Instituto Politécnico Nacional

Centro de Investigación en Computación
Laboratorio de Robótica y Mecatrónica

Método para el entrenamiento de neuronas morfológicas con procesamiento dendral

TESIS

que para obtener el grado de:
Doctorado en Ciencias de la Computación
presenta:

M. en C. Elizabeth Guevara Martínez

Director de Tesis:
Dr. Juan Humberto Sossa Azuela

México, D.F. Enero 2016

SIP-14

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 16:00 horas del día 14 del mes de diciembre de 2015 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Método para el entrenamiento de neuronas morfológicas con procesamiento dendral"

Presentada por la alumna:

GUEVARA	**MARTÍNEZ**	**ELIZABETH**
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

A	1	2	0	4	3	4

aspirante de: **DOCTORADO EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Director de tesis

Dr. Juan Humberto Sossa Azuela

Dr. Sergio Suárez Guerra

Dr. Herón Molina Lozano

Dr. Ricardo Barrón Fernández

Dra. Elsa Rubio Espino

Dr. Francisco Hiram Calvo Castro

PRESIDENTE DEL COLEGIO DE PROFESORES

Dr. Luis Alfonso Villa Vargas

ESTADOS UNIDOS MEXICANOS
INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION EN COMPUTACION
DIRECCION

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de **México D. F.** el día **18** del mes **diciembre** del año **2015**, el (la) que suscribe **Elizabeth Guevara Martínez** alumno (a) del Programa de con número de registro **A120434**, adscrito al **Centro de Investigación en Computación**, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección del **Dr. Juan Humberto Sossa Azuela** y cede los derechos del trabajo intitulado **Método para el entrenamiento de neuronas morfológicas con procesamiento dendral**, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección **eli.guevara@gmail.com**. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Elizabeth Guevara Martínez

Nombre y firma

Resumen

El reconocimiento de patrones se ha estudiado ampliamente en las últimas décadas. No obstante, sigue siendo un tema importante de investigación, ya que la mayoría de las propuestas presentan inconvenientes al enfrentarse con problemas complejos de la vida real.

Entre las técnicas más utilizadas para la clasificación de patrones se encuentran las redes neuronales artificiales. Sin embargo, los perceptrones clásicos tienen ciertas desventajas como problemas de convergencia, tiempos de entrenamiento largos y la posible generación de superficies no cerradas. Estas características no las presentan las redes neuronales morfológicas con procesamiento dendral, por lo que son una herramienta apropiada para el reconocimiento de patrones.

Las redes neuronales morfológicas se basan en el Álgebra Reticular (*Lattice Algebra*). No emplean multiplicaciones como lo hacen los modelos clásicos, sino máximos y mínimos, lo cual favorece implementaciones rápidas y eficientes. Además, el modelo de perceptrones morfológicos se ha extendido incorporando estructuras dendríticas que permiten resolver problemas no lineales sin necesidad de múltiples capas.

Este trabajo se enfoca en el desarrollo de un método para el entrenamiento de neuronas morfológicas con procesamiento dendral que permita resolver aplicaciones reales de reconocimiento de patrones. El desempeño del algoritmo se evalúa desde una perspectiva práctica –con diferentes conjuntos de datos sintéticos, bases de datos y experimentos reales– y se compara con diversas técnicas que se han aplicado para resolver problemas de clasificación como: perceptrones multicapa, redes de base radial y máquinas de vector soporte.

Los resultados experimentales muestran que el algoritmo de entrenamiento propuesto en esta investigación tiene un desempeño comparable y en algunos casos superior al de los métodos evaluados.

Abstract

Pattern recognition has been studied thoroughly in recent decades. However, it remains an important research topic, since most of the proposals have drawbacks when tackling real life tasks.

Among the techniques used for pattern classification are artificial neural networks, but classical perceptrons have certain disadvantages such as convergence problems, extensive training times and possible generation of non-closed surfaces. On the other hand, morphological neural networks do not have these disadvantages; therefore, this type of artificial neural networks is an appropriate tool for pattern recognition.

Morphological neural networks are based on Lattice Algebra; they do not utilize multiplication operations only maximum, minimum and addition, allowing fast and efficient implementations. Furthermore, the morphological perceptron model has been extended to incorporate dendritic structures that solve non-linear problems without multiple layers.

This work focuses on the development of a training method for morphological neurons with dendritic processing that solve pattern classification problems. The algorithm performance is evaluated from a practical perspective, with different sets of synthetic data, databases and real-world experiments and compared with several techniques that have been applied to classification such as: multilayer perceptrons, radial basis networks and support vector machines. Experimental results show that the proposed training algorithm has a comparable performance, and in some cases it is superior to the considered methods.

En memoria de mi padre,

Marcos Guevara

Agradecimientos

Mi más sincero agradecimiento al Centro de Investigación en Computación del IPN, por todas las oportunidades que me brindó.

Al CONACYT, por la beca otorgada para realizar mis estudios de doctorado, a la SIP-IPN en el marco de los proyectos con número de registro SIP 20151187, 20151769, CONACYT 155014 y CONACYT 65 (Fronteras de la Ciencia).

A mi asesor, el Dr. Humberto Sossa, de quien he aprendido mucho, tanto en el aspecto académico como personal. Muchas gracias por creer en mí y aceptarme como estudiante. Un privilegio haber trabajado con usted.

A los investigadores miembros de mi comité tutorial y sinodales: Dra. Elsa Rubio, Dr. Hiram Calvo, Dr. Ricardo Barrón, Dr. Sergio Suárez y Dr. Herón Molina, gracias por sus comentarios y consejos. Estoy muy agradecida por tener la oportunidad de conocer a personas tan valiosas.

A mi familia, no solo me refiero a mi mamá María Teresa, a mis hermanas Teresa y Erendira, a mis sobrinos Andy, Santi y Antonio, sino además a ustedes, María Luisa, Jorge y Ana, por su cariño y apoyo incondicional en todo momento.

A todas las personas que me ayudaron y ofrecieron su amistad durante mi estancia en el CIC; ustedes saben quiénes son, les agradezco profundamente.

A mi esposo, Luis Toxqui, gracias por estar siempre conmigo a pesar de todo. Un reto más que cumplimos, te amo.

Y principalmente, gracias a Dios, por permitirme concluir esta etapa y estar siempre a mi lado.

Elizabeth Guevara

Índice general

Lista de figuras

Lista de tablas

Lista de acrónimos

IPC	Índice de Precios y Cotizaciones
MLP	Multilayer Perceptron, Perceptrón multicapa
NMPD	Neurona Morfológica con Procesamiento Dendral
PM	Perceptrón Morfológico
PMPD	Perceptrón Morfológico con Procesamiento Dendral
RBN	Radial Basis Network, Red de Base Radial
RNA	Red Neuronal Artificial
RNM	Red Neuronal Morfológica
RNMPD	Red Neuronal Morfológica con Procesamiento Dendral
SVM	Support Vector Machine, Máquina de vector soporte
TLU	Threshold Logic Unit, Unidad lógica de umbral

Capítulo 1

Introducción

El reconocimiento de patrones es un área de gran interés para la investigación en el campo de la Inteligencia Artificial (IA). De acuerdo con el ganador del premio Nobel de Economía, Herbert Simon, el reconocimiento de patrones es fundamental en la mayoría de las tareas de toma de decisiones del ser humano [1], [2]. La actividad de adquirir datos brutos (patrones) y seleccionar una acción basada en la información obtenida de ellos (reconocimiento o clasificación) ha sido crucial para el aprendizaje.

Por tanto, el reconocimiento de patrones es de gran importancia para el desarrollo de sistemas inteligentes que se utilizan no solo en ingeniería, sino en diversas disciplinas como: biología, medicina, finanzas, mercadotecnia, psicología, entre muchas otras.

Antes de la década de 1960, la mayoría de la investigación teórica de reconocimiento de patrones se realizaba en el ámbito estadístico. Sin embargo, los esfuerzos multidisciplinarios fomentaron nuevas ideas, metodologías y técnicas que enriquecieron el paradigma tradicional, como las Redes Neuronales Artificiales (RNA). La aplicación de estas técnicas en el reconocimiento de patrones ha sido posible gracias al desarrollo tecnológico de los últimos años, que permitió la aplicación de algoritmos de aprendizaje, de búsqueda y optimización complejos que no podían realizarse hace algunas décadas, así como abordar problemas reales que pueden implicar miles de muestras en espacios de alta dimensión.

Dentro de las RNA los perceptrones multicapa (MLP) se han utilizado para resolver tareas de reconocimiento, pero se demostró que los MLP no son adecuados para este tipo de problemas [3], puesto que pueden generar superficies de separación abiertas en el espacio de patrones en aplicaciones que requieren un rechazo fiable.

Una alternativa para resolver problemas de clasificación son las Redes Neuronales Morfológicas (RNM) basadas en el marco de la denominada Álgebra Reticular (*Lattice Algebra*), a diferencia de las RNA que se basan en el Álgebra Lineal.

Las Redes Neuronales Morfológicas con Procesamiento Dendral (RNMPD) surgieron a partir de las RNM al agregar estructuras dendríticas. Este tipo de RNA presenta diversas ventajas en comparación con los MLP. Algunas de éstas son:

1. Las RNMPD no requieren de capas ocultas para resolver problemas no lineales, solo son necesarias las dendritas.
2. La estructura neuronal se construye dinámicamente con base en el conjunto de datos de entrenamiento, en lugar de que dicha estructura se predefina.
3. En la etapa de entrenamiento de las RNMPD no es necesario ajustar parámetros previamente y el porcentaje de reconocimiento siempre es del 100 % para el conjunto de entrenamiento.
4. Con las RNMPD se pueden generar superficies de separación cerradas, lo cual no es posible garantizar con los MLP.
5. El no emplear multiplicaciones en su modelo favorece implementaciones rápidas y eficientes.

Todas estas características hacen a las RNMPD una buena herramienta para la comunidad de reconocimiento de patrones.

1.1. Planteamiento del problema y consideraciones

Si bien, el reconocimiento de patrones se ha estudiado ampliamente, en numerosos casos las soluciones propuestas solo resuelven parcialmente los problemas debido a las restricciones impuestas para obtener un rendimiento óptimo.

A pesar de sus ventajas con respecto a otros tipos de RNA como el MLP, los algoritmos de entrenamiento existentes para las RNMPD presentan ciertas limitaciones como: sensibilidad al ruido, dependencia del orden de presentación de los patrones de entrenamiento, traslape entre las regiones y la necesidad de agregar una capa al Perceptrón Morfológico con Procesamiento Dendral (PMPD) para resolver problemas de múltiples clases sin ambigüedad.

El objetivo de esta tesis es explotar las ventajas de las RNMPD y proponer un método para el entrenamiento de neuronas morfológicas con procesamiento dendral (NMPD) que permita disminuir las limitaciones mencionadas y que sea capaz de resolver problemas multiclase con una sola neurona.

En la fase experimental se utiliza la metodología para un sistema de reconocimiento de patrones utilizando la clasificación para la toma de decisiones, que consiste en una etapa de

entrenamiento y una de prueba. En este trabajo se resolverán problemas de clasificación, no solamente de patrones sintéticos y bases de datos, sino experimentos en el contexto de sistemas en tiempo real.

Para validar el desempeño del algoritmo de entrenamiento se usan técnicas de validación cruzada con el fin de obtener los datos de entrenamiento y prueba y se realizan comparaciones con otros modelos usados para clasificación como Perceptrones Multicapa (*Multilayer Perceptrons*, MLP), Redes de Base Radial (*Radial Basis Networks*, RBN) y Máquinas de Vector Soporte (*Support Vector Machine*s, SVM).

1.2. Objetivo

Definir una nueva forma de entrenar redes neuronales morfológicas con procesamiento dendral que permita mejorar su desempeño, con el propósito de resolver problemas reales de reconocimiento de patrones.

1.3. Objetivos particulares

Los objetivos particulares de este trabajo de investigación son:

1. Proponer un algoritmo de entrenamiento eficiente para redes neuronales morfológicas con procesamiento dendral.

2. Definir las propiedades del modelo de entrenamiento planteado.

3. Aplicar el método de entrenamiento para resolver problemas de clasificación de patrones usando:

   a) datos sintéticos,

   b) bases de datos y

   c) aplicaciones reales:

      1) Predicción diaria del movimiento del IPC.
      2) Reconocimiento de objetos utilizando Kinect para la segmentación.
      3) Reconocimiento de objetos para que un robot humanoide realice una acción.
      4) Control compartido para el seguimiento de una trayectoria.

1.4. Justificación

El reconocimiento de patrones es un área de gran importancia para los sistemas automáticos de toma de decisiones. Sin embargo, a pesar de que se han realizado numerosas investigaciones en el área, aún representa un problema a resolver, puesto que la mayoría de los estudios muestran dificultades al enfrentarse a circunstancias complejas de la vida real. Es por esto que en la presente tesis se propone un método de entrenamiento de NMPD capaz de solucionar problemas reales de reconocimiento de patrones, demostrando ser una herramienta eficaz que puede competir, e incluso en algunos casos mejorar otras alternativas de RNA.

1.5. Contribuciones

Durante el desarrollo de este trabajo de investigación se realizaron las siguientes contribuciones:

1. Propuesta de un algoritmo de entrenamiento eficiente para NMPD.
2. Mejora del algoritmo para poder resolver problemas en espacios de alta dimensión.
3. Validación del desempeño del algoritmo de entrenamiento en contextos reales.

1.6. Organización de la tesis

Este documento se divide en tres partes. La primera sección comprende los cuatro capítulos iniciales que incluyen la introducción, estado del arte y el marco teórico, en los cuales se presentan los aspectos fundamentales de la investigación. El segundo apartado se centra en la propuesta de un método de entrenamiento para NMPD, expuesto en el capítulo 5. Los resultados experimentales, así como las aplicaciones del algoritmo de entrenamiento se presentan en los capítulos 6 y 7, respectivamente. Por último, en la tercera sección se enuncian las conclusiones, el trabajo futuro y las publicaciones originadas a partir de esta investigación. A continuación se describe el contenido de cada capítulo:

- Capítulo 1. Se presenta una introducción, el planteamiento del problema, así como los objetivos, justificación y las contribuciones de este trabajo.
- Capítulo 2. Se muestra un breve estado del arte en el que se describen los trabajos relacionados con las RNM y la necesidad de seguir investigando en RNMPD.

- Capítulo 3. Se expone el marco teórico y se definen los conceptos fundamentales de RNA y de reconocimiento de patrones necesarios para solucionar el problema establecido en esta investigación.
- Capítulo 4. Se explica la teoría básica para comprender el funcionamiento de las RNMPD, así como sus métodos de entrenamiento.
- Capítulo 5. Se detalla la propuesta del método de entrenamiento para NMPD, sus ventajas y desventajas. Además de una mejora de dicho algoritmo.
- Capítulo 6. Se presentan los resultados experimentales obtenidos al aplicar el método de entrenamiento a diversos problemas que involucran bases de datos, tanto sintéticas como reales. Asimismo, se evalúan los resultados en comparación con otras técnicas de IA como MLP, RBN y SVM.
- Capítulo 7. Se muestran los experimentos utilizando plataformas reales como *Kinect*, robots humanoides (*Bioloid* y *NAO*) y un robot móvil construido con *Lego Mindstorm*.
- Capítulo 8. Se mencionan las conclusiones, así como futuras líneas de investigación que podrían derivar de este trabajo.

Capítulo 2

Estado del arte

En este capítulo se presenta una reseña de la historia de las RNA y se describen brevemente las principales investigaciones que se han realizado sobre Redes Neuronales Morfológicas (RNM) y sus aplicaciones en diferentes áreas, haciendo énfasis en el reconocimiento de patrones.

El estudio de las RNA se inspiró originalmente en los avances en neurociencia comenzando con la investigación de McCulloch y Pitts publicada en 1943 [4], en la que mostraron que incluso los tipos más simples de redes neuronales podían calcular funciones aritméticas o lógicas. El siguiente avance importante fue el perceptrón, introducido por Frank Rosenblatt en su artículo de 1958 [5]. Sin embargo, las primeras investigaciones en RNA llegaron casi a un punto muerto en la década de 1970. En el libro publicado por Minsky y Papert [6] demuestran que el perceptrón, con las reglas propuestas en ese momento, no podía calcular funciones esenciales como la XOR. Con esta investigación parecía que el campo de las redes neuronales no tenía futuro.

En los años de 1982-1986 John Hopfield, un físico de reputación internacional, se interesó en las redes neuronales. Hopfield escribió tres documentos sobre redes neuronales en 1982 [7], 1984 [8] y 1986 [9], estos trabajos persuadieron a cientos de científicos, matemáticos y técnicos altamente calificados para unirse al campo emergente de las redes neuronales. Desde estos días las RNA se han convertido en una herramienta importante en el aprendizaje automático y la IA y se han aplicado en áreas tan diversas como el reconocimiento de patrones, control, robótica, procesamiento de imágenes y voz, visión artificial, almacenamiento y recuperación de datos, sistemas expertos y muchas otras.

Como parte de estas nuevas investigaciones en el área de las RNA surgen las RNM.

2.1. Redes neuronales morfológicas

Las RNM fueron introducidas originalmente por Davidson en 1993 [10] y después retomadas por Ritter y Sussner en 1996 [11]. Las RNM están estrechamente relacionadas con el "Cómputo Reticular" o "Lattice Computing" [12], que se define como la colección de herramientas que hacen uso de operadores mínimo y máximo para la construcción de los algoritmos computacionales y reemplazan el operador de multiplicación por el operador de adición.

Históricamente, el desarrollo de las RNM se ha extendido por cerca de una década durante la cual se pueden distinguir dos etapas:

1. La primera etapa comienza con la introducción de las RNM de una y múltiples capas, y algunos algoritmos de entrenamiento [13], [14], [15], [16]. Además de diversas aplicaciones entre las que destacan las siguientes:

   - Desde su inicios las RNM se utilizaron para construir memorias asociativas resistentes a distorsiones morfológicas (como por ejemplo el ruido de sal y pimienta, etc.). Algunas investigaciones en este sentido son las de Ritter et al. [17], Raducanu et al. [18] y Sussner [19], [20]. Una extensión de memorias binarias a escala de grises fue propuesta por Sussner y Valle en [21].
   - Otros trabajos de procesamiento de imágenes usando RNM son los de Davidson y Hummer quienes diseñan RNM generalizadas de dilatación y erosión, de dilatación booleana y de dilatación en escala de grises; además de resultados teóricos presentan soluciones en imágenes reales [10]. Por otra parte, en [22] Aráujo et al. utilizan un algoritmo genético para entrenar una RNM con el objetivo de restaurar imágenes ruidosas.
   - Won y colegas proponen en [23] una RNM de pesos compartidos y la aplican al reconocimiento de vehículos oruga en imágenes infrarrojas y a la detección de vehículos específicos en imágenes de un estacionamiento. Cabe destacar que el entrenamiento de la red no se realiza en tiempo real. Otras aplicaciones de detección de objetivos (*targets*) en imágenes se presentan en [24] y [25].
   - Raducanu et al. aplican una RNM heteroasociativa para la auto-localización de robots móviles en un marco de navegación basado en visión artificial [26]. Tratan la auto-localización como un problema de clasificación donde la entrada es la imagen captada por la cámara del robot y el vector de salida especifica una clase que codifica la posición del robot. Las pruebas únicamente se realizan en las imágenes adquiridas y no durante el movimiento del robot móvil.
   - Urcid y colegas clasifican correctamente, usando RNM, la paridad de todas las cadenas de bits de longitud n [27]. Este problema de reconocimiento de patrones es la extensión de n dimensiones del problema clásico XOR en el plano euclidiano y se usa comúnmente como un punto de referencia para probar el rendimiento de algoritmos de entrenamiento de las RNA.

- Villaverde y colegas utilizan RNM para la localización simultánea y mapeo (SLAM), que es un problema clave en robótica. Ellos resuelven el problema de SLAM en interiores con la ayuda de la información visual obtenida a partir de imágenes morfológicamente independientes. Y determinan su correspondencia con los vértices de la envolvente convexa en un espacio de alta dimensión [28].

2. La siguiente etapa estuvo marcada por la incorporación de estructuras dendríticas en el modelo original [29], [15], y la propuesta de diversos métodos de entrenamiento [30], [31], [32], [33], [34], [35]. La variación dendrítica de los perceptrones morfológicos también comprende redes de una y múltiples capas. Las redes neuronales morfológicas con procesamiento dendral de una capa se utilizan principalmente para resolver problemas de clasificación mientras que las de múltiples capas se emplean para la construcción de memorias asociativas [36], [37], [38]. Debido a que el área de las RNMPD es relativamente reciente, los investigadores se han enfocado en proponer nuevos modelos que incorporen el procesamiento dendrítico más que en las aplicaciones, algunas de ellas son:

   - En 2007, Roberson y Dankel II proponen la creación de un motor de consulta que transforma preguntas de los usuarios en una RNMPD capaz de filtrar documentos. La red de consulta es un perceptrón morfológico de una sola capa con una sola dendrita diseñada para tener como entrada un vector del documento y devolver una medida de la relevancia de la consulta examinada. Para cada término distinto de cero en el vector de consulta se realiza una conexión excitadora a la dendrita, y el peso de conexión se determina por el término de peso del vector de consulta. Debido a que la fuerza de la respuesta de la dendrita es importante, la función de activación de la neurona en lugar de ser la función limitador duro, como en el modelo original, es una función lineal. Para sus pruebas usan una base de datos de la revista TIME [39].
   - Chyzhyk et al. tratan el problema de la detección temprana de la enfermedad de Alzheimer a partir de imágenes de resonancia magnética usando procesamiento dendrítico. Los datos son extraídos de la base de datos OASIS de imágenes de resonancia magnética de pacientes con Alzheimer y realizan un análisis de morfometría basado en *voxels* (unidad cúbica que compone un objeto tridimensional) para determinar la ubicación de los grupos de voxels más afectados por la enfermedad. Para calcular los vectores de características usados para la clasificación, se utiliza la media y la desviación estándar de los voxels de las agrupaciones detectadas.[34], [40], [41].
   - En 2013, Caro-Contreras y Méndez investigadores del CINVESTAV Campus Guadalajara presentan un nuevo algoritmo, que denominan PIRA, para calcular Retículas de Conceptos utilizando una RNMPD. En este trabajo usan la relación entre los rectángulos de altura y anchura máxima y una RNMPD para clasificar las anti-cadenas máximas en la estructura reticular. Presentan una comparación con algunos algoritmos conocidos para la construcción de retículas de conceptos [35].

- La mayoría de las aplicaciones de RNMPD resuelven problemas sintéticos de reconocimiento de patrones [42], [43], problemas típicos de clasificación de patrones como las bases de datos del Iris, Vino, etc. [44] y recuperación de imágenes y patrones a partir de entradas ruidosas [45],[46].

En general, son pocas las aplicaciones reales de las RNMPD por lo que es importante seguir explorando las ventajas de este tipo de RNA que incorporan estructuras dendríticas. Investigaciones han demostrado que las dendritas son importantes ya que por sí mismas son capaces de resolver funciones linealmente no separables [47].

2.2. Resumen

En este capítulo se presenta una reseña de los principales artículos sobre RNM. Los trabajos están organizados tomando en consideración dos etapas principales de desarrollo de las RNM. La segunda etapa está determinada por el surgimiento de las RNMPD. Como se muestra en el presente apartado, las aplicaciones de las RNMPD son escasas, por lo que uno de los objetivos de esta investigación es contribuir en la solución de problemas utilizando este modelo neuronal. Cabe mencionar que el método de entrenamiento propuesto para neuronas morfológicas con procesamiento dendral ya se ha aplicado en otros trabajos para resolver problemas reales, contribuyendo así al estado del arte.

Capítulo 3

Marco teórico

En este capítulo se presentan los aspectos fundamentales para el desarrollo de esta investigación. Se tratan brevemente conceptos básicos de RNA y reconocimiento de patrones.

3.1. Redes neuronales artificiales

Los RNA buscan solucionar problemas cotidianos tal como lo hace el cerebro humano, que aprende sin instrucciones explícitas de ninguna clase. Estos sistemas llamados también conexionistas se basan en la eficiencia de los procesos llevados a cabo por el cerebro y se inspiran en su funcionamiento, es decir "emulan" las redes de neuronas biológicas. Las RNA no requieren que la tarea a ejecutar se programe sino que generalizan y aprenden de la experiencia [48].

Al ser las RNA un intento de modelar las capacidades de procesamiento de información de los sistemas nerviosos, se deben tener en cuenta las propiedades esenciales de las redes neuronales biológicas desde este punto de vista. En las redes neuronales biológicas la información se almacena en los puntos de contacto entre las diferentes neuronas, las denominadas sinapsis. Las neuronas reciben señales y producen una respuesta.

La estructura general de una neurona genérica se muestra en la figura 3.1. Las ramas de la izquierda son los canales de transmisión de la información entrante y se denominan dendritas. Las dendritas reciben las señales en las regiones de contacto con otras células, en las sinapsis. Mientras que las señales de salida se transmiten por el axón.

La estructura mínima que se considera en las neuronas artificiales son los canales de entrada, un cuerpo celular y un canal de salida. Las sinapsis se simulan mediante puntos

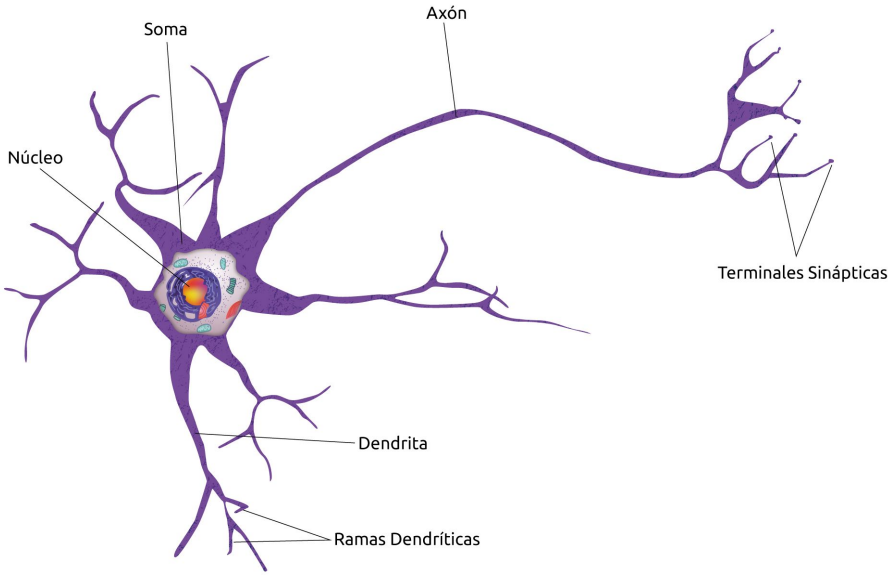

Figura 3.1: Estructura de una neurona biológica.

de contacto entre el cuerpo de la célula y las conexiones de entrada o salida, a las cuales generalmente se les asigna un peso. De esta forma, el conocimiento de una red de neuronas artificiales está en su topología (número de neuronas que componen una RNA y forma en que están conectadas entre sí) y en los valores de las conexiones (pesos) entre neuronas. En la figura 3.2 se muestra la semejanza entre la arquitectura de una RNA y una neurona biológica.

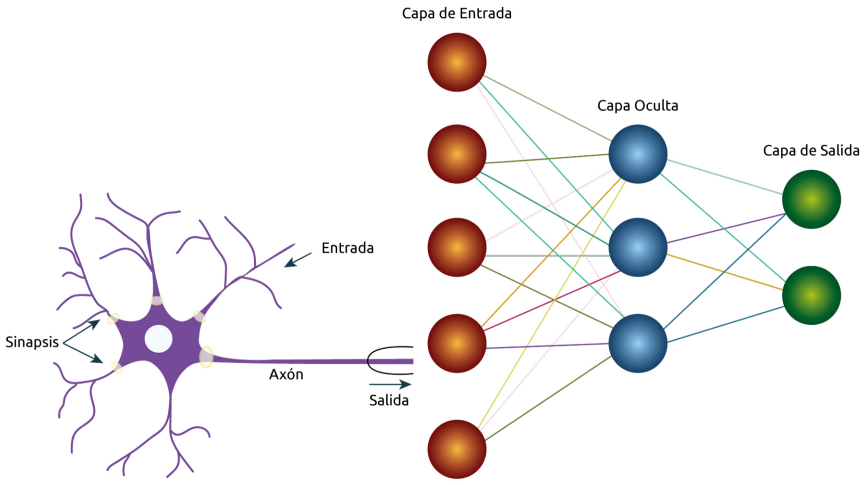

Figura 3.2: Neurona biológica y RNA.

Las técnicas tradicionales de programación requieren la generación de un algoritmo para solucionar un problema. Las RNA en lugar de algoritmos necesitan ser entrenadas. Como el proceso de aprendizaje humano, el de las RNA generalmente está basado en el uso de ejemplos que representan el problema. A este conjunto de ejemplos se le conoce como conjunto de entrenamiento. Es importante recalcar que el objetivo del aprendizaje no

es memorizar las relaciones entrada/salida que hay en el conjunto de entrenamiento sino modelar el proceso que genera estos datos. Para ello es conveniente que el número y tipo de ejemplos disponibles para el entrenamiento de la red sea suficientemente representativo de la relación que se desea aprender. De este modo, una vez entrenada, la red será capaz de manejar no únicamente los datos de entrenamiento, también nuevos datos que no han sido presentados con anterioridad a la red (conjunto de prueba) sin que por ello se degrade su rendimiento. Esto se conoce como la capacidad de generalización de la red.

El conocimiento en la RNA se obtiene a través de un proceso o regla de aprendizaje o entrenamiento, que no es más que la modificación de los parámetros de las RNA mediante un procedimiento preestablecido con el fin de conseguir una mejora en su rendimiento.

En general existen dos tipos básicos de problemas que una RNA puede tratar de resolver: problemas de clasificación y problemas de regresión. En la clasificación el objetivo es discriminar entre dos o más clases o categorías de objetos, basándose en la similitud de las características o atributos que constituyen los datos de entrada a la RNA. Por otro lado, los problemas de regresión son problemas de ajuste de funciones, se trata de obtener un número en función de los atributos de entrada a la red. En este trabajo de investigación se resolverán problemas de clasificación o reconocimiento de patrones.

Entre las principales ventajas de las RNA se pueden mencionar las siguientes:

- Paralelismo: debido a sus características las RNA se pueden implementar en plataformas de cómputo paralelo.
- Memoria distribuida: la memoria de una RNA corresponde a un mapa de activación de las neuronas, por tanto se distribuye sobre muchos nodos siendo robusta al ruido.
- Tolerancia a fallos: es la característica de poder funcionar de forma aceptable tanto en presencia de información inexacta como cuando se producen deterioros o fallos en sus componentes, entonces el comportamiento general de la red solo se degrada pero es funcional.
- No requiere modelación matemática: para utilizar las RNA no es necesario conocer los detalles matemáticos, solamente se requiere estar familiarizado con los datos de trabajo.
- Capacidad de adaptación: las RNA tienen la habilidad de aprender y en algunos casos de auto-organizarse.
- Capacidad de generalización: estas redes pueden ofrecer, dentro de un margen, respuestas correctas a entradas que presentan pequeñas variaciones debido a los efectos de ruido o distorsión.

Las limitaciones más severas para el uso de las RNA son las siguientes:

- Simulaciones secuenciales: aunque las RNA son inherentemente sistemas paralelos, estos se simulan normalmente en máquinas secuenciales.
- Problema de escalamiento: el tiempo de procesamiento puede incrementarse rápidamente cuando el tamaño del problema crece, además las RNA se deben entrenar para cada problema.
- Hardware vs. software: Las implementaciones en hardware son más rápidas pero menos flexibles que las implementaciones en software.
- Sensibilidad a datos: el desempeño de una RNA depende de la calidad de los datos de entrada.
- Estimación del desempeño: el desempeño se mide mediante métodos estadísticos y no matemáticamente.
- Ajuste de parámetros: muchas de las decisiones de diseño de una RNA no se pueden justificar totalmente.

En las siguientes subsecciones se presenta una descripción de algunos de los principales modelos de redes neuronales. Una explicación más detallada se puede consultar en [49], [48].

3.1.1. TLU

Las redes neuronales se basan en el modelo matemático de neurona propuesto por McCulloch y Pitts en 1943 [4] conocido como unidad lógica de umbral TLU (*Threshold Logic Unit*). En dicho modelo (figura 3.3) cada neurona recibe un conjunto de entradas $\{x_1, x_2, ..., x_n\}$ y devuelve una única salida y obtenida mediante la comparación de su entrada total con un umbral θ.

Las redes de McCulloch-Pitts cuentan con sinapsis no ponderadas de excitación o inhibición y solo producen resultados binarios, además cada unidad está provista de un cierto valor de umbral. En primera instancia, el modelo de McCulloch-Pitts parece muy limitado, ya que únicamente se puede transmitir y generar información binaria pero ya cuenta con todas las características necesarias para implementar modelos más complejos. La figura 3.3 muestra una unidad de cálculo abstracta de McCulloch-Pitts.

La regla para evaluar una unidad de McCulloch-Pitts es la siguiente:

- Supóngase que una unidad de McCulloch-Pitts recibe una entrada $x_1, x_2, \ldots, x_n$ a través de n sinapsis excitadoras y una entrada $y_1, y_2, \ldots y_m$ a través de m sinapsis inhibitorias.

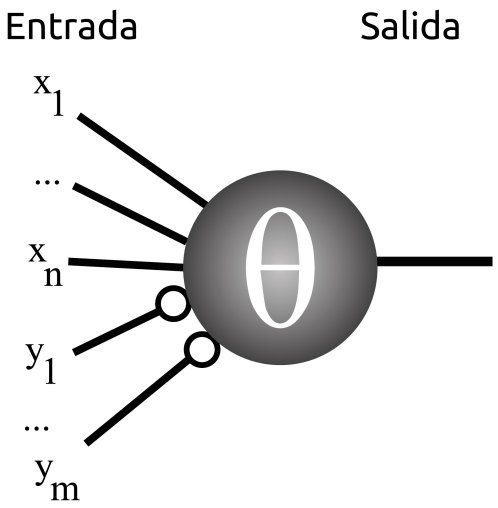

Figura 3.3: Representación de la TLU de McCulloch-Pitts.

- Si $m \geq 1$ y al menos una de las señales $y_1, y_2, \ldots y_m$ es 1, el resultado del cálculo es 0.
- De lo contrario, la excitación total de $x = x_1 + x_2 + \ldots + x_n$ se calcula y se compara con el umbral θ de la unidad (si $n = 0$, entonces $x = 0$). Si $x \geq 0$, la unidad dispara un 1, si $x < 0$ el resultado del cálculo es 0.

Esta regla implica que una unidad de McCulloch-Pitts puede inactivarse por un inhibidor de señal único, como es el caso de algunas neuronas. Cuando no hay señales inhibitorias presentes, las unidades actúan como una compuerta de umbral capaz de implementar muchas funciones lógicas de n argumentos.

La interpretación geométrica del objetivo de las TLU de McCulloch-Pitts es separar el espacio de entrada en dos espacios. Para los puntos que pertenecen a un espacio el resultado del cálculo es 0, y para puntos que pertenecen a la otra región, el resultado es 1. En el caso general, una RNA debe aprender a distinguir entre regiones del espacio y asociarlas con la respuesta correcta.

3.1.2. Perceptrón

En 1958 Frank Rosenblatt, basándose en el modelo de McCulloch-Pitts, presentó el perceptrón, el primer modelo de RNA [5]. En la figura 3.4 se muestra un diagrama del perceptrón que consta de una retina de perceptores que pueden ser 0 ó 1. Estas entradas pasan a un conjunto de unidades A o unidades asociativas. Las conexiones entre los perceptores y las unidades A tienen como valores posibles +1, 0 y –1. Cuando aparece una serie de estímulos en la retina, las unidades asociativas se activan si la suma de sus

entradas sobrepasa algún valor de umbral fijo, obteniéndose de esta manera las salidas s_j. El nodo de suma del modelo neuronal del perceptrón realiza una combinación lineal de las entradas aplicadas a las sinapsis e incorpora un sesgo con valor de -1. El resultado de la suma se aplica a una función de disparo, en este caso una función que produce una salida igual a +1 si la entrada es positiva o igual a cero, y –1 si es negativa. Dentro de una RNA existen numerosas conexiones entre las distintas neuronas que la forman. Estas conexiones simulan las conexiones interneuronales del cerebro y, al igual que éstas, pueden establecerse con mayor o menor intensidad dependiendo de los pesos sinápticos w_i.

