



Entrenamiento de Red Neuronal Artificial Morfológica de Dendritas con Algoritmo de Optimización por Inteligencia de Enjambres

presentado por:

Omar Jordán Jordán

Universidad del Valle

Facultad de Ingeniería

Escuela de Ingeniería Eléctrica y Electrónica

Programa Académico de Ingeniería Electrónica

Santiago de Cali, Colombia – agosto de 2019



Entrenamiento de Red Neuronal Artificial Morfológica de Dendritas con Algoritmo de Optimización por Inteligencia de Enjambres

presentado por:

Omar Jordán Jordán

dirigido por:

Prof. Eduardo Caicedo, Ph.D.

Prof. Wilfredo Alfonso, Ph.D.

Universidad del Valle

Facultad de Ingeniería

Escuela de Ingeniería Eléctrica y Electrónica

Programa Académico de Ingeniería Electrónica

Santiago de Cali, Colombia – agosto de 2019

RESUMEN

El presente trabajo de grado explora las capacidades de la llamada Red Neuronal Morfológica de Dendritas, haciendo una apropiación del conocimiento e identificando diferentes algoritmos para su entrenamiento. Para ello, se ponen a prueba tres algoritmos de aprendizaje: Gradiente Descendente Estocástico, algoritmo de Evolución Diferencial y el algoritmo de Optimización por Enjambre de Partículas; este último, aplicado en redes morfológicas por primera vez en este tipo de trabajos. Adicionalmente, se desarrolla una interfaz de usuario que permite realizar la exploración de esta arquitectura tal que facilita el aprendizaje y la comprensión de la misma; la Interfaz de usuario permite poner a prueba la red neuronal, con conjuntos de datos asociados (*datasets*) a problemas de clasificación, y selecciona el algoritmo requerido junto con sus parámetros ajustables. Los *datasets* usados dentro de la aplicación, demuestran la versatilidad de esta arquitectura para la solución de problemas de clasificación.

Como resultados, se obtuvo un aplicativo capaz de crear y entrenar la red neuronal planteada, a partir de conjuntos de datos de diversos problemas de clasificación; se halló que el algoritmo de gradiente descendente es el más eficiente para la mayoría de problemas, aun así el de partículas es también una opción adecuada, superior al evolutivo, con mejor generalización que el de gradiente descendente y con capacidad de hallar en algunos casos soluciones superiores a las de este, a causa de su habilidad para escapar a mínimos locales.

Palabras Clave: *red neuronal morfológica de dendritas, evolución diferencial, gradiente descendente estocástico, optimización por enjambre de partículas, red neuronal artificial, clasificación de patrones.*

ABSTRACT

This work is about the Dendral Morphological Neural Network, a different type of artificial neural network capable of solve pattern classification problems whit no ocult layers, was develop in the 90's and over last years, several research progresively approach to optimum management of it, this work is in first an acquisition of knowledge, that ends with the execution of the net with three learning algorithms: Stochastic Gradient Descent, Differential Evolution and Particle Swarm Optimization; the latter applied with morphological networks for first time. All this was implemented in Python, resulting in a Graphic User Interface toolbox useful for future research, the software can import any pattern classification dataset and try to solve it, using two initialization algorithms, and applying some of the three learning algorithms mentioned.

As conclusions, the application was created and with it, diverse datasets was imported to create, train and evaluate the network; was found that, the stochastic gradient descent is the best train method for the most problems; in the other hand, the particle swarm optimization show adequate performance training this kind of net, being superior to the evolutive one, showing better generalization that gradient one, and in some cases, showing better results than it, especially when scape of local minima is an priority.

Keywords: *dendral morphological neural network, differential evolution, stochastic gradient descent, particle swarm optimization, artificial neural network, pattern classification.*

AGRADECIMIENTOS

Sin lugar a dudas, es una posición privilegiada el estar escribiendo esta página, agradecer así sea por cumplir no está de más, y debo decir ciertamente que, de no ser por el apoyo de mi familia y su gran paciencia, no sé qué habría sido de esta historia. Me alegra haber dado con directores de tesis tan entusiastas y motivadores, de igual forma, sentir esa mano siniestra que a través de los años ha girado los dados a mi favor.

Pienso que hay dos tipos de agradecimiento, el consciente de su antítesis y el engatusado con lo placentero, este último es el que siente una persona despreocupada y serena, que come un postre en un tranquilo chalet al atardecer, el primero es el de alguien distraído que a último segundo dio un brinco atrás para permitirle al tren pasar libremente por su curso.

Vivimos en una de las épocas más pacíficas, creativas e innovadoras de la humanidad, sin embargo, el precio de la supervivencia cambia de moneda, es la naturaleza en su cruda selección la que aún en medio de las ciudades de concreto, dictamina que unos pocos caminen orgullosos, sobre los crujientes huesos de esos individuos que no pasaron a la siguiente generación.

Sin embargo, siguen vivos, lo sé porque en medio del vacío, frío, silencio y oscuridad aún se pueden sentir, susurros en la noche sin luna perturban levemente la tranquilidad de los habitantes de la superficie, de los hijos del sol intermitente; por eso agradezco con desconfianza, porque algo anda mal en este mundo y no es la ciencia, el arte, la política, la filosofía, no es siquiera la humanidad, son las raíces macabramente programadas de este teatro.

Me agrada el elitismo, no confundas mi discurso con socialismo de insectos, también disfruto pisar los cadáveres, pero sé que un elitismo basado en el mérito y una población que no muera con las manos vacías y los sueños rotos, es lo que deseo para todas las especies del multiverso, solo cuando eso suceda, cuando las reglas se hayan revertido, agradeceré de verdad, no por mí, sino por todos.

LISTA DE CONTENIDOS

1. ¿QUÉ ESTÁ PASANDO AQUÍ?	1
1.1. INTRODUCCIÓN	1
1.2. IDENTIFICACIÓN DEL PROBLEMA.....	2
1.3. MOTIVACIÓN	3
1.4. OBJETIVOS	3
1.4.1. General.....	3
1.4.2. Específicos	4
1.5. SOLUCIÓN PROPUESTA	4
1.6. ESTRUCTURA DEL DOCUMENTO	5
2. MARCO TEÓRICO	6
2.1. INTRODUCCIÓN	6
2.2. TRABAJOS RELACIONADOS.....	6
2.3. Redes Neuronales Morfológicas de Dendritas (DMNN)	9
2.3.1. Origen.....	9
2.3.2. Modelo de Dendritas Biológicas	11
2.4. INICIALIZACIÓN	17
2.4.1. K-medias	17
2.4.2. División de Hiper-cajas	18
2.5. ADECUACIÓN DE SALIDA.....	21
2.5.1. Capa Softmax.....	21
2.5.2. Función de Error	21
2.6. ENTRENAMIENTO	22
2.6.1. Gradiente Descendente.....	22
2.6.2. Evolución Diferencial	25

2.6.3.	Enjambre de Partículas	28
2.7.	OPTIMIZACIÓN DE DENDRITAS.....	31
2.8.	CRITERIOS DE EVALUACIÓN.....	32
2.9.	RESUMEN	34
3.	IMPLEMENTACIÓN.....	35
3.1.	INTRODUCCIÓN.....	35
3.2.	REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES	36
3.3.	DISEÑO DE LA <i>GUI</i> Y CONCEPTUALIZACIÓN	37
3.4.	MODELO DE INFORMACIÓN	38
3.4.1.	Formato Patrones	39
3.4.2.	Formato Red Neuronal	39
3.4.3.	Formato Matriz de Confusión	40
3.4.4.	Formato Resultados	41
3.4.5.	Formato Pesos Sinápticos.....	41
3.5.	DIAGRAMA DE CLASES	42
3.6.	FUNCIONALIDAD DE LOS MÓDULOS.....	45
3.6.1.	Menú.....	45
3.6.2.	Problema	46
3.6.3.	Inicialización	47
3.6.4.	Post-Entreno.....	49
3.6.5.	Entreno	51
3.6.6.	Análisis	53
3.7.	OBSERVACIONES DEL APLICATIVO SOFTWARE	55
4.	ANÁLISIS DE RESULTADOS	56
4.1.	INTRODUCCIÓN	56
4.2.	RESULTADOS DATASETS	56

4.3.	PROBLEMA “PAYASOLANDER”	60
4.4.	PRUEBA DE USABILIDAD	62
4.5.	PRUEBA DE INTEGRACIÓN.....	64
4.6.	ANÁLISIS DE RESULTADOS	65
4.6.1.	Análisis de Resultados Generales	65
4.6.2.	Análisis de Algoritmos de Inicialización	66
4.6.3.	Análisis de Algoritmos de Entrenamiento	67
4.6.4.	Análisis de problema “PayasoLander”	68
4.6.5.	Análisis Test de Usabilidad.....	69
4.7.	CONCLUSIONES.....	70
5.	¿A QUÉ SE LLEGÓ?.....	71
5.1.	CONCLUSIONES GENERALES.....	71
5.2.	TRABAJOS FUTUROS	72
6.	REFERENCIAS	74
7.	ANEXOS.....	1

LISTA DE FIGURAS

Figura 1 Menú principal del aplicativo creado	4
Figura 2 Doble hélice: cajas y superficie de decisión	9
Figura 3 Estructura de la neurona biológica	11
Figura 4 Superficies de decisión para perceptrón clásico y morfológico	12
<i>Figura 5 Representación de la ecuación para el modelo de una sola caja</i>	<i>13</i>
Figura 6 Ejemplo con tres dendritas	14
Figura 7 Superficie de decisión para dos neuronas con dos dendritas cada una..	15
Figura 8 Diagrama de la DMNN	16
Figura 9 Diagrama de bloques del algoritmo K-medias.....	18
Figura 10 Inicialización K-medias para una neurona (una clase)	18
Figura 11 Diagrama de bloques del algoritmo de División de Hiper-Cajas	19
Figura 12 Diagrama de bloques del algoritmo que divide una caja	20
Figura 13 Inicialización por división de hiper-cajas	20
Figura 14 Superficie de error y camino de descenso, tomado de Mathworks	22
Figura 15 Superficie de error compleja, tomado de Mathworks	24
Figura 16 Diagrama de bloques algoritmo evolutivo genético	26
Figura 17 Diagrama de bloques del algoritmo de evolución diferencial	27
Figura 18 Actualización velocidad de la partícula.....	30
Figura 19 Diagrama de bloques del algoritmo de enjambre de partículas	30
Figura 20 Modelo metodológico del sistema	35
Figura 21 Diseño de partida, GUI's conceptuales	38
Figura 22 Formato patrones TXT	39
Figura 23 Formato patrones XML.....	39
Figura 24 Formato red TXT	40
Figura 25 Formato red XML	40
Figura 26 Formato matriz de confusión XML.....	40
Figura 27 Formato resultados TXT.....	41
Figura 28 Formato resultados XML	41
Figura 29 Vector de pesos w	42
Figura 30 Diagrama de clases para las GUI's	43

Figura 31 Diagrama de la clase Motor	43
Figura 32 Diagrama de clases de la red DMNN	44
Figura 33 Diagrama de Clases.....	45
Figura 34 GUI Menú.....	45
Figura 35 GUI llamada Problema	46
Figura 36 Diagramas de secuencia para: Normalizar Patrones / Calcular porcentajes.....	47
Figura 37 Diagramas de secuencia para: Importar Patrones / Mezclar y Exportar Patrones.....	47
Figura 38 GUI Inicialización.....	48
Figura 39 Diagramas de secuencia para: Importar Red / Exportar Red	48
Figura 40 (izquierda) Diagrama de secuencia para inicializar por Kmedias	49
Figura 41 (derecha) Diagrama K-medias auto tuneado	49
Figura 42 GUI Post-Entreno	49
Figura 43 Diagrama de secuencia para: Ejecutar Red.....	50
Figura 44 Diagrama de secuencia para: Unir Dendritas.....	50
Figura 45 Diagrama de secuencia para: Quitar Dendritas	50
Figura 46 GUI's Entreno.....	51
Figura 47 Diagrama de secuencia para Entrenamiento (botón Play)	52
Figura 48 Campana de Gauss	53
Figura 49 GUI Análisis	53
Figura 50 Ejemplo ROC	54
Figura 51 Matriz de confusión para el problema Mammographic Mass	58
Figura 52 Matrices de confusión para el problema Iris.....	59
Figura 53 Matrices de confusión para el problema Glass	60
Figura 54 Físicas PayasoLanderANN	61
Figura 55 Matriz de confusión para PayasoLander	62
Figura 56 Test de usabilidad	63

LISTA DE TABLAS

Tabla 1: comparación de clasificación con MLP, SVM, RBN, SLMP-P (Humberto Sossa, 2014)	7
Tabla 2: Comparación de clasificación con MLP, SVM, RBN, DE-DMNN (Arce, Zamora, Sossa, & Barron, 2018).....	7
Tabla 3: Comparación de clasificación con SLMP-P, DE-DMNN (Arce, Zamora, Sossa, & Barron, 2018)	8
Tabla 4 Estructura general de la matriz de confusión	32
Tabla 5 Requerimientos funcionales	36
Tabla 6 Requerimientos No funcionales.....	37
Tabla 7 Desbloqueo de los módulos	46
Tabla 8 Datasets para testear la red DMNN.....	56
Tabla 9 Óptima solución hallada	57
Tabla 10 Comparación algoritmos de inicialización.....	57
Tabla 11 Comparación algoritmos de entreno.....	58
Tabla 12 Resultados 30 vuelos PayasoLander	61
Tabla 13 Resumen prueba de integración generalizada	64

