(i) \leq T_1$, $a_{i,u(s)} \geq 0$, asignar $k_2 \leftarrow k_2 + 1$

Paso 3 : Si $k_1 > 0$, aplicar procedimiento 3.4.2**Paso 4 :** Si $k_2 > 0$, aplicar procedimiento 3.7.2**Paso 5 :** Calcular $p_2 = T_1 - k_2$

El siguiente procedimiento se utiliza para calcular los valores de las variables auxiliares.

Procedimiento 3.9: *Calculo de los valores de las variables auxiliares*

Paso 1 : Asignar a las variables libres $z_{u(s)} = 0$, $z_{u(j)} = 1$, $(j = 1, \dots, n+1, j \neq s)$

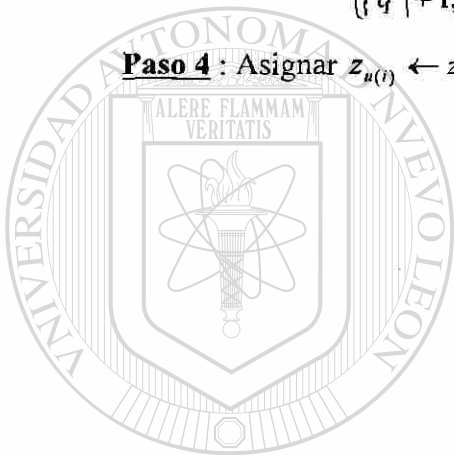
Paso 2 : Calcular los valores de las variables dependientes mediante las relaciones de

$$\text{dependencia lineal } z_{u(i)} = - \sum_{j=1}^{n+1} \tilde{a}_{i,u(j)} z_{u(j)} , (i = n+2, \dots, T) .$$

Paso 3 : $\forall \tilde{a}_{i,u(s)} < 0$ tal que $z_{u(i)} < 0$, calcular $q = \max_i \left\{ \frac{z_{u(i)}}{a_{iu(s)}} \right\}$ y asignar

$$z_{u(s)} \leftarrow \begin{cases} \lceil q \rceil , & \text{si } q \notin \mathbf{N} \\ \lceil q \rceil + 1 , & \text{si } q \in \mathbf{N} \end{cases}$$

Paso 4 : Asignar $z_{u(i)} \leftarrow z_{u(i)} - \tilde{a}_{i,u(s)} z_{u(s)}$, $(i = n+2, \dots, T)$.



UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

DIRECCIÓN GENERAL DE BIBLIOTECAS



APENDICE B

PROCEDIMIENTOS AUXILIARES PARA EL DISEÑO DE LA RED DE MEMORIA ENTERA

Los procedimientos 5.1.1, 5.2.1, 5.1.2 y 5.2.2 permiten determinar elementos pivotes que transforman la matriz de dependencia lineal para obtener diferentes ecuaciones de hiperplanos con coeficientes reales, que separan los mismos patrones.

Procedimiento 5.1.1: *Determinar Pivote 1* (Para el problema P5.3).

Paso 1: Determinar

$$b_i = \begin{cases} 1, & \text{si } a_{i+n+1,u(r)} > 0 \text{ y } a_{i+n+1,u(s)} < 0 \\ -1, & \text{si } a_{i+n+1,u(r)} < 0 \text{ y } a_{i+n+1,u(s)} \geq 0, \quad (i = 1, \dots, T-n-1) \\ 0, & \text{en caso contrario} \end{cases}$$

Paso 2: $\forall b_i \neq 0$, calcular

$$c_i = \frac{|a_{i+n+1,u(s)}|}{|a_{i+n+1,u(r)}|}.$$

Paso 3: Inicializar $max = 0$.