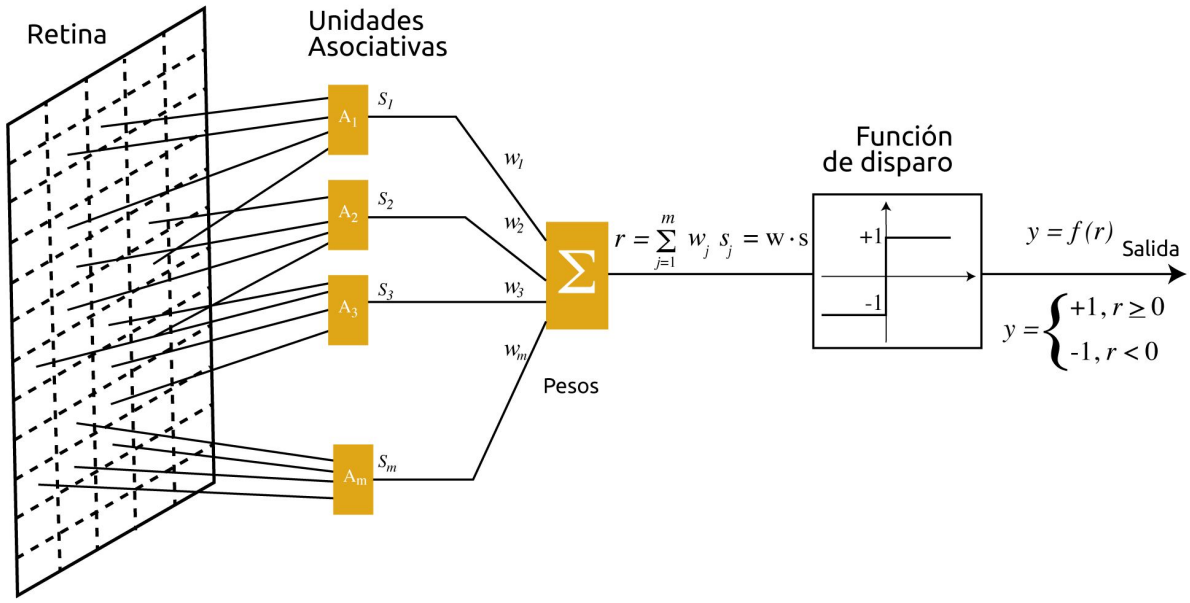

Figura 3.4: Modelo del perceptrón.

Al igual que en el caso de las TLU de McCulloch-Pitts, la interpretación geométrica es separar el espacio de entrada en dos espacios. En este caso, el umbral en el perceptrón se convierte en un peso -θ de un canal de entrada adicional conectado a la constante 1. Este peso extra conectado a una constante se denomina *bias*. En la figura 3.5 se presenta una simplificación del perceptrón, como se puede observar, el primer paso para obtener la salida y del perceptrón es calcular la suma ponderada de las entradas:

$$\left(\sum_{i=1}^{n} w_i x_i\right) + \theta \tag{3.1}$$

donde θ es un umbral o *bias*. Posteriormente, a partir de esta suma ponderada se obtiene la salida y de la neurona mediante la aplicación de una función f llamada función de activación o disparo, en este caso es la función limitador duro simétrica (ver figura 3.4). Como se puede observar en la ecuación 3.1, todas las operaciones ejecutadas por una neurona artificial o perceptrón son operaciones algebraicas lineales por lo que la aplicación de la función de activación es el único componente que puede ser no lineal en este modelo.

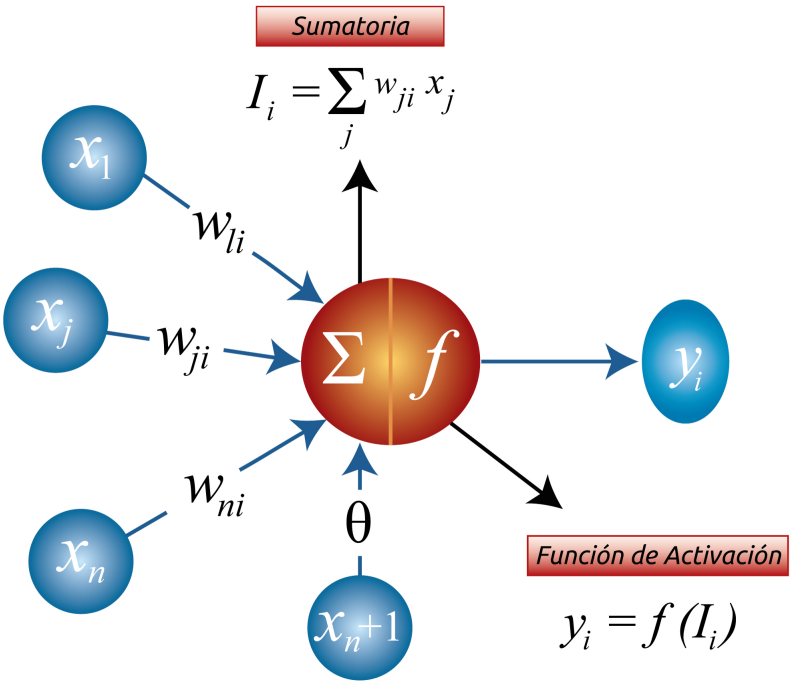

Figura 3.5: Perceptrón.

Por otra parte, el aprendizaje se realiza mediante la adaptación de los pesos con un algoritmo numérico que consiste en determinar los pesos de las conexiones de manera que las salidas del perceptrón coincidan con las salidas deseadas, o por lo menos sean lo más próximas posibles. La idea general de este tipo de algoritmos de aprendizaje es que a partir de la presentación de algunos ejemplos de la salida deseada dada cierta entrada se ejecuta una etapa de corrección iterativa hasta que la RNA aprende a producir la respuesta requerida. Por tanto, el algoritmo de aprendizaje es un ciclo cerrado de presentación de ejemplos y de correcciones de los parámetros del modelo.

Este tipo de aprendizaje se conoce como aprendizaje supervisado. En general existen dos clases de aprendizaje:

- Aprendizaje supervisado: este tipo de aprendizaje requiere de un conjunto con vectores de entrada y sus correspondientes vectores de salida. El entrenamiento consiste en presentar un vector de entrada, calcular la salida y compararla con la salida deseada, el error o diferencia resultante se utiliza para realimentar la red y cambiar los pesos de acuerdo con un algoritmo que tiende a minimizar el error. Las parejas de vectores de entrada se aplican secuencialmente y de forma cíclica, se calcula el error y se ajustan los pesos hasta que el error para el conjunto de entrenamiento sea un valor pequeño y aceptable.
- Aprendizaje no supervisado: los sistemas neuronales con aprendizaje supervisado han tenido éxito en muchas aplicaciones. Sin embargo, debido a que desde el punto de vista biológico resulta difícil creer que existe un mecanismo en el cerebro que

compare las salidas deseadas con las salidas reales, en los métodos de aprendizaje no supervisado el conjunto de datos de entrenamiento consiste solo en los patrones de entrada. La red aprende a adaptarse basada en las experiencias adquiridas de los patrones de entrenamiento anteriores. En algunos métodos de aprendizaje, los parámetros de la red se determinan como resultado de un proceso de auto-organización.

Específicamente, la regla de entrenamiento básico del perceptrón parte de dos conjuntos, C^0 y C^1 en el espacio de entrada n-dimensional. El objetivo es encontrar el vector de pesos $\mathbf{w}$ capaz de separar ambos conjuntos en el espacio mediante el siguiente algoritmo:

Algoritmo 3.1 Aprendizaje del perceptrón

1. Se genera un vector de pesos $\mathbf{w}_0$ aleatoriamente en el tiempo $t = 0$.
2. Se selecciona al azar un vector de entrada $\mathbf{x} \in C^1 \cup C^2$ y se analizan los siguientes casos:
   a) si $\mathbf{x} \in C^0$ y $\mathbf{w}_t \cdot \mathbf{x} > 0$ ir al paso 2,
   b) si $\mathbf{x} \in C^0$ y $\mathbf{w}_t \cdot \mathbf{x} \leq 0$ ir al paso 3,
   c) si $\mathbf{x} \in C^1$ y $\mathbf{w}_t \cdot \mathbf{x} < 0$ ir al paso 2,
   d) si $\mathbf{x} \in C^1$ y $\mathbf{w}_t \cdot \mathbf{x} \geq 0$ ir al paso 4.
3. $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{x}$ y $t = t + 1$, ir al paso 2.
4. $\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{x}$ y $t = t + 1$, ir al paso 2.

La rutina se detiene cuando todos los vectores se clasifican correctamente. Esta regla generará un vector de pesos que define una superficie de decisión que separa a las clases si el problema es linealmente separable.

3.1.3. Perceptrón multicapa

Para aislar una región convexa en el espacio, una alternativa es la combinación de varios perceptrones conocida como perceptrón multicapa MLP (Multilayer Perceptron), que incluye una o varias capas intermedias de unidades procesadoras también denominadas capas ocultas (figura 3.6).

El MLP utiliza el algoritmo de entrenamiento de retropropagación (*backpropagation algorithm*) [50] que usa la información del gradiente, proporcionado por la derivada de una función de error con respecto a los pesos, para encontrar los valores de éstos que generan un mínimo de la función de error. El ajuste se realiza comenzando por la capa de

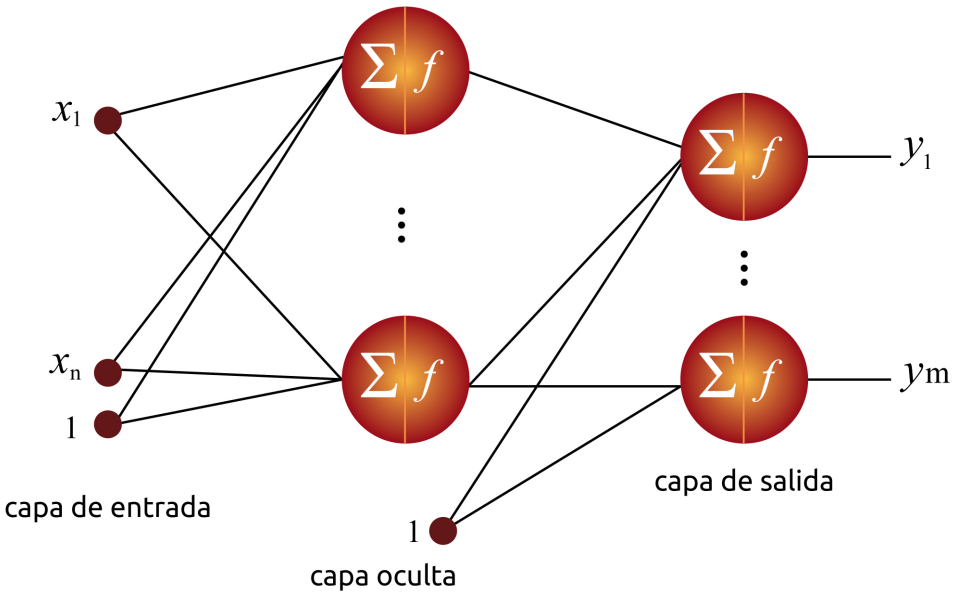

Figura 3.6: Arquitectura de un perceptrón multicapa con una capa oculta.

salida, según el error obtenido y propagando este error a las capas anteriores, hasta llegar a la capa de las unidades de entrada. De ahí que se denomine algoritmo de retropropagación.

Dado que este método requiere el cálculo de la pendiente de la función de error en cada paso de iteración se debe garantizar la continuidad y diferenciabilidad de la misma. Por tanto, se tiene que utilizar un tipo de función de activación diferente a la función de paso empleada en los perceptrones. De lo contrario la función compuesta producida por los perceptrones interconectados es discontinua y por consecuencia la función de error también. Una de las funciones de activación más populares para las redes de retropropagación es el sigmoide, cuya ecuación es la siguiente:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{3.2}$$

esta función proporciona la diferenciabilidad que se requiere en el algoritmo de entrenamiento de retropropagación del MLP que consta de los siguientes pasos:

Algoritmo 3.2 Algoritmo de retropropagación

1. Presentar el patrón de entrenamiento $\mathbf{x}$ a la capa de entrada.
2. Evaluar las salidas de los nodos intermedios.
3. Evaluar las salidas de las neuronas de la capa de salida usando los resultados del paso 2.
4. Comparar las salidas obtenidas con los valores deseados.
5. Calcular los valores δ^i para los nodos de salida mediante la siguiente ecuación de ajuste del error: $\delta^i = \sigma^{'}(x_i)(t_i - y_i)$, en este caso se está utilizando la función sigmoidal como función de activación, t es la salida deseada y y es la salida.
6. Entrenar cada nodo de salida mediante la siguiente ecuación, donde α es la razón de aprendizaje automático que determina qué tan grandes son los cambios en los pesos y puede tomar valores entre 0 y 1.:
$$\triangle w_{ji} = \alpha\sigma^{'}(x_i)(t_i - y_i)x_{ji} = \alpha\delta^j x_{ji} \tag{3.3}$$
7. Calcular los δ^i para los nodos intermedios aplicando la ecuación:
$$\delta^i = \sigma^{'}(x_i)\sum_{j=1}\delta^j w_{ji} \tag{3.4}$$
8. Entrenar cada nodo intermedio.

Los primeros tres pasos realizan lo que se conoce como propagación hacia delante y los pasos del 4 al 8 la propagación hacia atrás. La función de activación utilizada es la función sigmoidal y derivándola usando la regla de la cadena se obtiene que: $\sigma^{'}(x) = \sigma(x)(1 - \sigma(x))$. Al usar la función sigmoidal la salida estará entre 0 y 1, por lo que se considera que si la salida obtenida es mayor o igual a 0.5 entonces $y = 1$ y si la salida es menor a 0.5 entonces $y = 0$.

3.1.4. Redes neuronales morfológicas

Los perceptrones morfológicos (PM) a diferencia de los perceptrones clásicos se basan en el álgebra reticular (ver apéndice C) y al igual que el modelo clásico pueden ser de una capa o de múltiples capas.

En la teoría clásica de RNA, el elemento principal de procesamiento es la neurona. Los cálculos en la neurona se realizan en el contexto del anillo de los números reales $(\mathbb{R}, +, \times)$

sumando los productos de los valores de entrada con los pesos sinápticos. Generalmente se aplica una función de activación a la suma, la cual proporciona la no linealidad de la salida neuronal.

En contraste, los modelos neuronales morfológicos no están basados en el álgebra lineal sino en el álgebra reticular $(\mathbb{R}_{\pm}, \vee, \wedge, +)$. El elemento de procesamiento básico es también la neurona, o dendritas en el caso de los perceptrones morfológicos con estructuras dendríticas; y el cálculo consiste en máximos o mínimos de sumas.

En los modelos basados en álgebra lineal la función de activación f proporciona la no linealidad del modelo por lo que es indispensable en estos casos. Sin tal función, un modelo de este tipo se reduciría a un simple discriminador lineal con capacidades computacionales limitadas. Por otra parte, las Redes Neuronales Morfológicas (RNM) se basan en el álgebra reticular y las operaciones de máximo y mínimo son por sí mismas no lineales antes de la aplicación de una función de activación. La función de activación limitador duro se aplica solo para convertir el valor de entrada a la red a un valor de salida de 0 ó 1.

El cómputo en una red morfológica se define de la siguiente manera: dado un vector de entrada $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ el cálculo en una neurona morfológica es:

$$\tau_j(\mathbf{x}) = p_j \bigvee_{i=1}^{n} r_{ij}(x_i + w_{ij}) \tag{3.5}$$

o

$$\tau_j(\mathbf{x}) = p_j \bigwedge_{i=1}^{n} r_{ij}(x_i + w_{ij}) \tag{3.6}$$

donde $r_{ij} = \pm 1$ denota si la i-ésima neurona provoca excitación o inhibición en la j-ésima neurona y $p_j = \pm 1$ denota la respuesta de salida (de excitación o inhibición) de la j-ésima neurona a las neuronas a las cuales está conectada.

Después de calcular la entrada total $\tau_j(\mathbf{x})$, el valor de salida de la neurona j-ésima se obtiene mediante la aplicación de una función de activación $y_j = f(\tau_j(\mathbf{x}))$ que es generalmente la función limitador duro $f : \mathbb{R}_{\pm\infty} \to \{0, 1\}$:

$$f(a) = \begin{cases} 1 & \text{si } a \geq 0 \\ 0 & \text{si } a < 0 \end{cases} \tag{3.7}$$

En la figura 3.7 se muestra la comparación entre los perceptrones clásicos y los perceptrones morfológicos, como se puede observar, las principales diferencias son que el perceptrón clásico utiliza pesos multiplicativos, suma de productos (combinación lineal) y una función de activación no lineal mientras que el perceptrón morfológico se basa en

pesos aditivos, máximos o mínimos de sumas y su salida ya es no-lineal antes de aplicar la función de activación.

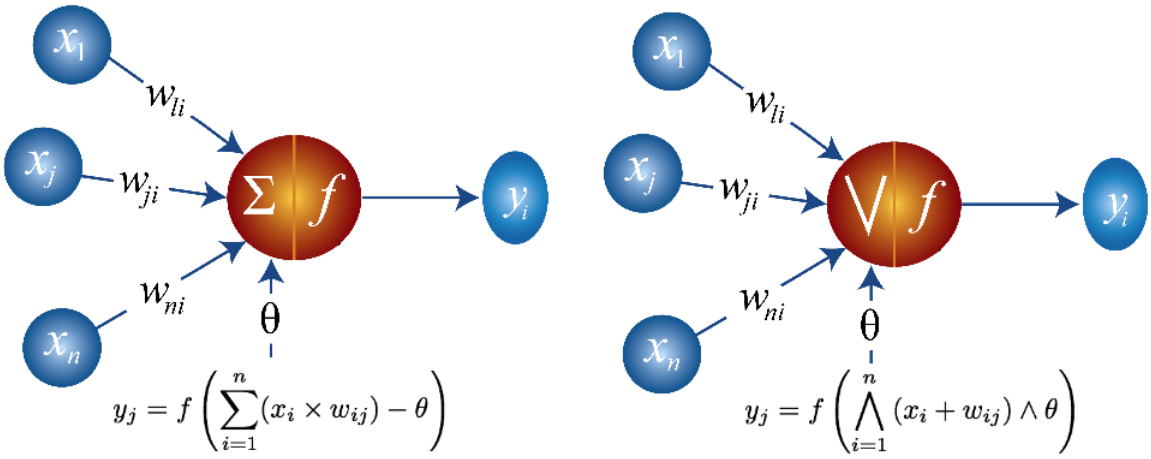

Figura 3.7: Comparación entre un perceptrón clásico y un perceptrón morfológico.

Comparativamente, los perceptrones clásicos y morfológicos presentan desempeños similares desde el punto de vista de la partición del espacio de características en problemas de clasificación. Sin embargo, el hecho de que las RNM no usen la multiplicación sino solo comparaciones (máximo o mínimo) y adición o substracción permite realizar implementaciones rápidas. Otra ventaja de las RNM son los algoritmos de entrenamiento eficientes que no presentan problemas de convergencia [14], [11], [13].

Recientes investigaciones en Neurocomputación proponen que la unidad computacional autónoma primaria en el cerebro capaz de realizar operaciones lógicas son las dendritas. Ritter y colegas agregaron una estructura dendrítica a los PM surgiendo un nuevo paradigma llamado Redes Neuronales Morfológicas con Procesamiento Dendral (RNMPD) que considera cálculos en las dendritas así como en el cuerpo de la neuronas [29]. En este trabajo de investigación se decidió utilizar las RNMPD debido a sus ventajas para usarlas en aplicaciones en tiempo real por lo que se dedicará el siguiente capítulo para explicar los conceptos fundamentales de este tipo de RNA.

3.1.5. Redes neuronales de base radial

Generalmente, las RNA se caracterizan por utilizar neuronas cuya salida es una función (normalmente no lineal) del producto escalar entre el vector de entradas y un vector de pesos. En las redes de base radial (Radial Basis Network, RBN) la activación de una neurona oculta está determinada por la distancia entre el vector de entrada a la red y un vector prototipo asociado a dicha neurona [51], [52]. La figura 3.8 muestra la arquitectura general de una RBN, se reciben los vectores de entrada y a través de funciones base (ϕ_i) se realiza una transformación no lineal del espacio de entradas hacia otro espacio llamado espacio de características, donde el problema puede ser linealmente separable.

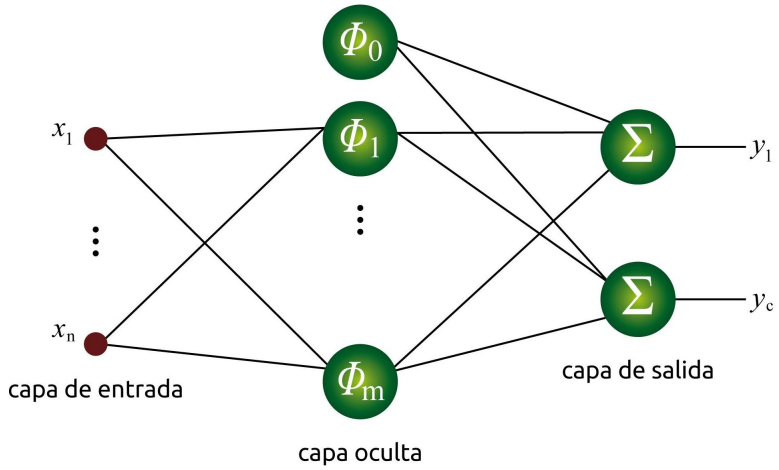

Figura 3.8: Arquitectura general de una red de base radial.

La función base puede ser cualquier función no lineal basada en una medida de distancia y debe cumplir dos propiedades: ser localizada y radialmente simétrica. La función Gaussiana cumple estas características y es la función de base radial más empleada; su ecuación es:

$$\phi_i(\mathbf{x}) = exp\left(-\frac{\|\mathbf{x}-\boldsymbol{\mu}_i\|^2}{2\sigma_i^2}\right) \tag{3.8}$$

donde $\boldsymbol{\mu}_i$ es el vector que determina el centro de la función base ϕ_i y σ_i es la desviación estándar, es importante mencionar que cada neurona tendrá sus propios parámetros.

El entrenamiento de una RBN se lleva a cabo en dos etapas. La primera se basa en técnicas de aprendizaje no supervisado, se divide el conjunto de datos de entrada en grupos o *clusters*, generalmente se utiliza el algoritmo *k-means* para este proceso. En la segunda etapa se aplica un método de mínimos cuadrados para encontrar el hiperplano que divide las clases, en [48] se recomienda utilizar mínimos cuadrados recursivos (*Recursive Least Squares*, RLS).

3.1.6. Mapas auto-organizados

Entre las RNA con aprendizaje no supervisado se encuentran los mapas auto-organizados. Los mapas auto-organizados, SOM (Self-Organizing Maps) o redes de Kohonen fueron propuestos en 1982 [53]. Este modelo es un tipo de red neuronal con un aprendizaje no supervisado competitivo y está formado por una malla de neuronas, generalmente bidimensional. Se trata de un sistema que permite visualizar e interpretar conjuntos de datos que se encuentran en un espacio de alta dimensionalidad convirtiendo las relaciones estadísticas entre los datos de entrada en relaciones geométricas en otro espacio de pocas

dimensiones, generalmente los vectores de entrada se representan mediante una matriz bidimensional.

La red auto-organizada determina rasgos comunes, correlaciones o categorías en los datos de entrada e incorpora esta información a su estructura interna de conexiones. Por tanto, se dice que las neuronas deben auto-organizarse en función de los estímulos (datos) procedentes del exterior. En el aprendizaje competitivo las neuronas compiten unas con otras por activarse, quedando finalmente una como neurona vencedora y desactivándose el resto. El objetivo de este aprendizaje es clasificar los datos similares de entrada en la misma categoría al activarse la misma neurona de salida. Las clases o categorías se determinan por la propia red a través de las correlaciones entre los datos de entrada.

Por tanto, durante el proceso de aprendizaje, el mapa auto-organizado emplea los datos de entrenamiento para adaptar su estructura mediante un proceso competitivo entre las neuronas; y una vez entrenado, cuando el sistema recibe un nuevo dato lo asocia a la neurona ganadora del mapa. Esta neurona es la que presenta la menor distancia respecto del dato recibido y la salida que produce es la salida de la red para esa muestra de entrada.

El algoritmo de aprendizaje de las redes de Kohonen se puede resumir en lo siguiente:

Algoritmo 3.3 Algoritmo de aprendizaje de mapas auto-organizados

1. Seleccionar al azar un vector $\mathbf{x}$ del conjunto de datos de entrenamiento y calcular su distancia a un vector de pesos de referencia, usando, por ejemplo, la distancia euclidiana.

2. Una vez que se ha encontrado el vector más próximo o BMU (*Best Matching Unit*) se actualiza el resto de los vectores de peso de referencia. El BMU y sus vecinos (en sentido topológico) se mueven cerca del vector $\mathbf{x}$ en el espacio de datos. La magnitud de dicha atracción depende de una tasa de aprendizaje.

3.1.7. Máquinas de vector soporte

Algunas técnicas utilizadas en IA se centran más en el método de entrenamiento o algoritmo de aprendizaje que en un modelo neuronal. Como consecuencia, la tendencia actual es distinguir entre: a) neurociencia computacional donde el énfasis está en los modelos de las neuronas biológicas, b) cómputo neuronal que tiene que ver con RNA basadas en abstracciones de la biología pero con aplicaciones tecnológicas y c) aprendizaje de máquina que se enfoca en la habilidad de un sistema para aprender independientemente de si el modelo tiene estructura neuronal o no. Entre estas últimas se encuentran las máquinas de vector soporte que en la actualidad es uno de los principales modelos utilizados en IA.

Las máquinas de vector soporte (Support Vector Machines, SVM) propuestas en 1995 [54], [55], [56] proporcionan un modo de encontrar modelos subyacentes a un conjunto de datos mediante un proceso de ajuste de parámetros. Las SVM mejoran el rendimiento en tareas de clasificación, ya que de todos los posibles hiperplanos que minimizan el error sobre el conjunto de entrenamiento, se elige aquél que presenta un mayor margen entre los datos y los hiperplanos. Este hecho se puede apreciar visualmente en la figura 3.9, donde se puede ver la recta discriminatoria entre ambas clases. También se presentan dos líneas discontinuas que pasan por los puntos de cada clase más cercanos a la recta de decisión. Estas rectas paralelas definen un margen entre dichos puntos. Intuitivamente se puede apreciar que la recta que presenta un margen mayor es la que producirá menos errores con nuevos datos, y por tanto tendrá un menor error de generalización.

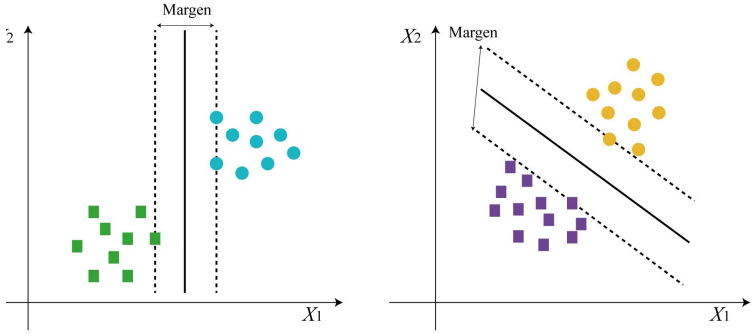

Figura 3.9: Ejemplos de los hiperplanos y del margen asociado.

Para encontrar el hiperplano con el mayor margen o distancia entre clases, se plantea un problema de optimización que busca:

$$Minimizar: \qquad \phi(\mathbf{w}) = \tfrac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$sujeto\ a: \qquad y^n\left(\mathbf{w}^T\mathbf{x}^n + b\right) - 1 \geq 0;\ \forall n = 1, ..., N$$

Este problema se resuelve mediante métodos de optimización de funciones.

Es importante mencionar que las SVM no pueden resolver problemas no lineales directamente sino en conjunto con otra técnica. Tales problemas se pueden resolver usando una función núcleo (una proyección no lineal). En el espacio de características y con una elección apropiada del núcleo, los datos pueden ser linealmente separables a pesar de no ser así en el espacio de entradas original, lo que permite que SVM pueda generar discriminadores lineales (hiperplanos) en el espacio de características. Cuando se hace el mapeo al espacio original, los discriminadores pueden ser curvas y otros separadores complejos, permitiéndole a la SVM resolver problemas de decisión no lineales. Además, un clasificador basado en SVM no maneja directamente el problema de clasificación de múltiples clases, entre las técnicas más utilizadas para resolver estos problemas están la de uno contra todos y uno contra uno.

3.2. Reconocimiento de Patrones

El reconocimiento de patrones es la disciplina científica que estudia cómo los sistemas artificiales pueden observar el entorno, obtener información y tomar decisiones acertadas acerca de la categoría o clase a la que pertenecen los objetos. Dependiendo de la aplicación, estos objetos pueden ser imágenes o formas de onda de señal o cualquier tipo de mediciones y se denotan usando el término genérico de patrones.

Aunque el ser humano realiza fácilmente actividades de reconocimiento como identificar el rostro de una persona, reconocer sonidos, olores, sabores, etc., los procesos involucrados para lograrlo son muy complejos. De ahí que emular esta capacidad humana para describir y clasificar objetos siga siendo un problema abierto y de gran interés en el campo de la IA.

En general, un sistema de reconocimiento de patrones implica esencialmente las siguientes etapas que se representan en la figura 3.10:

a) Adquisición de datos y pre-procesamiento: se obtienen los datos de entrada (patrones o muestras) que pueden ser señales, imágenes o bases de datos, por ejemplo, y se realizan procesos de acondicionamiento como eliminación de ruido, normalización, etc.

b) Representación de datos: se realiza una extracción de las características adecuadas que permitan identificar a un objeto de la mejor manera posible. Generalmente estas características se representan por medio de vectores y se conocen también como rasgos o atributos.

c) Toma de decisiones: esta etapa es la unidad principal de un sistema de reconocimiento de patrones y puede ser un proceso de: clasificación, *clustering* o de regresión.

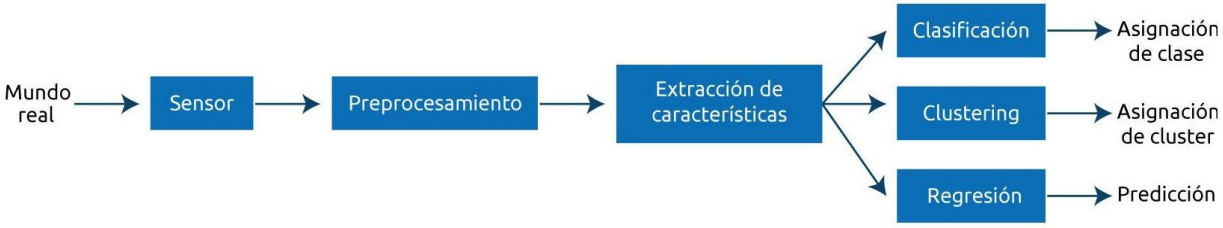

Figura 3.10: Diagrama general de un sistema de reconocimiento de patrones.

Antes de la década de 1960 la mayoría de la investigación teórica de reconocimiento de patrones era en el ámbito estadístico [57], [58], [59], [54], [56]. Sin embargo, los esfuerzos multidisciplinarios han impulsado nuevas ideas, metodologías y técnicas que han enriquecido el paradigma tradicional de reconocimiento de patrones, como las RNA y el aprendizaje automático [60], [61], [62]. Los enfoques más conocidos para la solución de

problema de reconocimiento de patrones son: 1) comparación de plantillas, 2) estadístico, 3) sintáctico o estructural, 4) difuso y 5) RNA:

1. Enfoque de comparación de plantillas: Uno de los primeros enfoques de reconocimiento de patrones se basa en la correspondencia (*matching*) de plantillas que consiste en tener un prototipo (típicamente una forma en 2D) del patrón a ser reconocido. El patrón que se quiere reconocer se compara con la plantilla almacenada teniendo en cuenta los posibles cambios de traslación, rotación y escala. Este método es computacionalmente exigente y aunque existen modelos deformables de plantillas [63], si los patrones presentan cambios que no se pueden modelar fácilmente no se obtendrán buenos resultados de reconocimiento.

2. Enfoque estadístico: El objetivo es elegir las características que permitan a los vectores de patrones pertenecientes a diferentes categorías ocupar regiones compactas y disjuntas, que permitan establecer límites de decisión para separar las diferentes clases. En este enfoque los límites de decisión se determinan por las distribuciones de probabilidad de los patrones que pertenecen a cada clase, que pueden ser especificadas o aprendidas [64], [57]. Algunos de los clasificadores estadísticos más comunes se basan en la regla de Bayes.

3. Enfoque sintáctico: En el paradigma sintáctico cada patrón se representa mediante un conjunto de primitivas o subpatrones y un conjunto de operadores. Se utiliza una gramática para generar una colección de oraciones o cadenas donde cada cadena corresponde a un patrón. El reconocimiento estructural de patrones es atractivo porque, además de la clasificación, este enfoque también proporciona una descripción de cómo se construye el patrón dadas las primitivas [65]. Aunque esta técnica es adecuada para tratar con patrones estructurados, los modelos aprendidos son muy sensibles al ruido.

4. Enfoque difuso: Kandel [66] afirma que existe una estrecha relación entre la teoría de conjuntos difusos y la teoría de reconocimiento de formas; se basa en el hecho de que la mayoría de las clases del mundo real son de naturaleza difusa y define diversas técnicas de reconocimiento difuso de patrones. En el planteamiento difuso se utiliza una medida de similitud basada en la distancia ponderada para obtener el grado de semejanza entre la descripción difusa de la forma desconocida y la forma de referencia.

5. Enfoque de redes neuronales: Las RNA se han utilizado con éxito para resolver problemas de reconocimiento de patrones; ya que ofrecen procedimientos flexibles para encontrar buenas soluciones basándose en el aprendizaje de una relación que transforma entradas en salidas deseadas dado un conjunto de ejemplos. Para la clasificación se utiliza el conocimiento codificado en los pesos y estructura de la RNA.

En este trabajo de investigación se resolverán problemas de reconocimiento de patrones donde la toma de decisiones se hará mediante un proceso de clasificación y se utilizará el

enfoque de RNA, específicamente se usarán RNM con procesamiento dendral para atacar el problema.

3.2.1. Clasificación

Para definir la clasificación de patrones se partirá de las siguientes definiciones:

Definición 3.2.1. Los rasgos son características, atributos o primitivas derivadas de los patrones, que se usan para su representación. Se define a n como el número de rasgos representados por números reales. Un patrón se define mediante un vector de rasgos $\mathbf{x} \in \mathbb{R}^n$:

$$\mathbf{x} = (x_1, x_2, ..., x_n)^T \tag{3.9}$$

El conjunto de rasgos se conoce como el espacio de representación.

Definición 3.2.2. Las clases o categorías son estados naturales de los objetos asociados con conceptos y se denotan como C^j donde $j = 1, 2, \ldots p$. El conjunto de todas las clases se denomina como espacio de interpretación.

Definición 3.2.3. Dado un patrón $\mathbf{x} = (x_1, x_2, ..., x_n)^T$ o una versión distorsionada $\tilde{\mathbf{x}}$, la clasificación consiste en determinar, de alguna manera, su clase de pertenencia $C^1, C^2, ..., C^p$. La clasificación realiza un mapeo del espacio de representación al espacio de interpretación.

En la etapa de clasificación de un sistema de reconocimiento de patrones se determina la categoría o clase a la que pertenece un objeto con base en los vectores de características que representan a los objetos. Dicha clasificación se puede realizar de manera supervisada o no supervisada.

Clasificación supervisada: La clasificación supervisada es aquella en el que cada patrón de entrada se identifica como miembro de una clase predefinida, por lo tanto, a cada vector de entrada se le debe asignar una etiqueta. El desempeño se obtiene al comparar la clase conocida de cada patrón contra la clase asignada por el clasificador.