GLOSARIO

DMNN	<i>Dendral Morphological Neural Network</i> , es el modelo de red neuronal artificial central de este libro.
SGD	<i>Stochastic Gradient Descent</i> , el gradiente descendente usado en un algoritmo iterativo que añade cierta heurística a su comportamiento.
DE	<i>Differential Evolution</i> , algoritmo genético que utiliza la diferencia de tres individuos al azar para formar un descendiente.
PSO	<i>Particle Swarm Optimization</i> , algoritmo que lanza en la superficie de error varias partículas, que son el modelo en movimiento.
RUP	<i>Rational Unified Process</i> , proceso de desarrollo de ingeniería, enfocado en la administración de un proyecto.
MLP	<i>Multi Layer Perceptrón</i> , conexión de varios perceptrones, donde hay capas ocultas.
SVM	<i>Support Vector Machine</i> , clasificador por regresión, separa los parámetros de entrada en dos clases, se enfoca en los límites entre patrones.
RBN	<i>Radial Basis Network</i> , red neuronal que calcula la salida en función a un punto central, es un aproximador universal.
GPU	<i>Graphics Processing Unit</i> , hardware que procesa gráficos computacionales y operaciones de coma flotante, para aligerar la carga de la unidad central.
FPGA	<i>Field Programmable Gate Array</i> , hardware que se programa desde un computador para funcionar como diversos sistemas digitales.
GUI	<i>Graphic User Interface</i> , gráficos de un software que se muestran en pantalla para que el usuario interactúe amigablemente.
ANN	<i>Artificial Neural Network</i> , modelos computacionales o de hardware basados en el funcionamiento de la neurona y sistema nervioso animal.
Perceptrón	Modelo más simple de la neurona biológica, es un discriminador lineal, las entradas se multiplican por un peso sináptico y luego se suman.
Gradiente	Vector que define la dirección y magnitud con que cambia una variable en el espacio de su dominio, matemáticamente es la función derivación.
Evolutivo	Algoritmo de optimización inspirado en la biología, hace iteraciones llamadas generaciones, donde cruza o muta individuos de una población.
Estocástico	Proceso no determinista, dado el estado actual no podemos predecir el siguiente ya que posee componentes aleatorias.
Heurístico	Solución intuitiva, aproximada o que evade la complejidad matemática.
Dataset	Grupo de datos (patrones) que definen un problema de clasificación.
Agente	Componente dinámica de un sistema complejo, usualmente multiagente.

CAPÍTULO 1

1. ¿QUÉ ESTÁ PASANDO AQUÍ?

1.1.INTRODUCCIÓN

Las Redes Neuronales Artificiales (*ANN*), han sido implementadas en diversas aplicaciones, logrando diversificarse en distintas formas. Es por ello, que surgen nuevas propuestas matemáticas y algoritmos más ligeros computacionalmente, dedicados a la sintonización de sus parámetros, diferentes a los métodos comúnmente conocidos como el gradiente descendente.

En el marco de las *ANN*, otro tipo de arquitecturas se presentan desde los avances en neurobiología, donde se ha planteado la participación de las dendritas como parte importante del procesamiento de la información, incluso se ha determinado que quizá es la parte más importante (Ritter, Lancu, & Urcid, 2003). Este tipo de redes se encuentran dentro de las arquitecturas de redes neuronales artificiales morfológicas, cuyos inicios se presentaron en el trabajo de (Davidson & Ritter, Theory of Morphological Neural Networks, 1990).

Sin embargo, ¿Cuál es la ventaja que tienen estas redes? Aunque es más común utilizar perceptrones multi capa (*MLP*), redes de base radial (*RBN*), máquinas de soporte vectorial (*SVM*), redes de aprendizaje profundo (*Deep Learning*), entre otras, las redes morfológicas de dendritas han demostrado un potencial que está en plena vanguardia, logrando:

- Ciclos de entrenamiento finitos debido al crecimiento iterativo del número de dendritas donde se puede alcanzar un error de entrenamiento cero sin estar supeditado al sobre-entrenamiento (Humberto Sossa, 2014).
- Pueden solucionar la compuerta XOR con tres dendritas de una neurona sin capas ocultas (Ritter, Lancu, & Urcid, 2003).
- Implementación más veloz en hardware y software debido a las operaciones morfológicas que son básicamente lógicas, siendo la estructura de la neurona fácilmente operable en paralelo, lo que se puede hacer por ejemplo en una *FPGA* o *GPU* (Ritter & Sussner, 1996).

- Las superficies de decisión siempre son cerradas y con alta complejidad, lo que implica que pueden tener innumerables zonas aisladas (Zamora & Sossa, 2017).

A pesar de sus múltiples beneficios, este tipo de arquitecturas requieren algoritmos de entrenamiento más sofisticados que el usual gradiente descendente. Para ello, este trabajo de grado, reconociendo las ventajas de los algoritmos de inteligencia computacional como mecanismos para la sintonización de parámetros en las arquitecturas de redes neuronales, lleva a cabo un proceso de estimación de los pesos sinápticos de las dendritas, donde se pone a prueba el algoritmo basado en enjambre de partículas (*PSO*) y a otros algoritmos como el método de gradiente descendente estocástico (*SGD*) (Zamora & Sossa, 2017) y el algoritmo de evolución diferencial (*DE*) (Arce, Zamora, Sossa, & Barron, 2018).

1.2. IDENTIFICACIÓN DEL PROBLEMA

A la hora de buscar los parámetros que mejoren las características de clasificación de una red neuronal morfológica tipo *DMNN* (*Dendral Morphological Neural Network*) se han utilizado algoritmos de optimización como *SGD* (Stochastic Gradient Descent) o algoritmos evolutivos como el *DE* (Differential Evolution), los cuales han presentado inconvenientes en las búsquedas estocásticas debido al fenómeno denominado estancamiento prematuro; es decir, solo consiguen llegar a soluciones óptimas locales.

El grupo de investigación en percepción y sistemas inteligentes (PSI) ha trabajado desde sus inicios en el campo de la inteligencia computacional, adquiriendo conocimiento de vanguardia en distintas áreas como lo son *Deep Networks* y la Inteligencia de Enjambres, donde se busca llegar cada vez a soluciones óptimas, que requieran menores requisitos de hardware y por consiguiente menores costos; para esto las redes neuronales morfológicas ofrecen operaciones más simples de computar; sin embargo, los algoritmos de aprendizaje están aún en una importante etapa de investigación, ya que los valores de los parámetros de sintonización del modelo resultan de operaciones matemáticas, donde la sumatoria de un *MLP* es reemplazada por operadores máximo o mínimo, siendo evidente que la ecuación de

esta neurona artificial no es derivable ni lineal, incluso desde antes de la función de activación. A partir de esta necesidad se plantea entonces la siguiente pregunta:

¿Qué tan eficiente es la técnica de inteligencia de enjambres PSO en la sintonización de parámetros de una red neuronal DMNN en comparación con los algoritmos recientemente utilizados (SGD, DE)?

1.3. MOTIVACIÓN

El conocer las características y las capacidades de las redes neuronales morfológicas de tipo *DMNN* en la solución de problemas de clasificación, las cuales están en la capacidad de crear límites de decisión complejos para sistemas no-lineales con solo una capa de neuronas, motiva a comprender mejor el funcionamiento de este tipo de arquitecturas ya que dan a entender que las *DMNN* superan las capacidades de las redes neuronales *feedforward* multicapa clásicas.

Por otro lado, el uso de inteligencia de enjambres ofrece ventajas al enfocarse en el trabajo de búsqueda colectiva realizado por unidades simples, que adicionalmente supone una solución significativa al problema de estancamiento prematuro en mínimos locales, dado que es capaz de encontrar soluciones que un solo individuo podría no resolver (Kiranyaz, Ince, & Gabbouj, 2014)

Finalmente, y teniendo en cuenta que están en pleno desarrollo estas redes morfológicas, es necesario para grupos investigativos como PSI, hacer una apropiación del conocimiento para su reproducción y experimentación a nivel local, así como llevarlo a los estudiantes y profesionales interesados en esta área, teniendo en cuenta que la institución tiene entre sus fines el desarrollo tecnológico e intelectual de la comunidad a nivel de ciudad y país.

1.4. OBJETIVOS

1.4.1. General

Desarrollar un algoritmo de sintonización de los pesos sinápticos de una *DMNN* mediante un mecanismo de Optimización por Enjambre de Partículas (*PSO*), el cual debe ser comparado con los algoritmos: Gradiente Descendente Estocástico (*SGD*) y Evolución Diferencial (*DE*).

1.4.2. Específicos

1. Realizar una búsqueda bibliográfica de los algoritmos de aprendizaje utilizados para la sintonización de una *DMNN*.
2. Desarrollar una herramienta software para poner a prueba la sintonización de parámetros de la red *DMNN* mediante los algoritmos *SGD*, *DE* y *PSO*.
3. Evaluar los algoritmos de aprendizaje para la red neuronal morfológica a través de la comparación de métricas de desempeño estandarizadas.

1.5. SOLUCIÓN PROPUESTA

Para cumplir con los objetivos propuestos en el trabajo de grado, se diseñó una interfaz que permite comprender el funcionamiento de una red morfológica *DMNN*, aplicando tres algoritmos de aprendizaje: Gradiente Descendente Estocástico (*SGD*), Algoritmo Genético de Evolución Diferencial (*DE*) y el Algoritmo de Optimización por Enjambre de Partículas (*PSO*). La Figura 1 muestra el menú principal de la aplicación, donde se cumple la secuencia de acciones: ingresar patrones del problema, inicializar la red neuronal con alguno de los dos algoritmos disponibles, entrenarla usando alguno o varios de los tres algoritmos ya mencionados, ejecutar pruebas con la red y/u optimizar el número de dendritas, finalmente hacer un análisis de resultados con matrices de confusión.

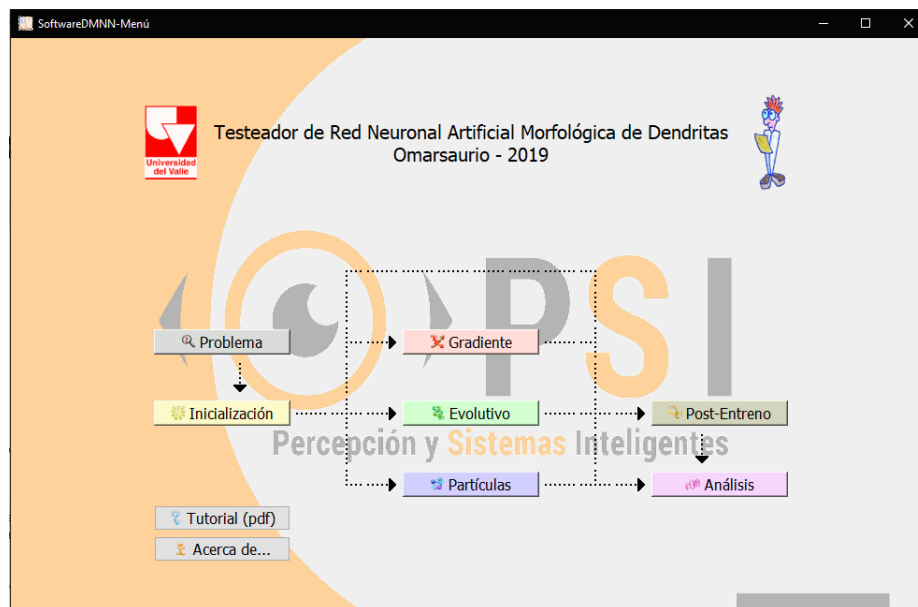


Figura 1 Menú principal del aplicativo creado

Además, el software cuenta con una amplia guía en formato *PDF*, que explica cómo usarlo y el funcionamiento de la *DMNN* con sus algoritmos.

1.6. ESTRUCTURA DEL DOCUMENTO

El libro se divide en cinco capítulos, siendo éste el primero, que corresponde al marco del problema con los objetivos de este trabajo ya presentado. El segundo capítulo presenta una explicación teórica de toda la temática involucrada, se ahonda en el concepto clave de red neuronal artificial morfológica de dendritas y luego en los algoritmos necesarios para manejar dicha red, su inicialización, entrenamiento, optimización y evaluación. En el tercer capítulo se explica la metodología usada para el desarrollo del trabajo, incluyendo cada uno de los bloques de procesamiento y haciendo uso del protocolo *RUP* (Rational Unified Process) para explicar el software desarrollado, presentando: el modelo conceptual, los requerimientos, los casos de uso real, el modelo de información y los pseudocódigos en diagramas de flujo; también se expone el lenguaje o motor de programación que se decidió utilizar. Pasando al cuarto capítulo tenemos las pruebas de integración del *RUP*, una adquisición de *datasets* obtenidos de la nube, para problemas de clasificación que coincidan con los requisitos de nuestra red neuronal; luego se presentan una serie de pruebas hechas con estos *datasets*, viendo como el software cumple sus funciones y genera las gráficas que serán analizadas luego; estas graficas mostrarán una comparativa de desempeño entre los diferentes algoritmos a la hora de entrenar la red neuronal. Finalmente, en el quinto capítulo se lleva a cabo el análisis de la información obtenida en las pruebas, se verifica la fiabilidad de los criterios de evaluación y se contrasta con los estudios hechos en el estado del arte, todo esto desenlaza en las conclusiones que plantean luego preguntas, hipótesis y autocríticas para tener en cuenta al hacer trabajos futuros.