Paso 4: Entre los valores de i que cumplen $u(i+n+1) \leq T_1$, $b_i = 1$, determinar

$$c_{m1} = \min\{c_i\}.$$

Paso 5 : Si $c_{m1} < c_i$, $\forall i \neq m1 : u(i+n+1) \leq T_1$, $b_i = 1$, entonces

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_{m1}$, $b_j = -1$, determinar

$$l_1 = \sum_j |b_j|$$

- Para los valores de j que cumplen $c_j \leq c_{m1}$, $b_j = 1$, determinar

$$l_2 = \sum_j b_j$$

- Calcular $l = l_1 - l_2$
- Si $l \geq \max$, entonces $\max \leftarrow l$, $k \leftarrow m1 + n + 1$

Paso 6 : Para cada valor de i que cumpla $u(i+n+1) > T_1$, $b_i = 1$, $c_i < c_{m1}$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_i$, $b_j = -1$, determinar

$$l_1 = \sum_j |b_j|$$

- Para los valores de j que cumplen $c_j \leq c_i$, $b_j = 1$, determinar

$$l_2 = \sum_j b_j$$

- Calcular $l = l_1 - l_2$
- Si $l \geq \max$, entonces $\max \leftarrow l$, $k \leftarrow i + n + 1$

Paso 7 : Para cada valor de i que cumpla $b_i = -1$, $c_i < c_{m1}$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Entre los valores de j que cumplen $c_j < c_i$, $b_j = -1$, determinar

$$l_1 = \sum_j |b_j| + 1.$$

- Entre los valores de j que cumplen $c_j \leq c_i$, $b_j = 1$, determinar

$$l_2 = \sum_j b_j$$

- Calcular $l = l_1 - l_2$
- Si $l \geq \max$, entonces $\max \leftarrow l$, $k \leftarrow i + n + 1$

Procedimiento 5.2.1: *Determinar Pivote 2 (Para el problema P5.3).*

Paso 1 : Determinar

$$b_i = \begin{cases} 1, & \text{si } a_{i+n+1,u(r)} > 0 \text{ y } a_{i+n+1,u(s)} \geq 0 \\ -1, & \text{si } a_{i+n+1,u(r)} < 0 \text{ y } a_{i+n+1,u(s)} < 0 \\ 0, & \text{en caso contrario} \end{cases}, \quad (i = 1, \dots, T-n-1)$$

Paso 2 : $\forall b_i \neq 0$, calcular

$$c_i = \frac{|a_{i+n+1,u(s)}|}{|a_{i+n+1,u(r)}|}.$$

Paso 3 : Inicializar $max = 0$.

Paso 4 : Entre los valores de i que cumplen $u(i+n+1) \leq T_1$, $b_i = -1$, determinar

$$c_{ml} = \min\{c_i\}.$$

Paso 5 : Para cada valor de i que cumpla $u(i+n+1) > T_1$, $b_i = 1$, $c_i < c_{ml}$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_i$, $b_j = 1$, determinar

$$l_1 = \sum_j b_j$$

- Para los valores de j que cumplen $c_j \leq c_i$, $b_j = -1$, determinar

$$l_2 = \sum_j |b_j|$$

- Calcular $l = l_1 - l_2$
- Si $l \geq max$, entonces $max \leftarrow l$, $k \leftarrow i+n+1$

Paso 6 : Para cada valor de i que cumpla $b_i = -1$, $c_i < c_{ml}$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_i$, $b_j = 1$, determinar

$$l_1 = \sum_j b_j.$$

- Para los valores de j que cumplen $c_j \leq c_i$, $b_j = -1$, determinar

$$l_2 = \sum_j |b_j| + 1$$

- Calcular $l = l_1 - l_2$
- Si $l \geq \max$, entonces $\max \leftarrow l$, $k \leftarrow -i + n + 1$

Procedimiento 5.1.2: Determinar Pivote 1 (Para el problema P5.4).

Paso 1: Determinar

$$b_i = \begin{cases} 1, & \text{si } a_{i+n+1,u(r)} > 0 \text{ y } a_{i+n+1,u(s)} < 0 \\ -1, & \text{si } a_{i+n+1,u(r)} < 0 \text{ y } a_{i+n+1,u(s)} \geq 0 \\ 0, & \text{en caso contrario} \end{cases}, \quad (i = 1, \dots, T - n - 1)$$

Paso 2: $\forall b_i \neq 0$, calcular

$$c_i = \frac{|a_{i+n+1,u(s)}|}{|a_{i+n+1,u(r)}|}$$

Paso 3: Inicializar $\max = 0$.