Clasificación no supervisada: en la clasificación no supervisada el objetivo es encontrar el agrupamiento natural en un conjunto de patrones sin conocer su respectiva etiqueta de clase. La técnica que comúnmente se asocia al aprendizaje no supervisado es el *clustering*, que detecta posibles grupos con base en la similitud de los patrones no etiquetados de entrada. En la clasificación no supervisada, en algunas ocasiones, el número de clases debe aprenderse según la estructura del problema.

El conjunto de datos presentado a un sistema de reconocimiento de patrones se divide en dos:

1. Conjunto de entrenamiento: son los patrones de entrada al sistema que se representan por medio de un vector de características, atributos o rasgos. En cuanto a la elección inicial de características es necesario señalar que de la selección apropiada de rasgos va a depender en gran medida el éxito del clasificador, pues de esto dependerá la distribución de los patrones en el espacio de características que permitirá que los patrones de diferentes clases puedan separarse.

2. Conjunto de prueba: la eficacia del sistema se determina mediante la evaluación del conjunto de prueba que es independiente del conjunto de patrones de entrenamiento. Este comportamiento del sistema en el conjunto de prueba da una idea de la capacidad de generalización del sistema.

Por tanto, un sistema de reconocimiento opera en dos etapas:

1. Etapa de entrenamiento: esta etapa consiste en presentar al sistema el conjunto de patrones de entrenamiento con el objetivo de que se genere la separación de patrones de diferentes clases. Es decir, se realiza el proceso de aprendizaje que permite diseñar al clasificador.

2. Etapa de prueba: en esta etapa se le presenta al clasificador diseñado el conjunto de prueba con el fin de estimar su desempeño, esto se logra al comparar la clase conocida de cada patrón contra la clase asignada por el clasificador, en el caso de la clasificación supervisada, y obtener una tasa de error.

En la figura 3.11 se muestra el diagrama de las etapas de un sistema de reconocimiento de patrones que usa clasificación para la toma de decisiones.

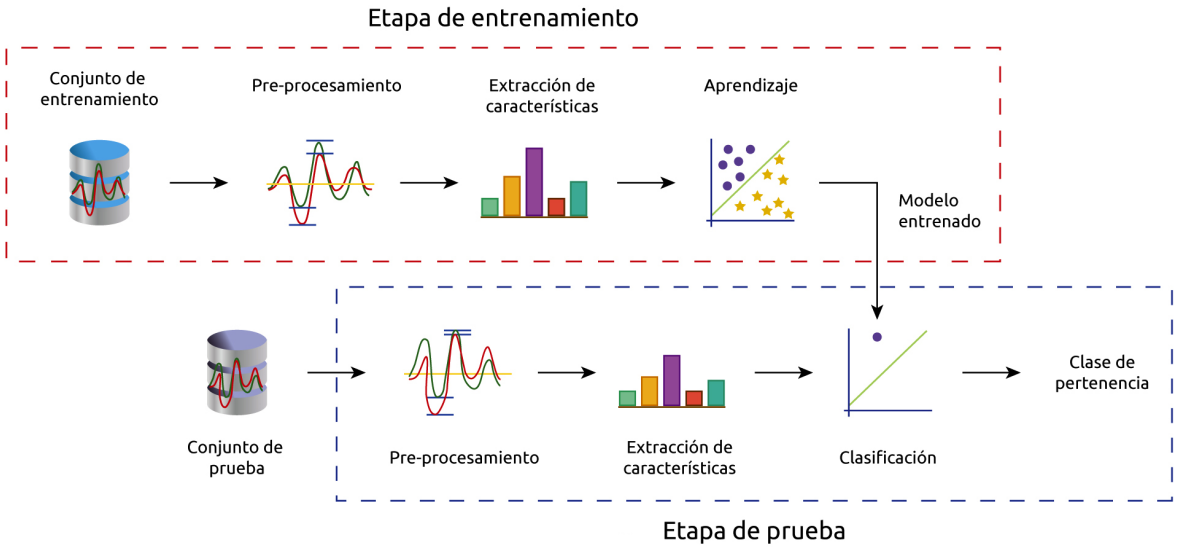

Figura 3.11: Diagrama de un sistema de reconocimiento de patrones usando clasificación para la toma de decisiones.

En la práctica los sistemas de clasificación no son perfectos y los errores se pueden deber a que:

- Las características utilizadas para la representación son inadecuadas o insuficientes.
- Las muestras de entrenamiento utilizadas para diseñar el clasificador no son suficientemente representativas, por lo que el clasificador no es capaz de encontrar la separación de las clases.
- Hay una superposición intrínseca de las clases que ningún clasificador puede resolver.

3.2.1.1. Validación experimental

Para medir el rendimiento alcanzado por un algoritmo de aprendizaje supervisado se utiliza un conjunto de prueba y el rendimiento se determina mediante la comparación de las etiquetas obtenidas con las etiquetas verdaderas. En ciertas bases de datos ya se tiene un conjunto de entrenamiento y un conjunto de prueba dados. Otras veces, simplemente se tienen ejemplos etiquetados, en este caso se tiene que dividir la base de datos en dos subconjuntos independientes: entrenamiento y prueba; para esto se pueden seguir varios criterios [67]:

- Una regla común es dividir aleatoriamente el conjunto de patrones en dos subconjuntos, generalmente el 70 % de la base de datos se usa para el entrenamiento y el 30 % para la prueba, a esto se le conoce como *hold-out*.
- Con el fin de garantizar que los dos subconjuntos sean muestras aleatorias con la misma distribución se utiliza el muestreo estratificado que asegura que cada clase está presente en la misma proporción en los subconjuntos de entrenamiento y prueba. Un ejemplo de muestreo estratificado es la validación cruzada *k-folds*, en la que el conjunto de entrenamiento se divide aleatoriamente en k conjuntos disjuntos de igual tamaño, donde cada parte tiene más o menos la misma distribución de clases. El clasificador se entrena k veces, cada partición se utiliza para prueba exactamente una vez y las particiones restantes $k-1$ se usan para entrenamiento. Al repetir la prueba k veces, cada vez se toma un conjunto de prueba distinto. El error estimado es el promedio de estos k errores, en lugar de la media se puede usar la estimación de intervalos que emplea la distribución de probabilidad.
- Un caso extremo de la validación *k-folds* es *leave-one-out*, con $k = n$, donde n es el número total de muestras en el conjunto de entrenamiento. Por tanto, se realizan n experimentos con $n-1$ muestras para el entrenamiento y la muestra restante para la prueba. Este método es muy costoso computacionalmente y no garantiza estratificación.

- Una alternativa a la validación cruzada es el *bootstrap* que utiliza el muestreo con reemplazo para formar el conjunto de entrenamiento y el conjunto original de datos se usa como el conjunto de prueba.

Una de las funciones de criterio más utilizadas para evaluar el desempeño de un clasificador es la precisión o tasa de error. La precisión es el porcentaje de clasificaciones correctas y la tasa de error es el porcentaje de clasificaciones incorrectas, es decir, precisión = 1 - tasa de error. Este criterio asume que se tiene una distribución de las clases relativamente uniforme, por lo que si esto no ocurre en el experimento no es un buen criterio para evaluar el desempeño del sistema. En estos casos se usa información de la llamada matriz de confusión o tabla de contingencia que se muestra en la figura 3.12, donde las filas de la matriz representan los valores de predicción, mientras que las columnas representan los valores reales. Por tanto, se comparan los valores reales con los de predicción resultando las siguientes categorías: falso positivo, verdadero positivo, falso negativo y verdadero negativo; que permiten evaluar rápidamente los resultados de clasificación.

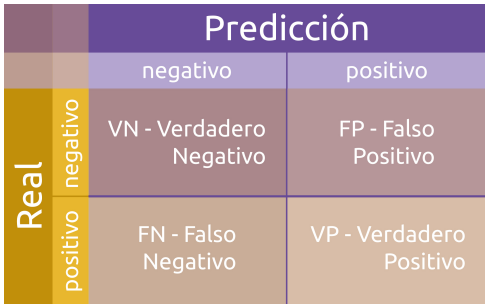

Figura 3.12: Matriz de confusión.

De acuerdo con lo establecido en el teorema de *No Free Lunch* [68], no existe "el mejor algoritmo de aprendizaje". Para cualquier algoritmo, hay un conjunto de datos donde es muy preciso y otro conjunto de datos donde es deficiente. Entonces solo se puede decir que un algoritmo de aprendizaje es bueno para ciertos problemas en los que sus resultados coinciden con las propiedades de los datos con los que se está trabajando.

Por tanto, se debe recalcar que la conclusión que se obtiene del análisis de resultados está condicionada al conjunto de datos que se tiene, es decir, no se realizan comparaciones de algoritmos de aprendizaje de una manera independiente del dominio sino en alguna aplicación en particular. El resultado únicamente es cierto para la aplicación particular y en la medida en que el problema esté bien representado por la muestra de entrada.

3.3. Resumen

En esta sección se presenta la base teórica necesaria para explicar el desarrollo de la investigación, así como los experimentos realizados sobre los que se sustenta esta te-

sis. Se puntualizan brevemente los conceptos básicos de RNA, y se describen la TLU, el perceptrón clásico, MLP, RBN, RNM, mapas auto-organizados y la técnica de SVM. Asimismo, se explica en qué consiste el reconocimiento de patrones, específicamente usando la clasificación para la toma de decisiones y se recapitulan algunas técnicas de validación experimental que servirán para determinar el desempeño del algoritmo de entrenamiento propuesto.

Capítulo 4

Redes neuronales morfológicas con procesamiento dendral

Las Redes Neuronales Morfológicas con Procesamiento Dendral (RNMPD) son redes neuronales que constan de una capa de elementos de procesamiento donde cada elemento es una neurona equipada con dendritas y realiza operaciones morfológicas basadas en el álgebra reticular. La capa de procesamiento es también la capa de salida de la red. Análogamente a otros modelos de redes neuronales, la capa de entrada no se cuenta convencionalmente como una capa por sí misma, porque ésta solo tiene el propósito de alimentar a las neuronas de procesamiento con los componentes del vector de entrada.

Las estructuras dendríticas representan los procesos neuronales específicos que dan al perceptrón morfológico sus nuevas capacidades computacionales. Una RNM de una capa es capaz de resolver problemas de clasificación no lineales cuando sus neuronas tienen dendritas mientras que los perceptrones clásicos requieren de múltiples capas para lograr la misma tarea.

Las RNMPD intentan modelar redes neuronales artificiales con un mayor parecido a las redes neuronales biológicas, por lo que en este intento no se puede ignorar a las dendritas que constituyen el componente más importante, tanto en superficie como en volumen del cerebro. El número de sinapsis en una sola neurona en la corteza cerebral oscila entre 500 a 200,000 y la mayor parte de las sinapsis se producen en el árbol dendrítico de la neurona, donde se procesa información [69], [70]. Además, las dendritas son subunidades funcionales capaces de implementar operaciones lógicas, como AND, OR, NOT y XOR [71], [72], [73], [69], [70], [74], [47]. Una explicación más detallada de la importancia de las dendritas biológicas se puede encontrar en [75], [76], [77].

Por todas estas características de las dendritas es conveniente incluirlas en un modelo de RNA como lo hacen las RNMPD propuestas por Ritter y colegas [29]. Este modelo cuenta con neuronas con procesamiento dendrítico. Para modelar los cálculos dendrales se

usan, al igual que en las RNM, las operaciones $\vee$ (máximo) o $\wedge$ (mínimo) y $+$ (suma), de los semianillos $(\mathbb{R}_{-\infty}, \vee, +)$ o $(\mathbb{R}_{\infty}, \wedge, +)$.

Una RNMPD de una capa consta de un conjunto de n neuronas de entrada $N_1, \ldots, N_n$, donde n es el número de atributos de cada patrón, m neuronas de salida $M_1, \ldots M_m$ donde m es el número de clases y un número finito de dendritas D. Al igual que con las estructuras neuronales biológicas y en contraste con el modelo clásico de un perceptrón de una sola capa, una neurona de salida puede tener múltiples sitios sinápticos de contacto de la misma neurona de entrada.

En el modelo, la neurona de salida M_j recibe la información de las neuronas de entrada $\{N_i\}_{i\in I}$, $I = \{1, 2, \cdots, n\}$ donde n es el número de atributos de cada patrón de entrada, a través de un número finito de dendritas D_{jk} donde j se refiere a la $j-$ésima clase y k a la $k-$ésima dendrita. Cada neurona de entrada N_i tendrá a lo más dos conexiones a una dendrita dada D_{jk}. Si existen dos conexiones entonces una será excitadora y la otra inhibitoria.

La arquitectura de una RNMPD se presenta en la figura 4.1, en esta y en todas las ilustraciones siguientes, las entradas excitadoras se representan con círculos llenos y las inhibitorias con círculos vacíos.

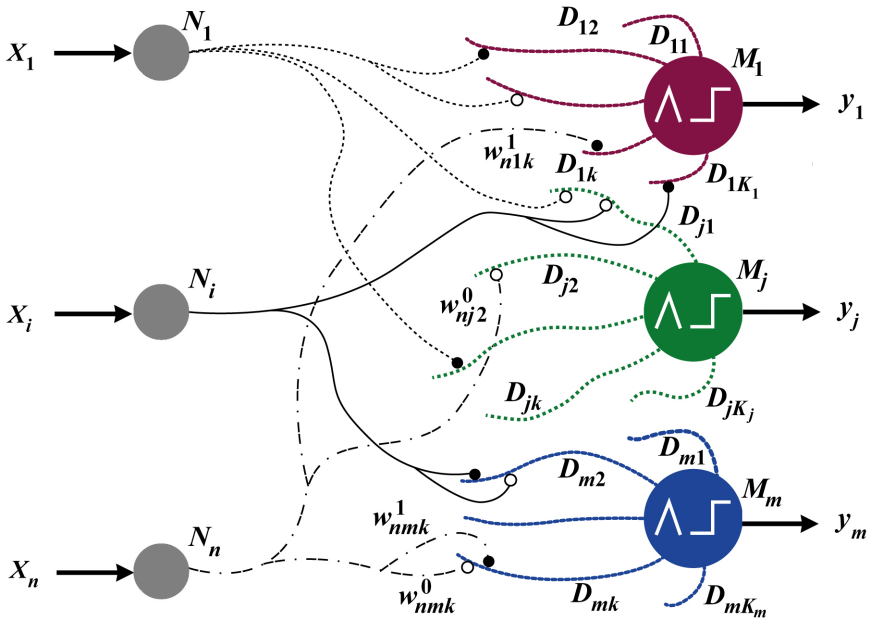

Figura 4.1: Arquitectura de una RNMPD.

El peso de la rama axonal de la neurona N_i que termina en la dendrita D_{jk} de la neurona de salida M_j se denota como w^l_{ijk}, donde $l \in \{0, 1\}$ distingue entre una entrada excitadora $(l = 1)$ y una entrada inhibitoria $(l = 0)$. La respuesta de entrada de la rama w^l_{ijk} se denota por $r^l_{ik} = \pm 1$, donde $r^1_{ik} = +1$ representa excitación y $r^0_{ik} = -1$ denota inhibición. Dado

que $r_{ik}^{l} = (-1)^{1-l}$ se puede hacer el reemplazo para simplificar la notación. Por tanto, el cálculo de la dendrita D_{jk} es:

$$\begin{aligned} \tau_k^j(\mathbf{x}) &= p_{jk} \bigwedge_{i \in I(k)} \bigwedge_{l \in L(i)} r_{ik}^{l} \left(x_i + w_{ijk}^{l}\right) \\ &= p_{jk} \bigwedge_{i \in I(k)} \bigwedge_{l \in L(i)} (-1)^{1-l} \left(x_i + w_{ijk}^{l}\right) \end{aligned} \tag{4.1}$$

donde $\tau_k^j(x)$ denota la salida de la dendrita D_{jk}; $\mathbf{x} = (x_1, \ldots, x_n)$ representa el valor de entrada a las neuronas $N_1, N_2, \ldots, N_n$; $I(k) \subseteq \{0, 1\}$ corresponde al conjunto de todas las neuronas de entrada con fibras terminales que tienen sinapsis en la $k-$ésima dendrita D_{jk} de M_j; $L(i) \subseteq \{0, 1\}$ corresponde al conjunto de fibras excitadoras o inhibitorias de N_i que tienen sinapsis con D_{jk}; y $p_{jk} \in \{-1, 1\}$ denota la respuesta excitadora o inhibitoria de la dendrita D_{jk} a la entrada recibida.

Cuando se compara con la ecuación 3.6, que describe al perceptrón morfológico, se tiene que la ecuación 4.1 presenta un operador mínimo adicional $\bigwedge_{l \in L(i)}$, que representa el cálculo realizado en la dendrita D_{jk} y es específico para el modelo del perceptrón morfológico con procesamiento dendral.

Mediante el uso de pesos $w_{ijk}^{1} = +\infty$ para las fibras excitadoras y $w_{ijk}^{0} = -\infty$ para las inhibitorias, se pueden sustituir las conexiones sinápticas que falten en la red de manera que todas las neuronas de entrada tengan sinapsis con las dendritas de una neurona de salida. En este caso, $I(k) = \{1, \ldots, n\}$ y la ecuación 4.1 se convierte en:

$$\tau_k^j(\mathbf{x}) = p_{jk} \bigwedge_{i=1}^{n} \bigwedge_{l \in L(i)} (-1)^{1-l} \left(x_i + w_{ijk}^{l}\right) \tag{4.2}$$

Además, cuando cada conexión sináptica de una neurona de entrada a una dendrita utiliza ambas fibras terminales, es decir, cuando la red está totalmente conectada desde la capa de entrada a la capa de salida, la ecuación 4.1 se convierte en:

$$\tau_k^j(\mathbf{x}) = p_{jk} \bigwedge_{i=1}^{n} \bigwedge_{l=0}^{1} (-1)^{1-l} \left(x_i + w_{ijk}^{l}\right) = p_{jk} \bigwedge_{i=1}^{n} \left(x_i + w_{ijk}^{1}\right) \wedge - \left(x_i + w_{ijk}^{0}\right) \tag{4.3}$$

El valor de entrada de la red $\tau_k^j(\mathbf{x})$ que se calcula en la dendrita D_{jk} de acuerdo con la ecuación 4.1 se pasa entonces al soma de las neuronas M_j y el estado de M_j es una función de la entrada recibida de todas sus dendritas. Por tanto, la respuesta de la neurona M_j,

$\tau^j(x)$ se define mediante la ecuación:

$$\tau^j(\mathbf{x}) = p_j \bigvee_{k=1}^{K_j} \tau_k^j(\mathbf{x}) \tag{4.4}$$

o

$$\tau^j(\mathbf{x}) = p_j \bigwedge_{k=1}^{K_j} \tau_k^j(\mathbf{x}) \tag{4.5}$$

donde K_j representa el número total de dendritas en la neurona M_j; $\tau_k^j(x)$ es la salida de la dendrita k de la neurona M_j calculada usando la ecuación 4.1; $p_j = 1$ denota que la entrada particular es aceptada y $p_j = 0$ indica que la entrada particular es rechazada.

La salida de la neurona, como en los perceptrones morfológicos, se determina mediante la función de activación limitador duro:

$$y_j(\mathbf{x}) = f\left(\tau^j(\mathbf{x})\right) = \begin{cases} 1 & si\ \tau^j(\mathbf{x}) \geq 0 \\ 0 & si\ \tau^j(\mathbf{x}) < 0 \end{cases} \tag{4.6}$$

Combinando las ecuaciones 4.1, 4.4 y 4.6 en una relación, el cálculo total de la neurona de salida M_j está definido por:

$$y_j(\mathbf{x}) = f\left(p_j \bigvee_{k=1}^{K_j} \left[p_{jk} \bigwedge_{i \in I(k)} \bigwedge_{l \in L(i)} (-1)^{1-l}\left(x_i + w_{ijk}^l\right)\right]\right) \tag{4.7}$$

En el caso de una red totalmente conectada, es decir, cuando todas las neuronas de entrada tienen una sinapsis a través de dos fibras axonales en cada una de las K_j dendritas de una neurona de salida M_j, el estado de salida de la neurona M_j a partir de la ecuación 4.7 se convierte en:

$$y_j(\mathbf{x}) = f\left(p_j \bigvee_{k=1}^{K_j} \left[p_{jk} \bigwedge_{i=1}^{n} \left(x_i + w_{ijk}^1\right) \wedge - \left(x_i + w_{ijk}^0\right)\right]\right) \tag{4.8}$$

El papel de cada dendrita es habilitar el procesamiento de la neurona para reconocer una región del espacio de patrones euclidiano. Cuando se tienen todas las conexiones sinápticas de las neuronas de entrada a una dendrita específica y todos los pesos son

finitos tanto para fibras excitadoras como inhibitorias, la región en el espacio de patrones que se calcula con la dendrita es un hipercubo cerrado. En esta investigación nos centramos en este tipo de redes, totalmente conectadas y de pesos finitos.

Por tanto, la interpretación geométrica del cálculo realizado por una dendrita es que cada dendrita por medio de los pesos w_{ijk} permite a la neurona reconocer una región en el espacio de patrones, la cual está delimitada por hasta $2n$ hiperplanos, donde n es la dimensionalidad del espacio. Así, esta región representa un intervalo en $\mathbb{R}^1$, un rectángulo en $\mathbb{R}^2$ y un paralelepípedo en $\mathbb{R}^3$. En $\mathbb{R}^n$ para $n > 3$, la región representa un hipercubo cuyas caras son todas perpendiculares a los ejes del sistema de coordenadas.

Los pesos w^l_{ijk} asociados a cada conexión definirán las fronteras del hipercubo sobre cada eje i. El valor más bajo de la frontera será asignado a $-w^1_{ijk}$, mientras que el valor más alto será asignado a $-w^0_{ijk}$. La figura 4.2 muestra un ejemplo de asignación de pesos en el caso de un rectángulo, $n = 2$.

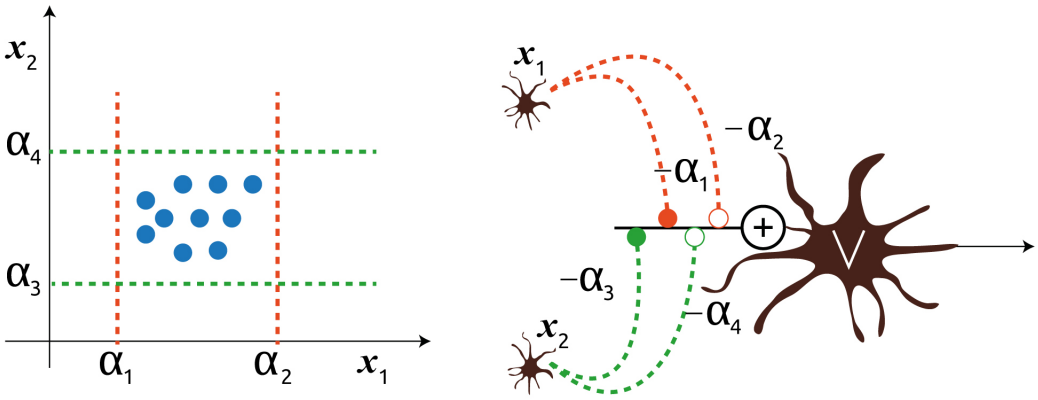

Figura 4.2: Interpretación geométrica de los pesos en una RNMPD.

Geométricamente, la neurona de la figura 4.2 se disparará para todos los valores de entrada que cumplan que $x_1 \in [\alpha_1, \alpha_2]$ y $x_2 \in [\alpha_3, \alpha_4]$.

4.1. Ejemplos numéricos

Después de haber explicado el modelo de las RNMPD se presentan algunos ejemplos que ilustran su funcionamiento.

Ejemplo 1

Considérese el PMPD que se muestra en la figura 4.3 que consta solo de una neurona de entrada, la cual recibe la entrada x, conectada a una neurona de salida a través de una

respuesta excitadora ($r_1 = 1$). El peso de esta conexión se denota como $w = -\alpha, \alpha \in \Re$, entonces cualquier valor de x en el conjunto $[\alpha, +\infty)$ activará a la neurona.

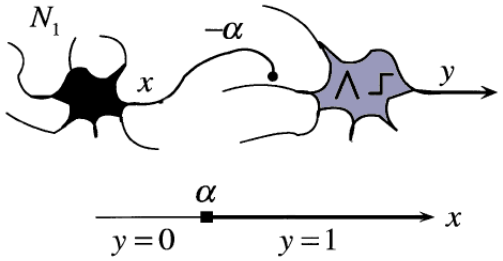

Figura 4.3: Ejemplo 1 (adaptado de [15]).

De acuerdo con las ecuaciones 4.1 y 4.6 para que la neurona se active se debe cumplir que:

$\tau(x) = r_1(x + w) \geq 0$, entonces $x \geqslant -w = \alpha$.

Geométricamente, la neurona se disparará para todos los valores del eje x que sean mayores o iguales a α.

Ejemplo 2

Los valores que disparan a una neurona se pueden delimitar usando una respuesta inhibitoria. Considérese el PMPD que aparece en la figura 4.4.

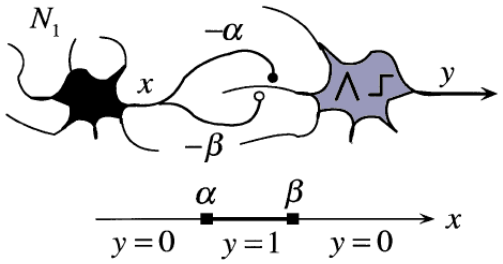

Figura 4.4: Ejemplo 2 (adaptado de [15]).

En este caso se tiene una neurona con dos conexiones cuyos pesos son $w_1 = -\alpha$, $w_2 = -\beta$ donde $\alpha < \beta$, la rama de w_1 tiene una respuesta excitadora ($r_1 = 1$) y la rama de w_2 tiene una respuesta inhibitoria ($r_2 = -1$). Por tanto, la neurona disparará cuando:

$\tau(x) = p \bigwedge_{i=1}^{2} r_i(x + \omega_i)$, donde $p = 1$

$\tau(x) = min\{r_1(x + w_1), r_2(x + w_2)\} \geq 0$

$\tau(x) = min\{(x - \alpha), -(x - \beta)\} \geq 0$

Esta condición se satisface solo por los puntos en el intervalo $[\alpha, \beta]$. Si en lugar de $p = 1$ se tiene $p = -1$ entonces se intercambia la región de encendido, en este caso la neurona se activa solamente cuando $x \leq \alpha$ o $x \geq \beta$.

Ejemplo 3

Sea el PMPD que se muestra en la figura 4.5b) el cual define la región que aparece en 4.5a). Esta región está determinada por los dos puntos extremos α y β, entonces $C^1 = \{x : x \in \mathbb{R}^2, \alpha \leq \mathrm{x} \leq \beta\}$. Como se puede notar en la figura 4.5b), la neurona de entrada N_2 no tiene conexión a la neurona de salida M. Por lo tanto, los índices de las fibras terminales que hacen sinapsis en las dendritas de M son $L_1 = \{0, 1\}$ y $L_2 = \emptyset$.

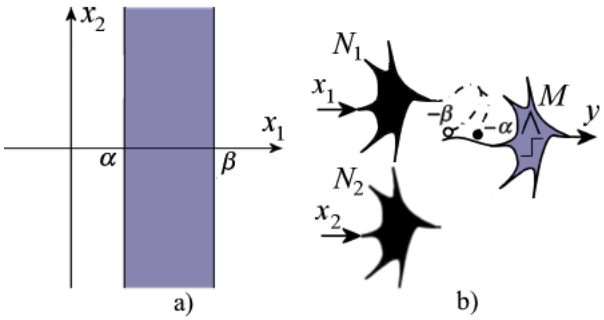

Figura 4.5: Ejemplo 3 (adaptado de [15]).

En este caso, aplicando la fórmula 4.1 se tiene que:

$$\tau(\mathbf{x}) = \bigwedge_{i=1}^{2} \bigwedge_{l \in L_i} (-1)^{1-l} \left(x_i + w_i^l\right)$$
$$= [(x_1 + w_1^1) \wedge -(x_1 + w_1^0)] \wedge [(x_2 + w_2^1) \wedge -(x_2 + w_2^0)]$$

Como en este caso no se tienen conexiones de la neurona N_2 entonces sus pesos se consideran como $w_2^1 = +\infty$ y $w_2^0 = -\infty$ (ver propiedades de los operadores reticulares en el apéndice C). Por tanto:

$$\tau(\mathbf{x}) = [(x_1 - \alpha) \wedge -(x_1 - \beta)] \wedge [(x_2 + \infty) \wedge -(x_2 - \infty)]$$
$$= [(x_1 - \alpha) \wedge -(x_1 - \beta)] \wedge [+\infty \wedge -(-\infty)] = (x_1 - \alpha) \wedge -(x_1 - \beta)$$

En el espacio de patrones, como el PMPD no tiene fibras axonales de N_2, la región de decisión es infinito en ambas direcciones en el eje de coordenadas x_2.

Ejemplo 4.

Finalmente se resolverá el problema de la XOR de dos entradas, el cual para ser resuelto por un perceptrón clásico se requiere de una capa oculta que no es necesaria con el PMPD. En este problema se tienen dos clases: $C^1 = \{(0, 1), (1, 0)\}$ y $C^2 = \{(0, 0), (1, 1)\}$. La figura 4.6a) muestra la representación gráfica en $\mathbb{R}^2$ y la figura 4.6b) presenta el PMPD

que resuelve este problema.

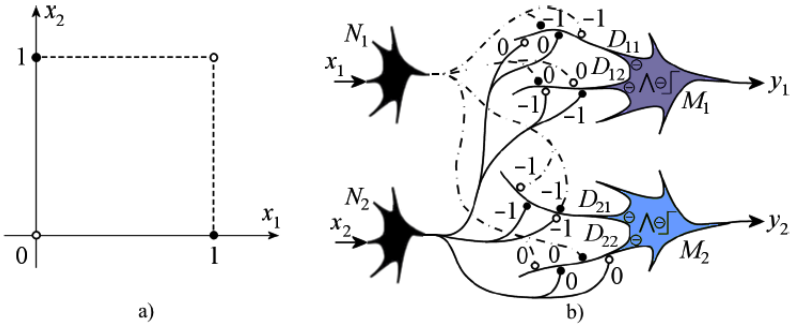

Figura 4.6: Ejemplo 4 (adaptado de [15]).

Como se puede observar, en este caso se tienen 4 dendritas D_{11}, D_{12}, D_{21} y D_{22}. En la tabla 4.1 se presentan los parámetros de la red, las respuestas de todas las dendritas y de las neuronas de salida son inhibitorias.

Tabla 4.1: Pesos sinápticos del PMPD que resuelve el problema XOR de dos entradas.

D_{jk}	w^1_{1jk}	w^0_{1jk}	w^1_{2jk}	w^0_{2jk}
D_{11}	-1	-1	0	0
D_{12}	0	0	-1	-1
D_{21}	-1	-1	-1	-1
D_{22}	0	0	0	0

La salida de la red es $\mathbf{y} = \left(f\left[\tau^1\left(\mathbf{x}\right)\right], f\left[\tau^2\left(\mathbf{x}\right)\right]\right)$ donde:

$$\begin{gathered}\tau^1\left(\mathbf{x}\right) = -\left[\tau^1_1\left(\mathbf{x}\right) \wedge \tau^1_2\left(\mathbf{x}\right)\right] = -\left(-\left[(x_1-1) \wedge -(x_1-1) \wedge (x_2-0) \wedge -(x_2-0)\right]\right. \\ \left. \wedge -\left[(x_1-0) \wedge -(x_1-0) \wedge (x_2-1) \wedge -(x_2-1)\right]\right) = \\ \left[(x_1-1) \wedge -(x_1-1) \wedge x_2 \wedge -x_2\right] \wedge \left[x_1 \wedge -x_1 \wedge (x_2-1) \wedge -(x_2-1)\right]\end{gathered}$$

$$\begin{gathered}\tau^2\left(\mathbf{x}\right) = -\left[\tau^2_1\left(\mathbf{x}\right) \wedge \tau^2_2\left(\mathbf{x}\right)\right] = -\left(-\left[(x_1-1) \wedge -(x_1-1) \wedge (x_2-1) \wedge -(x_2-1)\right]\right. \\ \left. \wedge -\left[(x_1-0) \wedge -(x_1-0) \wedge (x_2-0) \wedge -(x_2-0)\right]\right) = \\ \left[(x_1-1) \wedge -(x_1-1) \wedge (x_2-1) \wedge -(x_2-1)\right] \wedge \left[x_1 \wedge -x_1 \wedge x_2 \wedge -x_2\right]\end{gathered}$$

Entonces:

$\tau^1\left(\mathbf{x}\right) \geq 0 \iff \mathbf{x} = (0,1)$ ó $\mathbf{x} = (1,0)$ y

$\tau^2\left(\mathbf{x}\right) \geq 0 \iff \mathbf{x} = (0,0)$ ó $\mathbf{x} = (1,1)$

Las dos clases que reconoce el PMPD constan exactamente de un par de puntos cada una, por lo que la salida es la unión de dos regiones cada una compuesta de un único punto en el plano $\mathbb{R}^2$.

Cuando el vector de entrada x tiene n dimensiones, y por tanto, se necesitan n neuronas de entrada, la neurona se disparará cuando x se encuentra en el hipercubo creado por las respuestas excitadoras e inhibitorias de las dendritas de la neurona de salida M_j.

4.2. Algoritmos de entrenamiento

Las características geométricas de la RNMPD de una capa se usan para definir estrategias de partición del espacio de patrones y para el desarrollo de algoritmos de entrenamiento. Los algoritmos de entrenamiento de las RNMPD generan dendritas dinámicamente a partir de los patrones del conjunto de entrenamiento por lo que la estructura de la red no se elige a priori, esto es una ventaja ya que no hay necesidad de refinar la topología de la red después de la evaluación de su rendimiento en tiempo de ejecución, como es usualmente el caso con los MLP tradicionales.

Es importante recalcar que el aprendizaje es supervisado. La estrategia utilizada para determinar la región que encierra a los patrones de una sola clase es lo que hace diferentes a los métodos de entrenamiento de las RNMPD. De manera general existen dos enfoques de entrenamiento: eliminación y unión.

4.2.1. Algoritmos de entrenamiento de eliminación y unión

La estrategia de entrenamiento de eliminación inicia con una región que es lo suficientemente grande para contener todos los patrones de entrenamiento de una clase, seguido de la eliminación de los patrones ajenos (patrones de otra u otras clases) que se encuentran en esa región. La eliminación se logra mediante la construcción de hipercubos más pequeños alrededor de los patrones ajenos y el establecimiento de los parámetros de la red de tal manera que estas regiones se excluyan. La eliminación se realiza mediante el cálculo de la intersección de las regiones reconocidas por las dendritas (ecuación 4.5).

Por otra parte el enfoque de entrenamiento de unión consiste en crear una región pequeña alrededor de cada patrón, después se identifican los hipercubos que están cerca de acuerdo con una medida de distancia y los hipercubos con patrones de la misma clase se unen en regiones más grandes que evitan incluir patrones de otras clases. El entrenamiento se termina cuando las regiones unidas solo tienen elementos de una misma clase. La fusión se lleva a cabo mediante el cálculo de la unión de las regiones reconocidas por las dendritas (ecuación 4.4).