CAPÍTULO 2

2. MARCO TEÓRICO

2.1. INTRODUCCIÓN

Este capítulo muestra el marco de referencia, seguido por la exposición de manera detallada de los principios de funcionamiento de una red neuronal artificial morfológica de dendritas (*DMNN*). Para esto último, se describe la estructura interna de una de estas neuronas, comparándola con el clásico perceptrón y se explica cómo se interconecta esta neurona con otras generando la compleja superficie de decisión que promete este tipo de *ANN*, posteriormente, se expone el proceso de inicialización de los pesos sinápticos de la red antes del entrenamiento, debido a que estos no corresponden a distribuciones pseudo-aleatorias como sucede en la mayoría de redes, como en la *MLP*. Esto genera ventajas que son aprovechadas por los algoritmos de entrenamiento que actúan como refinadores (Zamora & Sossa, 2017).

Con la arquitectura *DMNN* bien definida se procede a describir tres algoritmos: *SGD*, *DE*, *PSO*; los cuales, requieren de funciones de costo o *fitness* que deben ser seleccionadas adecuadamente. Para finalizar, el capítulo presenta los criterios de evaluación para medir la efectividad del entrenamiento de la *ANN* de forma que sean comparables los diferentes algoritmos implementados.

2.2. TRABAJOS RELACIONADOS

Los trabajos más recientes, en el marco de las redes morfológicas de dendritas, han utilizado algoritmos meramente heurísticos, evolutivos y de gradiente descendente. Para el primer caso tenemos el algoritmo de división de hiper-cajas (*SLMP-P*) publicado en (Humberto Sossa, 2014) donde se exponen ventajas como:

- Las iteraciones son finitas, siempre converge a error cero para la situación de los patrones de entrenamiento.
- No hay traslape de hiper-cajas (se explicará esto más adelante).
- Los resultados de entrenamiento son totalmente repetibles, si uno de los parámetros se mantiene fijo.
- No depende de la complejidad de la forma geométrica de los patrones.

- No tiene áreas de indecisión en la superficie de decisión.
- Fácil escalabilidad al aumentar la dimensionalidad del problema.
- Pese a la ralentización que supone resolver problemas de mayor complejidad, el algoritmo puede ser fácilmente paralelizado en dispositivos como *GPUs* o *FPGAs*.

En el entrenamiento con datasets reales, de dicha publicación, la *DMNN* demostró alcanzar un performance superior al *MLP*, *SVM*, *RBN*, como se ve en Tabla 1.

Tabla 1: comparación de clasificación con *MLP*, *SVM*, *RBN*, *SLMP-P* (Humberto Sossa, 2014)

Problem	MLP	SVM		RBN		SLMP-P		
	%error	Degree	%error	#clusters	%error	#dendrites	M	%error
Iris	6.77	1	4.00	2	5.33	20	0.30	2.67
Glass	31.46	5	31.68	6	31.68	64	0.19	30.69
Liver disorders	39.50	3	38.95	4	35.47	132	0.88	35.47
Page blocks	4.93	6	5.27	4	5.12	304	0.29	4.86
Image segmentation	27.33	1	26.71	2	21.71	92	0.54	24.14
Letter recognition	40.33	2	43.33	2	41.33	1439	0.72	38.75
Digits recognition	15.38	1	13.23	2	14.09	2259	0.25	9.55

En 2017 y 2018 se publicaron dos trabajos que marcarían un fuerte punto de partida.

En uno, la *DMNN* fue entrenada mediante el algoritmo de evolución diferencial (Arce, Zamora, Sossa, & Barron, 2018), y se comparó el desempeño con otras redes: *MLP*, *SVM*, *RBN* y con el *SLMP-P*, ver Tabla 2 y

Tabla 3.

Tabla 2: Comparación de clasificación con *MLP*, *SVM*, *RBN*, *DE-DMNN* (Arce, Zamora, Sossa, & Barron, 2018)

Problem	MLP		SVM		RBN		DE-DMNN	
	E train	E test	E train	E test	E train	E test	E train	E test
A	20.7	24.0	20.8	22.0	21.8	23.5	21.7	20.5
B	15.5	16.7	15.7	16.7	15.5	17.0	16.6	15.2
Spiral 2	6.6	7.4	45.1	44.4	47.4	45.2	7.3	6.4
Iris	1.7	0.0	4.2	0.0	4.2	0.0	3.3	0.0
Mammographic mass	15.7	11.2	18.4	11.2	17.9	16.0	15.8	10.4
Liver Disorders	40.3	40.6	40.0	40.2	29.0	37.8	37.6	31.1
Glass identification	14.1	20.4	12.3	18.2	0.0	20.4	4.7	13.6
Wine quality	34.3	39.0	40.6	43.0	41.5	44.3	42.1	40.0
Mice protein expression	0.0	0.6	0.1	0.5	11.4	13.9	6.6	4.5

Tabla 3: Comparación de clasificación con *SLMP-P*, *DE-DMNN* (Arce, Zamora, Sossa, & Barron, 2018)

Problem	SLMP-P			DE-DMNN		
	#dendrites	E train	E test	#dendrites	E train	E test
Iris	28	0.0	3.3	3	3.3	0.0
Mammographic mass	26	0.0	19.2	8	15.8	10.4
Liver disorders	183	0.0	35.5	12	37.6	31.1
Glass identification	82	0.0	31.8	12	4.7	13.6
Wine quality	841	0.0	42.1	60	42.1	40.0
Mice protein expression	809	0.0	5.0	32	6.6	4.5

El trabajo mencionado, mediante las técnicas de optimización por algoritmos evolutivos, logra generar un menor error de testeo a la vez que utilizan un número mucho menor de dendritas, pese a que no llega a cero en el error de entrenamiento como lo hace la red *SLMP-P*; sin embargo, la *DMNN* es más veloz en su ejecución y por ende óptima.

En el otro trabajo, se utilizó la técnica del gradiente descendente estocástico (SGD) como algoritmo de aprendizaje (Zamora & Sossa, 2017), donde los autores enfrentaron al problema de la no derivabilidad de la *DMNN* a causa de sus funciones min y max, haciendo uso de una capa *softmax* a la salida de la red. El algoritmo de gradiente descendente conserva una mayor capacidad de generalización en comparación con los algoritmos basados en heurísticas; sin embargo, los algoritmos heurísticos siguen siendo utilizados para la inicialización de la red donde el K-medias resulta ser el más complejo y óptimo seguido por el de división de hiper-cajas planteado por (Humberto Sossa, 2014) con el algoritmo *SLMP-P*.

Los resultados al comparar la *DMNN* con el *MLP*, *SVM* y *RBN* muestran que todas las técnicas de clasificación funcionan adecuadamente para la clasificación, siendo la *DMNN* apenas superada por la *MLP*. Por otro lado, la ventaja en el uso de redes morfológicas, radica en su inicialización ya que presenta un error relativamente bajo, por lo que el entrenamiento se convierte en una etapa de refinación, mientras el perceptrón multicapa, con su inicialización aleatoria, requiere más ciclos para darle una forma geométrica a su solución.

Finalmente, para presentar la potencialidad de las redes DMNN se muestra en la Figura 2 el clásico problema de la doble hélice, dónde esta arquitectura demuestra su versatilidad.

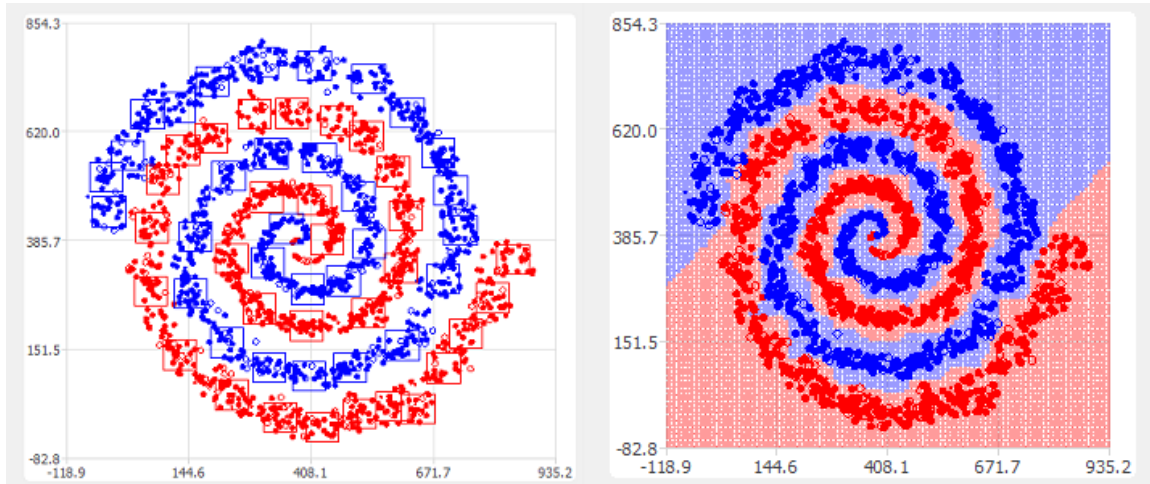


Figura 2 Doble hélice: cajas y superficie de decisión

2.3. Redes Neuronales Morfológicas de Dendritas (DMNN)

2.3.1. Origen

Hoy en día existen muchos modelos de *ANN*, algunos más fieles a los recientes descubrimientos de la neurociencia, otros más enfocados en hallar topologías de interconexión y algoritmos de aprendizaje óptimos; por ejemplo, los algoritmos de aprendizaje profundo ajustan los parámetros de redes basadas en el *MLP* con gran cantidad de capas ocultas. Otras de las estrategias más ampliamente usadas hoy en día para la resolución de problemas de clasificación son las *MLP*, las redes de base radial (*RBN*), las máquinas de soporte vectorial (*SVM*), estas últimas están basadas en la estadística y matemática intrínseca a los problemas de clasificación.

Por su parte, las redes neuronales morfológicas de dendritas (*DMNN*) comienzan su historia aproximadamente con el trabajo (Davidson & Ritter, 1990), basado en sus conocimientos previos sobre procesamiento digital de imagen con operaciones matemáticas morfológicas; ellos proponen un nuevo tipo de neurona donde se cambia la operación de multiplicación de entradas y pesos sinápticos por sumas, y la sumatoria por valores máximos o mínimos.

Posteriormente, (Davidson & Hummer, 1993) presentan un perceptrón morfológico teórico con una técnica de entrenamiento específico que garantiza ciclos finitos y sin problemas de convergencia, donde demuestran con un ejemplo teórico de memorias asociativas un mejor desempeño a las clásicas de Hopfield; también presentan una demostración matemática para la formación de la compuerta lógica NAND, determinando que es posible resolver cualquier problema computacional mediante esta red.

(Won & Gader, 1996) muestran usos prácticos de las redes morfológicas de pesos compartidos en el reconocimiento de imágenes, contrastándolas con redes lineales, donde claramente se lograron ventajas y eficiencia en la clasificación. (Raducanu & Grana, 2000) muestra redes entrenadas en solo un ciclo de computación, usadas en configuración de memorias hetero-asociativas, y como un paso de pre-procesamiento en detección de imágenes de forma humana para robots de navegación, donde estas redes, funcionando como memorias, demuestran una capacidad ilimitada.

El estudio de (Ritter, Lancu, & Urcid, 2003) sobre un perceptrón morfológico con estructura de dendritas, resalta las ventajas sobre el perceptrón al poder solucionar la compuerta XOR con solo una neurona que posee tres dendritas. La arquitectura no tiene problemas en dibujar siempre superficies de decisión cerradas y puede alcanzar sin problemas un error de entrenamiento cero, siempre y cuando el algoritmo de entrenamiento haga crecer el número de dendritas. Ese mismo año, (Ritter, Lancu, & Urcid, 2003) hacen una comparación más rigurosa entre la neurona biológica y la artificial propuesta, haciendo varios ejemplos para baja dimensionalidad de la misma; para determinar si las dendritas cuentan realmente como una capa oculta, a lo que se concluye que no, dado que no tienen función de activación, pueden variar su número con el entreno y funcionan con base a operaciones lógicas básicas para llevar a cabo el álgebra morfológica.

Por otro lado, otros estudios más recientes se han enfocado en los algoritmos de aprendizaje de esta arquitectura de red morfológica de dendritas, la mayoría de naturaleza heurística, se mencionan los tres más importantes: división de hiper-cajas

(*SLMP-P*) publicado en 2014, evolución diferencial en 2017 y gradiente descendente estocástico (*SGD*) en 2017; se habla de ellos al inicio del presente capítulo.

2.3.2. Modelo de Dendritas Biológicas

Muchos tipos de dendritas poseen pequeñas ramificaciones llamadas espinas, cuando están presentes, hacen una post-sinapsis del impulso excitatorio; las dendritas son la entrada de datos de la neurona, hacen parte de su procesamiento junto con el soma, para finalmente entregar la salida obtenida a través del axón, ver Figura 3.

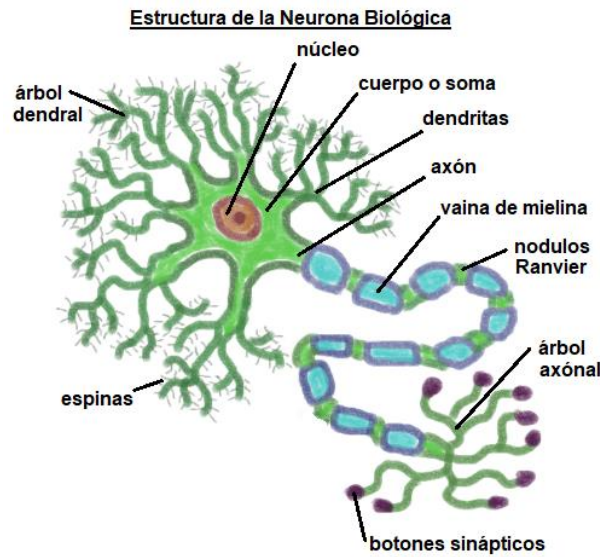


Figura 3 Estructura de la neurona biológica

Se ha propuesto que las dendritas son las unidades informáticas elementales del cerebro, en lugar de serlo la neurona (Gaia, 2012); así mismo, mecanismos basados en XOR, AND y NOT son utilizados para explicar matemáticamente el mecanismo lógico de las dendritas y es donde las operaciones morfológicas tienen cabida.