Paso 4: Entre los valores de i que cumplen $u(i+n+1) > T_1$, $b_i = 1$, determinar

$$c_m = \min\{c_i\}.$$

Paso 5: Si $c_m < c_i$, $\forall i \neq m$: $u(i+n+1) > T_1$, $b_i = 1$, entonces

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_m$, $b_j = -1$, determinar

$$l_1 = \sum_j |b_j|$$

- Para los valores de j que cumplen $c_j \leq c_m$, $b_j = 1$, determinar

$$l_2 = \sum_j b_j$$

- Calcular $l = l_1 - l_2$
- Si $l \geq \max$, entonces $\max \leftarrow l$, $k \leftarrow m + n + 1$

Paso 6: Para cada valor de i que cumpla $u(i+n+1) \leq T_1$, $b_i = 1$, $c_i < c_m$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_i$, $b_j = -1$, determinar

$$l_1 = \sum_j |b_j|$$

- Para los valores de j que cumplen $c_j \leq c_i, b_j = 1$, determinar

$$l_2 = \sum_j b_j$$

- Calcular $l = l_1 - l_2$
- Si $l \geq \max$, entonces $\max \leftarrow l, k \leftarrow i + n + 1$

Paso 7: Para cada valor de i que cumpla $b_i = -1, c_i < c_{m_i}$, hacer

- Inicializar $l_1 = 0, l_2 = 0$
- Entre los valores de j que cumplen $c_j < c_i, b_j = -1$, determinar

$$l_1 = \sum_j |b_j| + 1.$$

- Entre los valores de j que cumplen $c_j \leq c_i, b_j = 1$, determinar

$$l_2 = \sum_j b_j$$

- Calcular $l = l_1 - l_2$
- Si $l \geq \max$, entonces $\max \leftarrow l, k \leftarrow i + n + 1$

Procedimiento 5.2.2: Determinar Pivote 2 (Para el problema P5.4).

Paso 1: Determinar

$$b_i = \begin{cases} 1, & \text{si } a_{i+n+1,u(r)} > 0 \text{ y } a_{i+n+1,u(s)} \geq 0 \\ -1, & \text{si } a_{i+n+1,u(r)} < 0 \text{ y } a_{i+n+1,u(s)} < 0, \quad (i = 1, \dots, T - n - 1) \\ 0, & \text{en caso contrario} \end{cases}$$

Paso 2: $\forall b_i \neq 0$, calcular

$$c_i = \left| \frac{a_{i+n+1,u(s)}}{a_{i+n+1,u(r)}} \right|.$$

Paso 3: Inicializar $\max = 0$.

Paso 4: Entre los valores de i que cumplen $u(i+n+1) > T_1, b_i = -1$, determinar

$$c_{m_i} = \min\{c_i\}.$$

Paso 5: Para cada valor de i que cumpla $u(i+n+1) \leq T_1$, $b_i = 1$, $c_i < c_m$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_i$, $b_j = 1$, determinar

$$l_1 = \sum_j b_j$$

- Para los valores de j que cumplen $c_j \leq c_i$, $b_j = -1$, determinar

$$l_2 = \sum_j |b_j|$$

- Calcular $l = l_1 - l_2$
- Si $l \geq \max$, entonces $\max \leftarrow l$, $k \leftarrow i+n+1$

Paso 6: Para cada valor de i que cumpla $b_i = -1$, $c_i < c_m$, hacer

- Inicializar $l_1 = 0$, $l_2 = 0$
- Para los valores de j que cumplen $c_j < c_i$, $b_j = 1$, determinar

$$l_1 = \sum_j b_j$$

- Para los valores de j que cumplen $c_j \leq c_i$, $b_j = -1$, determinar

$$l_2 = \sum_j |b_j| + 1$$

- Calcular $l = l_1 - l_2$
- Si $l \geq \max$, entonces $\max \leftarrow l$, $k \leftarrow i+n+1$

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

DIRECCIÓN GENERAL DE BIBLIOTECAS

Los procedimientos 5.3.1 y 5.3.2 son para discretizar los pesos y ajustar el valor de umbral en hiperplanos con pesos reales que son solución de los problemas (P.53) y (P5.4) respectivamente.

Procedimiento 5.3.1: *Discretización de los pesos de un hiperplano separador*

(Para el problema P5.3).

Paso 1: Discretización los pesos.

- Seleccionar $w = \max_{1 \leq j \leq n} |w_j|$.