A continuación se presentan los algoritmos de los métodos de entrenamiento de eliminación y unión:

Algoritmo 4.1 Entrenamiento de eliminación de una RNMPD de una capa [15]

Dadas dos clases de patrones, C^0 y C^1:

1. Calcular los valores que generan al hipercubo que cubre a todos los patrones x^ξ de la clase C^1 y obtener la primera dendrita.
2. Obtener la respuesta de la dendrita actual mediante la ecuación 4.1.
3. Calcular la respuesta total de la neurona de salida N usando la ecuación 4.5.
4. Verificar si con la dendrita generada se obtiene un aprendizaje exitoso, usando la función limitador duro. Si esto es cierto, entonces se genera la RNMPD con los pesos calculados y se concluye el algoritmo de entrenamiento. Si no es cierto, entonces el algoritmo continúa con los siguientes pasos para la obtención de las siguientes dendritas.
5. Agregar una nueva dendrita a la neurona, el conjunto inicial tiene a todos los patrones encerrados por el hipercubo obtenido en el paso 1.
6. Seleccionar aleatoriamente un patrón de la clase C^0 que está clasificado incorrectamente $\mathbf{x}^\gamma$.
7. Calcular la distancia mínima de Chebyshev del patrón $\mathbf{x}^\gamma$, seleccionado en el paso 6, a los patrones x^ξ de la clase C^1. La distancia de Chebyshev entre dos patrones n-dimensionales x^ξ y x^γ se define como $d\left(x^\xi, x^\gamma\right) = max_{i=1,\ldots n}\left|x_i^\xi - x_i^\gamma\right|$.
8. Mantener los índices y las coordenadas de los patrones en el conjunto inicial pertenecientes a la clase C^1 que están en el borde del hipercubo centrado en $\mathbf{x}^\gamma$.
9. Asignar los pesos a la dendrita actual que proporciona la clasificación correcta de los patrones mal clasificados.
10. Actualizar los índices de los conjuntos I y L, de manera que solo se tengan los patrones mal clasificados.
11. Conservar en el conjunto inicial únicamente a los patrones $\mathbf{x}^\gamma$ que no pertenecen a la nueva región creada en el paso 9.
12. Verificar si no hay necesidad de generar más dendritas, si ya no hay patrones mal clasificados regresar al paso 2 y si todavía existen patrones clasificados erróneamente regresar al paso 6.

Algoritmo 4.2 Entrenamiento de unión de una RNMPD de una capa [33]

Dadas dos clases de patrones, C^0 y C^1:

1. Calcular la distancia d_{min} como la distancia de intersección mínima de Chebyshev entre las clases C^0 y C^1. La distancia de Chebyshev entre dos patrones n-dimensionales x^ξ y x^γ se define como $d\left(x^\xi, x^\gamma\right) = max_{i=1,\dots n}\left|x_i^\xi - x_i^\gamma\right|$.

2. Inicializar un contador de dendritas $K = 0$ y establecer todos los patrones en C^1 como no marcados.

3. Seleccionar un patrón no marcado x^ξ que pertenece a C^1 y marcarlo, esto se puede hacer eligiendo el primer patrón encontrado en el conjunto de entrenamiento o al azar.

4. Para cada patrón no marcado x^ζ situarse en la vecindad de x^ζ, i.e. para cada x^ζ tal que $d\left(x^\zeta, x^\xi\right) < d_{min} + d_\varepsilon$, donde un posible valor es $d_\varepsilon = \frac{1}{2}d_{min}$, ir al paso 5.

5. Identificar una región en el espacio de patrones que podría conectar a los patrones x^ξ y x^ζ. Si la región identificada de unión no contiene patrones de C^0, entonces incrementar K y generar una nueva dendrita de excitación D_K. En otro caso, si existe al menos un patrón de C^0 cercano a x^ζ no generar una dendrita en este paso.

6. Si no se generó una dendrita en el paso 5, entonces incrementar K y generar una dendrita de excitación que reconoce una región aislada alrededor de x^ζ. El tamaño de esta región deber ser menor que d_{min} en cada coordenada $i = 1, \dots, n$ para asegurar que no tocará a los patrones de la clase C^0.

7. Si aún hay patrones no marcados de C^1 repetir a partir del paso 3, en otro caso terminar.

En la figura 4.7 se muestra un ejemplo del resultado de aplicar los métodos de entrenamiento de eliminación y unión a un problema dado de clasificación de dos clases, la clase C^1 representada por lo círculos negros y la clase C^0 por los círculos vacíos.

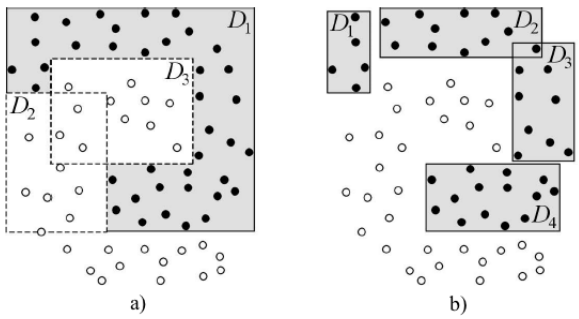

Figura 4.7: División de un espacio en $\mathbb{R}^2$ mediante el algoritmo a) de eliminación, b) de unión (tomada de [33]).

Los algoritmos de entrenamiento de las RNMPD de una capa no tienen problemas de convergencia ya que en las aplicaciones prácticas de clasificación se tienen conjuntos finitos de patrones y dado que cualquier conjunto finito es compacto, la existencia de una RNMPD de una capa para clasificar un problema finito de dos clases se garantiza con el siguiente teorema:

Teorema 4.2.1. *Si* $\mathbf{X} \subset \mathbb{R}^n$ *es compacto y* $\varepsilon > 0$ *entonces existe un perceptrón morfológico que asigna cada punto de* $\mathbf{X}$ *a la clase* C^1 *y cada* $\mathbf{x} \in \mathbb{R}^n$ *a la clase* C^0 *siempre y cuando* $d(\mathbf{x}, \mathbf{X}) > \varepsilon$.[1]

Lo anterior es cierto para dos clases, si se requieren clasificar múltiples clases entonces es necesario aumentar el número de neuronas de salida para clasificar cualquier colección de conjuntos compactos con p clases distintas y su convergencia también está garantizada por el siguiente teorema:

Teorema 4.2.2. *Si* $\{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_m\}$ *es una colección de subconjuntos compactos disjuntos de* $\mathbb{R}^n$ *y* ε *es un número positivo con* $\varepsilon < \varepsilon_0$*, entonces existe un perceptrón morfológico de una sola capa que asigna a cada* $\mathbf{x} \in \mathbb{R}^n$ *a la clase* C^j *siempre que* $\mathbf{x} \in X_j$ *y* $j \in \{1, \ldots, m\}$ *y a la clase* $C^0 = \neg \bigcup_{j=1}^{m} C_j$ *siempre que* $d(\mathbf{x}, \mathbf{X}_i) > \varepsilon$, $\forall i = 1, \ldots, m$. [2]

Como consecuencia de los teoremas mencionados en los que se basan los algoritmos de entrenamiento, las RNMPD entrenadas siempre reconocen correctamente el 100 % de los patrones en el conjunto de entrenamiento.

Es necesario mencionar que para resolver los problemas multiclase se usan los mismos algoritmos de entrenamiento que para el caso de dos clases considerando una clase a la vez y las demás como el espacio restante. La desventaja de este enfoque es que se pueden tener regiones de solapamiento donde los patrones son clasificados ambiguamente, como pertenecientes a ambas (o a varias) clases superpuestas. Para resolver este problema de superposición se agrega una capa oculta al perceptrón morfológico con procesamiento dendral [33]. Además, el resultado de estos métodos de entrenamiento depende del orden de presentación de los patrones. Estos problemas serán resueltos con la propuesta del algoritmo de entrenamiento que se explicará en el siguiente capítulo, el cual permite resolver problemas no lineales de múltiples clases con una sola neurona y con menor sensibilidad al ruido al agregar un factor de margen.

4.2.2. Otras propuestas de algoritmos de entrenamiento

Existen otras propuestas de métodos de entrenamiento reportados los cuales se describen brevemente a continuación.

[1] Ver demostración en [15]

- En 2003, Barrón y colegas [30] dan una explicación geométrica del algoritmo de entrenamiento y funcionamiento del PMPD que proporciona un punto de vista intuitivo de este proceso. A partir de esta interpretación diseñan un algoritmo de entrenamiento que parte del algoritmo de eliminación propuesto por Ritter [15] y lo mejoran de manera que su método de entrenamiento solo requiere de dos pasos que son:

Algoritmo 4.3 Entrenamiento de una RNMPD de una capa propuesto por Barrón et al.

Dadas dos clases de patrones, C^0 y C^1:

1. Construir un hipercubo que clasifique correctamente a todos los patrones de la clase C^1 aunque se incluyan también patrones de la clase C^0.
2. Aislar los puntos pertenecientes a la clase C^0 en vecindades máximas y construir dendritas con respuesta inhibitoria.

Los autores demuestran matemáticamente que aunque su método genera el mismo número de dendritas que el algoritmo original, su propuesta requiere de menos operaciones para obtenerlas.

- En 2005, Urcid [32] presenta no un nuevo método de entrenamiento sino una modificación al modelo de RNMPD original. Las neuronas biológicas solo transmiten una señal de excitación o inhibición, pero no simultáneamente como las RNMPD. Para evitar esto y obtener un modelo más parecido al biológico, los autores proponen añadir al PMPD una capa intermedia que es capaz de transformar los datos de entrada originales y enviar las señales transformadas a las sinapsis con las dendritas de una neurona de salida. Primero utilizan una transformación algebraica inversa y describen su efecto de desplazamiento del hiperplano con un objeto simple. Después aplican dos transformaciones no lineales, un polinomio de segundo grado y una función de coseno, para obtener solo señales excitadoras o inhibitorias en las sinapsis de las dendritas; y además reducir el número de dendritas necesarias para clasificar los puntos pertenecientes a conjuntos de entrada que presentan algún tipo de regularidad espacial tal como periodicidad. Los autores mencionan que la reducción del número de dendritas no es una limitación biológica inherente, ya que las neuronas reales pueden establecer desde 10^2 hasta 10^5 conexiones sinápticas con otras neuronas. Sin embargo, desde un punto de vista computacional, el tener un número menor de dendritas implica realizar menos operaciones durante la etapa de prueba.
- En 2006, Barmpoutis y Ritter [78] muestran cómo cada una de las dendritas pueden trabajar sobre una base ortonormal diferente a las demás dendritas. En otras palabras, considera rotaciones de los patrones durante el proceso de aprendizaje. El algoritmo propuesto también parte del algoritmo de eliminación y consiste en lo siguiente:

Algoritmo 4.4 Entrenamiento de una RNMPD de una capa propuesto por Barmpoutis y Ritter.

Dadas dos clases de patrones, C^0 y C^1:

1. Encontrar el hipercubo más pequeño posible que contiene todas las muestras de C^1 y asignar los valores apropiados $\mathbf{W}_1$ y $\mathbf{R}_1$ a la primera de las dendritas donde $\mathbf{R}$ es la matriz de rotación. Establecer $L = 1$.

2. Si hay puntos mal clasificadas de C^2, elegir arbitrariamente un punto mal clasificado x^γ e ir al paso 3.

3. Encontrar el hipercubo más grande que contiene a x^γ, pero que no contiene ningún punto de C^1. Establecer $L = L + 1$. Asignar los valores apropiados a $\mathbf{W}_L$ y $\mathbf{R}_L$ e ir al paso 2.

Como se puede notar el algoritmo es el mismo que el de eliminación pero ahora se consideran rotaciones de los hipercubos, esto ayuda a que se obtengan mejores resultados en problemas donde los patrones de entrenamiento tienen una estructura que involucra rotaciones.

- En 2011, Chyzhyk y Graña [34] proponen una modificación al algoritmo de eliminación. Esta modificación consiste en definir un factor de reducción $a \in [0, 1)$ que afecta el tamaño del hipercubo creado para excluir una región del espacio del hipercubo inicial que encierra a todos los elementos de la clase C^1. Este factor de reducción se introduce en el paso 9 del algoritmo 4.1 y permite que los hipercubos delimitadores estén más cerca de los elementos de la clase C^0 permitiendo así una mejor generalización para C^1. Los autores realizan experimentos para la detección de la enfermedad de Alzheimer en imágenes de resonancia magnética y comprueban las ventajas de agregar este factor de reducción.

4.3. Resumen

Este apartado es de gran importancia ya que sienta las bases para la propuesta del método de entrenamiento. Se enfoca primordialmente en la explicación del origen de las RNMPD de una capa a partir de las RNM, su base matemática fundamentada en operaciones de máximo, mínimo y suma (álgebra reticular), así como sus características principales de convergencia en un número finito de pasos, determinación de la estructura de la red durante el entrenamiento, solución de problemas no lineales sin necesidad de capas ocultas, entre otras. Además, se presentan los tipos de algoritmos de entrenamiento existentes.

Capítulo 5

Método de entrenamiento propuesto

En este capítulo se explica el algoritmo propuesto para entrenar una Neurona Morfológica con Procesamiento Dendral (NMPD). Como se mencionó en el capítulo anterior, los métodos de entrenamiento existentes presentan inconvenientes como el traslape de regiones cuando se tratan problemas de clasificación multiclase, por lo que es necesario agregar más neuronas y una capa oculta a las RNMPD para minimizar estos problemas. En cambio, el método de entrenamiento propuesto es capaz de resolver problemas no lineales de múltiples clases con una sola neurona. Además, el planteamiento es mucho más sencillo e intuitivo, el algoritmo consta de un número menor de pasos, los hipercubos obtenidos no se traslapan entre sí y se incluye un factor que permite mejorar la respuesta del algoritmo al ruido, premisa que se comprobará experimentalmente en el siguiente capítulo.

5.1. Descripción del algoritmo de entrenamiento

Antes de presentar los pasos del algoritmo de entrenamiento de la NMPD se darán las siguientes definiciones:

Definición 5.1.1. Sean $\mathbf{x}_1, \mathbf{x_2}, \ldots, \mathbf{x_m}$ un conjunto finito de patrones de muestra, donde cada patrón $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in}) \in \mathbb{R}^n$ para $i = 1, 2, \ldots, m$ es un vector n-dimensional con n atributos. Además, cada patrón $\mathbf{x}_i$ pertenece a una y solo una clase C^j, para $j = 1, 2, \ldots, p$, donde $p > 1$ es un número finito, es decir se tiene un conjunto finito de clases.

Definición 5.1.2. Un hipercubo H es una caja n-dimensional que contiene un conjunto finito de patrones $\mathbf{x} \subset \mathbb{R}^n$. Este hipercubo define los pesos w_i^l, $l = \{0, 1\}$ donde 1 es excitación y 0 inhibición:

$$H = \left\{ \mathbf{x} \epsilon \mathbb{R}^n : w_i^1 \leq x_i \leq w_i^0, i = 1, ..., n \right\}$$

Definición 5.1.3. El factor de margen M es un porcentaje que se agrega a cada una de las n-dimensiones del hipercubo con el propósito de mejorar la tolerancia al ruido de manera que:

$$H = \left\{ \mathbf{x} \epsilon \mathbb{R}^n : w_i^1 - M \times \left(w_i^0 - w_i^1 \right) \leq x_i \leq w_i^0 + M \times \left(w_i^0 - w_i^1 \right), i = 1, ..., n \right\}$$

El objetivo del algoritmo de entrenamiento es obtener un conjunto de hipercubos $H_k \in \mathbb{R}^n$ para $k \in \{1, 2, \ldots, K\}$ que contengan únicamente patrones de la misma clase C^j a partir de los siguientes pasos:

Algoritmo 5.1 Entrenamiento propuesto para una NMPD

1. Seleccionar los patrones de todas las clases y definir un primer hipercubo denotado como H_0 en $\mathbb{R}^n$ con un tamaño tal que todos los elementos $\{\mathbf{x}_i\}_{i=1}^m$ de todas las clases se encuentran dentro. Para tener mayor tolerancia al ruido en el momento de la clasificación, se agrega un margen M a cada lado de H_0.

2. Dividir a los hipercubos H_k, incluyendo a H_0, que contienen patrones de más de una clase en 2^n hipercubos más pequeños, para esto se divide el largo en cada dimensión de H_k a la mitad.

3. Verificar los hipercubos generados en el paso 2 y aplicar uno y solo uno de los siguientes pasos:

   a) Si los hipercubos generados encierran cada uno patrones de una sola clase, entonces se etiquetan con el nombre de la clase correspondiente, se detiene el proceso de aprendizaje y se va al paso 4.

   b) Si los hipercubos no contienen patrones se eliminan.

   c) Si al menos uno de los hipercubos generados tiene patrones de más de una clase, entonces se itera entre los pasos 2 y 3 hasta que el criterio de paro se satisface (que todos los hipercubos generados contengan elementos de una sola clase).

4. Simplificar, una vez que se generaron todos los hipercubos, si dos o más hipercubos de la misma clase comparten un lado común entonces se agrupan en una sola región.

5. Finalmente, a partir de las coordenadas de cada eje del hipercubo n-dimensional H_k, que encierra patrones que pertenecen a la clase C^j, se calculan los pesos para cada dendrita de la NMPD. Esto se hace para cada $k \in \{1, 2, \ldots, K\}$ donde K es el número total de dendritas.

Como se mencionó en el capítulo anterior los hipercubos se representan en la NMPD mediante las dendritas D_{jk}. Es necesario también explicar detalladamente lo siguiente:

a) El tamaño del hipercubo H_0 del paso 1 se determina mediante los puntos extremos en cada dimensión. Por lo tanto, $d_i = max\{x_i\} - min\{x_i\}$, donde d_i es la longitud del hipercubo a lo largo de la dimensión $i = 1, 2, \ldots, n$, $max\{x_i\}$ es el valor máximo de todos los patrones de muestra en la dimensión i, y $min\{x_i\}$ es el valor mínimo de todos los patrones de muestra en la dimensión i.

b) El margen M que se puede agregar a cada lado del hipercubo H_0 es un número mayor o igual a cero, y es una función de d_i. Por ejemplo, si $M = 0.1$, cada lado del hipercubo aumentará $0.1 \times d_i$ en el caso del $max\{x_i\}$ y disminuirá $0.1 \times d_i$ para el $min\{x_i\}$.

c) Los pesos w_i^l asociados a las conexiones dendríticas se definirán mediante las fronteras del hipercubo en cada eje i. El valor más bajo de la frontera será asignado a $-w_i^1$, mientras que el valor más alto será asignado $-w_i^0$. La figura 4.2 muestra un ejemplo de cómo se asignan los pesos para $n = 2$.

d) Para obtener la salida de la NMPD, como se tiene una red totalmente conectada y los hipercubos generados para cada clase deben unirse, se utilizará la ecuación 4.4 donde $p_j = +1$.

e) Para el buen funcionamiento del algoritmo es necesario que no se tengan vectores de patrones de entrenamiento iguales asignados a clases diferentes.

Se debe especificar también que la estructura propuesta del PMPD para resolver problemas multiclase se modificó, ya que no se tienen varias neuronas de salida como en el modelo original sino una sola. Esto ocurre porque el método de entrenamiento propuesto proporciona como salida el índice j de la clase a la que pertenece cada dendrita (hipercubo) y esta información se utiliza para simplificar la estructura.

En este caso se tiene una neurona totalmente conectada por lo que la salida de cada neurona se define por la siguiente ecuación:

$$\tau_k^j(\mathbf{x}) = \bigwedge_{i=1}^{n} \left(x_i + w_{ik}^1\right) \wedge - \left(x_i + w_{ik}^0\right) \tag{5.1}$$

donde $\tau_k^j(x)$ denota la salida de la dendrita, $\mathbf{x} = (x_1, \ldots, x_n)$ representa el valor de entrada a las neuronas, w_{ik}^l son los pesos, k es el índice del número de dendritas de la neurona y j es el índice de la clase.

Por otra parte, el valor de salida de la neurona se calcula mediante:

$$\tau^j(\mathbf{x}) = \bigvee_{k=1}^{K} \tau_k^j(\mathbf{x}) \tag{5.2}$$

donde K es el número total de dendritas.

Para obtener la salida $y(\mathbf{x}) = f(\tau^j(\mathbf{x}))$ en lugar de utilizar la función limitador duro se usa $f : \mathbb{R} \to \mathbb{A}$ donde $\mathbb{A} = \{1, 2, \ldots, p\}$ siendo p el número total de clases y

$$y(\mathbf{x}) = f\left(\tau^j(\mathbf{x})\right) = j. \tag{5.3}$$

Por tanto, la salida $y(\mathbf{x})$ es el índice j de la clase que corresponde a $\tau^j(\mathbf{x})$. En la figura se presenta la arquitectura de la NMPD propuesta.

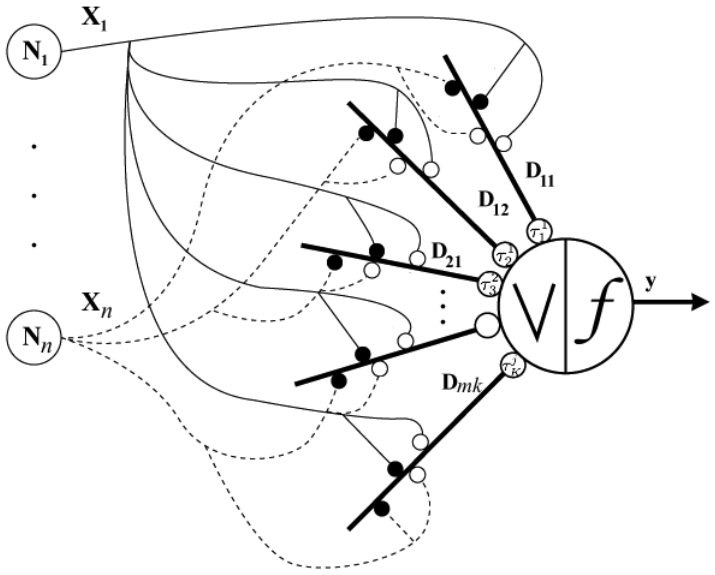

Figura 5.1: Arquitectura de la NMPD propuesta.

5.1.1. Ejemplos numéricos

Para explicar el algoritmo propuesto de manera detallada se explicarán dos ejemplos.

Ejemplo 1

El primer problema consta de 3 clases con 2 atributos. La figura 5.2 muestra los patrones pertenecientes a cada clase, los patrones de C^1 aparecen como diamantes rojos, los de C^2 como puntos verdes y los de C^3 como asteriscos negros. Los pasos del algoritmo de entrenamiento se detallan a continuación.

1. Seleccionar los patrones de todas las clases y generar un hipercubo H_0 con un tamaño tal que todos los elementos de todas las clases se encuentren dentro. La figura 5.2 presenta, en este ejemplo el rectángulo ya que $n = 2$, que cubre todos los patrones y $M = 0.1$.

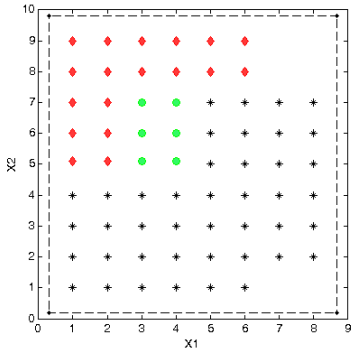

Figura 5.2: Paso 1: Rectángulo que encierra todos los patrones de entrenamiento.

2. Dividir el hipercubo en 2^n hipercubos más pequeños. La primera división se presenta en la figura 5.3.

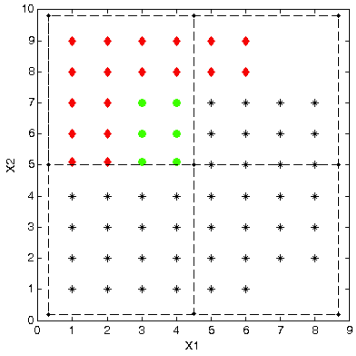

Figura 5.3: Paso 2: Primera división efectuada por el algoritmo de entrenamiento.

3. Aplicar el paso 3 hasta que se satisface el criterio de paro. La figura 5.4 muestra todos los rectángulos generados por el algoritmo de entrenamiento aplicado al ejemplo.

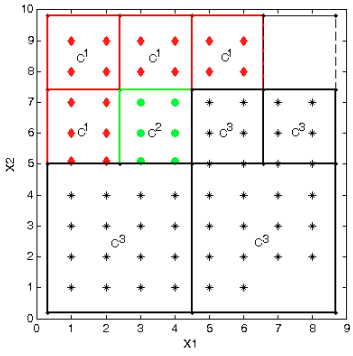

Figura 5.4: Paso 3: Rectángulos generados después del proceso iterativo de división y verificación.

4. Simplificar los hipercubos generados. La figura 5.5 presenta el resultado de la aplicación de este proceso de simplificación que automáticamente reduce el número de hipercubos y por lo tanto de dendritas.

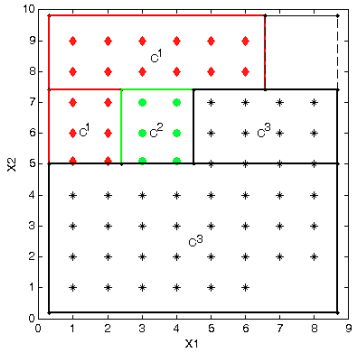

Figura 5.5: Paso 4: Rectángulos obtenidos después de la simplificación, los puntos de la clase C^1 se encierran por dos rectángulos rojos, los de la clase C^2 por un rectángulo verde y la clase C^3 por dos rectángulos negros.

5. Tomando como base las coordenadas de cada eje de los hipercubos que encierran a los patrones que pertenecen a la clase C^j, se calculan los pesos y se diseña la NMPD. La figura 5.6 muestra la estructura de la NMPD que separa las tres clases del ejemplo: C^1, C^2 y C^3. Como se puede observar cada neurona de entrada N_i se conecta a la neurona de salida a través de una dendrita D_{jk} en dos puntos diferentes: uno de excitación y otro de inhibición.

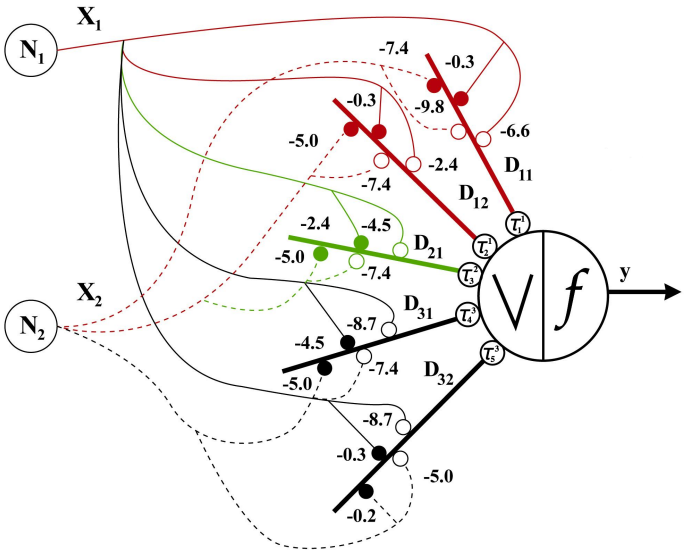

Figura 5.6: Paso 5: Neurona morfológica con procesamiento dendral que resuelve el problema de ejemplo.

En la tabla 5.1 se resumen los pesos de la NMPD.

Tabla 5.1: Pesos sinápticos de la NMPD que resuelve un problema de 3 clases y 2 atributos

D_{jk}	w^1_{1jk}	w^0_{1jk}	w^1_{2jk}	w^0_{2jk}
D_{11}	-0.3	-6.6	-7.4	-9.8
D_{12}	-0.3	-2.4	-5.0	-7.4
D_{21}	-2.4	-4.5	-5.0	-7.4
D_{31}	-4.5	-8.7	-5.0	-7.4
D_{32}	-0.3	-8,7	-0.2	-5.0

Para la etapa de prueba se seleccionan cuatro patrones ruidosos: $\tilde{\mathbf{x}}_1 = \begin{pmatrix} 4.5 \\ 8.5 \end{pmatrix}$ de la clase C^1, $\tilde{\mathbf{x}}_2 = \begin{pmatrix} 4 \\ 3.5 \end{pmatrix}$ de la clase C^3, $\tilde{\mathbf{x}}_3 = \begin{pmatrix} 4.5 \\ 6 \end{pmatrix}$ que está en un punto intermedio entre las clases C^2 y C^3 y $\tilde{\mathbf{x}}_4 = \begin{pmatrix} 8 \\ 9 \end{pmatrix}$ que no se encuentra dentro de ninguna región determinada por las dendritas.

Patrón ruidoso $\tilde{\mathbf{x}}_1$

Sea $\tilde{\mathbf{x}}_1 = \begin{pmatrix} 4.5 \\ 8.5 \end{pmatrix}$ de la clase C^1 . Aplicando la ecuación 5.1 para las dos primeras dendritas se obtienen los siguientes resultados:

$$\tau^1_1(\tilde{\mathbf{x}}_1) = \tau^1_1 \begin{pmatrix} \tilde{x}_{11} \\ \tilde{x}_{12} \end{pmatrix} = [(x_1 + w^1_{11}) \wedge -(x_1 + w^0_{11})] \wedge [(x_2 + w^1_{21}) \wedge -(x_2 + w^0_{21})]$$

$$\tau^1_1(\tilde{\mathbf{x}}_1) = \tau^1_1 \begin{pmatrix} 4.5 \\ 8.5 \end{pmatrix} = [(4.5 - 0.3) \wedge -(4.5 - 6.6)] \wedge [(8.5 - 7.4) \wedge -(8.5 - 9.8)]$$

$$= [2.1 \wedge 1.1] = 1.1$$

$$\tau^1_2(\tilde{\mathbf{x}}_1) = \tau^1_2 \begin{pmatrix} 4.5 \\ 8.5 \end{pmatrix} = [(4.5 - 0.3) \wedge -(4.5 - 2.4)] \wedge [(8.5 - 5) \wedge -(8.5 - 7.4)]$$

$$= [-2.1 \wedge -1.1] = -2.1$$

De la misma forma, se calcula las salida de todas las dendritas: $\tau^2_3(\tilde{\mathbf{x}}_1) = -1.1, \tau^3_4(\tilde{\mathbf{x}}_1) = -1.1, \tau^3_5(\tilde{\mathbf{x}}_1) = -3.5$. Con estos valores y mediante la ecuación 5.2 se obtiene:

$$\tau^j(\tilde{\mathbf{x}}_1) = \bigvee_{k=1}^{5} \tau^j_k(\tilde{\mathbf{x}}_1)$$

$$= (1.1 \vee -2.1 \vee -1.1 \vee -1.1 \vee -3.5) = 1.1$$

Por lo tanto, $\tau^1(\tilde{\mathbf{x}}_1) = \tau^1_1(\tilde{\mathbf{x}}_1) = 1.1$. Aplicando la ecuación 5.3 se tiene que $y(\tilde{\mathbf{x}}_1) = 1$ y por lo tanto, el patrón de entrada ruidoso se clasifica correctamente en la clase C^1.

Patrón ruidoso $\tilde{\mathbf{x}}_2$

Ahora sea $\tilde{\mathbf{x}}_2 = \begin{pmatrix} 4 \\ 3.5 \end{pmatrix}$ de la clase C^3. Aplicando la ecuación 5.1 para las dos primeras dendritas se tienen los siguientes resultados:

$$\tau_1^1(\tilde{\mathbf{x}}_2) = \tau_1^1 \begin{pmatrix} 4 \\ 3.5 \end{pmatrix} = [(4 - 0.3) \wedge -(4 - 6.6)] \wedge [(3.5 - 7.4) \wedge -(3.5 - 9.8)]$$

$$= [2.6 \wedge -3.9] = -3.9$$

$$\tau_2^1(\tilde{\mathbf{x}}_2) = \tau_2^1 \begin{pmatrix} 4 \\ 3.5 \end{pmatrix} = [(4 - 0.3) \wedge -(4 - 2.4)] \wedge [(3.5 - 5) \wedge -(3.5 - 7.4)]$$

$$= [-1.6 \wedge -1.5] = -1.6$$

Se calculan los valores de las dendritas restantes: $\tau_3^2(\tilde{\mathbf{x}}_2) = -1.5, \tau_4^3(\tilde{\mathbf{x}}_2) = -1.5, \tau_5^3(\tilde{\mathbf{x}}_2) = 1.5$. Con estos valores y mediante la ecuación 5.2, se obtiene:

$$\tau^j(\tilde{\mathbf{x}}_2) = \bigvee_{k=1}^{5} \tau_k^j(\tilde{\mathbf{x}}_2)$$

$$= (-3.9 \vee -1.6 \vee -1.5 \vee -1.5 \vee 1.5) = 1.5$$

Por lo tanto, $\tau^3(\tilde{\mathbf{x}}_2) = \tau_5^3(\tilde{\mathbf{x}}_2) = 1.5$, $y(\tilde{\mathbf{x}}_2) = f(\tau^3(\tilde{\mathbf{x}}_2)) = 3$ y el patrón de entrada se clasifica correctamente en la clase C^3.

Como se puede observar, se obtendrán valores positivos únicamente en la dendrita donde se encuentra el patrón a clasificar, en todos los demás casos el resultado es negativo.

Patrón ruidoso $\tilde{\mathbf{x}}_3$

Ahora se analizará el caso de un patrón ruidoso que se encuentra en un punto entre dos hipercubos.

Sea $\tilde{\mathbf{x}}_3 = \begin{pmatrix} 4.5 \\ 6 \end{pmatrix}$, aplicando la ecuación 5.1 para la primeras dendritas se obtienen los siguientes resultados:

$$\tau_1^1(\tilde{\mathbf{x}}_3) = \tau_1^1 \begin{pmatrix} 4.5 \\ 6 \end{pmatrix} = [(4.5 - 0.3) \wedge -(4.5 - 6.6)] \wedge [(6 - 7.4) \wedge -(6 - 9.8)]$$

$$= [2.1 \wedge -1.4] = -1.4$$

$$\tau_2^1(\tilde{\mathbf{x}}_3) = \tau_2^1 \begin{pmatrix} 4.5 \\ 6 \end{pmatrix} = [(4.5 - 0.3) \wedge -(4.5 - 2.4)] \wedge [(6 - 5) \wedge -(6 - 7.4)]$$

$= [-2.1 \wedge 1] = -2.1$

Calculando los valores de las dendritas restantes: $\tau_3^2(\tilde{\mathbf{x}}_3) = 0, \tau_4^3(\tilde{\mathbf{x}}_3) = 0, \tau_5^3(\tilde{\mathbf{x}}_3) = -1$ y aplicando la ecuación 5.2 se obtiene:

$\tau^j(\tilde{\mathbf{x}}_3) = \bigvee_{k=1}^{5} \tau_k^j(\tilde{\mathbf{x}}_3)$

$= (-1.4 \vee -2.1 \vee 0 \vee 0 \vee -1) = 0$

En este caso el valor máximo no es único, por convención se elige el primer valor máximo que aparece, entonces:

$\tau^3(\tilde{\mathbf{x}}_3) = \tau_3^2(\tilde{\mathbf{x}}_3) = 0 \Rightarrow y(\tilde{\mathbf{x}}_3) = 2$, el patrón de entrada se clasifica en la primera clase que se encuentra, en este caso C^2. Por tanto, se tiene que si el valor de la dendrita es positivo entonces el patrón se encuentra en la región correspondiente a esa dendrita. Si el valor de salida es cero significa que el patrón se encuentra en un punto entre hipercubos y en este caso se elige la primera dendrita que se analiza.

Patrón ruidoso $\tilde{\mathbf{x}}_4$

Sea $\tilde{\mathbf{x}}_4 = \begin{pmatrix} 8 \\ 9 \end{pmatrix}$, aplicando la ecuación 5.1 para las primeras dendritas se obtienen los siguientes resultados:

$\tau_1^1(\tilde{\mathbf{x}}_4) = \tau_1^1 \begin{pmatrix} 8 \\ 9 \end{pmatrix} = [(8 - 0.3) \wedge -(8 - 6.6)] \wedge [(9 - 7.4) \wedge -(9 - 9.8)]$

$= [-1.4 \wedge 0.8] = -1.4$

Calculando los valores de las dendritas restantes: $\tau_2^1(\tilde{\mathbf{x}}_4) = -5.6\ \tau_3^2(\tilde{\mathbf{x}}_4) = -3.5, \tau_4^3(\tilde{\mathbf{x}}_4) = -1.6, \tau_5^3(\tilde{\mathbf{x}}_4) = -4$ y aplicando la ecuación 5.2 se obtiene:

$\tau^j(\tilde{\mathbf{x}}_4) = \bigvee_{k=1}^{5} \tau_k^j(\tilde{\mathbf{x}}_4)$

$= (-1.4 \vee -5.6 \vee -3.5 \vee -1.6 \vee -4) = -1.4$

Como se puede observar, todos los valores obtenidos son negativos; con la propuesta el patrón ruidoso se clasificará en la clase más cercana, en este caso en la clase 1 pues $\tau_1^1(\tilde{\mathbf{x}}_4)$ es el valor máximo. Por tanto, $y(\tilde{\mathbf{x}}_4) = 1$ y el patrón de entrada se clasifica en C^1.

Ejemplo 2

Como segundo ejemplo se resuelve el problema de la XOR con dos entradas, $C^1 = \{(0,1),(1,0)\}$ y $C^2 = \{(0,0),(1,0)\}$. En la figura 5.7 se muestra el resultado del primer paso del algoritmo con $M = 0.1$ y del cuarto paso.