Para identificar las diferencias entre el modelo habitual de un perceptrón y el modelo matemático de la neurona morfológica básica (Ritter & Sussner, 1996), tenga en cuenta la ecuación (1):

$$y = f \left(b + \sum_{i=0}^I x_i \cdot w_i \right) \quad (1)$$

Donde, y es el valor de salida de la neurona, x_i es el vector de entradas con dimensión I , w_i son los pesos sinápticos asociados a cada entrada y corresponde a los parámetros a sintonizar mediante el entrenamiento junto con el bias b , y $f(\cdot)$ es la función de activación que suele ser escalón, rampa, sigmoideal, entre otras.

Ahora bien, las ecuaciones (2) y (3), cambian la operación de multiplicación por suma, y la sumatoria por mínimo o máximo respectivamente; este representa el principio de funcionamiento de un perceptrón morfológico. Adicionalmente, ya no se necesita el *bias*, pero se proponen nuevos parámetros, φ y ρ que adquieren valores -1 o 1, y son utilizados para inhibir las entradas ρ o la salida φ . La Figura 4 muestra la línea formada por la ecuación (1) a la izquierda, y el efecto de mínimo en (2) y máximo en (3); siendo las zonas rojas mayores a cero, los pesos w diferentes y los parámetros φ y ρ omitidos por ser opcionales, se omitirán de aquí en adelante.

$$y = f\left(\varphi \cdot \bigvee_{i=0}^I \rho_i \cdot (x_i + w_i)\right) \quad (2)$$

$$y = f\left(\varphi \cdot \bigwedge_{i=0}^I \rho_i \cdot (x_i + w_i)\right) \quad (3)$$

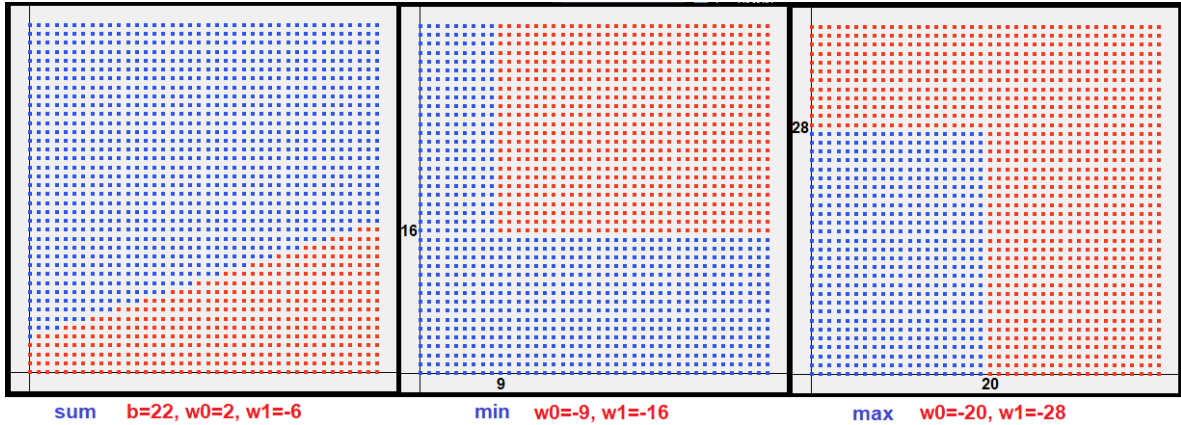


Figura 4 Superficies de decisión para perceptrón clásico y morfológico

La Figura 5 muestra la combinación de las ecuaciones (2) y (3), que dan como resultado la (4), esta última se ve en su forma simplificada o extendida, dado que l adquiere solo dos valores: 1 o 0, H o L respectivamente.

La ecuación extendida muestra dos parámetros w_i para cada dimensión o entrada i , es decir, alto w_i^H y bajo w_i^L , así pues, cada dimensión está acotada por dos valores entre los cuales su salida será positiva, en dos dimensiones esto representaría una caja, siendo una hiper-caja cuando la dimensionalidad es mayor. Así, se obtiene una caja, tal como se muestra en la Figura 5, siendo los valores w los mismos usados en la Figura 4, solo que esta vez, se adiciona el efecto de los límites usando $(-1)^l$ con $l \in [0, 1]$.

$$y = f \left(\bigvee_{i=0}^I \bigvee_{l=0:L}^{1:H} (-1)^l \cdot (x_i + w_i^l) \right) \quad (4)$$

$$y = f \left(\bigvee_{i=0}^I \{x_i + w_i^L | -(x_i + w_i^H)\} \right)$$

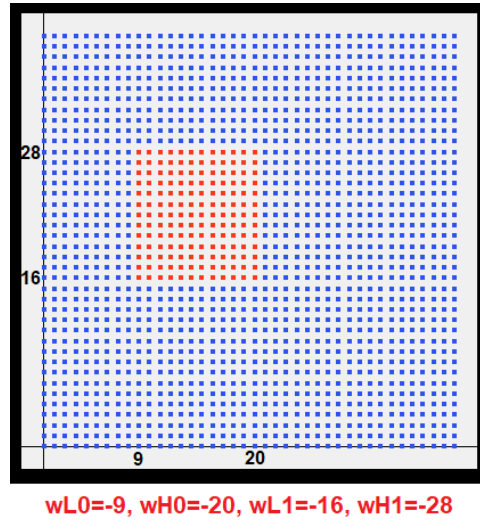
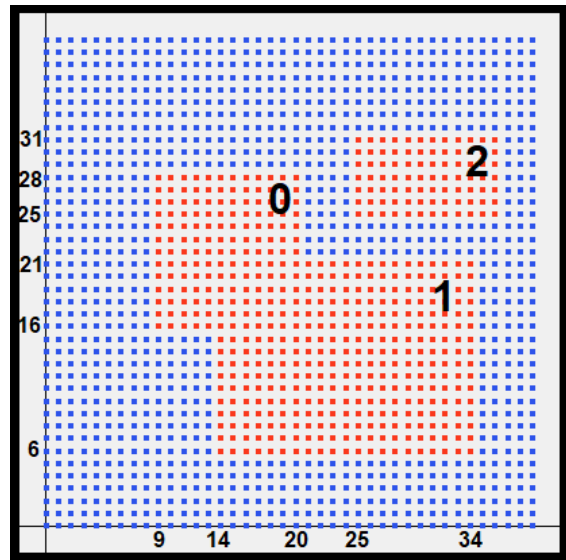


Figura 5 Representación de la ecuación para el modelo de una sola caja

Por lo tanto, en la *DMNN* cada dendrita dibuja una hiper-caja en el hiper-espacio, logrando que las soluciones siempre sean superficies cerradas y tomen una complejidad mayor cuando haya más dendritas; de nuevo, haciendo una versión simplificada de la ecuación anterior, y asumiendo que s_k es la salida de cada dendrita k , la ecuación (5) muestra la salida total z de la neurona; en otras palabras, la neurona representa el valor mayor de sus dendritas; note que para los casos presentados en los ejemplos, la función de activación $f(\cdot)$ es siempre la función escalón unitario.

$$z = f\left(\bigwedge_{k=0}^K s_k\right) \quad (5)$$

El ejemplo de la Figura 6 muestra el resultado para tres dendritas con valores arbitrarios, donde la dendrita “0” tiene los mismos valores de la Figura 5, y la función de activación sigue siendo el escalón unitario, por lo que el interior de las cajas es mayor igual que cero. Finalmente, en cada neurona podrían considerarse diferentes funciones de activación, aunque sigue careciendo de derivabilidad dado que posee operadores máximo y mínimo no diferenciables.



**wL00=-9, wH00=-20, wL01=-16, wH01=-28
wL10=-14, wH10=-34, wL11=-6, wH11=-21
wL20=-25, wH20=-36, wL21=-25, wH21=-31**

Figura 6 Ejemplo con tres dendritas

Para seleccionar dos clases, una neurona es suficiente, pero cuando el problema tiene múltiples clases son necesarias más neuronas; además, la superficie de decisión puede hacerse más compleja, es decir no solo basada en rectángulos concatenados; (Humberto Sossa, 2014) utilizan el algoritmo de división de hiper-cajas, que aunque éstas no se superponen, ocupan todo el espacio; por otra parte (Zamora & Sossa, 2017) optan por usar la función *argmax* a la salida de la red, la cual devuelve el índice asociado al mayor valor de un vector, para determinar cuál será la dendrita ganadora y determinar a qué superficie de decisión debe ser asignado el

dato, teniendo como característica no dejar zonas de indecisión. La Figura 7 muestra lo descrito, donde dos neuronas: roja (0) y azul (1), representan los límites de decisión de las dendritas y el fondo amarillo indica lo que no pertenece a las clases; al hacer uso de la función *argmax* las regiones son difuminadas de acuerdo a la proximidad de los datos a las regiones clasificadas, dando como resultado lo presentado en el lado derecho.

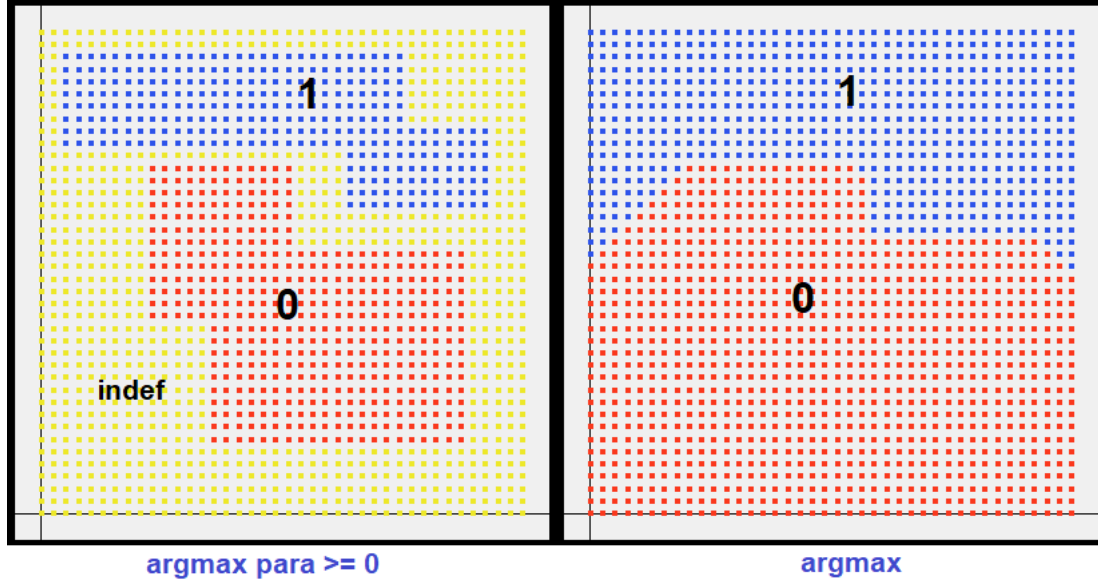


Figura 7 Superficie de decisión para dos neuronas con dos dendritas cada una

La ecuación (6) presenta el modelo matemático de toda la *DMNN*, donde los subíndices son: M el número de neuronas que a su vez es el número de clases, K es el número de dendritas que además puede depender de la neurona, siendo K_m el número total para la neurona m , y ya se había dicho que I son las entradas, o dimensionalidad del problema; los otros parámetros son: Z_m la salida total y sin función de activación de cada neurona, S_{mk} la salida total y sin función de activación de cada dendrita, x_i el vector con los datos de entrada, y la salida que es un número que representa a la clase, w_{mki}^l son los pesos sinápticos que serán sintonizados con algoritmos de entrenamiento; estos pesos tienen entonces cuatro subíndices, por lo que se leería la referencia a uno como: el peso alto H / bajo L de la neurona m , de la dendrita k para la entrada i .

$$s_{mk} = \bigvee_{i=0}^I \bigvee_{l=0:L}^{1:H} (-1)^l \cdot (x_i + w_{mki}^l)$$

$$z_m = \bigwedge_{k=0}^{K_m} s_{mk}$$

$$y = \operatorname{argmax}(z_m)$$
(6)

Con esto se comprende por qué no se requieren entonces capas ocultas, la red neuronal es capaz de dibujar en la superficie de decisión formas geométricas complejas y cerradas; la discusión de si ¿las dendritas cuentan realmente como una capa oculta? responde que: No porque no tienen función de activación, pueden variar su número con el entreno y funcionan con base en operaciones lógicas básicas para llevar a cabo el álgebra morfológica (Ritter, Lancu, & Urcid, 2003). En la Figura 8 se puede ver la estructura general de la red neuronal morfológica de dendritas.