- Asignar $w_j \leftarrow \frac{3w_j}{w}$, ($j = 1, \dots, n+1$).
- Aproximar los pesos al valor entero más cercano para ($j = 1, \dots, n$)
 - Si $\lceil w_j \rceil - w_j \leq 0.5$ entonces $w_j \leftarrow \lceil w_j \rceil$, en caso contrario $w_j \leftarrow \lfloor w_j \rfloor$

Paso 2: Selección del valor de umbral:

- Calcular $z_i = d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1} \right)$, para $i = 1, \dots, T_1$
- Inicializar $p_1 = 0$
- Si existen $z_i \leq 0$, determinar
 - $z_k = \min\{z_i : z_i \leq 0, i = 1, \dots, T_1\}$
 - $w_{n+1}^k = w_1 x_{k1} + w_2 x_{k2} + \dots + w_n x_{kn}$
 - $z_i = d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1}^k \right)$, si $z_i > 0 \Rightarrow p_1 \leftarrow p_1 + 1$,
 $i = T_1 + 1, \dots, T$

$$\bullet z_i = \min\{z_i : z_i > 0, i = T_1 + 1, \dots, T\}$$

$$\bullet w_{n+1}' = w_1 x_{11} + w_2 x_{12} + \dots + w_n x_{1n}$$

- en caso contrario, determinar

$$\bullet z_k = \min\{z_i : z_i > 0, i = 1, \dots, T_1\}$$

$$\bullet z_i = d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1}^k \right), \text{ si } z_i > -z_k \Rightarrow p_1 \leftarrow p_1 + 1,$$

$$i = T_1 + 1, \dots, T$$

$$\bullet z_i = \min\{z_i : -z_k < z_i \leq 0, i = T_1 + 1, \dots, T\}$$

$$\bullet w_{n+1}' = w_1 x_{11} + w_2 x_{12} + \dots + w_n x_{1n}$$

$$\bullet w_{n+1}^k = w_1 x_{k1} + w_2 x_{k2} + \dots + w_n x_{kn}$$

$$\bullet \text{ Calcular } w_{n+1} = \frac{w_{n+1}^k + w_{n+1}'}{2}$$

Procedimiento 5.3.2: *Discretización de los pesos de un hiperplano separador*

(Para el problema P5.4).

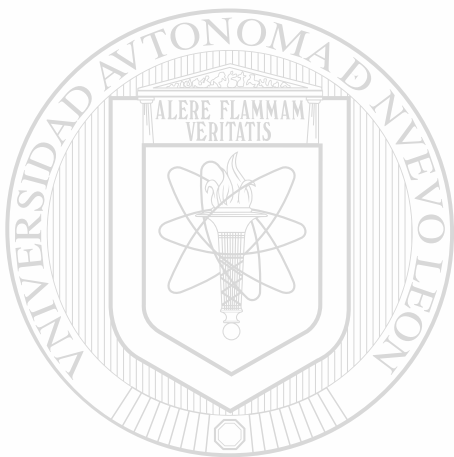
Paso 1: Discretización los pesos.

- Seleccionar $w = \max_{1 \leq j \leq n} |w_j|$.
- Asignar $w_j \leftarrow \frac{3w_j}{w}$, ($j = 1, \dots, n+1$).
- Aproximar los pesos al valor entero más cercano para ($j = 1, \dots, n$)
 - Si $\lceil w_j \rceil - w_j \leq 0.5$ entonces $w_j \leftarrow \lceil w_j \rceil$, en caso contrario $w_j \leftarrow \lfloor w_j \rfloor$

Paso 2: Selección del valor de umbral:

- Calcular $z_i = d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1} \right)$, para $i = T_1 + 1, \dots, T$
- Inicializar $p_2 = 0$
- Si existen $z_i \leq 0$, determinar
 - $z_k = \min\{z_i : z_i \leq 0, i = T_1 + 1, \dots, T\}$
 - $w_{n+1}^k = w_1 x_{k1} + w_2 x_{k2} + \dots + w_n x_{kn}$
 - $z_i = d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1}^k \right)$, si $z_i > 0 \Rightarrow p_2 \leftarrow p_2 + 1$, ($i = 1, \dots, T_1$)
 - $z_l = \min\{z_i : z_i > 0, i = 1, \dots, T_1\}$
 - $w_{n+1}^l = w_1 x_{l1} + w_2 x_{l2} + \dots + w_n x_{ln}$
- en caso contrario, determinar
 - $z_k = \min\{z_i : z_i > 0, i = T_1 + 1, \dots, T\}$
 - $z_i = d_i \left(\sum_{j=1}^n (x_{ij} \cdot w_j) - w_{n+1}^k \right)$, si $z_i > -z_k \Rightarrow p_2 \leftarrow p_2 + 1$, ($i = 1, \dots, T_1$)
 - $z_l = \min\{z_i : -z_k < z_i \leq 0, i = 1, \dots, T_1\}$