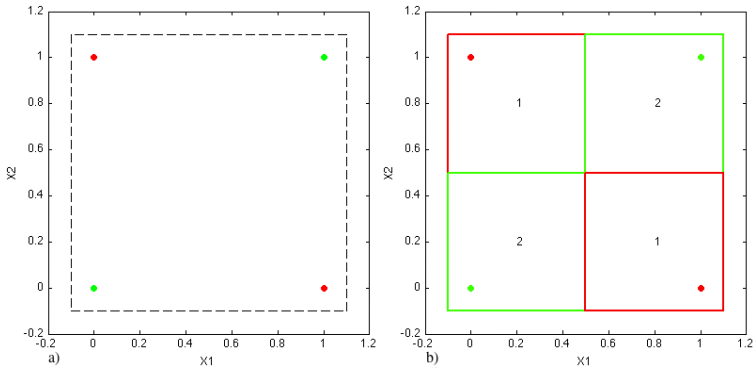

Figura 5.7: a) paso 1, b) paso 4 del algoritmo de entrenamiento.

Por tanto, la NMPD generada con el algoritmo de entrenamiento propuesto para resolver el problema de la XOR se muestra en la figura 5.8.

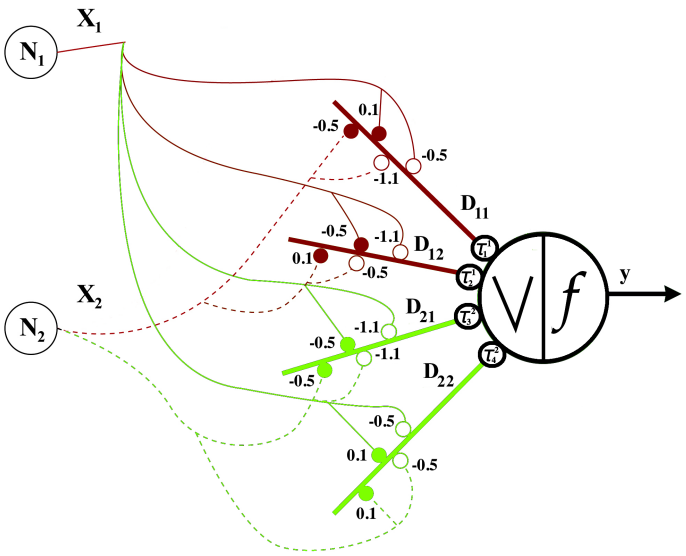

Figura 5.8: NMPD que resuelve el problema XOR de dos entradas.

Los pesos de la NMPD que resuelve el problema de la XOR de dos entradas se presentan en la tabla 5.2.

Tabla 5.2: Pesos sinápticos de la NMPD que resuelve el problema de XOR

D_{jk}	w^1_{1jk}	w^0_{1jk}	w^1_{2jk}	w^0_{2jk}
D_{11}	0.1	-0.5	-0.5	-1.1
D_{12}	-0.5	-1.1	0.1	-0.5
D_{21}	-0.5	-1.1	-0.5	-1.1
D_{22}	0.1	-0.5	0.1	-0.5

Para la etapa de prueba se selecciona el patrón ruidoso $\tilde{\mathbf{x}}_1 = \begin{pmatrix} 0.1 \\ 0.3 \end{pmatrix}$. Aplicando la ecuación 5.1 para las dendritas se obtienen los siguientes resultados:

$$\tau_1^1(\tilde{\mathbf{x}}_1) = \tau_1^1 \begin{pmatrix} 0.1 \\ 0.3 \end{pmatrix} = [(0.1 + 0.1) \wedge -(0.1 - 0.5)] \wedge [(0.3 - 0.5) \wedge -(0.3 - 1.1)]$$

$$= [0.2 \wedge -0.2] = -0.2$$

$$\tau_2^1(\tilde{\mathbf{x}}_1) = \tau_2^1 \begin{pmatrix} 0.1 \\ 0.3 \end{pmatrix} = [(0.1 - 0.5) \wedge -(0.1 - 1.1)] \wedge [(0.3 + 0.1) \wedge -(0.3 - 0.5)]$$

$$= [-0.4 \wedge 0.2] = -0.4$$

De la misma forma se obtienen: $\tau_3^2(\tilde{\mathbf{x}}_1) = -0.4, \tau_4^2(\tilde{\mathbf{x}}_1) = 0.2$. Con estos valores y mediante la ecuación 5.2:

$$\tau^j(\tilde{\mathbf{x}}_1) = \bigvee_{k=1}^{4} \tau_k^j(\tilde{\mathbf{x}}_1)$$

$$= (-0.2 \vee -0.4 \vee -0.4 \vee 0.2) = 0.2$$

Por lo tanto, $y(\tilde{\mathbf{x}}_1) = 2$ pues $\tau^2(\tilde{\mathbf{x}}_1) = \tau_4^2(\tilde{\mathbf{x}}_1) = 0.2$, el patrón de entrada ruidoso se clasifica en la clase correcta C^2.

El método de entrenamiento propuesto es capaz de resolver problemas de p clases y n atributos y tiene mayor tolerancia al ruido al agregar el factor M.

5.2. Características del algoritmo de entrenamiento propuesto

La propuesta del algoritmo de entrenamiento para NMPD cumple con ciertas características que se verifican de forma experimental en los siguientes capítulos.

Proposición 5.2.1. *Si $X \subset \mathbb{R}^n$ es un conjunto finito de patrones de p clases, C^j, $j = 1, 2, ..., p$, con n atributos, entonces el algoritmo de entrenamiento para la NMPD converge en un número finito de pasos.*

Demostración: ver apéndice D.

Proposición 5.2.2. *$\forall \mathbf{x}_E \epsilon X_{FE}$ donde X_{FE} es el conjunto fundamental de entrenamiento, $\mathbf{x}_E$ se clasifica correctamente una vez que se aplica el algoritmo de entrenamiento.*

Demostración: ver apéndice D.

Proposición 5.2.3. *Para cualquier par de hipercubos H generados por el algoritmo se cumple que* $H_l \cap H_m = \emptyset$ *donde* $l \neq m$, $l, m = 1, 2, ..., p$.

Demostración: ver apéndice D.

Proposición 5.2.4. *El número de dendritas K que genera el algoritmo de entrenamiento se encuentra en el intervalo* $p \leq K \leq m$, *donde p es el número total de clases y m es el número total de patrones de entrenamiento. De esta proposición se deducen los siguientes corolarios:*

Corolario 5.2.5. *El número mínimo de dendritas que puede generar el algoritmo es 2 ya que éste es el mínimo número de clases que se pueden tener en un problema de clasificación.*

Corolario 5.2.6. *El número máximo de dendritas que puede generar el algoritmo es m (cardinalidad del conjunto de entrenamiento) lo que implica que cada patrón del conjunto de entrenamiento produce una dendrita.*

Proposición 5.2.7. *Una vez que se establece el valor del margen M, si los patrones de entrenamiento no cambian, los hipercubos generados son siempre los mismos. No hay dependencia del orden de presentación de los datos de entrenamiento y la neurona generada.*

Proposición 5.2.8. *El algoritmo se puede aplicar a problemas de clasificación de p clases y n atributos.*

La validación de algunas de las proposiciones se hace de modo experimental. Todas estas características hacen que el algoritmo propuesto sea eficiente para resolver diferentes problemas de clasificación de manera exitosa.

5.3. Mejora del algoritmo de entrenamiento

La primera propuesta de entrenamiento para NMPD publicada en [79] proporciona buenos resultados de clasificación pero su aplicación no es eficiente para problemas con muchos rasgos ya que el tiempo de ejecución crece exponencialmente al incrementarse el número de atributos.

Después de realizar un análisis experimental se definió una nueva forma de determinar los hipercubos. En lugar de siempre dividir el espacio de patrones en 2^n regiones ahora solo se obtienen los hipercubos que cubren los datos.

Esta implementación permite resolver problemas sin restricciones estrictas en el número de características n, ya que se observa un crecimiento lineal con respecto al número de rasgos.

Los pasos del nuevo algoritmo de entrenamiento son:

Algoritmo 5.2 Entrenamiento modificado para una NMPD

Dadas p clases de patrones, C^j, $j = 1, 2, ..., p$ cada una con n atributos:

1. Seleccionar los patrones de todas las clases y abrir un primer hipercubo denotado como H_0 en $\mathbb{R}^n$ con un tamaño tal que todos los elementos $\{\mathbf{x}_i\}_{i=1}^m$ de todas las clases se encuentran dentro. Para tener mayor tolerancia al ruido en el momento de la clasificación, se le agrega un margen M a cada lado de H_0.

2. Dividir el largo de cada dimensión de H_k, incluyendo a H_0, a la mitad, $\frac{d_i}{2} = \frac{max\{x_i\}-min\{x_i\}}{2}$, obteniéndose dos partes en cada dimensión $min\{x_i\} \leq h_{i1} \leq min\{x_i\} + \frac{d_i}{2}$, $min\{x_i\} + \frac{d_i}{2} \leq h_{i2} \leq max\{x_i\}$. Determinar en qué intervalos se encuentra la primera muestra y formar el hipercubo correspondiente y así sucesivamente hasta que se terminen todas las muestras. Si algún o algunos datos se encuentran en los puntos medios se generan todos los hipercubos correspondientes.

3. Verificar los hipercubos generados en el paso 2 y aplicar uno y solo uno de los siguientes pasos:

   a) Si los hipercubos generados encierran cada uno patrones de una sola clase, entonces se etiquetan con el nombre de la clase correspondiente, se detiene el proceso de aprendizaje y se va al paso 4.

   b) Si al menos uno de los hipercubos generados tiene patrones de más de una clase, entonces se itera entre los pasos 2 y 3 hasta que el criterio de paro se satisface (que todos los hipercubos generados contengan elementos de una sola clase).

4. Simplificar, una vez que se generaron todos los hipercubos, si dos o más hipercubos de la misma clase comparten un lado común, entonces se agrupan en una sola región.

5. Finalmente, a partir de las coordenadas de cada eje del hipercubo n-dimensional H_k, que encierra patrones que pertenecen a la clase C^j, se calculan los pesos para cada dendrita de la NMPD. Esto se hace para cada $k \in \{1, 2, \ldots, K\}$ donde K es el número total de dendritas.

Para ilustrar el nuevo algoritmo de entrenamiento se utilizará un ejemplo específico de clasificación de dos 2 clases con 2 atributos y se resaltarán las diferencias y similitudes entre las dos propuestas. La figura 5.9 muestra los patrones pertenecientes a cada clase, las muestras de C^1 aparecen como puntos rojos y las de C^2 como puntos verdes.

Los pasos generales del algoritmo son los siguientes:

1. Seleccionar los patrones de todas las clases y abrir el hipercubo H_0, este paso es el mismo que en la propuesta inicial. La figura 5.9 presenta el rectángulo que cubre todos los

patrones de todas las clases con $M = 0$ para el ejemplo considerado.

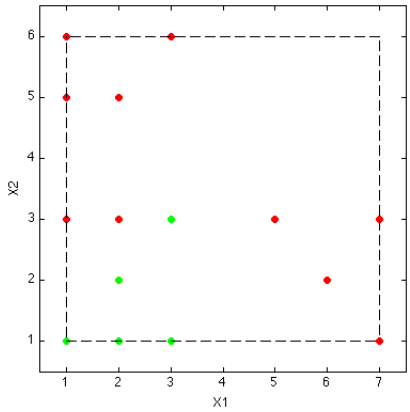

Figura 5.9: Paso 1: Rectángulo que encierra todos los patrones de entrenamiento.

2. En la propuesta original el hipercubo se divide en 2^n hipercubos más pequeños. Para el ejemplo, la división se presenta en la figura 5.9.

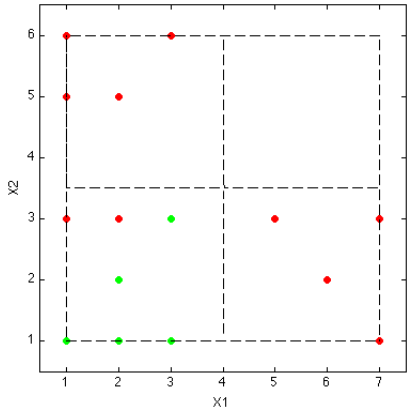

Figura 5.10: Paso 2: División del rectángulo efectuada por el algoritmo de entrenamiento.

En cambio, con el nuevo algoritmo se consideran los datos en lugar del espacio. Cada dimensión (atributo) se divide a la mitad obteniéndose dos intervalos en cada eje. Para un dato específico se determina en qué rangos se encuentra, formándose así el primer hipercubo a partir de los intervalos de cada dimensión. Posteriormente se verifica si el siguiente dato se encuentra en la misma región, si no es así se genera otro hipercubo y así sucesivamente hasta que se terminen los datos. Si algún o algunos patrones se encuentran en los puntos medios se generan todos los hipercubos correspondientes.

A diferencia de la propuesta original, en este caso no siempre se obtienen 2^n hipercubos sino solamente los hipercubos que cubren los datos. Para el ejemplo, supóngase que el patrón a verificar es el de la clase C^1 con valor (1,6), por lo tanto se genera el primer rectángulo que corresponde a los intervalos en x_1 de $[1, 4]$ y en x_2 de $[3.5, 6]$, ver la figura

5.11 a). Posteriormente se analiza el resto de los datos obteniéndose tres rectángulos que aparecen en la figura 5.11 b) a diferencia de los cuatro rectángulos que se calculan con la primera propuesta (figura 5.10).

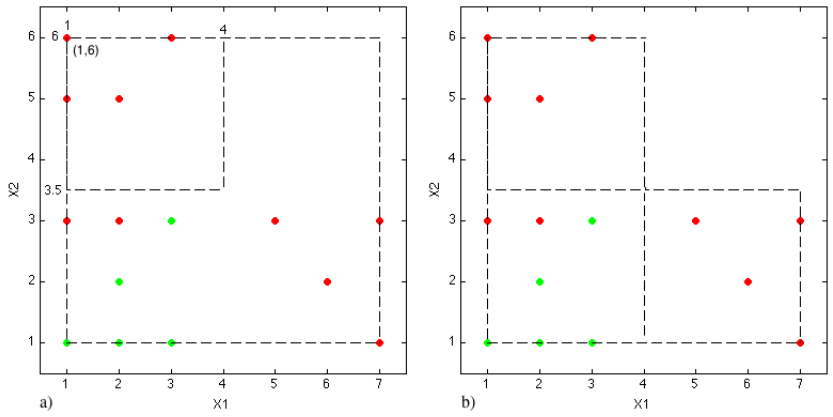

Figura 5.11: a) primer rectángulo generado considerando los datos y la división a la mitad en cada dimensión, b) resultado del paso 2 del algoritmo.

3. Al igual que en el algoritmo original este es un paso de iteración. Si al menos uno de los hipercubos generados tiene patrones de más de una clase entonces se repite el proceso del paso 2. En caso de que solamente existan patrones de una sola clase en un hipercubo, éste se guarda y se etiqueta con la clase correspondiente. Hay que notar que en este caso no hay proceso de eliminación de hipercubos que no contengan patrones ya que estos no se generan. En la figura 5.12a) se presentan los primeros pasos de este procedimiento para el ejemplo considerado. El proceso de verificación se repite iterativamente en cada hipercubo hasta que el criterio de paro se satisface. La figura 5.12 b) muestra todos los rectángulos generados por el algoritmo de entrenamiento aplicado al ejemplo, que son los mismos que los que se obtienen con el primer algoritmo.

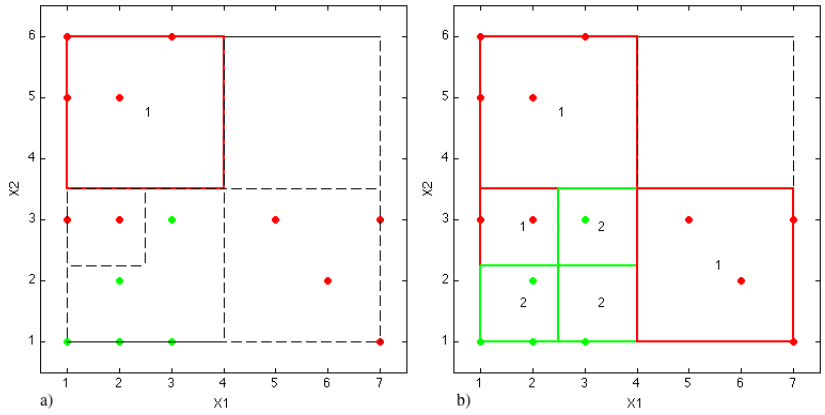

Figura 5.12: a) primeros resultados del paso 3 del algoritmo de entrenamiento, b) rectángulos generados después del proceso iterativo de división y prueba de datos.

4. Este es el paso de simplificación igual al del algoritmo original. La figura 5.13 presenta el resultado.

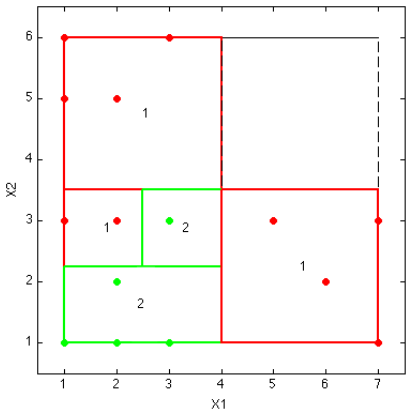

Figura 5.13: Paso 4) : Rectángulos obtenidos después de la simplificación, los puntos de la clase C^1 se encierran por tres rectángulos rojos y los de la clase C^2 por dos rectángulos verdes.

5. En este paso, tomando como base las coordenadas de cada eje de los rectángulos, se calculan los pesos para cada dendrita que encierra a los patrones que pertenecen a la clase C^k, y de esta forma se diseña la NMPD que se muestra en la figura 5.14.

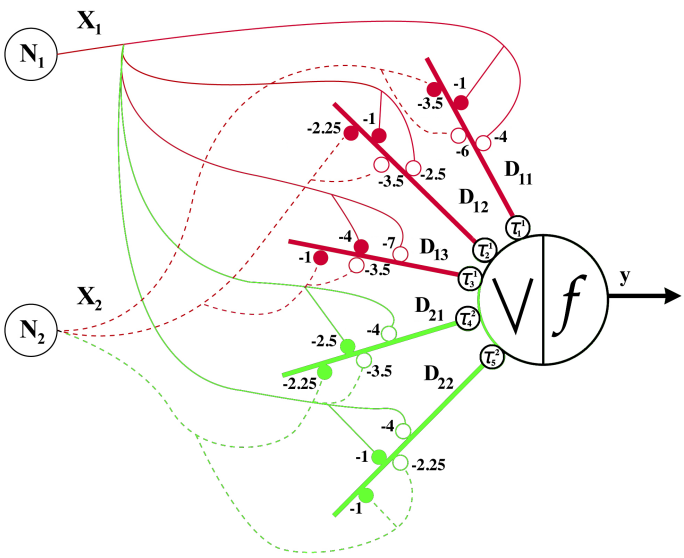

Figura 5.14: Paso 5) NMPD que resuelve el problema

La NMPD que generan ambos algoritmos es la misma, por lo tanto se obtiene el mismo resultado para la clasificación, la diferencia es la manera de generar los hipercubos de separación.

5.4. Comparación

La nueva implementación permite resolver problemas de clasificación cuyos patrones tienen muchos rasgos. Para el caso de pocos atributos no hay una diferencia significativa en los tiempos de ejecución de ambos algoritmos. Sin embargo, si el número de atributos se incrementa, el tiempo de la propuesta original (algoritmo 1, A1) crece de forma exponencial lo que no ocurre con la nueva propuesta (algoritmo 2, A2). Para comprobar lo anterior se generó un ejemplo de dos clases con n atributos y 100 muestras para cada clase generadas aleatoriamente de manera uniforme entre -1 y 1, variando n desde 2 hasta 25. Ambos algoritmos se implementaron usando MATLAB 7.11 en una computadora portátil con procesador Intel Core i5 a 2.3 GHz, con 4GB en RAM. Los errores de clasificación obtenidos muestran las ventajas de las mejoras como se puede ver en la tabla 6.2.

La figura 5.15 presenta la comparación de la eficiencia de ambos algoritmos aumentando el número de atributos.

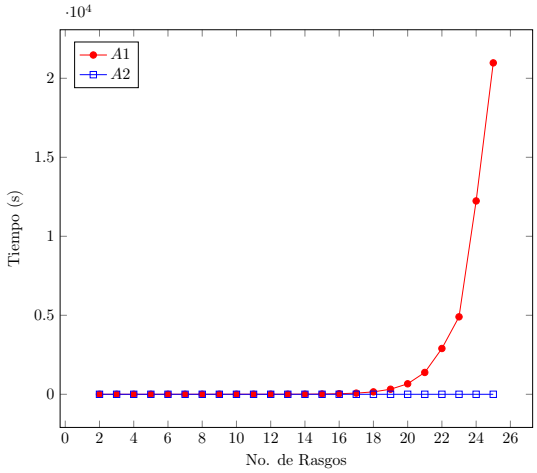

Figura 5.15: Comparación de la eficiencia de los algoritmos al aumentar el número de atributos.

Como se puede observar en la figura 5.15, el algoritmo 1 requirió de casi 6 horas para entrenar la neurona para un problema de 2 clases y 25 atributos. Debido a esta restricción se realizaron pruebas con más de 25 atributos solo con el algoritmo 2.

Para comprobar el comportamiento lineal del algoritmo 2 al aumentar el número de rasgos se incrementó el número de atributos desde 100 hasta 1000 con incrementos de 100, el resultado aparece en la figura 5.16, como se puede observar el crecimiento ya no es exponencial como en el primer algoritmo.

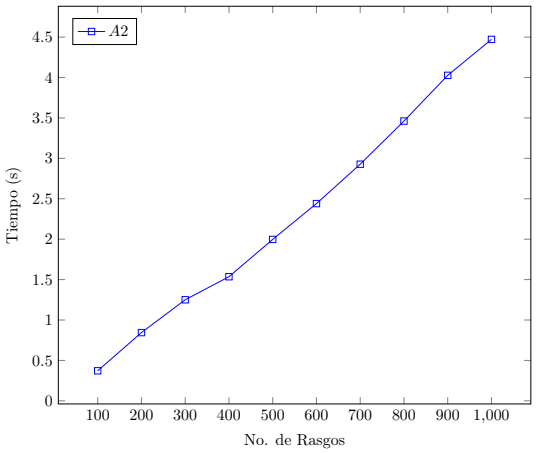

Figura 5.16: Gráfica de tiempo contra número de atributos.

Para verificar que el crecimiento se mantiene lineal se realizó una prueba con 500 mil atributos, igual que en los experimentos anteriores se consideraron dos clases con 100 muestras cada una. Como se puede observar en la figura 5.17 el comportamiento es lineal y para resolver este problema el algoritmo tarda aproximadamente una hora.

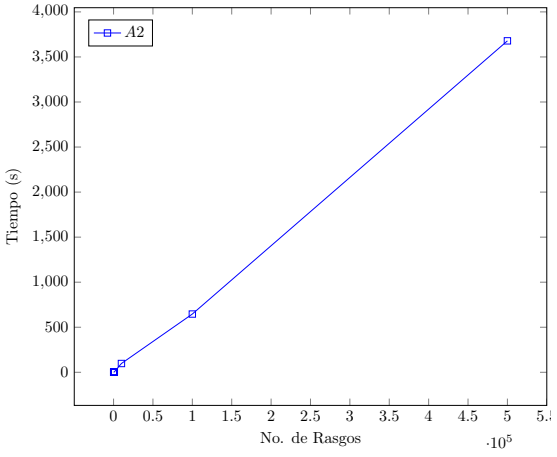

Figura 5.17: Gráfica de tiempo contra número de atributos considerando hasta 500 mil rasgos.

Por otra parte, el comportamiento de ambos algoritmos con respecto al incremento en el número de clases es similar y no exponencial, como se puede observar en la figura 5.18.

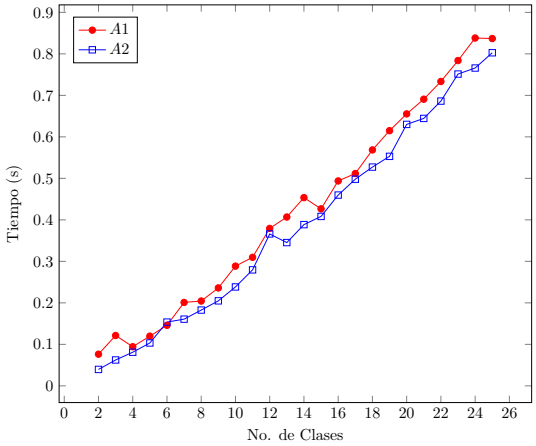

Figura 5.18: Comportamiento de los algoritmos al incrementar el número de clases.

Con la modificación del algoritmo ahora es posible tratar problemas con un mayor número de atributos, lo que hace que la propuesta de este trabajo se pueda utilizar en un campo más amplio de aplicaciones.

5.5. Resumen

La explicación del método de entrenamiento propuesto para NMPD se presenta en este capítulo. Se realiza la descripción detallada del algoritmo de entrenamiento y se mencionan sus características principales. Una desventaja importante del método formulado inicialmente es que el tiempo de entrenamiento crece exponencialmente al aumentar el número de rasgos. Sin embargo, en la última etapa de la presente investigación se encontraron modificaciones que permitieron eliminar esta característica para ciertos problemas, de tal manera que se obtiene la misma NMPD, pero sin una dependencia exponencial respecto al incremento del número de atributos utilizados para representar a los patrones de entrenamiento. Las modificaciones y comparaciones de ambos algoritmos se muestran también en esta sección.

Capítulo 6

Resultados experimentales

En este capítulo se presenta un estudio experimental que permite establecer las ventajas del algoritmo de entrenamiento propuesto en esta investigación y mostrar sus características. Se realizaron experimentos con datos generados artificialmente y con problemas reales obtenidos de bases de datos conocidas, además de una aplicación para el mercado financiero mexicano.

6.1. Resultados experimentales usando datos generados sintéticamente

Para comprobar la eficiencia del método propuesto se utilizaron dos problemas sintéticos que por su naturaleza son difíciles de resolver, el problema de los espirales por su forma y el problema de Ripley por la distribución de sus clases. Los resultados del algoritmo se compararon con los del algoritmo de entrenamiento de eliminación de Ritter [15] y MLP.

6.1.1. Problema de los espirales de Arquímedes

El espiral de Arquímedes está definido por la expresión:

$$x_c(\theta) = \frac{2(-1)^{1-c}}{\pi}\theta\cos(\theta) \tag{6.1}$$

$$y_c(\theta) = \frac{8(-1)^{1-c}}{3\pi}\theta\sin(\theta). \tag{6.2}$$

donde $c \in \{0, 1\}$ denota la etiqueta de la clase espiral, θ es el ángulo en radianes y (x_c, y_c) son las coordenadas de los puntos del espiral. Una explicación más detallada se puede consultar en [15].

Usando las fórmulas anteriores, se generaron las muestras del conjunto de entrenamiento. La figura 6.1 muestra el conjunto de 50 muestras de entrenamiento. En las siguientes figuras, los círculos rellenos representan a los patrones de entrenamiento de la clase C^j, $j = 1, 2, \ldots, p$.

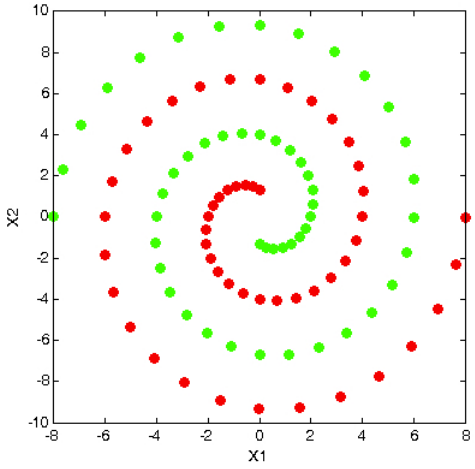

Figura 6.1: Conjunto de 50 muestras de entrenamiento de los espirales.

En la propuesta del algoritmo de entrenamiento original, para la neurona morfológica con procesamiento dendral (NMPD-PO), el valor del factor M se selecciona empíricamente. En una segunda aproximación (NMPD-P1), el valor de M que minimiza el error se determina por medio de un algoritmo evolutivo, evolución diferencial (DE/rand/1/bin) [80], con los parámetros que se muestran en la tabla 6.1.

Tabla 6.1: Valores de los parámetros del algoritmo de evolución diferencial usados para encontrar el factor M.

Parámetros	Valor
Máximo número de generaciones	100
Tamaño de la población	20
CR (probabilidad de cruce)	[0.5, 0.95]
F (peso diferencial)	[0.3, 0.8]

Para asegurar que no hay sensibilidad a los parámetros F y CR, los valores de estos parámetros se generaron aleatoriamente (usando una distribución uniforme) entre los intervalos que se muestran en la tabla 6.1. Los intervalos para ambos parámetros se definieron

empíricamente. Como revelan los resultados (ver tabla 6.2), la aplicación del algoritmo de evolución diferencial para encontrar el mejor valor de M, ayuda a reducir significativamente el error de clasificación del algoritmo propuesto.

Además de la propuesta para encontrar el valor de M se realizó otra mejora. El modelo original de la RNMPD usa la función de activación del limitador duro que se cambió por la función de la ecuación 5.3 (NMPD-P2), esto disminuyó el error en un porcentaje considerable (ver tabla 6.2). Lo anterior ocurre porque con el limitador duro pueden existir patrones sin clasificar y con el uso de la función propuesta, estos patrones se asignan a la clase más cercana. En las siguientes figuras, los círculos vacíos denotan a las muestras de prueba y los asteriscos representan a las muestras de prueba clasificadas en cada clase. La figura 6.2 muestra, a la izquierda, los resultados del algoritmo NMPD-P1 (función de activación limitador duro), en este caso los patrones representados con los asteriscos amarillos no se clasifican en ninguna de las clases, lo que produce un error. Sin embargo, si se utiliza la arquitectura neuronal y la función propuesta, el error obtenido en el problema del espiral ($\sigma = 0.2$) es cero y todos los patrones de prueba son asignados a una clase.

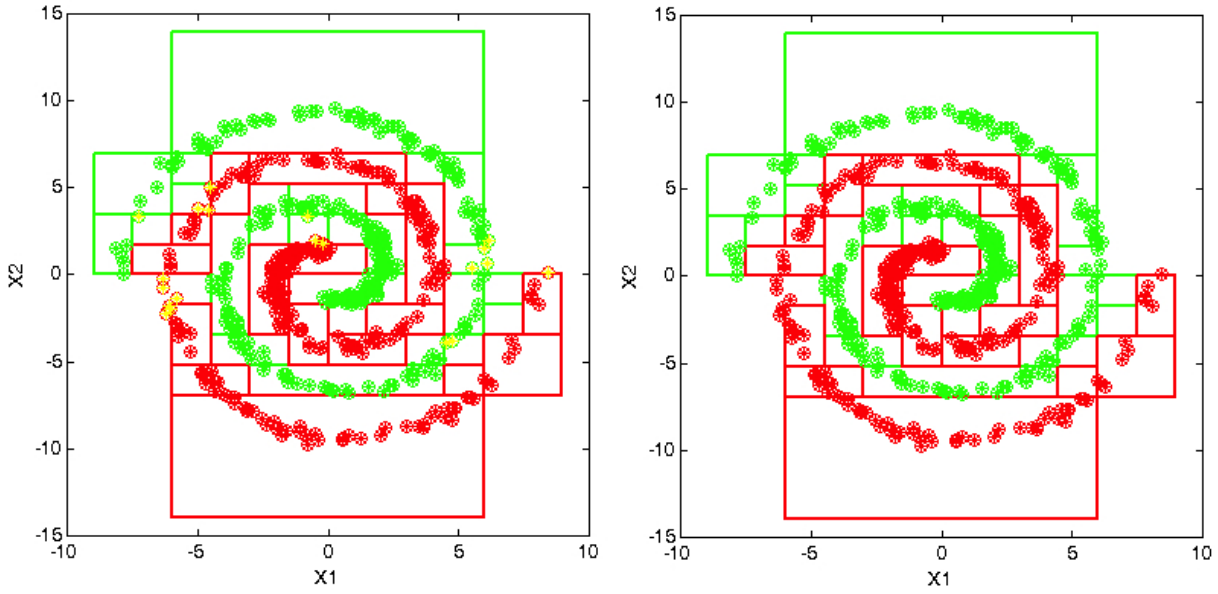

Figura 6.2: Resultados de clasificación del NMPD-P1 (función limitadora) y NMPD-P2 (función propuesta) con 50 muestras de entrenamiento y 500 muestras de prueba para el problema del espiral.

La tabla 6.2 presenta los errores de clasificación obtenidos para el problema del espiral. Todos los experimentos de esta sección se realizaron usando MATLAB 7.11 en una computadora de escritorio con procesador Intel i7 a 2.2 GHz, con 8GB en RAM. Para obtener el conjunto de prueba, se le agregó ruido a los patrones de entrenamiento con una distribución normal con media en el centro del espiral y con diferentes valores de desviación estándar (σ), cada conjunto de prueba consta de 500 muestras. Los errores de clasificación disminuyen al aplicar las mejoras propuestas, como se puede ver en la tabla

6.2.

Tabla 6.2: Tabla de comparación de los errores de clasificación obtenidos para el problema del espiral con NMPD-PO, NMPD-P1 y NMPD-P2; donde PO se refiere a la propuesta original, P1 es el algoritmo con M encontrada mediante evolución diferencial y P2 es el algoritmo con la función propuesta y M obtenido mediante el algoritmo evolutivo.

	NMPD-PO			**NMPD-P1**			**NMPD-P2**		
σ	# Dendritas	M	% Error	# Dendritas	M	%Error	# Dendritas	M	%Error
0.2	28	0.3	4.4	28	0.22	3	34	0.25	0
0.3	28	0.3	6.44	28	0.06	4.03	30	0.27	1.17
0.4	28	0.3	11.31	28	0.63	7.54	45	0.21	3.87

Es importante mencionar que los resultados de la clasificación solo se presentan para los patrones de prueba, porque una propiedad del algoritmo de entrenamiento de la NMPD es que los patrones de entrenamiento son siempre reconocidos al 100 %. Esta característica se debe a la manera en que se construyen los hipercubos a partir de estos patrones por lo que todos quedan dentro de los hipercubos construidos. Además de que los errores de entrenamiento no se pueden usar para comparar del desempeño de los algoritmos ya que éste siempre será menor que el error obtenido en la etapa de prueba [67].

Para verificar la eficiencia del algoritmo de entrenamiento propuesto (NMPD-P) se realizaron comparaciones con el algoritmo de entrenamiento de eliminación propuesto por Ritter (NMPD-R) [15] y el perceptrón multicapa (MLP). Para el MLP con una capa escondida, los parámetros de entrenamiento se establecieron como: razón de aprendizaje= 0.25, momento=0.2 y la función de activación usada fue la sigmoide. Hecht-Nielsen [81] sugirieron que un límite superior para el número de neuronas en la capa escondida es un número menor a $2N + 1$ donde N es el número de neuronas de entrada. Este límite se consideró como un punto de partida (4 neuronas) pero en este problema el error obtenido con 4 neuronas fue considerable (entre 40-50 %). Se encontró experimentalmente que con 9 neuronas en la capa escondida el error es aceptable.

El algoritmo usado en los siguientes experimentos es el algoritmo NMPD-P2 al que se referirá simplemente como NMPD-P. En todos los casos, los errores de clasificación obtenidos con el algoritmo propuesto fueron menores que los obtenidos con el algoritmo de Ritter y el MLP. Se puede notar que el número de dendritas generadas por el algoritmo propuesto es mayor que el número de dendritas del algoritmo de entrenamiento de Ritter, esto es porque en el algoritmo propuesto los hipercubos generados cubren todas las clases.

La figura 6.3 muestra una gráfica de comparación del porcentaje de error de los tres algoritmos. Esta gráfica muestra la ventaja del algoritmo propuesto sobre el NMPD-R y el MLP.

Tabla 6.3: Tabla de comparación entre MLP, NMPD-R y NMPD-P para el problema del espiral.