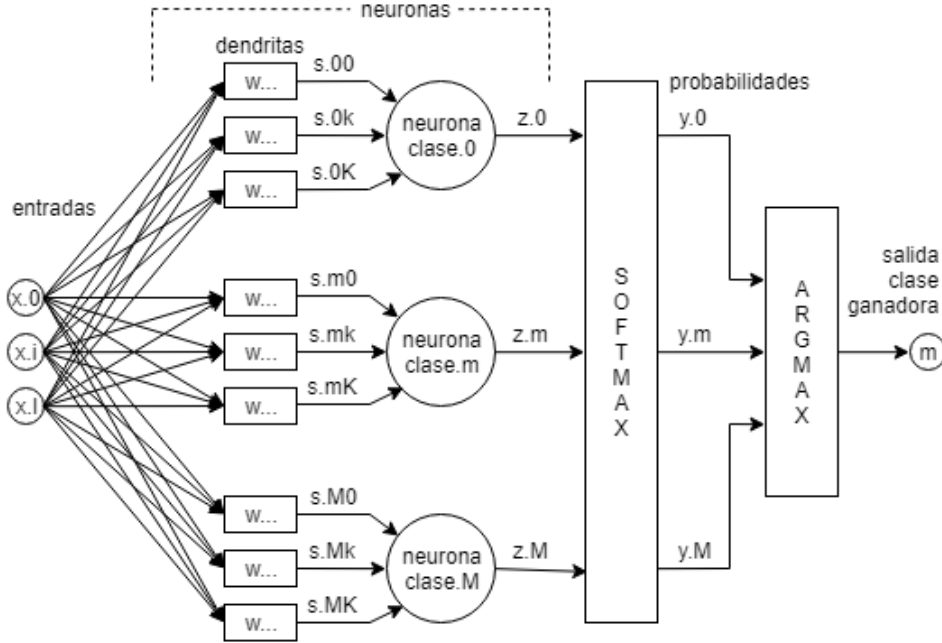


Figura 8 Diagrama de la DMNN

Al inicio de la explicación del modelo se habló de unos parámetros extras, como el φ y ρ ; ya que el entreno de la *DMNN* requiere optimizar el número de hiper-cajas, se definirá la variable $b_{mk} \in [0, 1]$ como un vector de booleanos referidos a cada dendrita, donde 0 implica inhibición; con el fin que durante el entreno se desactiven o activen hiper-cajas para testear su utilidad en el resultado final.

2.4. INICIALIZACIÓN

A diferencia del *MLP* u otros tipos de redes neuronales que inicializan sus pesos al azar (usualmente con distribuciones de probabilidad que favorecen), la *DMNN* requiere una inicialización más adecuada dada su interpretación geométrica (Arce, Zamora, Sossa, & Barron, 2018). Entre ellos se destacan: el algoritmo de K-medias y la división de hiper-cajas, los cuales garantizan una excelente fase inicial, ya que los errores respecto a los datos de entrenamiento son lo suficientemente bajos para considerarse como algoritmos de configuración de los pesos sinápticos.

2.4.1. K-medias

Este algoritmo ampliamente usado en minería de datos (Aggarwal & Reddy, 2013), lo que hace es agrupar conjuntos de datos para cualquier dimensionalidad de entrada, estos clústers generados, para el caso de inicialización de la *DMNN* se convertirán en las hiper-cajas que representan a las dendritas; es comprensible de acuerdo a lo visto en el modelo de esta red neuronal, que se saque provecho de este algoritmo, a causa de su explotación geométrica del problema de agrupamiento; en la Figura 9 se puede apreciar el diagrama de bloques.

Para empezar, el algoritmo recibe los patrones que definen al problema, y un valor K que sería el número de clústers o agrupamientos deseados, en el caso de inicializar la *DMNN* este valor usualmente se pone por ensayo y error, pues depende de la geometría del problema, aunque una buena práctica es poner un número alto dado que posteriormente otro algoritmo haría la reducción de hiper-cajas (Arce, Zamora, Sossa, & Barron, 2018); por otra parte, el algoritmo K-medias opera los patrones de cada clase del problema por separado.

La Figura 10 muestra el procedimiento de creación de dendritas con el algoritmo K-medias para una neurona, donde se observa un ejemplo sencillo que consta de 165 patrones con dimensionalidad dos, 24 centroides elegidos arbitrariamente y un total de 9 ciclos; las cajas al final son creadas con una dimensionalidad proporcional al tamaño máximo de los datos de entrada para cada dimensión.

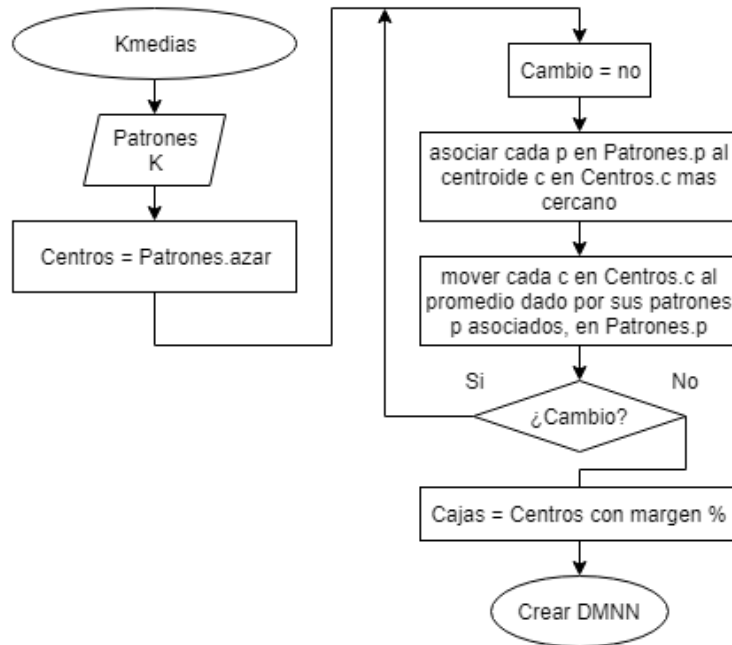


Figura 9 Diagrama de bloques del algoritmo K-medias

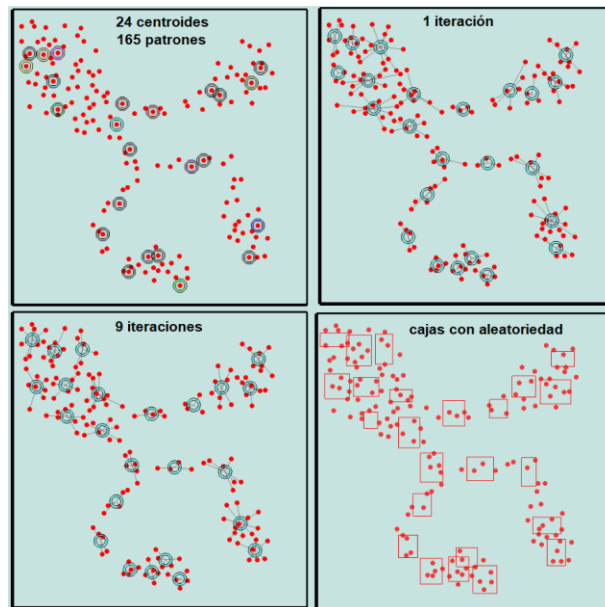


Figura 10 Inicialización K-medias para una neurona (una clase)

2.4.2. División de Hiper-cajas

La división de hiper-cajas, propuesto en (Humberto Sossa, 2014), es un procedimiento de segregación iterativa que busca discriminar cada clase en grupos pequeños de datos de la misma clase. En la Figura 11 donde se distinguen dos ciclos del algoritmo, se muestra el procedimiento descrito anteriormente.

Inicialmente, se encierran a todos los patrones de datos en una sola hiper-caja, esto lo logra encontrando los valores máximo y mínimo para cada dimensión de entrada, luego pone un margen extra para abarcar más allá de los datos ((Humberto Sossa, 2014) proponen un 10% a cada lado). Luego se procede a realizar una verificación del tipo de clase de datos, donde se determina si divide la caja en dos más pequeñas cuando los datos no corresponden a la misma clase, y termina este ciclo una vez no pueda realizar alguna división.

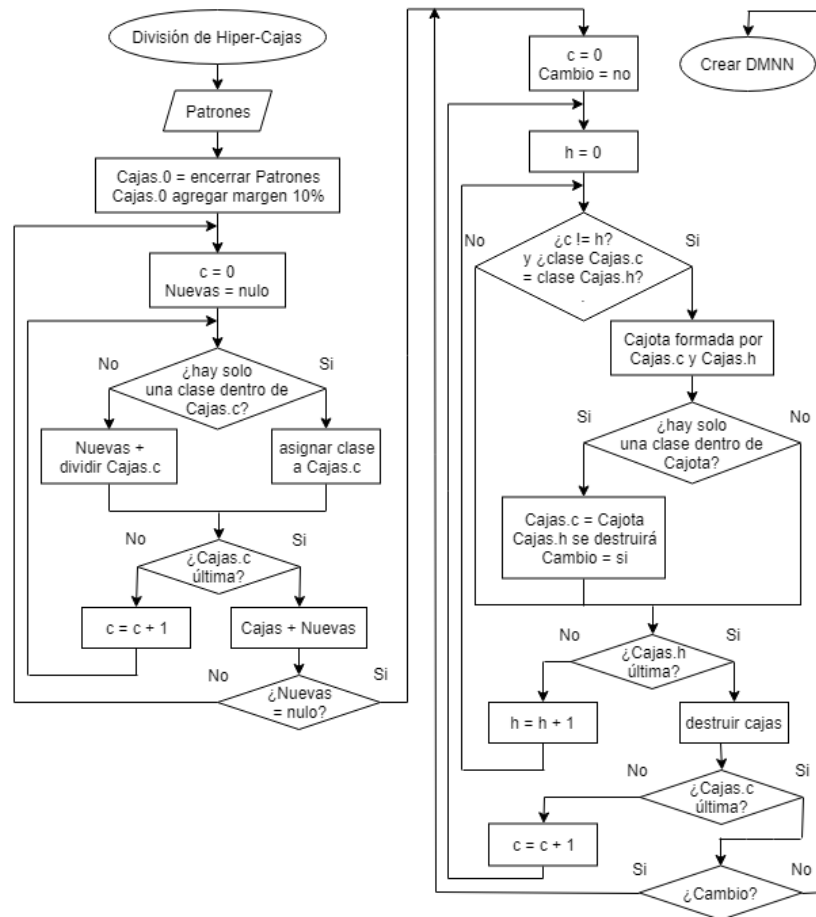


Figura 11 Diagrama de bloques del algoritmo de División de Hiper-Cajas

El segundo ciclo del algoritmo de la Figura 11 se encarga de unir las cajas que lo permitan para disminuir su número; este procedimiento intenta encontrar los límites de la hiper-caja que mantenga el conjunto convexo durante un ciclo que se repite hasta no poder realizar otra unión; en cada unión se verifica si se mantiene una relación de datos de la misma clase que terminarían por eliminar las cajas más pequeñas para ser reemplazadas por la nueva caja más grande.

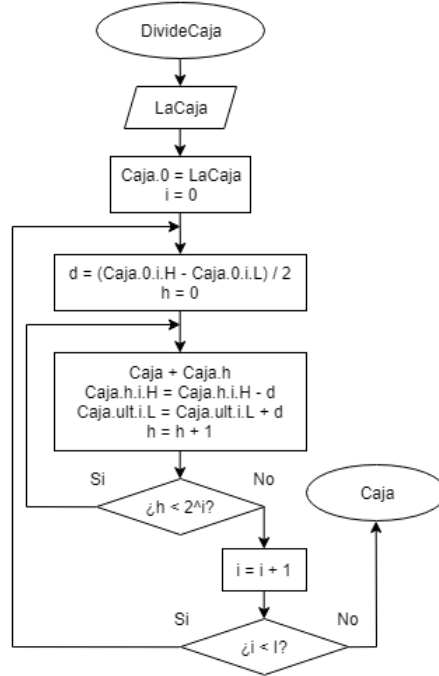


Figura 12 Diagrama de bloques del algoritmo que divide una caja

El algoritmo es agresivo ya que la división de esta geometría multidimensional podría llegar a masificarse hasta el orden de 2^I , siendo I la dimensionalidad del problema, en otras palabras, el algoritmo se ralentiza exponencialmente con la dimensionalidad; pudiendo sobrecargar la memoria dinámica del hardware. En la Figura 12, se muestra el algoritmo de división de hiper-cajas entregando como resultado un vector con las nuevas hiper-cajas, siendo H y L las partes alta y baja de cada dimensión i . La Figura 13 presenta una respuesta del algoritmo con error de entrenamiento de cero en la generación de las hiper-cajas.

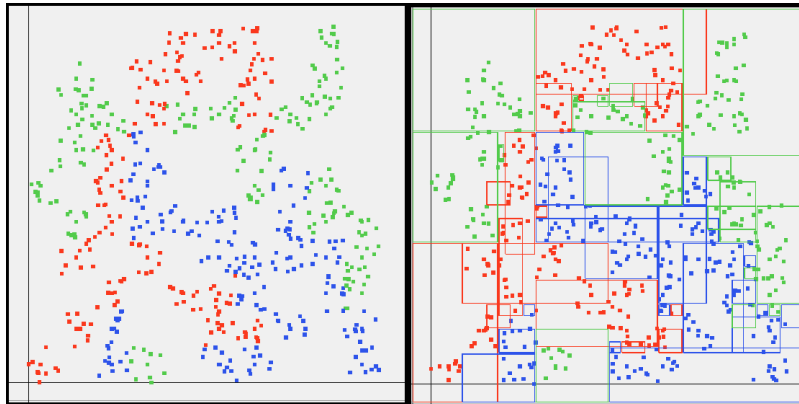


Figura 13 Inicialización por división de hiper-cajas

2.5. ADECUACIÓN DE SALIDA

2.5.1. Capa Softmax

La función softmax es utilizada en las redes neuronales, como capa de salida, cuando es necesario clasificar excluyentemente (Bishop, 2006). Este tipo de capa ayuda a no perder información probabilística y permite hacer uso del algoritmo *backpropagation* para la actualización de los pesos sinápticos al ser diferenciable (Zamora & Sossa, 2017). La función *softmax* recibe un vector de datos y los ajusta a valores entre 0 y 1 para indicar la probabilidad con que cada dato pertenece a alguna de las clases así:

$$y_m = e^{z_m} / \sum_{n=1}^M e^{z_n} \quad (7)$$

$$\frac{\partial y_m}{\partial z_m} = y_m \cdot (1 - y_m) \quad (8)$$

En la ecuación (7), z_m representa la salida de la neurona m -ésima, la cual es normalizada sobre la sumatoria exponencial de todas las neuronas que representan cada clase hasta M , y_m representa entonces la probabilidad ponderada de la clase m , que posteriormente podría pasar por una capa de selección *argmax* que identificaría el índice de la clase ganadora. Por otro lado, la ecuación (8) muestra la derivada de la función softmax.