- $w_{n+1}^j = w_1 x_{j1} + w_2 x_{j2} + \dots + w_n x_{jn}$
- $w_{n+1}^k = w_1 x_{k1} + w_2 x_{k2} + \dots + w_n x_{kn}$
- Calcular $w_{n+1} = \frac{w_{n+1}^k + w_{n+1}^j}{2}$.



UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

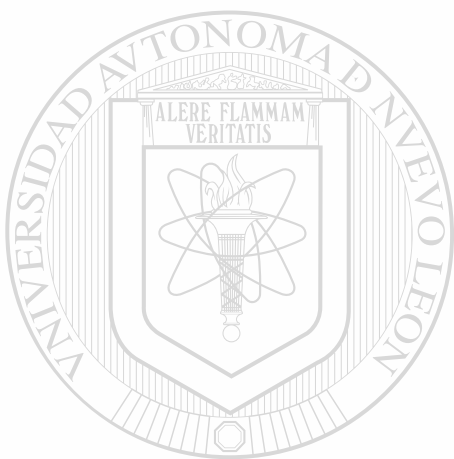


DIRECCIÓN GENERAL DE BIBLIOTECAS

LISTA DE FIGURAS

Figura	Página
1. Partes de la neurona	12
2. Flujo de información en el sistema nervioso	13
3. Simbología para las neuronas artificiales	14
4. Tipos de asociación	17
5. Diagrama de bloques para sistemas de reconocimiento y clasificación	18
6. Esquema de un clasificador multicategoría	18
7. Diagrama de bloques de un clasificador básico	20
8. Dicotomizador	20
9. Ilustración de una función discriminante lineal bidimensional	21
10. Clasificador lineal	22
<hr/>	
11. Esquema del aprendizaje supervisado en una neurona	25
12. Dicotomizador lineal con TLU	27
13. Aprendizaje en un perceptrón discreto	28
14. Aprendizaje en un perceptrón continuo	29
15. Estructura de un perceptrón discreto multicategoría	31
16. Estructura de un perceptrón continuo multicategoría	33
17. Red neuronal con dos capas ocultas	35
18. Arbol asociado al proceso de división	48
19. Red neuronal transformadora a un conjunto linealmente separable	65
20. Red neuronal clasificadora con funciones discretas de activación	65

Figura	Página
21.Red neuronal con funciones discretas de activación	70
22.Red neuronal con funciones continuas de activación	71
23.Red neuronal transformadora a un conjunto linealmente separable	93
24.Red neuronal de memoria entera con funciones discretas de activación	94



UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



DIRECCIÓN GENERAL DE BIBLIOTECAS

RESUMEN AUTOBIOGRAFICO

Francisco Román Angel Bello Acosta

Candidato para el Grado de

Doctor en Ingeniería con Especialidad en Ingeniería de Sistemas

Tesis: **NUEVO ENFOQUE EN EL DISEÑO Y ENTRENAMIENTO DE REDES NEURONALES PARA LA CLASIFICACION**

Campo de Estudio: Ingeniería

Biografía: Nacido en Ciego de Avila, Cuba el 29 de Octubre de 1961, hijo de Arbelio Andrés Angel Bello Pardo y de Blanca Rosa Acosta Darias.

DIRECCIÓN GENERAL DE BIBLIOTECAS

Educación: Egresado de la Universidad Estatal de Daguestán, Rusia. Grados obtenidos: Licenciado en Matemáticas y Maestro en Ciencias Físico-Matemáticas con especialidad en Ecuaciones Diferenciales, en 1985.

Experiencia profesional: Maestro de Tiempo Completo del Departamento de Matemáticas de la Universidad de Camagüey desde 1985 hasta 1997. Profesor Invitado en la Facultad de Ciencias de la Universidad de Angola, desde 1988 hasta 1990.