	MLP		NMPD-R		NMPD-P		
σ	# Neuronas	% Error	# Dendritas	% Error	# Dendritas	M	%Error
0.2	9	20	11	25	34	0.25	0
0.3	9	23.8	12	27.44	30	0.27	1.17
0.4	9	28.61	14	31.47	45	0.21	3.87

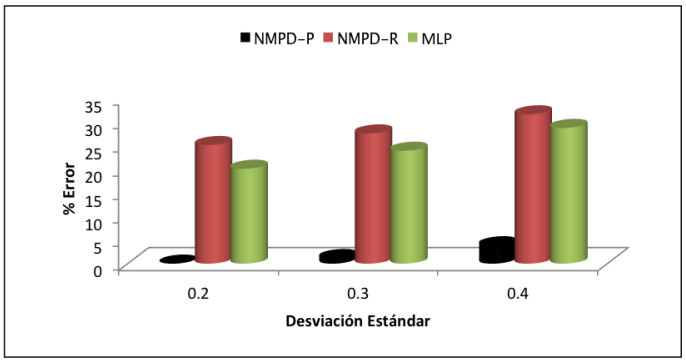

Figura 6.3: Gráfica de comparación del error de clasificación del MLP, NMPD-R y NMPD-P con diferentes valores de desviación estándar de los patrones de prueba para el problema del espiral.

Las figuras 6.4, 6.5, 6.6 y 6.7 muestran las cajas formadas por el proceso de entrenamiento usando 50 muestras y los resultados de la clasificación de los patrones de prueba obtenidos por NMPD-R y NMPD-P, respectivamente. En la figura 6.4 se presenta el resultado del entrenamiento de la NMPD-R y las 50 muestra utilizadas para el entrenamiento.

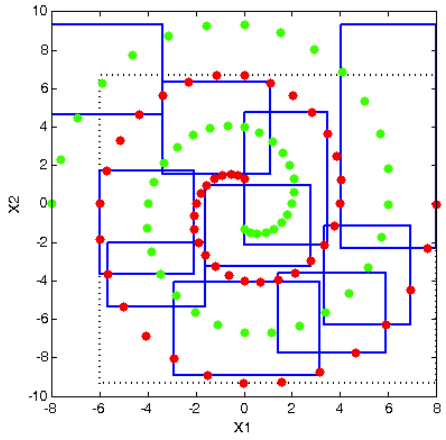

Figura 6.4: Resultado de entrenamiento del NMPD-R con 50 muestras de entrenamiento para el problema del espiral.

La figura 6.5 presenta a la izquierda los patrones de prueba representados por círculos vacíos y los hipercubos generados por el entrenamiento y a la derecha se presentan nuevamente los patrones de prueba y con asteriscos se presentan los resultados de la clasificación de cada muestra de prueba, se puede observar que si el círculo y el asterisco coinciden en color, la muestra está clasificada correctamente. En caso de que los colores del círculo y el asterisco no sean iguales se tiene un error.

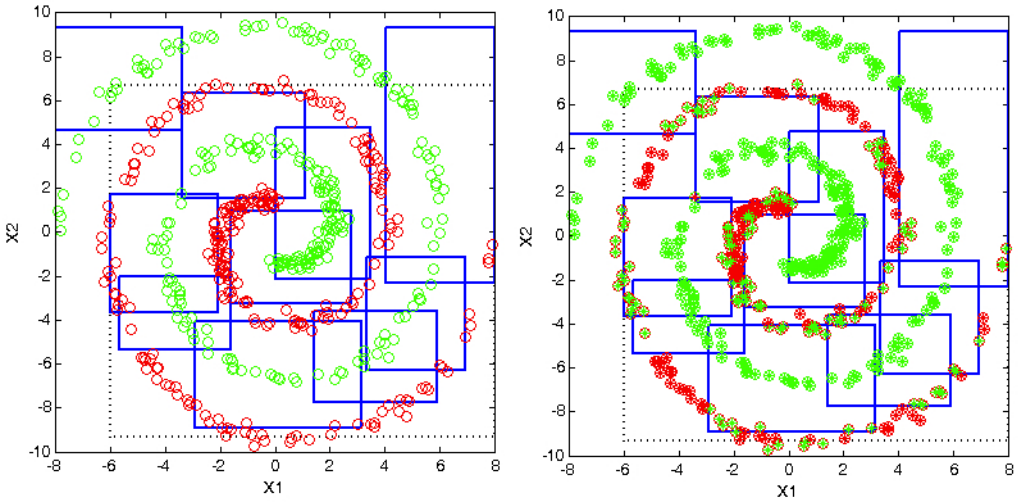

Figura 6.5: Resultados de clasificación del NMPD-R con 50 muestras de entrenamiento y 500 muestras de prueba para el problema del espiral.

Las figuras 6.6 y 6.7 muestran los resultados del NMPD-P para el problema del espiral.

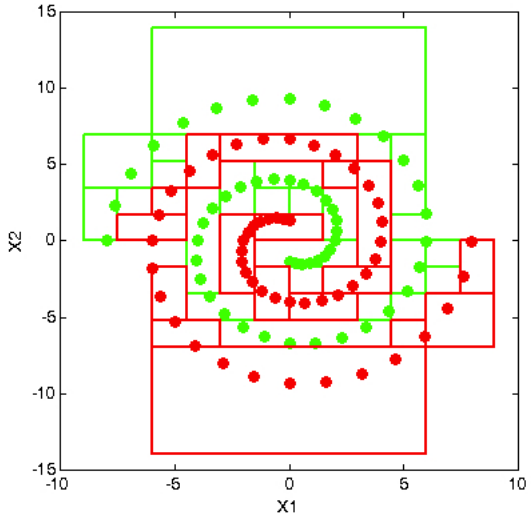

Figura 6.6: Resultados de entrenamiento del NMPD-P con 50 muestras de entrenamiento para el problema del espiral.

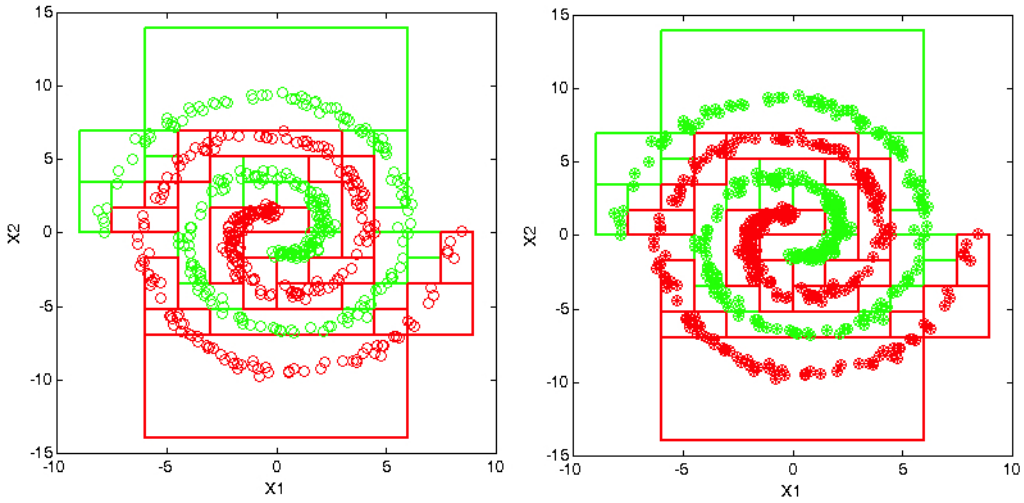

Figura 6.7: Resultados de clasificación del NMPD-P con 50 muestras de entrenamiento y 500 muestras de prueba para el problema del espiral.

6.1.2. Problema de Ripley

El segundo problema a tratar es el conjunto sintético de Ripley [82] que consta de dos clases. Los datos se dividen en un conjunto de entrenamiento y un conjunto de prueba que constan de 250 y 1000 muestras respectivamente, con el mismo número de muestras pertenecientes a cada una de las dos clases. El conjunto de entrenamiento aparece en la figura 6.8.

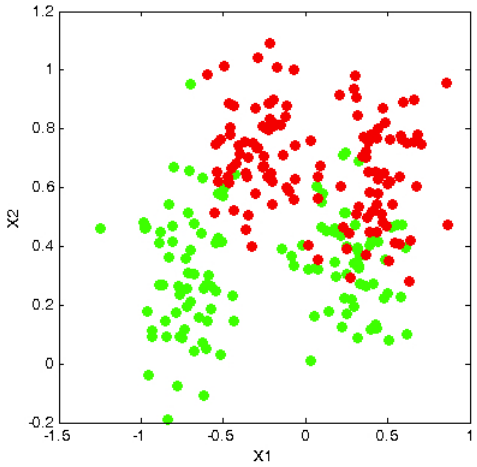

Figura 6.8: Conjunto de entrenamiento de Ripley.

Los resultados de clasificación de los algoritmos aplicados al conjunto de Ripley aparecen en las figuras 6.9, 6.10, 6.11 y 6.12.

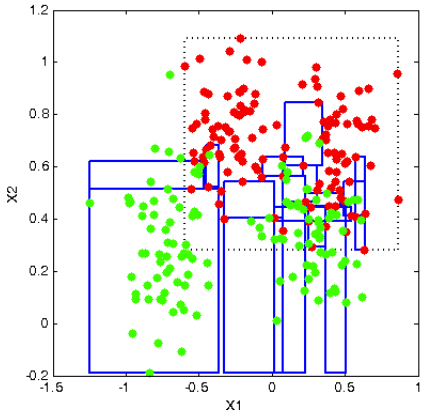

Figura 6.9: Resultado del entrenamiento del NMPD-R con 50 muestras de entrenamiento para el problema de Ripley.

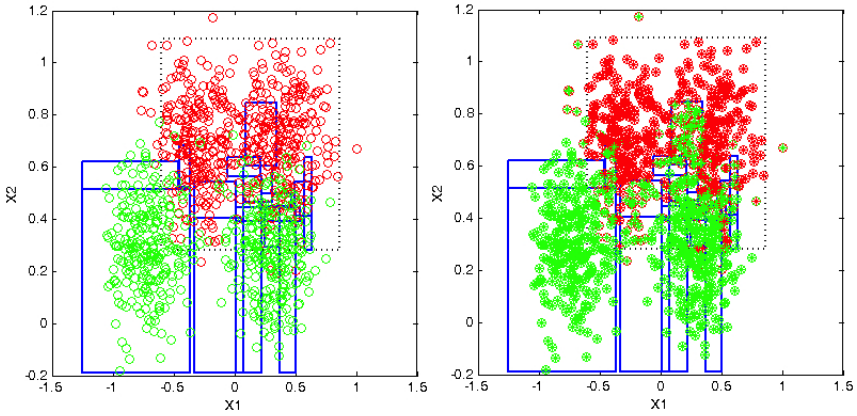

Figura 6.10: Resultados de la clasificación de 1000 muestras de prueba del conjunto Ripley usando el algoritmo NMPD-R.

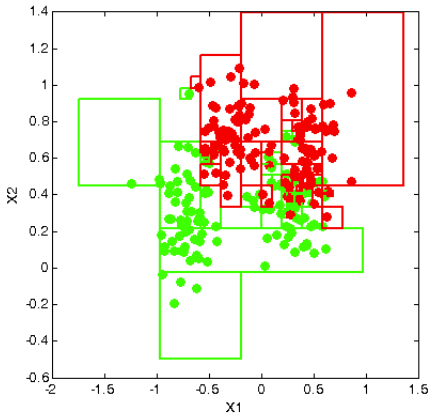

Figura 6.11: Resultado del entrenamiento del NMPD-P con 50 muestras de entrenamiento para el problema de Ripley.

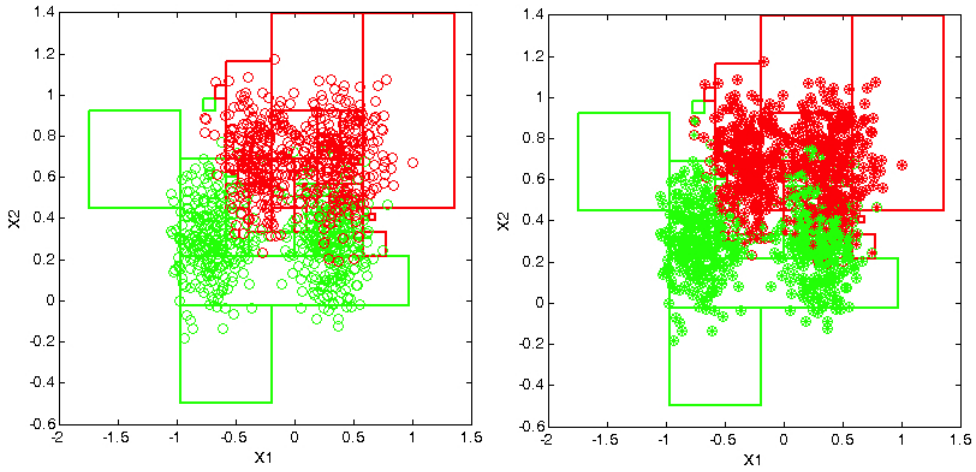

Figura 6.12: Resultados de clasificación de 1000 muestras de prueba del conjunto de datos Ripley usando el algoritmo NMPD-P.

La tabla 6.4 presenta los errores de clasificación para los datos de Ripley obtenidos por cada uno de los algoritmos. Como se puede ver, para este problema, el error generado con el algoritmo de entrenamiento propuesto es mayor que el error obtenido en el problema del espiral. Esto es porque algunas de los datos de la clase C^1 están mezclados en la clase C^2 y viceversa. Sin embargo, el NMPD-P mejora los resultados del NMPD-R y el MLP, por tanto el comportamiento del algoritmo es satisfactorio para este tipo de problemas.

Tabla 6.4: Tabla de comparación de los algoritmos MLP, NMPD-R y NMPD-P para el conjunto de datos Ripley.

MLP		NMPD-R		NMPD-P		
# Neuronas	% Error	# Dendritas	% Error	# Dendritas	M	% Error
9	14	16	18.8	70	0.238	12.2

6.2. Resultados experimentales usando datos reales

El algoritmo de entrenamiento propuesto se aplicó a datos reales con p clases y n atributos para cada clase, y los resultados se compararon con MLP, RBN y SVM. Primero se consideraron experimentos de reconocimiento de objetos y posteriormente se utilizaron algunas bases de datos del repositorio de aprendizaje de máquina de la Universidad de California, Irvine (UCI) [83]. El método de validación utilizado en la mayoría de los casos fue el de *hold-out* 50 %-50 %.

6.2.1. Clasificación en imágenes

El algoritmo propuesto se utilizó para resolver el problema de 5 clases y 2 atributos presentado en [84]. Este problema consiste en clasificar los objetos: rondana, cola de pato, tornillo, armella y alcayata. Para caracterizar a los objetos se usaron como rasgos el primer y segundo momentos de Hu (ver subsección 7.1.2.1). La figura 6.13 muestra las imágenes de los objetos y sus momentos de Hu.

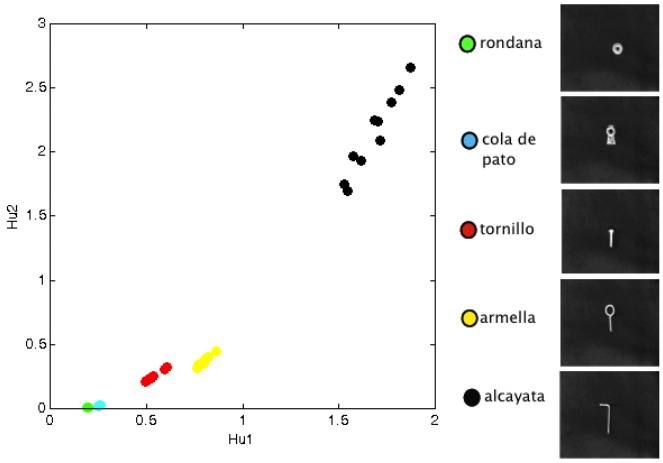

Figura 6.13: Momentos de Hu de las 5 clases.

Los conjuntos de entrenamiento y de prueba tienen 50 muestras cada uno. El algoritmo de entrenamiento propuesto obtuvo un porcentaje de error de clasificación del 0 % con un valor de $M = 0.7$ y 6 dendritas. La figura 6.14 muestra los resultados.

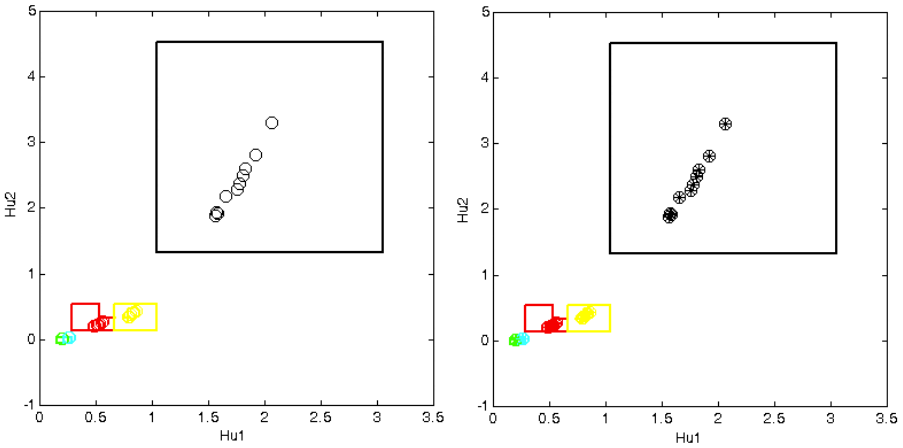

Figura 6.14: Resultados de clasificación del problema de reconocimiento de objetos.

Para probar el desempeño del algoritmo de entrenamiento en imágenes se utilizó un subconjunto de la base de datos ETH80 [85]. Esta base de datos consta de imágenes

a color de 80 objetos agrupados en 8 categorías diferentes, el subconjunto seleccionado consta de 10 objetos y cada objeto está representado por 41 imágenes tomadas desde diferentes puntos de vista. El tamaño de las imágenes es de 128x128 pixeles y las imágenes se convirtieron a escala de grises. El subconjunto de 10 objetos se muestra en la figura 6.15 y consta de los siguientes objetos: manzana, carro, taza-1, taza-2, perro-1, perro-2, pera-1, pera-2, tomate y vaca.

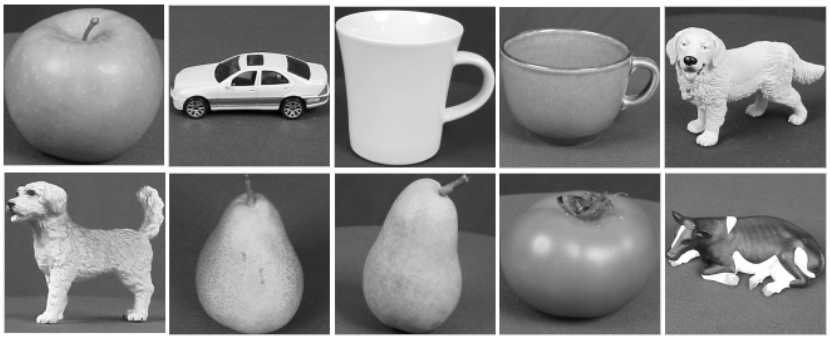

Figura 6.15: Subconjunto de la base ETH80.

Para la representación de los objetos, se obtuvieron los primeros 4 momentos de Hu y se usaron los rasgos de 21 imágenes de cada objeto para formar el conjunto de entrenamiento y 20 para prueba. Los resultados del algoritmo se compararon con un MLP de una capa escondida, SVM y RBN. Estos algoritmos se aplicaron usando el software WEKA 3-6-9 [86]. Para el MLP con una capa oculta, los parámetros de entrenamiento se establecieron como: razón de aprendizaje=0.3, momento=0.2 y la función de activación sigmoide. Para el SVM, se utilizó un núcleo polinomial de grado 2 y para la RBN se utilizaron dos *clusters*. En los experimentos, para seleccionar el grado del núcleo polinomial de SVM y el número de clusters de la RBN, se hicieron pruebas cambiando manualmente estos valores y se seleccionaron los que generaron los mejores resultados. El porcentaje de error obtenido con los diferentes algoritmos se presentan en la tabla 6.5.

Tabla 6.5: Tabla de comparación de resultados de clasificación obtenidos por MLP, SVM, RBN y NMPD-P para el subconjunto de la base ETH80.

MLP	SVM	RBN	NMPD-P		
% Error	% Error	% Error	# Dendritas	M	% Error
33.66	39.51	45.36	114	0.438	29.76

Como se puede ver en la tabla 6.5, el algoritmo de entrenamiento de la NMPD mejora los resultados del resto de los algoritmos. Sin embargo, el porcentaje de error se podría reducir si se utilizan más o mejores características para describir a los objetos.

6.2.2. Bases de datos de UCI

El desempeño del algoritmo propuesto también se evaluó en 7 bases de datos conocidas, las cuales se pueden encontrar en el repositorio de aprendizaje de máquina de la UCI [83]. Primero se consideró la base de datos del Iris. Esta base de datos tiene 3 clases (Iris setosa, Iris virginica e Iris versicolor) y 4 características (el largo y ancho de los sépalos y pétalos en centímetros). El conjunto de datos tiene 50 muestras por clase, la base de datos original se dividió en dos subconjuntos del mismo tamaño tomando muestras aleatorias para el entrenamiento y la prueba.

Para desplegar los resultados, solo se usaron 3 atributos (largo y ancho de los sépalos y largo de los pétalos), de esta forma los patrones están en el espacio tridimensional y los resultados se pueden ver gráficamente. La figura 6.16 presenta los resultados del entrenamiento y la figura 6.17 muestra los resultados de la clasificación. El algoritmo obtuvo un error del 4 % con $M = 0.7$ y 22 dendritas.

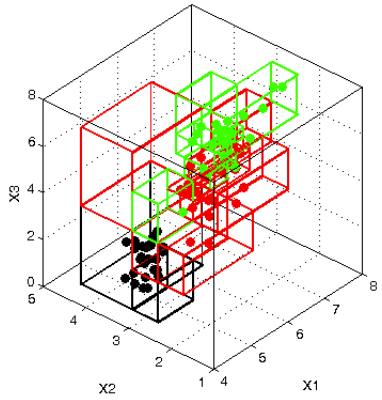

Figura 6.16: Resultados del entrenamiento para el problema del Iris con 3 rasgos.

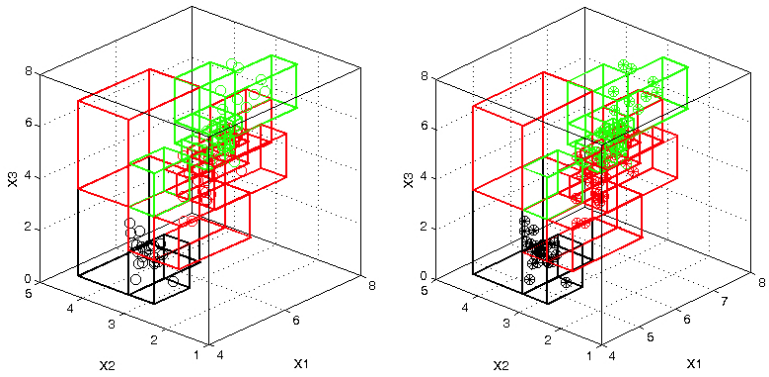

Figura 6.17: Resultados de la clasificación para el problema del Iris con 3 rasgos.

El algoritmo de entrenamiento propuesto se aplicó a las siguientes bases de datos:

Iris, Identificación del Vidrio, Desórdenes del Hígado, Bloques de Página, Segmentación de Imagen, Reconocimiento de Dígitos y Reconocimiento de Letras. La base de datos de Identificación del Vidrio tiene 6 clases, 9 atributos y 214 muestras (la clase 6 no se considera debido a que tiene muy pocas muestras). La base de datos de Desórdenes del Hígado tiene 2 clases, 6 atributos y 345 muestras mientras que la de Bloques de Página tiene 5 clases, 10 atributos y 5473 muestras. Como en el caso de la base de datos del Iris, todas las bases de datos anteriores se dividieron en dos subconjuntos para obtener los patrones de entrenamiento y de prueba. La base de datos de Segmentación de Imagen tiene 7 clases, 19 atributos y 210 muestras para el entrenamiento y 2100 muestras para la prueba. Por otra parte, la base de datos de Reconocimiento de Dígitos tiene 10 clases, 16 atributos y 10992 muestras con un conjunto de entrenamiento de 7494 muestras y un conjunto de prueba de 3498 muestras. La base de datos de Reconocimiento de Letras tiene 27 clases, 16 atributos y 20000 muestras, generalmente 16000 muestras son para entrenamiento y 4000 para prueba. Para las bases de datos anteriores solo se utilizó el 30 % de las muestras de entrenamiento y prueba. Para las bases de datos de Segmentación de Imagen y Reconocimiento de Dígitos solo se usaron 10 atributos y para la de Reconocimiento de Letras se usaron 5 rasgos.

Para seleccionar los atributos más importantes, se aplicó el algoritmo "InfoGainAttributeEval" del software WEKA, este algoritmo evalúa la importancia de un atributo midiendo la ganancia de información con respecto a la clase. La tabla 6.6 presenta los resultados de clasificación obtenidos con el algoritmo propuesto, MLP, SVM y RBN aplicados a las diferentes bases de datos. Como se puede ver, en casi todos los casos el algoritmo propuesto obtiene el menor error de clasificación. Además, en el algoritmo propuesto, el valor de los parámetros no se establece de forma manual para el proceso de entrenamiento, lo cual es una ventaja para el usuario.

Tabla 6.6: Tabla de comparación de resultados de clasificación para problemas de p clases y n atributos.

	MLP	SVM		RBN		NMPD-P		
Base de datos	% Error	Grado	%Error	# Clusters	%Error	# Dendritas	M	% Error
Iris	6.77	1	4.00	2	5.33	20	0.30	2.67
Vidrio	31.46	5	31.68	6	31.68	64	0.19	30.69
Hígado	39.50	3	38.95	4	35.47	132	0.88	35.47
Bloques de Página	4.93	6	5.27	4	5.12	304	0.29	4.86
Seg. de Imagen	27.33	1	26.71	2	21.71	92	0.54	24.14
Rec. de Letras	40.33	2	43.33	2	41.33	1439	0.72	38.75
Rec. Dígitos	15.38	1	13.23	2	14.09	2259	0.25	9.55

6.3. Resultados experimentales con datos financieros

El algoritmo de entrenamiento se aplicó para tratar de predecir la dirección del movimiento diario (a la alza o a la baja) del mercado mexicano. El índice seleccionado para los experimentos es el IPC "Índice de Precios y Cotizaciones" que es uno de los principales indicadores de la Bolsa Mexicana de Valores. El IPC se integra por las acciones de las empresas más representativas, pertenecientes a varios sectores de la economía, que cotizan en la Bolsa Mexicana de Valores. Los datos utilizados en este trabajo muestran la dirección del movimiento diario del precio de cierre del IPC en la Bolsa Mexicana.

Para predecir el movimiento de la dirección del IPC, se utilizaron diez indicadores técnicos como variables de entrada, dado que estudios empíricos muestran que los precios de las acciones se correlacionan con diversos indicadores técnicos [87]. Las entradas utilizadas en este estudio se seleccionaron de un conjunto de indicadores técnicos que incluyen una variedad de medias móviles, indicadores de fuerza relativa, osciladores y rendimientos retrasados. Para la selección de los atributos más útiles, se utilizó el algoritmo "InfoGainAttributeEval" del software WEKA. La tabla 6.7 muestra los indicadores técnicos seleccionados, los rendimientos retrasados se calcularon para 4 días.

Tabla 6.7: Indicadores técnicos seleccionados.

Indicadores	Fórmula
Larry William's R	$\frac{H_n - C_t}{H_n - L_n}$
K estocástica	$\frac{C_t - LL_{t-n}}{HH_{t-n} - LL_{t-n}}$
D estocástica	$\frac{\sum_{i=0}^{n-1} K_{t-i}}{n}$
Índice de fuerza	$(C_t - C_{t-1})\, Volume_t$
CCI (Commodity Channel Index)	$\frac{M_t - SM_t}{0.015 D_t}$
RSI (Relative Strength Index)	$100 - \frac{100}{1+\left(\sum_{i=0}^{n-1} Up_{t-1}/n\right)/\left(\sum_{i=0}^{n-1} Dw_{t-1}/n\right)}$
Rendimientos retrasados	$\frac{S_t - S_{t-1}}{S_{t-1}}$

C_t precio al cierre, L_t precio bajo, H_t precio alto en t, LL_t y HH_t son el precio más bajo y más alto en los últimos t días, Up_t es el cambio de precio a la alta, Dw_t es el cambio de precio a la baja en el tiempo t, S_t es el precio en t, $M_t = H_t + L_t + \frac{C_t}{3}$, $SM_t = \frac{\left(\sum_{i=1}^{n} M_{t-i+1}\right)}{n}$, $D_t = \frac{\left(\sum_{i=1}^{n} |M_{t-i+1} - SM_t|\right)}{n}$.

Para predecir la dirección del movimiento del IPC se utilizó un conjunto de datos que cubre el período comprendido entre el 3 enero de 2010 hasta el 31 diciembre de 2012, un total de 755 días de negociación. Los datos históricos se obtuvieron de la página de finanzas de Yahoo. La dirección del cambio diario en el IPC se categorizó como "alza" o "baja" . Si el índice IPC en el tiempo t es mayor que en el tiempo t - 1, la dirección en t es "alza" que es una dirección de incremento. Si el índice IPC en el momento t es menor que en el momento t - 1 , la dirección en t es "baja", es decir una dirección descendente. El número de días con dirección a la alza es 409, mientras que el número de días con dirección a la baja es 346. Es decir, 54.17 % de todos los casos tienen una dirección a la alza y 45.83 % de ellos tienen una dirección a la baja.

Para realizar el análisis, el conjunto de datos se dividió de la siguiente manera: un conjunto de entrenamiento que comprende datos entre enero de 2010 y diciembre de 2011 (503 observaciones) y un conjunto de prueba que se utiliza para predecir el comportamiento del IPC en enero de 2012. En el siguiente análisis, los datos de enero de 2012 se agregan al conjunto de entrenamiento y se predice el comportamiento en febrero de 2012. De esta forma se incorporan los datos del mes anterior para predecir el siguiente hasta diciembre de 2012. La figura 6.18 muestra la gráfica del porcentaje de la exactitud de predicción obtenida por MLP, SVM y NMPD-P, respectivamente.

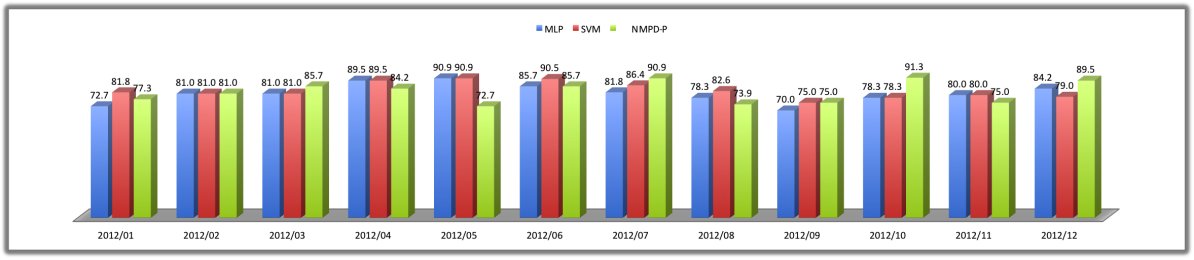

Figura 6.18: Porcentaje de predicción obtenido para cada mes de 2012, mediante MLP, SVM y NMPD-P, respectivamente.

Como se puede observar en la Figura 6.18, no es evidente un rendimiento superior de ninguno de los algoritmos. La tabla 6.8 muestra la media del porcentaje de la predicción de los doce meses como un indicador del rendimiento de los tres algoritmos. Los resultados revelan que la SVM tiene un porcentaje de predicción más alto que la NMPD-P y el MLP tuvo el peor rendimiento.

Tabla 6.8: Porcentaje promedio de predicción acertada del IPC durante 2012.

	MLP	SVM	NMPD-P
% Promedio de Predicción	81.11	82.98	81.85

Para este problema los falsos positivos son importantes. Un falso positivo significa que la inversión realizada en una acción que al final de la jornada financiera finalmente va a la baja implica un riesgo mucho más alto que solo no invertir en una acción que va a la alza. La tabla 6.9 muestra el total de falsos positivos obtenidos en los doce meses del 2012. Como se puede ver, aunque la SVM tiene el mejor porcentaje de predicción, la NMPD-P presentó el menor número de falsos positivos, por lo que el rendimiento del algoritmo propuesto es bueno y comparable al de SVM. Por otra parte, en el caso de SVM, fue necesario modificar el exponente del núcleo polinomial entre uno y tres para obtener el mejor rendimiento, si el exponente se mantiene fijo el rendimiento de SVM podría ser muy bajo.

Tabla 6.9: Número de falsos positivos de cada mes del 2012.

	MLP	SVM	NMPD-P
Total de falsos positivos	24	20	19

Como prueba final, se utilizaron los datos de los años 2010, 2011 y 2012 para predecir el movimiento de la dirección del IPC del 2 enero de 2013 al 15 de febrero de 2013. La tabla 6.10 presenta los resultados. Como puede verse, los resultados proporcionados por la NMPD son muy satisfactorios para este conjunto de datos.

Tabla 6.10: Tabla de comparación de MLP, SVM y NMPD, para la predicción del IPC en un período de seis semanas (2 de enero de 2013 a 15 de febrero de 2013).

MLP		SVM	NMPD-P		
# Neuronas	% Predicción	% Predicción	# Dendritas	M	% Predicción
8	84.375	84.375	454	0.33	90.7

La tabla 6.11 muestra las matrices de confusión, estos resultados revelan que la NMPD obtiene un buen resultado, ya que solo tiene un falso positivo, mientras que el SVM tiene tres falsos positivos y MLP presenta cuatro de ellos.

Tabla 6.11: Matriz de confusión para la predicción del IPC en un período de seis semanas (2 Enero de 2013 a 15 Febrero de 2013).

MLP	alza	baja
alza	16	1
baja	4	11

SVM	alza	baja
alza	15	2
baja	3	12

NMPD-P	alza	baja
alza	15	2
baja	1	14

6.4. Resumen

En este capítulo se detallan los experimentos realizados para determinar el desempeño del algoritmo de entrenamiento para NMPD con datos sintéticos y reales. Los datos sintéticos utilizados fueron el problema de los espirales y el de Ripley, con los que se encontraron resultados para mejorar el desempeño del algoritmo, como cambiar la función de activación del limitador duro por una función de mapeo lo cual permite que los patrones en áreas de indecisión se clasifiquen en la clase más cercana, y aplicar evolución diferencial para determinar el valor óptimo del margen M. Los resultados se compararon con el algoritmo de entrenamiento de eliminación de Ritter y MLP, obteniéndose un desempeño favorable. De igual forma, se presentan los resultados con datos reales para clasificación

en imágenes, bases de datos de la UCI y datos del mercado financiero mexicano. En estos casos, las comparaciones se realizaron con MLP, RBN y SVM, obteniéndose resultados equiparables o superiores en algunos problemas.

Capítulo 7

Resultados experimentales en aplicaciones con plataformas reales

En este capítulo se presentan aplicaciones del método de entrenamiento propuesto para NMPD en problemas de clasificación probados en plataformas físicas como el Kinect de Microsoft, los robots humanoides Bioloid y Nao y Lego Mindstorm.

7.1. Reconocimiento de objetos usando Kinect

En este experimento se busca implementar un sistema de reconocimiento de objetos que utilice la información de profundidad proporcionada por el Kinect para el proceso de segmentación [88].

El diagrama del sistema de reconocimiento de objetos se muestra en la figura 7.1. Para el reconocimiento, se utiliza una NMPD como clasificador, para representar al objeto se usan los dos primeros momentos invariantes de Hu y la información de color en el espacio HSI. Para el proceso de segmentación se utiliza la información de profundidad proporcionada por el Kinect.