2.5.2. Función de Error

En el caso de los algoritmos evolutivos esta función se conoce también como fitness, la cual mide el desempeño de un modelo. Dos alternativas para medir el error son: El error cuadrático medio dado por la ecuación (9), y el error logístico o máxima entropía presentado en la ecuación (10). (Bishop, 2006)

$$\varepsilon = \frac{1}{P} \cdot \sum_{p=1}^P \begin{cases} 0, & \text{if } \text{argmax}(y_{mp}) = y_{d_p} \\ 1, & \text{otherwise} \end{cases} \quad (9)$$

$$\varepsilon = \frac{-1}{P} \cdot \sum_{p=1}^P y_{d_p} \cdot \log(y_p) + (1 - y_{d_p}) \cdot \log(1 - y_p) \quad (10)$$

Siendo y_{d_p} la salida deseada a la clase de interés, y_p la salida de la red morfológica del p -ésimo patrón de entrada, para los P patrones de entrada.

2.6. ENTRENAMIENTO

2.6.1. Gradiente Descendente

El gradiente descendente es muy utilizado en problemas de optimización matemática y es eficiente con funciones multivariable como lo es el caso de las *ANN* (Bishop, 2006). El gradiente genera un campo vectorial de un campo escalar, en dirección y magnitud en el que campo escalar crece más rápido; además, es la generalización de la derivada, pues hace derivadas parciales del campo escalar respecto a cada una de sus componentes o dimensiones. La Figura 14 muestra un ejemplo de gradiente descendente, donde se asume que al cambiar la dirección del gradiente 180° se obtiene la dirección hacia el mínimo de la función de interés.

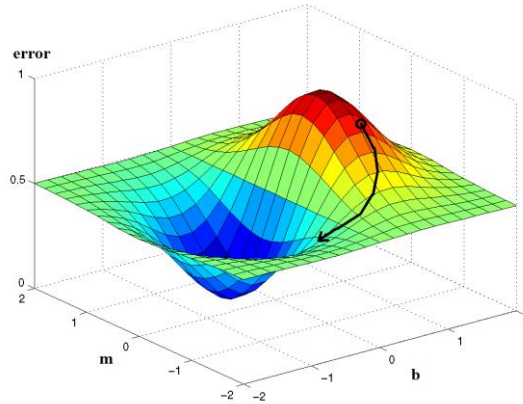


Figura 14 Superficie de error y camino de descenso, tomado de Mathworks

Bajo el concepto de gradiente descendente, la ecuación (11) muestra cómo se actualiza el valor de los pesos sinápticos en términos de los cambios de la función ε donde α es un parámetro de ajuste que estima los cambios de w en cada iteración.

$$w = w - \alpha \cdot \frac{\partial \varepsilon}{\partial w} \quad (11)$$

Para el caso de las redes neuronales MLP, este método es reconocido como *Back-Propagation* ya que considera los cambios parciales por cada capa oculta, evitando el sobre-costeo computacional al realizar el proceso de ajuste por cada capa y no por

el conjunto de todos los parámetros de toda la red neuronal. Este es el algoritmo básico de aprendizaje de cualquier red MLP, pero presenta dificultades, entre las cuales se encuentran:

- Definir el paso α puede implicar una muy lenta convergencia o una brusca oscilación entorno al mínimo a causa del gran cambio entre iteraciones.
- El mismo problema anterior, pero teniendo en cuenta que α es el mismo para todas las dimensiones, por lo que puede funcionar bien para unas y mal para otras.
- Las zonas donde la derivada es cero, además de los mínimos y máximos, en la Figura 14 pueden verse zonas de meseta (región verde).
- La convergencia a un mínimo local, también llamado estancamiento, como se observa en la Figura 15, dado que los problemas pueden ser de naturaleza más compleja.

Algunas alternativas para mejorar el desempeño del algoritmo de gradiente descendente consisten en variar el valor de α en términos de las iteraciones, de tal forma que haya movimientos más alargados en las primeras iteraciones y más finos en las etapas finales durante el proceso de aprendizaje. El segundo ítem puede ser solucionado con algoritmos como AdaGrad (John Duchi, 2010), AdaDelta (Zeiler, 2012), Adam (Diederik P. Kingma, 2014), que son algoritmos de gradiente descendente adaptativos, donde cada parámetro se actualiza individualmente de acuerdo a su propio gradiente; por ejemplo, en AdaGrad la ecuación (12) muestra como para cada j -ésimo w_j el valor de α se divide por g_j que es la suma de los cuadrados del gradiente para w_j en cada paso; ξ es solo un valor cercano a cero para evitar divisiones por cero. El algoritmo AdaDelta limita dicha acumulación a un número de iteraciones para evitar que se detenga prematuramente la actualización de pesos.

$$w_j = w_j - \frac{\alpha}{\sqrt{g_j} + \xi} \cdot \frac{\partial \varepsilon}{\partial w_j} \quad (12)$$

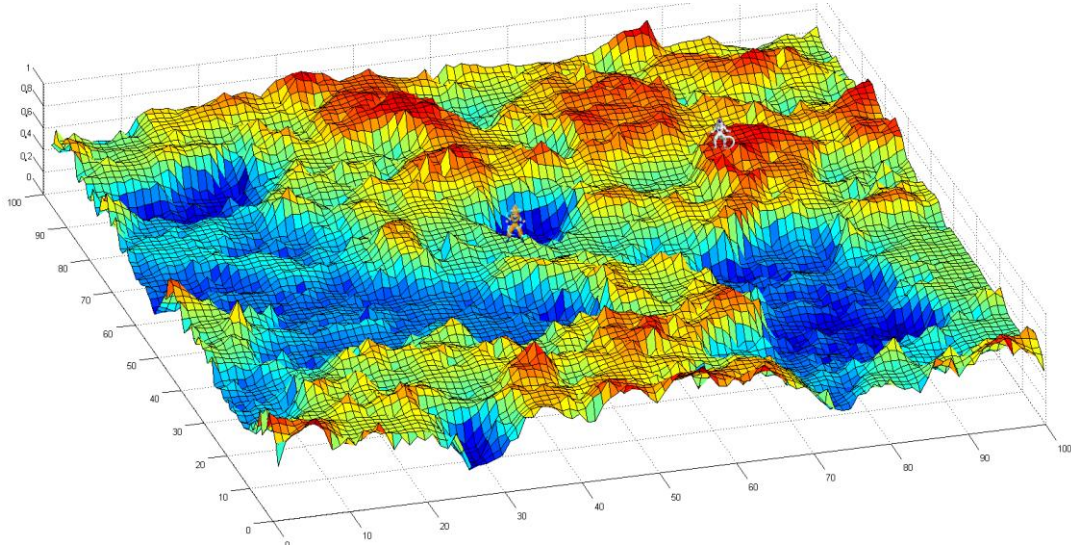


Figura 15 Superficie de error compleja, tomado de Mathworks

Para el resto de problemas, se tienen otras soluciones, entre ellas el uso del impulso o momento (momentum) en el cálculo de la actualización del peso; la ecuación (13) muestra el efecto del momentum, que además requiere la sintonización de otro hiper-parámetro $\beta \in [0, 1]$, este representa la fuerza del impulso; en general, la ventaja de usar esta modificación del gradiente descendente es que puede escapar de algunos mínimos locales y agrega componentes de momentum diferentes a cada dimensión.

$$u = \beta \cdot u + \frac{\partial \varepsilon}{\partial w} \quad (13)$$

$$w = w - \alpha \cdot u$$

La otra solución es a partir de la partición de los datos en mini-batches, donde se hace una estimación del gradiente con la respuesta de algunos patrones de entrada y sus respectivas salidas, esto se identifica como Batch-SGD (Gradiente Descendente Estocástico, por sus siglas en inglés).

La naturaleza estocástica en la actualización de los pesos sinápticos en los pesos de una red neuronal sobre el gradiente descendente, permiten aumentar la posibilidad de hallar un mínimo global del problema de optimización en el proceso de aprendizaje (Herbert Robbins, 1951). Al subdividir aleatoriamente los patrones en paquetes llamados batches, el proceso de búsqueda del mínimo evitaría estancamiento prematuro, por la misma naturaleza en la aproximación de la función de

costo, donde cada bache representa una función de costo aproximada del sistema original (donde se tienen en cuenta todos los patrones); en el caso particular en que el bache es de tamaño 1, el algoritmo sería el *SGD* puro, pues en cada iteración se actualizan los pesos usando solo un patrón tomado al azar.

(Zamora & Sossa, 2017) lograron entrenar a la red *DMNN* utilizando *SGD*, haciendo uso de una capa softmax y el error de regresión logística. Dada la naturaleza no derivable de este tipo de arquitectura de red neuronal, ellos propusieron la siguiente aproximación:

$$\frac{\partial \varepsilon}{\partial w_{mki}^l} = \frac{1 - y_d}{\ln(10)} \cdot \begin{cases} 1, & l = L \\ -1, & l = H \end{cases} \quad (14)$$

La ecuación (14) presenta la derivada del error respecto del peso sináptico w ganador, para un solo patrón de entrenamiento p por lo que no se especifica su subíndice, para llegar a esto se toma un valor de la sumatoria de error de la ecuación (10), la derivada del softmax de la ecuación (8) y se calcula la derivada de la ecuación (4), siendo esta última, un valor 1 o -1; en general, el error se propaga hacia atrás haciendo en las operaciones max y min, una selección de la procedencia del valor que resultó a su salida. Cuando se va a hacer el proceso de reajuste de pesos sinápticos con el algoritmo *SGD*, en cada iteración se toma un p y se ajusta el peso de solo una componente w de la red, como lo muestra la ecuación (11); complementariamente en la misma iteración, se ajusta el peso w proveniente de la derivada tomada de la clase ganadora, no de la deseada, y determinar si se hace el ajuste o no a partir de la coincidencia entre la clase ganadora con la clase deseada.

2.6.2. Evolución Diferencial

Cuando se enfrenta un problema de optimización, los algoritmos inspirados en la naturaleza ofrecen un abanico de posibilidades, entre ellos se encuentran los algoritmos evolutivos, genéticos, autómatas celulares, inteligencia de enjambres, colonias de hormigas, auto-ensamblamiento biomolecular, entre otros. Estos al ser heurísticos, no deterministas, multiagentes, adaptativos, son capaces de resolver problemas para los cuales no se tiene un modelamiento matemático detallado, ya sea por ser una caja negra o por poseer alta complejidad (Forbes, 2005).

En el caso de una arquitectura *DMNN* se ha utilizado un algoritmo de evolución diferencial (*DE*, por sus siglas en inglés) (Arce, Zamora, Sossa, & Barron, 2018), para el ajuste de los parámetros de las dendritas. La evolución diferencial, corresponde en general, a un algoritmo genético basado en poblaciones de individuos, que a su vez representan las soluciones candidatas del problema. Este algoritmo iterativo, crea nuevas generaciones de individuos, donde se describen operadores genéticos como la mutación y el cruce; cada generación nueva es el resultado de individuos con mejores características que la generación inmediatamente anterior (Eiben & Smith, 2003). Ver Figura 16.

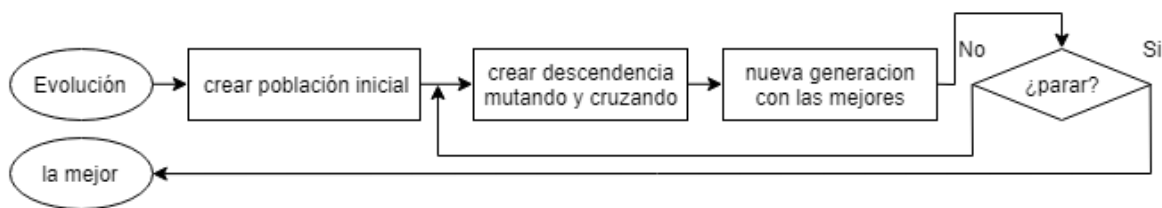


Figura 16 Diagrama de bloques algoritmo evolutivo genético

Nótese entonces que hay dos puntos críticos: el primero implica ¿Cómo se va a crear a la nueva generación? El segundo, ¿Cómo se va a medir el desempeño de una solución? Esta última pregunta refiere entonces a una función de error o *fitness*.

La *DE* fue introducida por (Storn & Price, 1997) y se encarga de la creación de la nueva generación, usando operaciones vectoriales que mezclan mutación y recombinación. La Figura 17 muestra los pasos del algoritmo completo. Inicialmente, recibe como parámetros de entrada a los patrones que serán necesarios para efectuar la función de error o *fitness*, el número de individuos que conformarán el tamaño constante de la población, la cantidad de ciclos o generaciones que durará el entrenamiento, si es que no alcanza antes un valor de error deseado, las dos constantes c y h que definen la probabilidad de recombinación y la escala de mutación respectivamente. Finalmente, recibe a los pesos sinápticos o más generalmente, el vector de características del modelo que se va a optimizar, lo cual es análogo a la cadena genética de cromosomas, tal como se ve en la biología micro-celular.