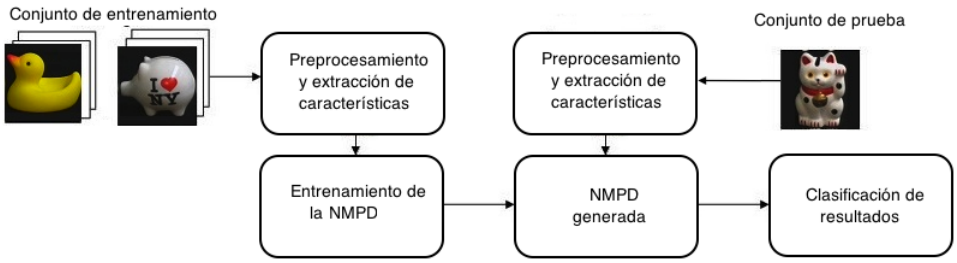

Figura 7.1: Proceso de reconocimiento de objetos.

En las siguientes secciones se explicarán cada una de las etapas que intervienen en el proceso de reconocimiento de objetos así como los resultados experimentales obtenidos. En la siguiente sección se explica el proceso de segmentación propuesto.

7.1.1. Proceso de segmentación

El proceso de segmentación en imágenes es la tarea de dividir una imagen en regiones consistentes y se utiliza típicamente para identificar objetos u otra información relevante en imágenes digitales. Existen varios algoritmos que solo usan información de color para realizar la segmentación lo que lleva consigo algunas dificultades. Para obtener una buena segmentación, usualmente los objetos a segmentar deben tener un gran contraste con el fondo para que sea fácil distinguir al objeto. También es importante que el fondo sea homogéneo para evitar que áreas erróneas sean marcadas como objetos, además las condiciones de iluminación deben ser constantes. Para resolver este problema se utilizó la información de profundidad proporcionada por el sensor del dispositivo Kinect. De esta forma, el problema de segmentación de objetos en escenarios reales es más simple porque los límites de segmentación del objeto se pueden definir como los bordes de los datos de profundidad.

En la figura 7.2 se muestran las características con las cuenta el sensor Kinect de Microsoft, en este trabajo se usó la versión para Windows cuyo sensor de profundidad tiene un rango efectivo de reconocimiento de aproximadamente 0.5 a 3 metros.

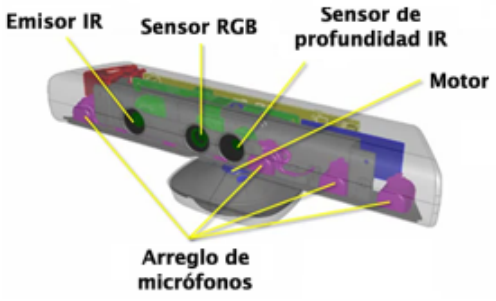

Figura 7.2: Características del Kinect

Para la segmentación es muy importante que las cámaras de color y de profundidad del Kinect estén alineadas ya que la detección del objeto en la imagen de profundidad (su contorno) debe coincidir con la imagen de color para lograr una buena segmentación. Para calibrar las dos cámaras se utilizó el proceso de calibración reportado en [89].

Así, la información de profundidad se usó para dividir la imagen en zonas donde solo la primera se toma en cuenta, esta zona de reconocimiento va de 0.5 a 0.9 metros y únicamente el objeto que se localiza a esta distancia del Kinect se considera para el reconocimiento.

Para explicar el proceso de segmentación se seleccionó un objeto como ejemplo. La figura 7.3a) presenta el objeto de interés en la escena, aplicando la división de profundidad, se reconoce la primera zona y se presenta en la figura 7.3b), como se puede ver, el objeto se segmentó correctamente. Sin embargo, los datos de profundidad tienen ruido, especialmente en las siluetas, además el Kinect también detecta puntos de la superficie. Para reducir estos problemas, la imagen de color se binariza (figura 7.3c)) y el ruido se elimina usando filtros de erosión y dilatación, la figura 7.3d) presenta la imagen filtrada y la figura 7.3e) muestra la correspondiente imagen de color. Para obtener una imagen con la menor cantidad de ruido para obtener sus rasgos característicos, se determinan los puntos mínimo y máximo en x y y de la imagen binaria filtrada y se calcula un área rectangular, esta área se selecciona de la imagen de color original y el resultado para este ejemplo se presenta en la figura 7.3f).

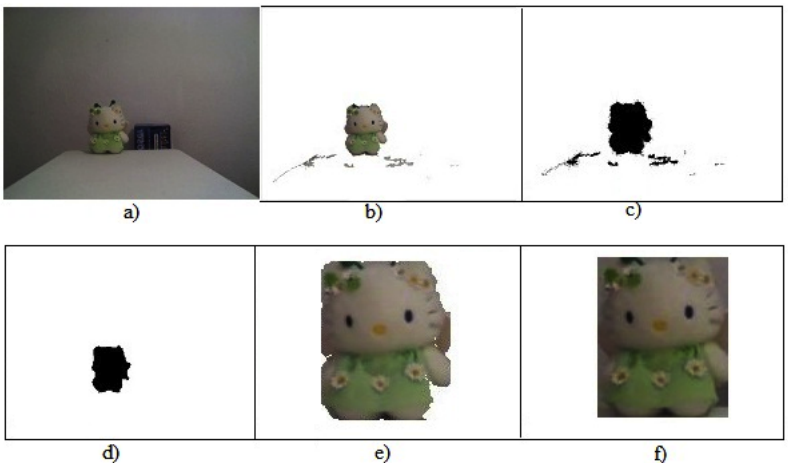

Figura 7.3: a) Escenario para la captura de la imagen, b) Segmentación de profundidad, c) Imagen segmentada binaria, d) Imagen segmentada filtrada, e) Imagen segmentada de color, f) Imagen de prueba.

7.1.2. Preprocesamiento y extracción de rasgos característicos

Las características utilizadas para representar al objeto son los dos primeros momentos invariantes de Hu y la información de color en el espacio HSI.

7.1.2.1. Momentos invariantes de Hu

Los momentos invariantes geométricos fueron introducidos por Hu basándose en la teoría de los invariantes algebraicos [90]. Las características invariantes de imagen o forma permanecen sin cambio si la imagen es afectada por cualquier combinación de las siguientes transformaciones: translación, rotación y escala. Por lo tanto, los momentos invariantes se pueden usar para reconocer un objeto aún si ha sido alterado por ciertas transformaciones.

Los momentos de orden 2D $(p+q)$ de una imagen digital $f(x,y)$ de tamaño $M \times N$ se definen como [91]

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^p y^q f(x,y) \tag{7.1}$$

donde $p,q = 0,1,2,..$ son enteros. Los momentos centrales correspondientes de orden $(p+q)$ se definen como:

$$\mu_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (x-\bar{x})^p (y-\bar{y})^q f(x,y) \tag{7.2}$$

para $p,q = 0,1,2,..$ donde $\bar{x} = \frac{m_{10}}{m_{00}}$ and $\bar{y} = \frac{m_{01}}{m_{00}}$.

Los momentos centrales normalizados se denotan como η_{pq} y se definen por:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \tag{7.3}$$

donde $\gamma = \frac{p+q}{2} + 1$ para $p+q = 2,3,...$

Un conjunto de siete momentos invariantes se derivan del segundo y tercer momentos

$$\phi_1 = \eta_{20} + \eta_{02} \tag{7.4}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{7.5}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - 3\eta_{03})^2 \tag{7.6}$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{7.7}$$

$$\begin{gathered}\phi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] + \\ (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]\end{gathered} \tag{7.8}$$

$$\phi_6 = (\eta_{20} - \eta_{02})\left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \tag{7.9}$$

$$\begin{gathered}\phi_7 = (3\eta_{21} - \eta_{30})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] + \\ (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]\end{gathered} \tag{7.10}$$

En este trabajo solo se usaron el primer y segundo momentos invariantes para representar al objeto.

7.1.2.2. Características de color

En esta investigación el espacio de color HSI se utiliza como una alternativa al espacio RGB ya que HSI es menos sensible a los cambios de luz que el RGB. El espacio de color HSI considera la imagen como una combinación de los componentes: matiz ("hue" H), saturación (S) e intensidad (I).

Los componentes RGB de una imagen se utilizan para obtener la componente H de representación de color HSI mediante:

$$H = cos^{-1}\left(0.5(R-G) + (R-B)/\sqrt{(R-G)^2 + (R-B)(G-B)}\right). \tag{7.11}$$

Para obtener un valor de H que esté en el rango de 0 a 360 grados es necesario restar H a 360° cuando $B > G$. Las fórmulas para la saturación y la intensidad son respectivamente:

$$S = 1 - \frac{3}{R+G+B} min(R,G,B), \quad I = \frac{1}{3}(R+G+B). \tag{7.12}$$

Para determinar los colores negro, blanco y gris, se utiliza la componente de saturación, si S es cercano a 0 entonces el color es negro, si es cercano a 1 el color es blanco y en otro caso el color es gris.

7.1.3. Resultados experimentales

La base de datos experimental que se generó consta de 10 objetos con 10 imágenes cada una. Las imágenes fueron capturadas usando la cámara RGB del Kinect para Windows con una resolución de 640x480 y la imagen de profundidad se obtuvo con la misma resolución habilitando el modo cercano del Kinect. La figura 7.4a) muestra las imágenes RGB del conjunto de objetos. Como se puede observar, algunos de los objetos son muy similares en forma, por lo tanto el color es muy importante para distinguirlos.

Figura 7.4: Conjunto de 10 objetos capturados con el dispositivo Kinect.

La figura 7.3a) presenta un ejemplo del fondo real donde se capturaron las imágenes para el experimento. Estas imágenes no fueron tomadas bajo condiciones de luz controladas y hay otros objetos en la escena además del objeto de interés. La figura 7.5 presenta 4 ejemplos de las 10 diferentes vistas del objeto, las imágenes muestran el objeto segmentado en cada escena aplicando el proceso de segmentación de profundidad.

Figura 7.5: Imágenes de muestra de un objeto.

Para la representación de los objetos se usaron los primeros dos momentos de Hu y para el color el sistema detecta 8 colores (rojo, amarillo, verde, azul, magenta, blanco, gris y negro) a partir de la escala de color HSI. Las características de color se representan como el porcentaje de cada color en el objeto. Por lo tanto, los objetos se caracterizaron mediante 10 rasgos (8 de color y 2 de forma).

Las etapas de preprocesamiento, extracción de características y el algoritmo de entrenamiento para la NMPD se implementaron en MATLAB 7.11. Para evaluar el desempeño de la NMPD, se realizaron comparaciones con MLP y SVM. Los algoritmos MLP y SVM se aplicaron usando el software Weka 3-6-9. Para el MLP con una capa oculta, los parámetros de entrenamiento se establecieron como: razón de aprendizaje=0.3, momento=0.2, función de activación sigmoide. Para el SVM se usó un núcleo polinomial de grado 2.

Para el conjunto de objetos de prueba, el sistema se entrenó con 3 imágenes tomadas al azar de cada objeto y el resto de las imágenes (7) se usaron para prueba. Para tener una generalización del error estimado, se realizaron 5 experimentos usando validación cruzada $k\text{-}folds$ con $k = 10$ para obtener las muestras de entrenamiento y de prueba. La tabla 7.1 presenta el promedio del porcentaje de reconocimiento y la desviación estándar obtenida. Como se puede ver, el algoritmo de entrenamiento propuesto para la NMPD y la SVM tienen el mismo porcentaje de reconocimiento y mejoran los resultados del MLP,

por lo que el desempeño del algoritmo propuesto es satisfactorio para el reconocimiento de objetos. También se puede notar que la desviación estándar de la NMPD es menor que la desviación estándar de SVM y MLP, lo cual es una buena característica porque el porcentaje de reconocimiento es más estable para diferentes conjuntos de entrenamiento.

Tabla 7.1: Tabla de comparación de MLP, SVM y NMPD para el conjunto de 10 objetos.

	MLP	SVM	NMPD
% Promedio de reconocimiento	80.71	81.85	81.85
Desviación estándar	5.17	7.01	4.96

Seleccionando vistas específicas de los objetos, el porcentaje de reconocimiento se incrementó a 90 % para la NMPD y SVM y 87.15 % para el MLP, por lo que es conveniente realizar el entrenamiento con vistas predeterminadas de los objetos.

7.2. Reconocimiento de objetos usando robots humanoides

Para probar el reconocimiento de objetos usando el algoritmo de entrenamiento propuesto para la NMPD en tiempo real, se desarrolló un sistema en C# y se utilizaron los robots humanoides Biolod y NAO.

El objetivo de este experimento es que el robot sea capaz de reconocer una serie de objetos, una vez que tales objetos se le han enseñado en el proceso de aprendizaje que se realiza con una NMPD en tiempo real. Cada objeto está asociado con un movimiento del robot, por lo que al reconocer cierto objeto el robot humanoide realizará una acción.

En la figura 7.6 se muestra un esquema del espacio de trabajo que debe cumplir con las siguientes condiciones:

- El robot humanoide se encuentra enfrente de una mesa donde se colocan los objetos.
- El Kinect se posiciona de manera que capte tanto al espacio de trabajo como al robot humanoide.

En este experimento solo se utiliza la cámara RGB del Kinect por lo que se aplica el proceso de reconocimiento de objetos presentado en [92]. En este caso, se utilizan imágenes en escala de grises y el proceso de segmentación se realiza mediante el algoritmo de Otsu [93]. Las características seleccionadas para representar a los objetos son los dos primeros momentos invariantes de Hu y la media y la desviación estándar obtenidas de la

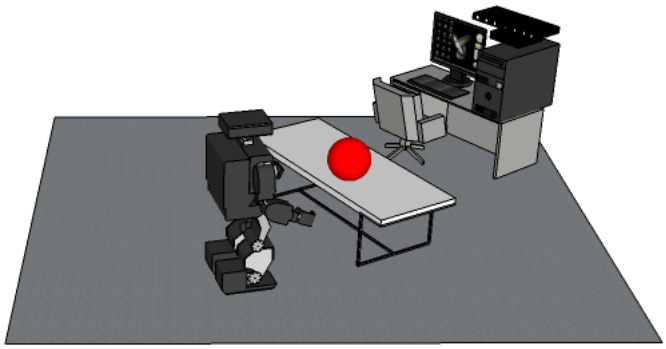

Figure 7.6: Esquema del espacio de trabajo.

distribución de los pixeles en las imágenes en escala de grises. Estas cuatro características se usan para entrenar la NMPD.

La figura 7.7 presenta una gráfica de los atributos de 5 objetos en la que solo se presentan 3 características (los dos primeros momentos de Hu y la media) para poder desplegar el resultado gráficamente.

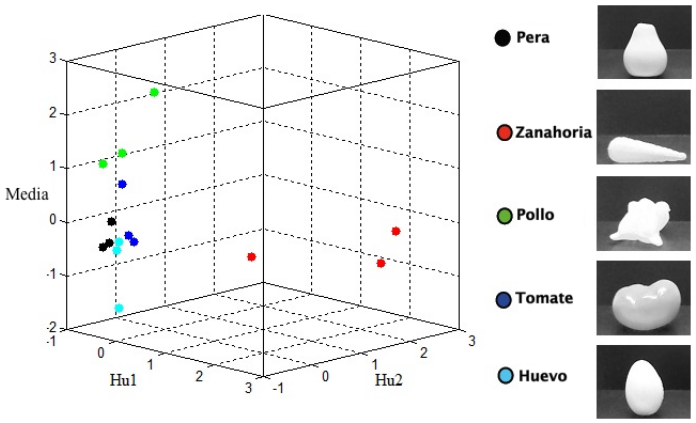

Figure 7.7: Características de 5 objetos.

En la figura 7.8 se presenta el resultado del entrenamiento de la NMPD con el algoritmo de entrenamiento propuesto para lograr la clasificación de los objetos.

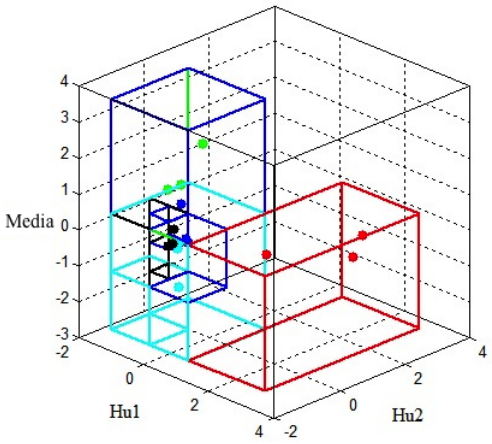

Figure 7.8: Resultado del entrenamiento para el reconocimiento de objetos.

En las siguientes subsecciones se explica cómo se utilizó este proceso de reconocimiento de objetos para lograr que los robots humanoides dependiendo del objeto reconocido realizaran cierta acción.

7.2.1. Reconocimiento de objetos usando el robot Bioloid

El robot humanoide Bioloid Premium Kit de Robotis cuenta con 18 grados de libertad, un sensor de distancia y un giroscopio. En la figura 7.9 se muestra el robot humanoide y sus características.

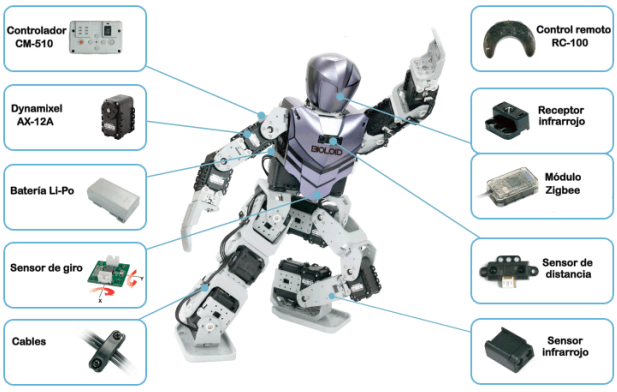

Figura 7.9: Robot humanoide Bioloid Premium Kit (imagen tomada del manual del Bioloid)

El experimento planteado consiste en colocar al robot en un espacio de trabajo específico y lograr que reconozca diferentes objetos y que con cada objeto, el robot realice una acción específica.

Físicamente, el espacio de trabajo utilizado para la realización de los experimentos se muestra en la figura 7.10.

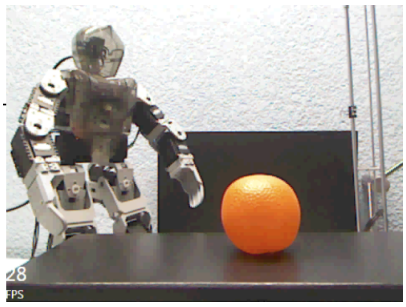

Figura 7.10: Espacio de trabajo real.

El funcionamiento del sistema es el siguiente:

1. Se realiza el entrenamiento de la red mostrando varias vistas del objeto, en este experimento específico se usaron 3 vistas de 3 objetos diferentes: naranja, pera y zanahoria.
2. Una vez obtenida la NMPD correspondiente, se hace la etapa de prueba mostrando cada uno de los objetos.
3. Si el objeto reconocido es la naranja, el robot levanta el brazo izquierdo y el sistema dice la palabra “orange”.
4. Si se muestra la pera, el robot levanta el brazo derecho y el sistema dice la palabra “pear”.
5. Finalmente, si el objeto reconocido es la zanahoria, el robot levanta ambos brazos y el sistema dice la palabra “carrot”.

En la figura 7.11 se muestran imágenes de los pasos 3, 4 y 5 respectivamente.

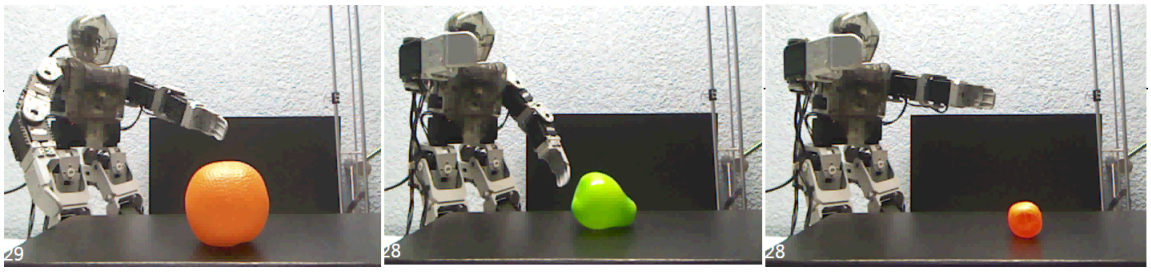

Figura 7.11: Reconocimiento de objetos.

Los resultados obtenidos al realizar el experimento fueron muy satisfactorios, ya que el sistema permitió realizar el reconocimiento exitoso en casos difíciles como el de la zanahoria que se presenta en la figura 7.11 en el que la imagen obtenida podría confundirse con la naranja. De 20 experimentos realizados, en 19 ocasiones el sistema reconoció correctamente al objeto presentado, por tanto se obtuvo un porcentaje de reconocimiento del 95 %.

7.2.2. Reconocimiento de objetos usando el robot NAO

El robot humanoide NAO cuenta con 25 grados de libertad, dos cámaras, sonares, sensores táctiles, sensores infrarrojos, sensor inercial, micrófonos, bocinas, un procesador ATOM Z530 1.6GHz y 1 GB de memoria RAM, todas estas características lo hacen una plataforma ideal para la investigación. En la figura 7.12 se muestran las características del robot humanoide NAO.

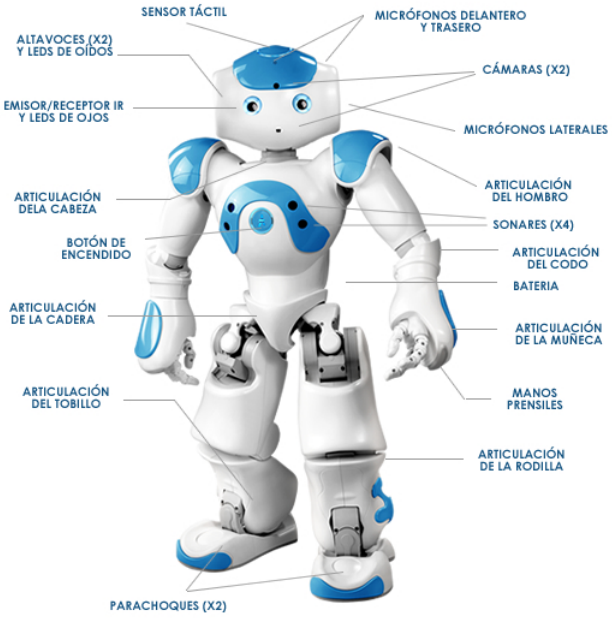

Figura 7.12: Robot NAO (imagen tomada del sitio del grupo Mediatec, http://www.grupo-mediatec.com/robotica/caracteristicas_nao.html).

El robot humanoide NAO se puede programar en diferentes plataformas como C++, Python, JAVA, etc., una de ellas es C# por lo que se seleccionó este programa para que fuera compatible con la parte del Kinect.

El experimento realizado con el robot humanoide NAO fue el mismo que el del robot Bioloid, en este caso de 20 experimentos realizados el porcentaje de reconocimiento fue de 100 %, esto se debe a que como se mencionó en la sección 7.1, si las imágenes de entrenamiento son las más adecuadas el porcentaje de clasificación se puede incrementar.

En la figura 7.13 se muestras las imágenes del reconocimiento de objetos usando al robot NAO.

Como dato técnico se debe mencionar que para lograr la comunicación entre los robots humanoides y el Kinect se utilizaron las librerías del Kinect de Microsoft [94] y las librerías del robot Bioloid [95] y el robot NAO [96] para C#.

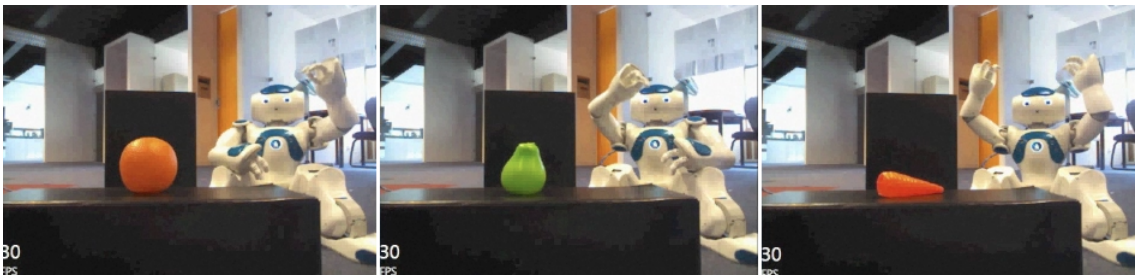

Figura 7.13: Reconocimiento de objetos.

7.3. Control compartido usando Lego

Este experimento se realizó en colaboración con la Dra. Kim Adams y el Dr. Patrick Pilarski en el Laboratorio de Tecnología de Asistencia de la Universidad de Alberta durante la estancia de investigación realizada como parte de este trabajo doctoral.

El objetivo es generar un sistema de control compartido que le permita a niños con habilidades diferentes mejorar su desempeño al realizar una tarea de seguimiento de trayectoria usando un robot móvil construido con Lego.

En la figura 7.14 se muestra una foto del kit de Lego Mindstorm NXT 2.0 utilizado en este experimento. La razón de usar esta plataforma es su bajo costo, que permite la posibilidad de que los niños puedan tener el robot en sus casas o escuelas.

Figura 7.14: Lego Mindstorm 2.0.

De acuerdo con [97], [98], los niños con habilidades diferentes pierden oportunidades de juego con la correspondiente pérdida de las ventajas del mismo, ya que en ocasiones no pueden alcanzar y manipular los juguetes debido a que presentan graves limitaciones en su capacidad de agarre o dificultad para controlar sus movimientos (por ejemplo, los niños con parálisis cerebral, atrofia muscular espinal, distrofia muscular). A menudo, la única interfaz que estos niños pueden controlar de forma fiable es un interruptor. Los interruptores pueden colocarse al lado de la parte del cuerpo que controlan más fácilmente,

por ejemplo, se pueden colocar interruptores a cada lado de la cabeza de un niño para que pueda activarlos con una inclinación. De esta manera los niños pueden controlar los robots de Lego usando interruptores, y dependiendo de su nivel físico y cognitivo, pueden hacer algunas tareas básicas con los robots como avanzar, retroceder y girar. Con el fin de realizar tareas más interesantes, los niños necesitan al menos ser capaces de entender la secuencia de las entradas de los botones. En la figura 7.15 se muestra un ejemplo de un espacio de trabajo real y de diferentes configuraciones para realizar actividades básicas.

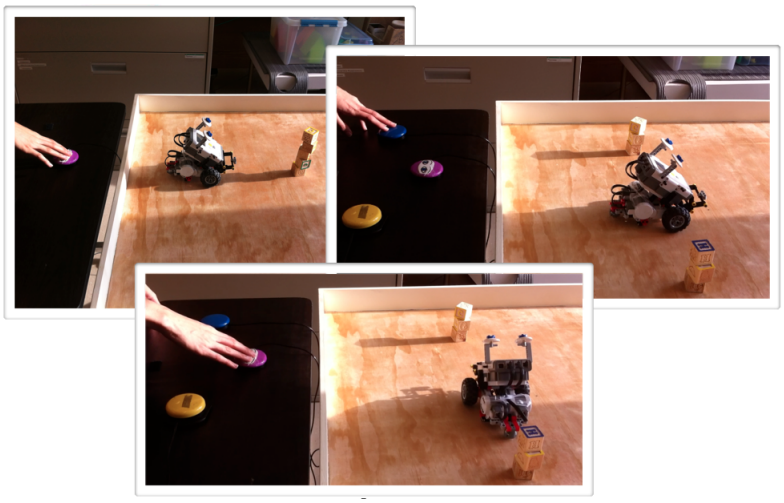

Figura 7.15: Espacio de trabajo real y ejemplos de configuraciones para realizar actividades básicas como avanzar, girar y retroceder.

El objetivo de este trabajo es que los niños que tienen un control muy limitado (ya sea debido a limitaciones físicas o cognitivas) puedan realizar tareas más desafiantes con los robots. Esto requiere que el robot asuma el control de la actividad. El nivel de control puede ser de plena autonomía del robot, totalmente teleoperado por el niño, o algún punto intermedio. En la actualidad, los terapeutas determinan qué nivel de control se le debe dar al niño, pero sería mejor si el sistema puede establecer automáticamente el nivel de control en función de las características y respuestas del usuario.

El problema que se plantea es seguir una trayectoria usando control compartido con el robot, en la figura 7.16 se presenta un ejemplo de un camino a seguir.

El sistema permite seguir la trayectoria con diferentes grados de autonomía:

- Nivel 1 - Autónomo: el robot hace toda la trayectoria después de que el niño pulsa y suelta el botón de avance.
- Nivel 2 - Inhibición: en la inhibición, el robot hace toda la trayectoria, siempre y cuando el niño presione y mantenga presionado el botón de avance. Si se suelta el botón, el robot se detiene.
- Nivel 3 - Lateralidad: para la lateralidad, el robot avanza si se pulsa y se suelta el botón verde (ver figura 7.17), y posteriormente se detiene en el primer punto de

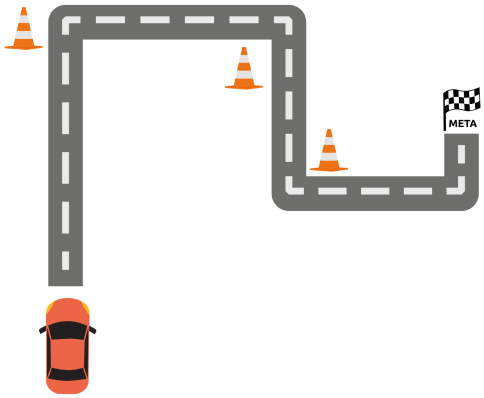

Figura 7.16: Ejemplo de una trayectoria a seguir.

decisión, donde el robot le da el control al niño y espera a que presione con éxito el interruptor de la izquierda o la derecha según corresponda. Si el niño pulsa el incorrecto, el robot toma el control y no se mueve y el sistema da un aviso con una luz que indica la dirección correcta para ayudarlo. Si se selecciona el botón correcto, el robot gira y pasa al siguiente punto de decisión.

- Nivel 4 - Secuenciación: los niños pueden utilizar la secuencia de botones (avance, izquierda, derecha) para seguir la trayectoria por sí mismos.

En la figura 7.17 se presenta la interfaz del programa desarrollado la cual tiene cuatro botones que realizan las siguientes acciones: verde - hacia adelante, rojo - hacia atrás, azul - girar a la izquierda y amarillo - girar a la derecha. En la aplicación de seguimiento de la trayectoria solo se utilizan los botones de avanzar, derecha e izquierda. El botón de retroceso se usa cuando se realizan actividades básicas para mover al robot. El programa se puede utilizar con los botones virtuales o mediante interruptores físicos conectados a la computadora para manipular al robot remotamente mediante conexión *bluetooth*. La interfaz se desarrolló en JAVA usando la librería de LeJOS [99] para manejar al Lego.

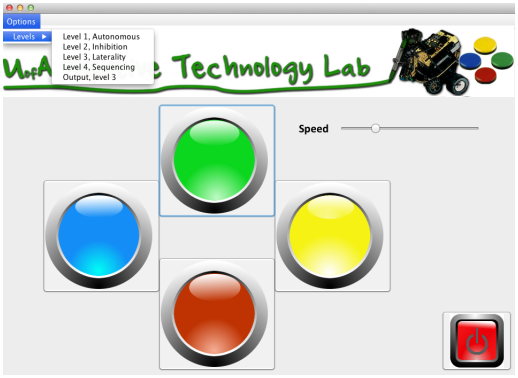

Figura 7.17: Interfaz del programa de control compartido.

La NMPD se utilizó para realizar el seguimiento de la trayectoria, que se planteó como un problema de clasificación de 4 clases (avanzar, girar a la izquierda, girar a la derecha

y parar) y como características se usaron las señales de 4 sensores de luz colocados en el carrito como se muestran en la figura 7.18 donde se presenta el espacio de trabajo real, el robot armado con Lego, el camino y los indicadores luminosos de ayuda.

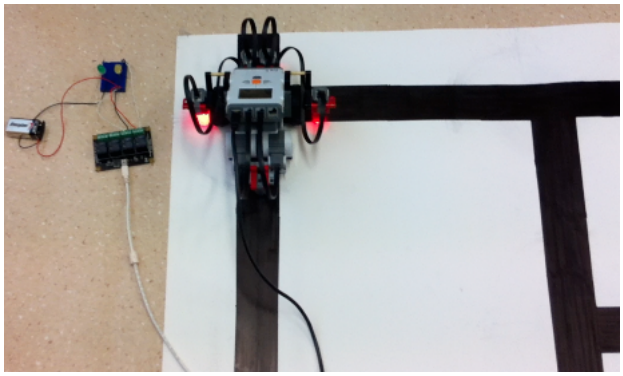

Figura 7.18: Montaje del experimento de control compartido.

En la tabla 7.2 se muestra un ejemplo de las clases y los valores obtenidos en cada uno de los sensores.

Tabla 7.2: Ejemplo de entradas al sistema de clasificación

Clase	Sensor 1	Sensor 2	Sensor 3	Sensor 4
Avanzar	41	32	30	50
Girar a la izquierda	28	33	33	59
Girar a la derecha	51	33	30	31
Parar	50	54	59	58

El sistema se entrenó tomando tres muestras de cada clase (12 muestras en el conjunto de entrenamiento) y la etapa de prueba se hizo en el momento en que el Lego sigue la trayectoria en el Nivel 1 de autonomía.

Se realizaron 10 pruebas y la trayectoria se completó satisfactoriamente 9 veces (90 % de reconocimiento). El problema se presenta cuando el sistema se prueba con diferentes condiciones de luz. Las pruebas se hicieron en la mañana (luz del sol) y al probarlo en la noche (luz artificial) la neurona entrenada ya no funcionó, esto se debe a la sensibilidad que tienen los sensores a los cambios de luz. Una opción es entrenar dos neuronas y seleccionar la que se va a utilizar dependiendo de si la prueba es en la mañana o en la noche, aunque lo más conveniente para incrementar la robustez del sistema es usar otro sensor, como una cámara, por ejemplo.

Durante el experimento en el Nivel 3 se guarda la información del número de errores en los puntos de decisión, el número de intentos hasta completar la trayectoria y el tiempo para recorrer la trayectoria satisfactoriamente. Esta información está disponible para el

terapista, el siguiente objetivo es usar nuevamente la NMPD para detectar el nivel de habilidad del niño y ajustar el nivel de control de forma automática.

7.4. Aplicaciones derivadas de la investigación

Cabe mencionar que el resultado de este trabajo de investigación ya ha sido utilizado por investigadores del ITESM Campus Guadalajara para resolver exitosamente problemas reales como la detección de vasos sanguíneos en imágenes de retina para ayudar a la detección de enfermedades como diabetes, arterioesclerosis e hipertensión [100], [101]. En la figura 7.19 se presenta el resultado de la segmentación de los vasos sanguíneos.

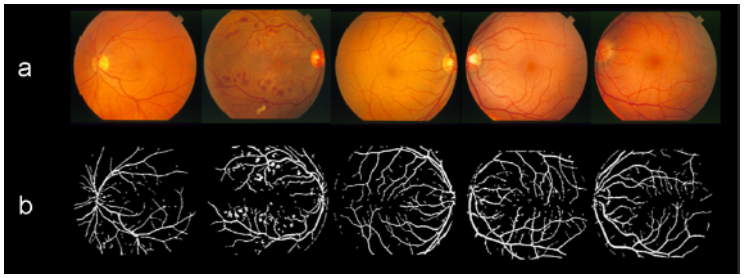

Figura 7.19: a) Imágenes originales de la retina, b) Resultado de la segmentación (imagen tomada de [101]).

Otra aplicación es la clasificación de movimientos de la mano usando señales cerebrales obtenidas mediante el sensor no invasivo EPOCH [102]. En la figura 7.20 se presenta la secuencia temporal de un ensayo durante la ejecución del experimento de detección de los movimientos de la mano y una foto del ambiente experimental utilizado.

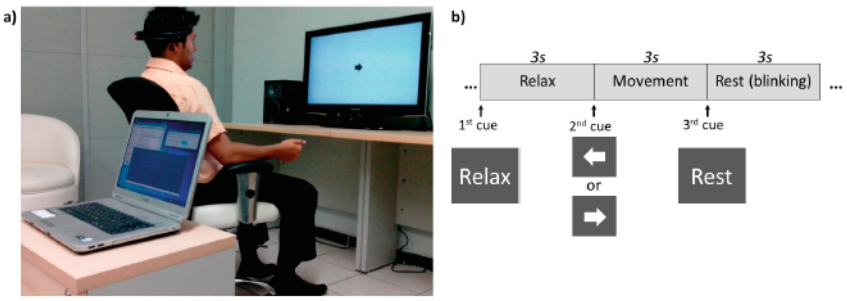

Figura 7.20: (a) Foto de la configuración experimental que muestra un participante sentado en frente a la pantalla del ordenador usando el sistema de registro de señales (b) secuencia temporal de un ensayo durante la ejecución del experimento (imagen tomada de [102]).

7.5. Resumen

En este apartado se presentan los experimentos que permiten validar el objetivo de utilizar el algoritmo de entrenamiento propuesto en aplicaciones en tiempo real. Para dicho propósito se emplearon las plataformas: Kinect, robots humanoides Bioloid y NAO y Lego Mindstorm, con las que se obtuvieron resultados aceptables en cada una de las aplicaciones. Además de los experimentos realizados en este trabajo, el algoritmo de entrenamiento ha sido utilizado por otros investigadores, quienes comprobaron los beneficios de la propuesta.

Capítulo 8

Conclusiones, trabajo a futuro y recomendaciones

En este capítulo se presentan las conclusiones a las que se llegó a lo largo del desarrollo de esta investigación. Se establecen las directrices para el desarrollo de investigaciones futuras y finalmente, se mencionan las recomendaciones consideradas para obtener mejores resultados en la solución de los problemas planteados.

8.1. Conclusiones

En este trabajo de investigación se propuso un método de entrenamiento eficiente para neuronas morfológicas con procesamiento dendral. Para mostrar la eficiencia del algoritmo se realizaron una serie de experimentos de clasificación de patrones, tanto en datos obtenidos de manera sintética como en bases de datos y aplicaciones reales.

Entre los datos sintéticos se encuentran el problema del espiral y los datos de Ripley, con los que se realizaron experimentos para comparar el desempeño con el algoritmo de entrenamiento de eliminación de Ritter, comprobándose de esta manera las ventajas del algoritmo propuesto.

Posteriormente, el método de entrenamiento se aplicó a problemas con datos reales de p clases y n dimensiones. Se utilizaron varias bases de datos del repositorio de la UCI (Iris, Vidrio, Hígado, etc.), así como datos obtenidos de imágenes de objetos reales e información del principal índice financiero mexicano (IPC). Los resultados de clasificación se compararon con los de MLP, SVM y RBN, obteniéndose resultados equiparables y en algunos casos superiores.

Asimismo, se realizaron experimentos en plataformas reales. Para lograr la segmentación exitosa de objetos tridimensionales en un ambiente no controlado se utilizó la información de profundidad proporcionada por el sensor de profundidad del Kinect, para realizar una separación por capas que permitió determinar la segmentación correcta del objeto.

El método de entrenamiento propuesto hizo posible realizar el entrenamiento en línea de una NMPD y su funcionamiento se probó en el reconocimiento de objetos, de manera satisfactoria, mediante el uso de dos robots humanoides (Bioloid y NAO).

Las NMPD se aplicaron al seguimiento de una trayectoria en un experimento de control compartido para ayudar a niños con capacidades diferentes a completar una tarea.

Todos estos experimentos permitieron comprobar las características principales del método propuesto:

- convergencia en un número finito de pasos,
- clasificación perfecta del conjunto de entrenamiento,
- no hay traslape entre los hipercubos,
- no existen áreas de indecisión en el proceso de prueba,
- no hay dependencia del orden de presentación de las muestras para el entrenamiento por lo que los resultados del mismo son reproducibles,
- la estructura dendrítica se construye dinámicamente con base en el conjunto de datos de entrenamiento, en lugar de que dicha estructura se predefina,
- no es necesario ajustar parámetros para el entrenamiento,
- con la mejora del algoritmo, el tiempo de entrenamiento ya no crece exponencialmente al incrementarse el número de rasgos,
- el método de entrenamiento es muy intuitivo y fácil de explicar, por lo que sirve como una herramienta eficaz para las personas con poca experiencia en algoritmos de aprendizaje automático.
- la propuesta permitió resolver problemas reales de clasificación de patrones que van desde la predicción de la dirección del movimiento diario del IPC hasta aplicaciones de robótica.

8.2. Trabajo a futuro y recomendaciones

Es necesario seguir realizando aplicaciones del método de entrenamiento propuesto con la mejora en tiempo para validar su funcionamiento en la clasificación de patrones en espacios de alta dimensión.

Las características del algoritmo de entrenamiento podrían permitir que el aprendizaje sea incremental, es decir, que si se agrega un patrón al conjunto de entrenamiento, en algunos casos, no se requiera ejecutar nuevamente la etapa completa de entrenamiento, sino solo realizar una redefinición de la NMPD donde sea necesario. Esto es deseable para reducir los requisitos de memoria y tiempo de cálculo, por lo cual se recomienda dotar de esta característica al algoritmo de entrenamiento. Las reglas que se pueden utilizar son: si la muestra pertenece a una clase conocida y se clasifica bien no se lleva a cabo una ninguna acción. Si el patrón queda afuera de todas las regiones se añade una nueva dendrita y si la muestra está en un hipercubo que no corresponde a su clase se realiza el proceso de división.

Una propuesta para tomar la decisión de clasificación en los puntos que se encuentran en los límites entre hipercubos es asignar el patrón a la clase con la mayor densidad de puntos en lugar de seleccionar la clase correspondiente a la primera dendrita generada.

Por otra parte, dadas la características del algoritmo de entrenamiento propuesto se considera que puede ser un método útil para el entrenamiento de diferentes tipos de RNA; la primera aproximación se realizó en RBN [103]. Es recomendable seguir investigando en esta vertiente.

Finalmente, aunque el método de entrenamiento propuesto para la NMPD obtiene resultados satisfactorios, los patrones clasificados son estáticos y como consecuencia se produce también una salida fija, por lo que sería conveniente incluir conceptos que incorporen señales variables en el tiempo, incluso señales que correspondan a un vector o imagen que cambia con el tiempo. Para lograr un enfoque dinámico también se podría investigar la incorporación de conceptos de redes pulsantes, tanto para incluir el tiempo como para crear sistemas más parecidos al modelo real de la neurona; o plantear un modelo dinámico capaz de simular la retracción dendrítica equivalente a la eliminación de las dendritas que reconocen regiones alrededor de patrones aislados.

Apéndice A

Publicaciones surgidas a partir de esta investigación

En este apéndice se presentan las referencias a los artículos publicados a partir de esta investigación.

A.1. Revistas indexadas por ISI-JCR

1. H. Sossa and E. Guevara, "Efficient training for dendrite morphological neural networks," *Neurocomputing*, vol. 131, pp. 132–142, 2014

2. R. Vega, G. Sanchez-Ante, L. E. Falcon-Morales, H. Sossa, and E. Guevara, "Retinal vessel extraction using lattice neural networks with dendritic processing," *Computers in Biology and Medicine*, vol. 58, pp. 20–30, 2015

A.2. Memorias en congresos internacionales

1. H. Sossa and E. Guevara, "Modified dendrite morphological neural network applied to 3D object recognition," *MCPR 2013, Lecture Notes in Computer Science, Springer*, vol. 7914, pp. 314–324, 2013

2. H. Sossa and E. Guevara, "Modified dendrite morphological neural network applied to 3D object recognition on RGB-D data," *Hybrid Artificial Intelligent Systems, Lecture Notes in Computer Science, Springer*, vol. 8073, pp. 304–313, 2013

3. R. Vega, E. Guevara, L. E. Falcon-Morales, G. Sanchez-Ante, and H. Sossa, "Blood vessel segmentation in retinal images using lattice neural networks," *Advances in Artificial Intelligence and Its Applications, 12th Mexican International Conference on*

Artificial Intelligence, MICAI 2013, Lecture Notes in Computer Science, Springer, vol. 8265, pp. 532–544, 2013

4. H. Sossa, G. Cortés, and E. Guevara, "New radial basis function neural network architecture for pattern classification: First results," *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, 19th Iberoamerican Congress, CIARP 2014, Lecture Notes in Computer Science, Springer*, vol. 8827, pp. 706–713, 2014

5. S. Valadez, H. Sossa, S.-M. Raúl, and E. Guevara, "Encoding polysomnographic signals into spike firing rate for sleep staging," *Pattern Recognition, 7th Mexican Conference, MCPR 2015, Lecture Notes in Computer Science, Springer*, vol. 9116, pp. 282–291, 2015

Apéndice B

Notación matemática

Atributos - $x_1, x_2, x_3, \ldots$

Clase - C^j donde j es el índice de la clase

Distancia - d

Hipercubo - H

Patrón o muestra - $\mathbf{x}$

Patrón ruidoso - $\tilde{\mathbf{x}}$

Número de atributos - n

Número de clases - p

Números reales - $\mathbb{R}$

Pesos - w

Margen - M

i-ésima neurona de entrada - N_i

j-ésima neurona de salida - M_j

Relación - R

Retícula - $\mathbb{L}$

Apéndice C

Álgebra reticular

George Birkhoff junto con su hijo Garrett, hicieron una importante contribución al desarrollo de la teoría reticular [105] que se ha aplicado en problemas que van desde el filtrado hasta la clasificación [106], [107].

Para establecer el marco matemático en el que se basan las RNMPD, en este apéndice se presentan una serie de definiciones importantes para este trabajo.

Definición C.0.1. Una relación binaria R en un conjunto $\mathbb{A}$ es un subconjunto de $\mathbb{A} \times \mathbb{A}$. Para denotar que a y b están relacionados se utiliza indistintamente $(a, b) \in R$ ó aRb. Una relación R en $\mathbb{A}$ es:

1. reflexiva si para cada $a \in \mathbb{A}, aRa$.

2. antisimétrica si $a, b \in \mathbb{A}, aRb$ y bRa entonces $a = b$

3. transitiva si $a, b, c \in \mathbb{A}, aRb$ y bRc entonces aRc.

4. asimétrica si $a, b \in \mathbb{A}, aRb$ entonces bRa no ocurre

5. lineal si para cada $a, b \in \mathbb{A}$ se tiene que aRb ó bRa.

Definición C.0.2. Un conjunto parcialmente ordenado en $\mathbb{A}$ es una relación binaria reflexiva, antisimétrica y transitiva en un conjunto $\mathbb{A}$ y se denota con el símbolo $\preceq$, y diremos que a precede o es igual a b $(a, b \in \mathbb{A})$ si $a \preceq b$. $(\mathbb{A}, \preceq)$ es un conjunto parcialmente ordenado. La definición no implica que dados $a, b \in \mathbb{A}$, entonces $a \preceq b$ o $b \preceq a$, es decir, en un conjunto parcialmente ordenado no todo par de elementos tiene que estar relacionado.

Definición C.0.3. Una retícula es un conjunto parcialmente ordenado $\mathbb{L}$ en el que para cada par de elementos a, b, existen el supremo y el ínfimo de $\{a, b\}$ los cuales se denotan como $a \vee b$ y $a \wedge b$ respectivamente. Simbólicamente la retícula se denota como $(\mathbb{L}, \vee, \wedge)$.

Si el símbolo $\preceq$ denota la relación de orden parcial de $\mathbb{L}$, entonces las operaciones de $\vee$ y $\wedge$ satisfacen las siguientes leyes para los elementos $a, b, c \in \mathbb{L}$ (siempre que las expresiones referidas existan):

1. Idempotencia: $a \vee a = a, a \wedge a = a$
2. Conmutativa: $a \vee b = b \vee a, a \wedge b = b \wedge a$
3. Asociativa: $a \vee (b \vee c) = (a \vee b) \vee c, a \wedge (b \wedge c) = (a \wedge b) \wedge c$
4. Absorción: $a \vee (a \wedge b) = a, a \wedge (a \vee b) = a$
5. Consistencia: $a \preceq b \Leftrightarrow a \vee b = a, a \preceq b \Leftrightarrow a \wedge b = a$
6. Elemento neutro: Si $\mathbb{L}$ tiene al menos un elemento O, entonces $O \vee a = a$ y $O \wedge a = O$ $\forall a \in \mathbb{L}$
7. Elemento identidad: Si $\mathbb{L}$ tiene al menos un elemento I, entonces $I \vee a = I$ y $I \wedge a = a$ $\forall a \in \mathbb{L}$

Las propiedades anteriores son válidas para cualquier conjunto parcialmente ordenado y, en consecuencia, para retículas también se cumplen:

1. Isotónica: $b \preceq c \Rightarrow a \wedge b \preceq a \wedge c$ y $a \vee b \preceq a \vee c$
2. Modularidad: $a \preceq c \Rightarrow a \vee (b \wedge c) \preceq (a \vee b) \wedge c$
3. Distributividad 1:$a \wedge (b \vee c) \preceq (a \wedge b) \vee (a \wedge c)$
4. Distributividad 2:$a \vee (b \wedge c) \preceq (a \vee b) \wedge (a \vee c)$

Definición C.0.4. El álgebra reticular cumple con las siguientes reglas, si $a, b, c \in \mathbb{R}_\infty$ entonces

1. $a + (b \wedge c) = (a + b) \wedge (a + c)$
2. $a \wedge \infty = \infty \wedge a = a$
3. $a + \infty = \infty + a = \infty$
4. $a + 0 = 0 + a = a$

En esta área es importante el concepto de espacio métrico, por lo que se presentarán algunas definiciones.

Definición C.0.5. Un espacio métrico es un conjunto no vacío cualquiera E provisto de una distancia d entre pares de puntos, que cumple las siguientes propiedades:

1. Positividad: $\forall a, b \in E$, $d(a, b)$ es un número real tal que $d(a, b) > 0$ si $a \neq b$ y $d(a, b) = 0$ si $a = b$
2. Simetría: $\forall a, b \in E$, $d(a, b) = d(b, a)$
3. Triangular: $\forall a, b, c \in E$, $d(a, b) \leq d(a, c) + d(c, b)$

Definición C.0.6. Un conjunto $Q \subset E$ es un conjunto cerrado en el espacio métrico E si y solo si los puntos de frontera o borde de Q, o bien no existen, o bien pertenecen todos a Q.

Definición C.0.7. Un conjunto $Q \subset E$ es un conjunto abierto en el espacio métrico E si y solo si los puntos de frontera o borde de Q, o bien no existen, o bien no pertenecen a Q.

Definición C.0.8. Un conjunto Q de reales se dice acotado si es vacío o si existe algún número real K llamado cota superior que es mayor o igual que todos los elementos de Q, y si existe algún número real k llamado cota inferior que es menor o igual que todos los elementos de Q.

Definición C.0.9. Un conjunto Q de reales se dice compacto si es cerrado y acotado.

Apéndice D

Demostraciones

Proposición 5.2.1. Si $X \subset \mathbb{R}^n$ es un conjunto finito de patrones de p clases, C^j, $j = 1, 2, ..., p$, con n atributos, entonces el algoritmo de entrenamiento para la NMPD converge en un número finito de pasos.

Demostración. Si $X \subset \mathbb{R}^n$ es un conjunto compacto, entonces para cada i existen:

$b_i = inf\{x_i : \mathbf{x} \in X\}$

$a_i = sup\{x_i : \mathbf{x} \in X\}$

Sea $H_0(X)$ el hipercubo más pequeño que contiene a X,

$H_0(X) = \{\mathbf{x} \in \mathbb{R}^n : b_i \leq x_i \leq a_i, i = 1, \ldots, n\}$

El algoritmo divide el hipercubo H_0 en 2^n hipercubos más pequeños. Sea:

$$\gamma_i = \frac{a_i - b_i}{2}$$

entonces para cada i se obtienen:

$H_{i1}(X) = \{\mathbf{x} \in \mathbb{R}^n : b_i \leq x_{i1} \leq \gamma_i, i = 1, \ldots, n\}$

$H_{i2}(X) = \{\mathbf{x} \in \mathbb{R}^n : \gamma_i \leq x_{i2} \leq a_i, i = 1, \ldots, n\}$

El caso trivial se da cuando los hipercubos generados contienen patrones de una sola clase C^k, $k \in \{1, 2, \ldots, p\}$, entonces se generan K dendritas cuyos pesos se determinan mediante:

$w_{i1}^1 = -b_i$, $w_{i1}^0 = -\gamma_i$ y $w_{i2}^1 = -\gamma_i$, $w_{i2}^0 = -a_i$

y por tanto el algoritmo converge a una solución.

El siguiente caso es cuando los hipercubos generados o algún hipercubo tiene patrones de más de una clase, entonces estos hipercubos vuelven a dividirse en 2^n hipercubos más pequeños y así sucesivamente hasta que se cumpla el criterio de paro, esto es, que todos los hipercubos solo tengan patrones de una sola clase.

Supónganse que el algoritmo no converge, esto implica que no se puede hacer la división:

$$\gamma_i = \frac{a_i - b_i}{2} \notin \mathbb{R}$$

esto es una contradicción ya que si $a_i, b_i \in \mathbb{R}$ entonces por propiedad de clausura de la suma de los números reales $(a_i + (-b_i)) \in \mathbb{R}$, entonces $\exists \gamma_i \in \mathbb{R}$ lo cual contradice la hipótesis de que el algoritmo no converge. □

Proposición 5.2.2. $\forall \mathbf{x}_E \epsilon X_{FE}$ donde X_{FE} es el conjunto fundamental de entrenamiento, $\mathbf{x}_E$ se clasifica correctamente una vez que el algoritmo de entrenamiento se aplica.

Demostración. El algoritmo de entrenamiento se detiene cuando en todos los hipercubos solo hay elementos de una sola clase. Sean $x_i \epsilon C^l$ y $x_j \epsilon C^m$, si se tiene que:

$$x_i, x_j \epsilon H_{C^k} \Rightarrow \gamma_{k+1} = \frac{\gamma_k}{2}$$

y así sucesivamente hasta que $x_i \epsilon H_{C^l}$ y $x_j \epsilon H_{C^m}$. Por lo tanto, el decir que un patrón está mal clasificado contradice la hipótesis del algoritmo. □

Proposición 5.2.3. Para cualquier par de hipercubos H generados por el algoritmo se cumple que $H_l \cap H_m = \emptyset$ donde $l \neq m$, $l, m = 1, 2, ..., p$.

Demostración. El algoritmo divide cada hipercubo de manera iterativa en 2^n hipercubos más pequeños. Entonces en la primera iteración se tendría que:

$$\gamma_1 = \frac{a_i - b_i}{2}$$

Para que existiera traslape entre los hipercubos se tendría que cumplir que

$$\gamma_k \neq \frac{x_i - x_j}{2}$$

lo que contradice el funcionamiento del algoritmo. □

Referencias

[1] H. A. Simon, "Rational choice and the structure of the environment," *Psychological Review*, vol. 63-2, pp. 129–138, 1956.

[2] H. A. Simon and Associates, "Report of the research briefing panel on decision making and problem solving," tech. rep., National Academic Press, Washington, DC., 1986.

[3] M. Gori and F. Scarselli, "Are multilayer perceptrons adequate for pattern recognition and verification?," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 1121–1132, 1998.

[4] W. S. McCulloch and W. H. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

[5] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, pp. 386–408, 1958.

[6] M. L. Minsky and S. A. Papert, "Perceptrons," *MIT Press, Cambridge*, 1969.

[7] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the USA*, vol. 79-8, pp. 2554–2558, 1982.

[8] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two state neurons," *Proceedings of the National Academy of Sciences of the USA*, vol. 81, pp. 3088–3092, 1984.

[9] J. J. Hopfield and D. W. Tank, "Computing with neural circuits: a model," *Science*, vol. 233-4764, pp. 625–633, 1986.

[10] J. L. Davidson and F. Hummer, "Morphology neural networks: An introduction with applications," *Circuits, Systems and Signal Processing*, vol. 12(2), pp. 177–210, 1993.

[11] G. X. Ritter and P. Sussner, "An introduction to morphological neural networks," *Proceedings of the 13th International Conference on Pattern Recognition*, vol. 4, pp. 709–717, 1996.

[12] M. Graña (Ed.), "Special issue on: Lattice computing and natural computing," *Neurocomputing*, vol. 72, No. 10-12, pp. 2065–2066, 2009.

[13] P. Sussner, "Morphological perceptron learning," *IEEE ISIC/CIRA/ISAS Joint Conference*, pp. 477–482, 1998.

[14] G. X. Ritter and T. W. Beaver, "Morphological perceptrons," *International Joint Conference on Neural Networks*, vol. 1, pp. 605–610, 1999.

[15] G. X. Ritter and G. Urcid, "Lattice algebra approach to single-neuron computation," *IEEE Transactions on Neural Networks*, vol. 14(2), pp. 282–295, 2003.

[16] P. Sussner and E. L. Esmi, "Morphological perceptrons with competitive learning: Lattice-theoretical framework and constructive learning algorithm," *Information Sciences*, vol. 181, pp. 1929–1950, 2011.

[17] G. X. Ritter, G. Urcid, and L. Iancu, "Reconstruction of patterns from noisy inputs using morphological associative memories," *Journal of Mathematical Imaging and Vision*, vol. 19-2, pp. 95–111, 2003.

[18] B. Raducanu, M. Graña, and F. X. Albizuri, "Morphological scale spaces and associative morphological memories: Results on robustness and practical applications," *Journal of Mathematical Imaging and Vision*, vol. 19-2, pp. 113–131, 2003.

[19] P. Sussner, "Observations on morphological associative memories and the kernel method," *Neurocomputing*, vol. 31, 1-4, pp. 167–183, 2000.

[20] P. Sussner and C. R. Medeiros, "An introduction to morphological associative memories in complete lattices and inf-semilattices," *IEEE International Conference on Fuzzy Systems*, pp. 1–8, 2012.

[21] P. Sussner and M. E. Valle, "Gray-scale morphological associative memories," *IEEE Transactions on Neural Networks*, vol. 17-3, pp. 559–570, 2006.

[22] R. De Aráujo, F. Madeiro, R. P. De Sousa, and L. F. C. Pessoa, "Modular morphological neural network training via adaptive genetic algorithm for designing translation invariant operators," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2006)*, vol. 2, pp. 873–876, 2006.

[23] Y. Won, P. D. Gader, and P. C. Coffield, "Morphological shared-weight networks with applications to automatic target recognition," *IEEE Transactions on Neural Networks*, vol. 8-5, pp. 1195–1203, 1997.

[24] Y. Nong, W. Hao, W. Changyong, L. Fanming, and W. Lide, "Morphological neural networks for automatic target detection by simulated annealing learning algorithm," *Science in China (Series F)*, vol. 46, pp. 262–278, 2003.

[25] S. Cheng-tian and W. Ke-yong, "Image target detection using morphological neural network," *International Conference on Computational Intelligence and Security*, pp. 234–236, 2009.

[26] B. Raducanu, M. Graña, and P. Sussner, "Morphological neural networks for vision based self-localization," *IEEE International Conference on Robotics and Automation*, pp. 2059–2064, 2001.

[27] G. Urcid, G. X. Ritter, and L. Iancu, "Single layer morphological perceptron solution to the n-bit parity problem," *Progress in Pattern Recognition, Image Analysis and Applications. 9th Iberoamerican Congress on Pattern Recognition, CIARP, Puebla, México,*, vol. 3287, pp. 171–178, 2004.

[28] I. Villaverde, M. Graña, and A. D'Anjou, "Morphological neural networks and vision based simultaneous localization and mapping," *Integrated Computer-Aided Engineering*, vol. 14-4, pp. 355–363, 2007.

[29] G. X. Ritter, L. Iancu, and G. Urcid, "Morphological perceptrons with dendritic structure," *12th IEEE International Conference in Fuzzy Systems (FUZZ 2003)*, vol. 2, pp. 1296–1301, 2003.

[30] R. Barrón, H. Sossa, and H. Cortés, "Morphological neural networks with dendrite computation: A geometrical approach," *LNCS 2905, Springer-Verlag*, pp. 588–595, 2003.

[31] G. X. Ritter and L. Iancu, "A lattice algebraic approach to neural computation," *Handbook of Geometric Computing. Applications in Pattern Recognition, Computer Vision, Neuralcomputing, and Robotics. Springer-Verlag.*, vol. ch. 4, pp. 97–127, 2005.

[32] G. Urcid, "Transformations of neural inputs in lattice dendrite computation," *Proc. SPIE, Mathematical Methods in Pattern and Image Analysis*, vol. 5916, pp. 201–212, 2005.

[33] G. X. Ritter and G. Urcid, "Learning in lattice neural networks that employ dendritic computing.," *Computational Intelligence Based on Lattice Theory*, vol. 67, pp. 25–44, 2007.

[34] D. Chyzhyk and M. Graña, "Optimal hyperbox shrinking in dendritic computing applied to Alzheimer's disease detection in MRI," *6th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2011). Advances in Intelligent and Soft Computing*, vol. 87, pp. 543–550, 2011.

[35] D. E. Caro-Contreras and V. A. Mendez, "Computing the concept lattice using dendritical neural networks," *Proc. 10th Int. Conf. on Concept Lattices and Their Applications*, pp. 141–152, 2013.

[36] G. X. Ritter and L. Iancu, "A morphological auto-associative memory based on dendritic computing," *IEEE Proceedings of International Joint Conference on Neural Networks*, vol. 2, pp. 915–920, 2004.

[37] G. X. Ritter, D. Chyzhyk, G. Urcid, and M. Graña, "A novel lattice associative memory based on dendritic computing," *Hybrid Artificial Intelligent Systems. 7th International Conference, HAIS 2012. LNCS, Salamanca, Spain, Springer-Verlag*, vol. 7209, pp. 491–502, 2012.

[38] G. X. Ritter, L. Iancu, and M. S. Schmalz, "A new auto-associative memory based on lattice algebra," *Proc. of 9th Iberoamerican Congress on Pattern Recognition, CIARP, LNCS, Springer-Verlag*, pp. 148–155, 2014.

[39] C. Roberson and D. D. Dankel II, "A morphological neural network approach to information retrieval," *Twentieth International Florida Artificial Intelligence Research Society Conference*, pp. 184–185, 2007.

[40] D. Chyzhyk, M. Graña, A. Savio, and J. Maiora, "Hybrid dendritic computing with kernel-LICA applied to Alzheimer's disease detection in MRI," *Neurocomputing*, vol. 75-1, pp. 72–77, 2012.

[41] D. Chyzhyk, "Bootstrapped dendritic classifiers for Alzheimer's disease classification on MRI features," *Advances in Knowledge-Based and Intelligent Information and Engineering Systems, IOS Press*, vol. 243, pp. 2251–2258, 2012.

[42] G. X. Ritter, L. Iancu, and G. Urcid, "Neurons, dendrites, and pattern classification," *Progress in Pattern Recognition, Speech and Image Analysis. Proceedings of 8th Iberoamerican Congress on Pattern Recognition, CIARP, LNCS, Springer*, vol. 2905, pp. 1–16, 2003.

[43] G. X. Ritter, G. Urcid, and J. Valdiviezo, "Two lattice metrics dendritic computing for pattern recognition," *IEEE International Conference on Fuzzy Systems*, pp. 45–52, 2014.

[44] G. Urcid, G. X. Ritter, and J. Valdiviezo, "Dendritic lattice associative memories for pattern classification," *Fourth World Congress on Nature and Biologically Inspired Computing*, pp. 181–187 181–187, 2012.

[45] G. X. Ritter and G. Urcid, "Perfect recall from noisy input patterns with a dendritic lattice associative memory," *Proccedings of International Joint Conference on Neural Networks*, pp. 503–510, 2011.

[46] G. Urcid, G. X. Ritter, and J. Valdiviezo, "Grayscale image recall from imperfect inputs with a two layer dendritic lattice associative memory," *Third World Congress on Nature and Biologically Inspired Computing*, pp. 261–266, 2011.

[47] R. D. Cazé, M. Humphries, and B. Gutkin, "Passive dendrites enable single neurons to compute linearly non-separable functions," *Computational Biology*, vol. 9-2, pp. 1–15, 2013.

[48] S. Haykin, *Neural Networks - A Comprehensive Foundation.* Prentice Hall, 1999.

[49] R. Rojas, *Neural Networks. A Systematic Introduction.* Springer-Verlag, 1996.

[50] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by backpropagating errors," *Parallel distributed processing: Explorations in the microstructure of cognition.*, vol. 1, pp. 318–362, 1986.

[51] D. S. Bromhead and D. Lowe, "Multivariate functional interpolation and adaptive networks," *Complex Systems*, vol. 2, pp. 321–355, 1988.

[52] J. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1(2), pp. 281–294, 1989.

[53] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43(1), pp. 59–69, 1982.

[54] V. Vapnik, "The nature of statistical learning theory," *Springer-Verlag, New York*, 1995.

[55] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[56] V. Vapnik, "Statistical learning theory," *John Wiley and Sons, Inc., New York*, 1998.

[57] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. Wiley, 1973.

[58] K. Fukunaga, *Introduction to Statistical Pattern Recognition, 2nd Edition*. Academic Press, 1990.

[59] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[60] B. Ripley, *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[61] E. Micheli-Tzanakou, *Supervised and unsupervised Pattern Recognition Feature Extraction and Computational*. CRC Press, 2000.

[62] M. Narasimha and D. V. Susheela, *Introduction to Pattern Recognition and Machine Learning*. IISc Press, 2015.

[63] R. Bacjsy and Kovacic, "Multiresolution elastic matching," *Computer Vision Graphics Image Processing*, vol. 46, pp. 1–21, 1989.

[64] L. Devroye, L. Győrfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*. Springer, 1997.

[65] K. S. Fu, *Syntactic Pattern Recognition and Applications*. Prentice-Hall, 1982.

[66] A. Kandel, *Fuzzy Techniques in Pattern Recognition*. Addison-Wesley, 1982.

[67] E. Alpaydin, *Introduction to Machine Learning, 2nd edition*. MIT Press, 2010.

[68] D. H. Wolpert, "The lack of a prior distinctions between learning algorithms and the existence of a priori distinctions between learning algorithms," *Neural Computation*, vol. 8, pp. 1341–1390, 1996.

[69] I. Segev, *The Handbook of Brain Theory and Neural Networks*, ch. Dendritic Processing, pp. 282–289. MIT Press, Boston, 1998.

[70] C. Koch and I. Segev, "Methods in neuronal modeling: From synapses to networks," *MIT Press, Cambridge*, 1989.

[71] W. R. Holmes and W. Rall, *Single Neuron Computation*, ch. Electronic Models of Neuron Dendrites and Single Neuron Computation, pp. 7–25. Academic Press, New York, 1992.

[72] B. W. Mel, "Synaptic integration in excitable dendritic trees," *Journal of Neurophysiology*, vol. 70-3, pp. 1086–1101, 1993.

[73] W. Rall and I. Segev, *Synaptic Function*, ch. Functional Possibilities for Synapses on Dendrites and Dendritic Spines Spines, pp. 605–636. John Wiley & Sons, New York, 1987.

[74] B. W. Mel, *Dendrites*, ch. Why have Dendrites? A Computational Perspective, pp. 271–289. Oxford University Press, 1999.

[75] J. C. Eccles, *The Understanding of the Brain*. McGraw-Hill, New York, 1977.

[76] M. A. Arbib, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1998.

[77] T. McKenna, J. Davis, and S. E. Zornetzer, *Single Neuron Computation*. Academic Press, 1992.

[78] A. Barmpoutis and G. X. Ritter, "Orthonormal basis lattice neural networks," *IEEE International Conference on Fuzzy Systems (FUZZ 2006)*, pp. 331–336, 2006.

[79] H. Sossa and E. Guevara, "Efficient training for dendrite morphological neural networks," *Neurocomputing*, vol. 131, pp. 132–142, 2014.

[80] K. Price, S. Rainer, and J. Lampinen, *Differential Evolution - A Practical Approach to Global Optimization*. Springer, 1998.

[81] R. Hecht-Nielsen, "Kolmogorov's mapping neural network existence theorem," *IEEE First Annual International Conference on Neural Networks*, vol. 3, pp. 11–13, 1987.

[82] B. Ripley, "Datasets for pattern recognition and neural networks," 1996.

[83] A. Asunción and N. D., "UCI Machine Learning Repository," 2007.

[84] R. A. Vázquez and H. Sossa, "A new associative model with dynamical synapses," *Neural Processing Letters*, vol. 28, pp. 189–207, 2008.

[85] B. Leibe and B. Schiele, "Analyzing appearance and contour based methods for object categorization," *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 409–415, 2003.

[86] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explorations*, vol. 11, 2009.

[87] W. Brock, J. Lakonishok, and B. LeBaron, "Simple technical trading rules and the stochastic properties of stock returns," *Journal of Finance*, vol. 47(5), pp. 1731–1764, 1992.

[88] H. Sossa and E. Guevara, "Modified dendrite morphological neural network applied to 3D object recognition on RGB-D data," *Hybrid Artificial Intelligent Systems, Lecture Notes in Computer Science, Springer*, vol. 8073, pp. 304–313, 2013.

[89] N. Burrus, "Kinect RGB Demo," 2011.

[90] M. K. Hu, "Visual pattern recognition by moment invariants," *IRE Transactions on Information Theory*, vol. 8, pp. 179–187, 1962.

[91] R. González and R. Woods, *Digital Image Processing*. Pearson, 2007.

[92] H. Sossa and E. Guevara, "Modified dendrite morphological neural network applied to 3D object recognition," *MCPR 2013, Lecture Notes in Computer Science, Springer*, vol. 7914, pp. 314–324, 2013.

[93] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9-1, pp. 62–66, 1979.

[94] Microsoft, "Kinect for Windows SDK." https://dev.windows.com/en-us/kinect.

[95] AgaveRobotics.com, "Dynamixel CSharp library." http://www.agaverobotics.com/services/dotnet/Dynamixel/default.aspx.

[96] Aldebaran, "CSharp NAOqi SDK." https://community.aldebaran.com/en/.

[97] A. Cook, K. Adams, J. Volden, N. Harbottle, and C. Harbottle, "Using lego robots to estimate cognitive ability in children who have severe physical disabilities," *Disability and Rehabilitation: Assistive Technology*, vol. 6-4, pp. 338–346, 2011.

[98] R. A. Ríos, K. Adams, J. Magill-Evans, and A. M. Cook, "Changes in playfulness with a robotic intervention in a child with cerebral palsy," *European Conference for the Advancement of Assistive Technology*, 2013.

[99] L. Mindstorm, "leJOS." http://www.lejos.org/nxj-downloads.php.

[100] R. Vega, E. Guevara, L. E. Falcon-Morales, G. Sanchez-Ante, and H. Sossa, "Blood vessel segmentation in retinal images using lattice neural networks," *Advances in Artificial Intelligence and Its Applications, 12th Mexican International Conference on Artificial Intelligence, MICAI 2013, Lecture Notes in Computer Science, Springer*, vol. 8265, pp. 532–544, 2013.

[101] R. Vega, G. Sanchez-Ante, L. E. Falcon-Morales, H. Sossa, and E. Guevara, "Retinal vessel extraction using lattice neural networks with dendritic processing," *Computers in Biology and Medicine*, vol. 58, pp. 20–30, 2015.

[102] L. Ojeda, R. Vega, L. E. Falcon-Morales, G. Sanchez-Ante, H. Sossa, and J. M. Antelis, "Classification of hand movements from non-invasive brain signals using lattice neural networks with dendritic processing," *Pattern Recognition, 7th Mexican Conference, MCPR 2015, Lecture Notes in Computer Science, Springer*, vol. 9116, pp. 23–32, 2015.

[103] H. Sossa, G. Cortés, and E. Guevara, "New radial basis function neural network architecture for pattern classification: First results," *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, 19th Iberoamerican Congress, CIARP 2014, Lecture Notes in Computer Science, Springer*, vol. 8827, pp. 706–713, 2014.

[104] S. Valadez, H. Sossa, S.-M. Raúl, and E. Guevara, "Encoding polysomnographic signals into spike firing rate for sleep staging," *Pattern Recognition, 7th Mexican Conference, MCPR 2015, Lecture Notes in Computer Science, Springer*, vol. 9116, pp. 282–291, 2015.

[105] G. Birkhoff, *Lattice Theory*. American Mathematical Society, 1940.

[106] V. G. Kaburlasos (Ed.), "Special issue on: Information engineering applications based on lattices," *Information Sciences*, vol. 181, No. 10, pp. 1771–1773, 2011.

[107] M. Graña, "A brief review of lattice computing," *Fuzzy Systems, IEEE World Congress on Computational Intelligence*, pp. 1777–1781, 2008.