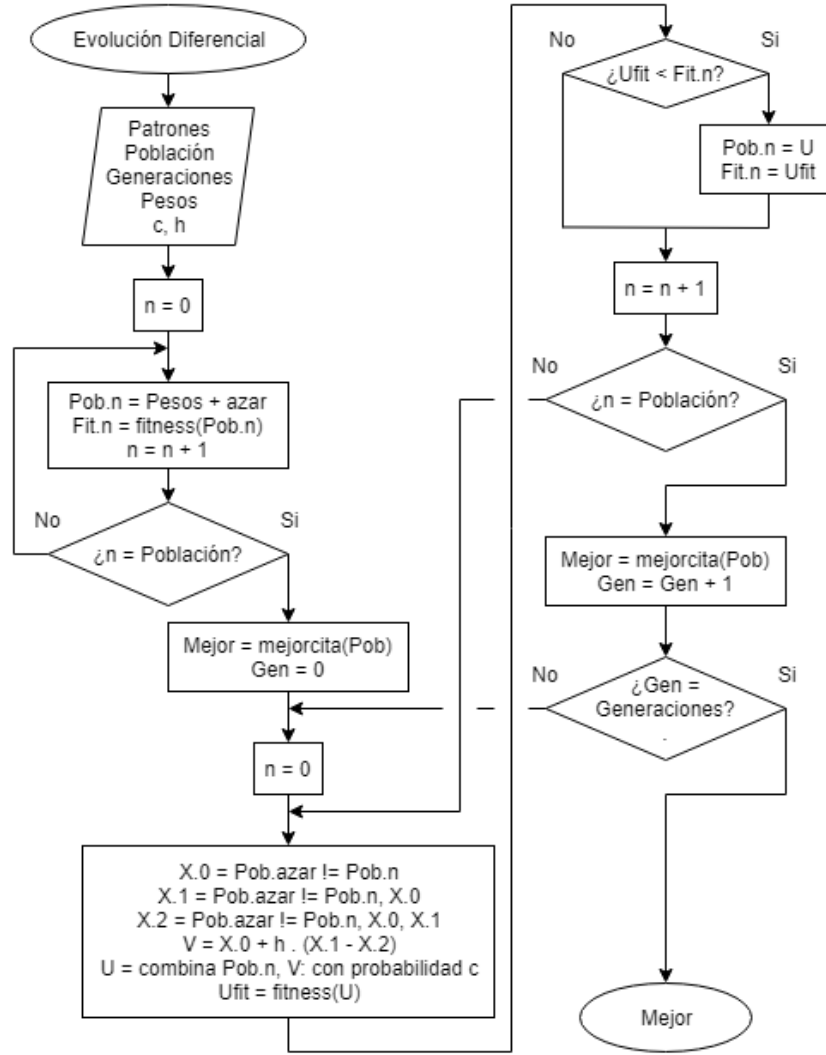


Figura 17 Diagrama de bloques del algoritmo de evolución diferencial

En el primer ciclo, el algoritmo crea una generación a partir del vector de genes dado, añadiéndole aleatoriedades, y evaluando su *fitness*; el algoritmo en general es de carácter elitista, pues las mejores soluciones no se perderán y será la mejor de éstas la que se entregue como resultado final. Comenzando con el ciclo de entrenamiento, tenemos que para cada individuo n de la población Pob se creará un descendiente u , en primer lugar se seleccionan al azar otros 3 individuos x_0, x_1, x_2 , todos diferentes, quienes formarán la descendencia u de i componentes como lo muestra la ecuación (15).

$$u_i = \begin{cases} Pob_{x_0i} + h \cdot (Pob_{x_1i} - Pob_{x_2i}), & rand < c \\ Pob_{ni}, & else \end{cases} \quad (15)$$

Luego se halla el valor de su *fitness* y se compara con el del padre n , quien será reemplazado si el hijo resulta un mejor candidato, al terminar este ciclo para toda la población, se tendrá una nueva generación cuyo mejor *fitness* será la mejor solución alcanzada hasta el momento (Kenneth V. Price, 2005).

Por otro lado, los parámetros h y c indican la escala de mutación y la probabilidad de recombinación, respectivamente. $c \in [0, 1]$ es un parámetro de mutación ya que, a pesar de que media un proceso cruzado, es la probabilidad de que uno de los parámetros herede de proceso de recombinación mutante (Kenneth V. Price, 2005); en general, si a una red neuronal se aplica la operación de evolución diferencial con un $c = 1/total_dendritas$, entonces teóricamente, al menos una hiper-caja de la red neuronal descendiente tendría cambios mutantes en una de sus características. Se recomienda cambiar solo un parámetro en cada mutación por lo que valores $0 \leq c \leq 0.2$ son usados frecuentemente, por otro, una población con altas mutaciones tendría valores y $0.9 \leq c \leq 1$. El parámetro $h \in [0, 1]$ escala el factor de mutación, es decir, la fuerza de la mutación. Un h grande supondría un cambio más brusco en los nuevos límites de la hiper-caja; un valor típico es 0.75, donde además se sobreentiende que un h alto favorece operaciones de exploración mientras uno bajo lo hace para efectos de explotación. Algunos trabajos como (Cecilia Sosa, 2012) proponen $h = f(iteración)$ dinámico, siendo $f(\cdot)$ una función gaussiana con altura 1, media 0 y alta varianza 500000.

2.6.3. Enjambre de Partículas

La optimización por enjambre de partículas nace al igual que los algoritmos evolutivos, de la inspiración biológica, unos ejemplos de ello son el comportamiento de las bandadas, los cardúmenes, el forrajeo de bacterias, búsqueda de recursos de las hormigas y abejas; se puede distinguir claramente que en estos comportamientos que implican movimiento espacial, hay alguna condición del sistema que debe optimizarse, por ejemplo el buscar la fuente de alimento más cercana y abundante, la ruta óptima, consumir menos energía al volar o nadar largas distancias, persuadir eficazmente a los depredadores, entre otras (James Kennedy, 1995).

Los estudios en inteligencia de enjambres fueron desarrollados históricamente con modelos computacionales. Se trata de la cooperación de varios agentes o partículas, donde hay una función de error o *fitness* que describe que tan bien ubicada está una partícula en el espacio de búsqueda, cada partícula tiene acceso a dos fuentes de información (los parámetros mencionados pertenecen a la ecuación (16)):

- Memoria local (PB), guarda la mejor posición que la partícula alcanzó, y le ocasiona una tendencia de volver allí. El parámetro $c1$ controla esta característica, se agrega aleatoriedad también.
- Memoria colectiva ($Mejor$), guarda la mejor posición obtenida globalmente, entonces todas las partículas van a tener una tendencia a ir hacia allí. El parámetro $c2$ controla esta característica, se agrega aleatoriedad también.

Viendo el algoritmo de la Figura 19, se emplean sencillos cálculos físicos de cinemática para cambiar la posición de cada partícula en cada iteración, como se ve en la ecuación (16). El parámetro $c3$ es una fricción que para el algoritmo básico se deja en 1, siendo menor, una atenuación del movimiento, PP_n es la posición de la partícula n y PV_n es su velocidad, los dos *rand* van entre 0 y 1 añadiendo aleatoriedad, ya que este es un algoritmo de naturaleza estocástica.

$$PV_n = c3.PV_n + c1.rand.(PB_n - PP_n) + c2.rand.(Mejor - PP_n) \quad (16)$$

$$PP_n = PP_n + PV_n$$

Se recomienda que los valores de $c1$ y $c2$ sean 1.7 (Trelea, 2003), un valor mayor de $c1$ respecto a $c2$ implicaría individuos aislados y errantes entorno a su propia mejor posición PB , mientras que un valor mayor de $c2$ respecto de $c1$ implicaría una rápida convergencia de todos los individuos al punto global, lo que prematuramente podría ser un mínimo local de la función de error.

La ecuación (16) puede verse gráficamente en la Figura 18 donde se tiene en azul a los datos de la partícula a operar, de rojo a los datos respecto de la mejor posición global y en verde respecto de la local; el vector amarillo es la velocidad que resulta al final de cada iteración para cada partícula, y definirá su nueva posición, donde procederá a evaluarse la función *fitness*.

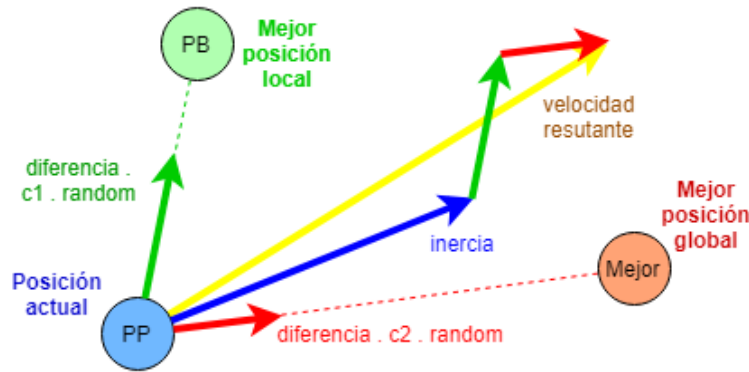


Figura 18 Actualización velocidad de la partícula

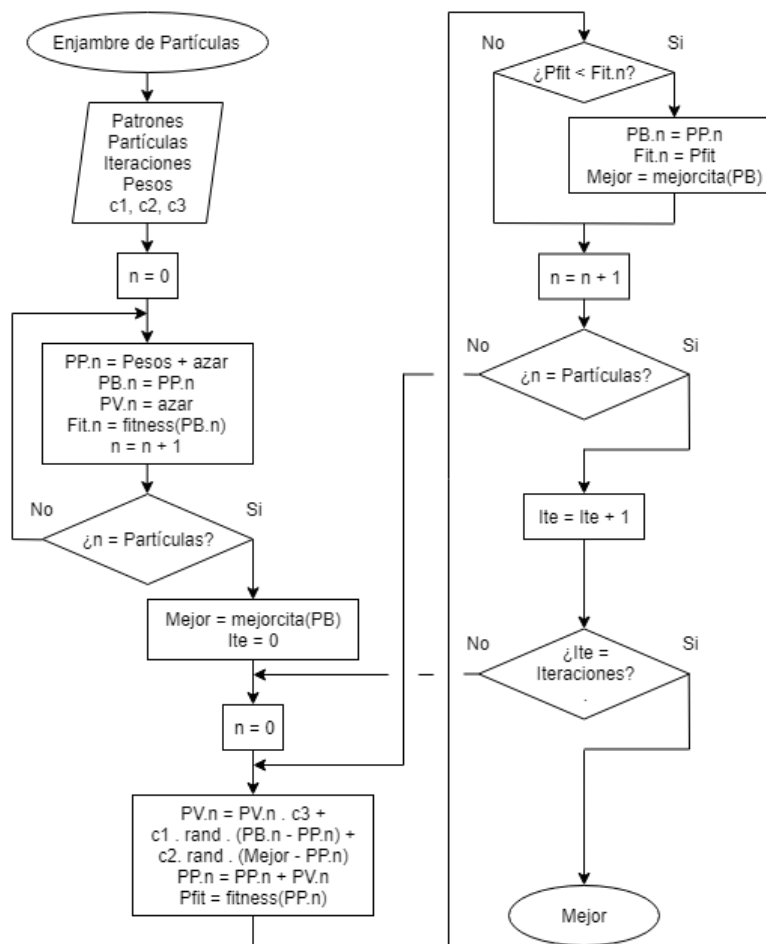


Figura 19 Diagrama de bloques del algoritmo de enjambre de partículas

Hay similitudes entre los algoritmos de inteligencia de enjambres y los de evolución genética, pero los primeros tienen la capacidad de encontrar soluciones globales, es decir, el mínimo global de la función de error, o al menos acercarse, esto debido

a que las partículas u agentes pueden recorrer la superficie de error del mismo modo que una abeja recorre el campo para hallar la mejor flor.

En general los algoritmos de aprendizaje dependen de varios parámetros, pero los heurísticos como *DE* o *PSO* tienden a tener más; encontrar los parámetros adecuados que lleven a una rápida convergencia cercana al mínimo global es un desafío, ya que depende mucho de la naturaleza del modelo, la función de error y el problema en general. En el caso de *PSO* se recomienda que c_1 y c_2 sumados no sobrepasen 4, usualmente reciben valores entre 0.7 y 2 (Yang, 2010), siendo en ocasiones tentativa y ligeramente mayor c_2 ; por otra parte, si c_1 y c_2 varían a lo largo del entreno, c_1 debería decrecer y c_2 crecer, para favorecer estrategias de exploración y luego explotación. En caso de usarse c_3 este estaría entre 0.4 y 0.9, aplicando variaciones lineales durante el entreno; la función de c_3 es mantener el movimiento inercial de la partícula (A. Serani, 2014).

2.7. OPTIMIZACIÓN DE DENDRITAS

La DMNN puede utilizar estrategias de aumento o disminución de dendritas (hiper-cajas) durante el entreno, para poder crear la compleja superficie de decisión o aligerar peso, tanto en memoria como computacionalmente. Como se ha mencionado la inicialización crea hiper-cajas que pueden cubrir todos los patrones y los algoritmos de entreno las reubican para optimizar la red, así que este apartado se enfocará en la reducción de hiper-cajas durante el entreno o post-entreno.

Se tienen tres algoritmos de inhibición:

- Desactivación Aleatoria:
En cada ciclo de entreno se elige una dendrita al azar de la mejor solución, si la dendrita está activa, se desactivará.
- Post-Entreno, Unión de Hiper-Cajas:
Similar al lado derecho de la Figura 11 recorre todas las dendritas, y para cada una hará la fusión con otra de su clase, eliminándose una dendrita y quedando otra de mayor hiper-volumen.
- Post-Entreno, Eliminación de Hiper-Cajas:
Recorre todas las dendritas inhibiéndolas.

Para los tres algoritmos, se operará el fitness luego de modificar una dendrita, esto para saber si se conserva el cambio en caso de que el error sea adecuado.

2.8. CRITERIOS DE EVALUACIÓN

Se entiende que entre menos dendritas necesite una *DMNN* para realizar la clasificación, más ligera será su ejecución y requerirá menos recursos de memoria. Por ello, y considerando alguna tolerancia en el error obtenido en el aprendizaje, los resultados de la evaluación se verificarán a partir de las matrices de confusión. Las matrices de confusión, usadas en los problemas de clasificación, son un arreglo de los datos reales y predichos, como se muestra en la Tabla 4, para un problema de M clases (Sofia Visa, 2011); en general lo ideal es que los valores más altos estén en la diagonal sombreada, mientras los demás tiendan a cero; esta diagonal se llamará: valores Verdaderos, y el resto son Falsos.

Para una clase m , las celdas de su columna se llamarán valores Positivos, y el resto son Negativos; de esta manera se tienen 4 combinaciones VP, VN, FP, FN.

Tabla 4 Estructura general de la matriz de confusión

Matriz de Confusión		Predichos		
		Clase 0	Clase m	Clase M
Reales	Clase 0	VN	FP	FN
	Clase m	FN	VP	FN
	Clase M	FN	FP	VN

Una vez se tiene la tabla, se procede a hacer operaciones estadísticas sobre sus datos:

- **Precisión** refiere al porcentaje de datos correctamente clasificados, ecuación (17), su inverso sería el **Error**.

$$precisión = \frac{verdaderos}{total} \quad (17)$$

Luego hay 3 mediciones estadísticas importantes, que son la exactitud, sensibilidad y especificidad, están referidas a cada clase m :

- **Exactitud** es la probabilidad de que un caso clasificado como perteneciente a la clase m realmente pertenezca a esa clase, la ecuación es (18).

$$exactitud = \frac{VP}{VP + FP} \quad (18)$$

- **Sensibilidad** es el porcentaje de casos pertenecientes a la clase m que fueron correctamente clasificados, la ecuación es (19), note FN refiere solo a la fila de m .

$$sensibilidad = \frac{VP}{VP + FN_m} \quad (19)$$

- **Especificidad** es el porcentaje de los datos NO pertenecientes a la clase m que fueron correctamente clasificados de esa forma, la ecuación es (20).

$$especificidad = \frac{VN + FN_{\neq m}}{VN + FN_{\neq m} + FP} \quad (20)$$

2.9. RESUMEN

Como puede observarse en la sección de Trabajos Relacionados, página 6, las redes *DMNN* están en pleno auge, sus capacidades se encuentran en investigación, aunque sus inicios rondan entre los años 1993 y 2003; por ello el marco teórico aquí presentado fue extraído de unos cuantos trabajos primordiales.

En cuanto a los algoritmos de inicialización, entreno y optimización, se puede decir que hay un amplio abanico bibliográfico, puesto que son métodos de optimización empleados en diversos sistemas y áreas del conocimiento; el presente trabajo hace un acople de estas técnicas matemáticas-computacionales, donde además se relacionan por primera vez la *DMNN* y el algoritmo de enjambre de partículas.

Es notorio también que la *DMNN* tiene una complejidad relativamente baja, como lo muestra su diagrama de bloques en la Figura 8, y además no posee capas ocultas, como ocurre con la mayoría de estructuras de redes neuronales artificiales.

Así mismo los algoritmos de optimización basados en modelos heurísticos (algoritmos evolutivos y la inteligencia de enjambres), presentan también una complejidad matemática baja, aunque se encuentra que es necesario sintonizar los parámetros más adecuados según sea el problema.

En cuanto al algoritmo de gradiente descendente, se ha llegado a una solución aproximada, como lo proponen en el estado del arte; esta se puede decir que es la componente matemáticamente más compleja de calcular para cualquier sistema, pero promete siempre resultados directos, así implique ahondar en un mínimo local.

CAPÍTULO 3

3. IMPLEMENTACIÓN

3.1. INTRODUCCIÓN

En este capítulo se lleva a cabo la programación de la red neuronal junto con sus algoritmos de entrenamiento, así como la realización de una interfaz *GUI* para la facilidad de manejo; todo ello guiado por el protocolo *RUP* que estará presente con el fin de organizar metodológicamente el proyecto; este protocolo de desarrollo de software es uno de los más utilizados en las empresas, divide el proyecto en 4 fases: iniciación, elaboración, construcción y transición; además, cuenta con: requerimientos funcionales y no funcionales, modelo conceptual o metodológico, casos de uso real, diagramas de secuencias, diagrama de clases, modelo de información y pruebas de integración (IBM Software Group, 2003); se fundamenta además en la modularidad y se acompaña del lenguaje gráfico de modelado *UML* para describir las partes que conforman todo el sistema. La totalidad del *RUP* se halla en los **anexos**.

En la Figura 20 se aprecia la totalidad del sistema desarrollado. La finalidad es dar solución a problemas de clasificación con base a patrones de entrenamiento, usando la red neuronal *DMNN*. Esta red requiere una inicialización, la cual se llevará a cabo con dos algoritmos seleccionables, y la fase de entrenamiento, que tiene tres algoritmos para seleccionar; los resultados serán analizados luego con funciones incluidas en la *GUI*.

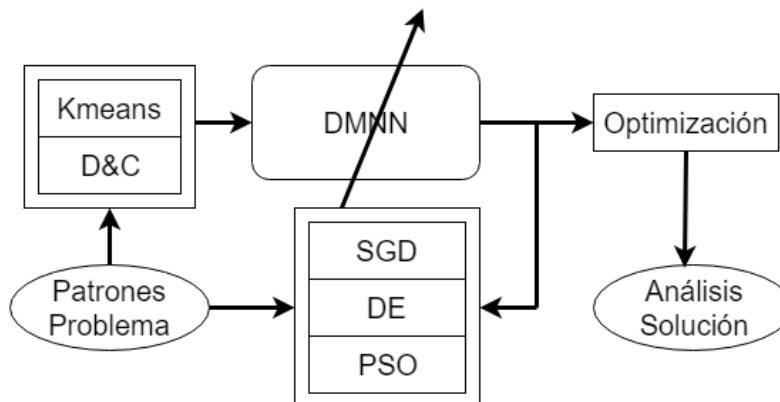


Figura 20 Modelo metodológico del sistema

3.2. REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES

La Tabla 5 y Tabla 6 muestran los requerimientos funcionales y No funcionales respectivamente. Ambas constan de una identificación “Id” del requerimiento, una breve descripción de la funcionalidad y la categoría “Ct” que puede ser E (elemental) u O (opcional); a la vez, se subdividen según el módulo al que pertenezcan. Se entiende que el requerimiento opcional refiere a una funcionalidad que, de no estar presente, no evitaría la ejecución del proyecto.

Tabla 5 Requerimientos funcionales

Id.	DESCRIPCIÓN DE LA FUNCIONALIDAD	Ct.
Módulo Menú, Graficación y Características Generales		
1.0	Permitir navegar desde el menú a las otras interfaces y devolverse.	E
1.1	Permitir seleccionar el tipo de grafica que se va a visualizar en el lienzo.	O
1.2	Graficar en 2D los patrones del problema, según dimensiones x, y dadas.	O
1.3	Graficar en 2D las cajas de las dendritas, según dimensiones x, y dadas.	O
1.4	Graficar en 2D la superficie de decisión como un muestreo de puntos, según dimensiones x, y dadas.	O
1.5	Exportar la red a un archivo especificado por el usuario.	E
1.6	Importar la red desde un archivo especificado por el usuario.	E
1.7	Mostrar cuadro de dialogo con la información del software.	O
1.8	Mostrar guía de usuario en PDF.	O
Módulo Problema		
2.0	Importar los patrones del problema de clasificación desde un archivo especificado por el usuario.	E
2.1	Partir los patrones entre entrenamiento, validación y testeo, con sus correspondientes baches.	E
2.2	Mezclar los patrones al azar y opcionalmente exportarlos a un archivo especificado por el usuario.	E
2.3	Poner un nombre o etiqueta a cada clase de salida.	O
2.4	Adecuar patrones por normalización Min-Max y Z-score.	E
Módulo Inicialización		
3.0	Ejecutar el algoritmo K-medias.	E
3.1	Ejecutar el algoritmo Divide y Conquista (D&C).	E
Módulo Entrenamiento		
4.0	Ejecutar algoritmo SGD.	E
4.1	Ejecutar algoritmo DE.	E
4.2	Ejecutar algoritmo PSO.	E
4.3	Poder pausar o detener la ejecución de los algoritmos.	E
4.4	Graficar en 2D el progreso de los errores, así como una barra de porcentaje de entreno.	E
4.5	Graficar en 2D la posición de los dos primeros pesos sinápticos para cada agente.	O
Módulo Post-entreno		
5.0	Ejecutar la red dándole las entradas y obteniendo la etiqueta de clase.	O
5.1	Ejecutar algoritmo de unión de dendritas.	E
5.2	Ejecutar algoritmo de quitar dendritas sobrantes.	E
5.3	Eliminar dendritas inhibidas o volverlas activas de nuevo.	E
Módulo Análisis		

6.0	Ejecutar el cálculo de las matrices de confusión.	E
6.1	Graficar en 2D las matrices de confusión.	O
6.2	Graficar en 2D la ROC.	O
6.3	Calcular los diferentes tipos de error de la red.	E
6.4	Calcular los diferentes tipos de estadísticas de las matrices de confusión para cada clase.	E
6.5	Exportar las matrices de confusión a un archivo especificado por el usuario.	E
6.6	Exportar salidas para todo el set de datos.	O

Los requerimientos No funcionales refieren a las consideraciones necesarias para el desarrollo del software, pero que no intervienen en su funcionamiento, tales como lo son: el sistema operativo y el hardware, el lenguaje de programación y software de desarrollo, las diferentes librerías (*API's*), entre otras cosas.

Tabla 6 Requerimientos No funcionales

Id.	DESCRIPCIÓN DE LA NO FUNCIONALIDAD	Ct.
1.0	Utiliza lenguaje Python v3.7.2	E
1.1	Utiliza la API de Qt Creator: PyQt5 v5.12.1	E
1.2	Utiliza la API numpy v1.16.1	E
1.3	Utiliza la API playsound v1.2.2	
2.0	Creado en CPU Intel Core i5-7200U con 2.7GHz, 8Gb de RAM, Windows 10, (mínimo 2Gb de RAM).	E

3.3. DISEÑO DE LA GUI Y CONCEPTUALIZACIÓN

El desarrollo del software se llevó a cabo subdividiéndolo, para facilitar la implementación y entendimiento del mismo por parte del usuario, se tienen entonces 6 módulos, que son: Menú, Problema, Inicialización, Entreno, Post-entreno, Análisis. La Figura 21 muestra el diseño conceptual / artístico que supone el punto de partida para la realización de las GUI's.

El diseño de las GUI's surge desde los requerimientos funcionales del proyecto, y se enfoca hacia un diseño intuitivo que favorezca su uso, con una relativamente baja curva de aprendizaje para el usuario final; se entiende además que el modelo metodológico de la Figura 20 Modelo metodológico del sistema, está empaquetada dentro del software y el usuario es el ente externo, así como los datos que él ingresa; por lo tanto dicha ilustración representa el modelo conceptual del proyecto.



Figura 21 Diseño de partida, GUI's conceptuales

3.4. MODELO DE INFORMACIÓN

Los datos del aplicativo se guardarán y abrirán en/con archivos de texto plano TXT o de marcado extensible XML, la información en cuestión refiere a: patrones, redes neuronales, matrices de confusión y resultados masivos.

3.4.1. Formato Patrones

Para guardar en TXT se ve la estructura en la Figura 22, donde “Patrones:”, “Salidas:”, “Entradas:” son campos obligatorios y deben ir en las 3 primeras líneas. El título, las etiquetas y los datos pueden variar o estar ausentes. En cuanto a la matriz de patrones, se tienen las primeras columnas como dimensiones de entrada, y la última como salida deseada (las clases inician contando desde cero).

```
Patrones: TiposDeRanas
Salidas: RanaAgridulce, RanaMaliciosa
Entradas: Longitud(cm), Grosor(cm), Peso(g)
4.5, 6.2, 9.8, 0
4.7, 5.8, 6.9, 1
5.1, 5.9, 7.7, 0
4.2, 6.0, 7.0, 1
```

Figura 22 Formato patrones TXT

Para guardar en XML se ve la estructura en la Figura 23, la información es análoga a la de la Figura 22 pero aquí se especifica el tamaño de los baches y particiones de patrones entre entreno, validación y testeo.

```
<Patrones>
  <Titulo>TiposDeRanas</Titulo>
  <Dimension>
    <Entradas>3</Entradas>
    <Clases>2</Clases>
    <TotalPatrones>4</TotalPatrones>
  </Dimension>
  <NombresSalidas>
    <A0>RanaAgridulce</A0>
    <A1>RanaMaliciosa</A1>
  </NombresSalidas>
  <NombresEntradas>
    <N0>Longitud(cm)</N0>
    <N1>Grosor(cm)</N1>
    <N2>Peso(g)</N2>
  </NombresEntradas>
  <ValoresDatos>
    <P0><E0>4.5</E0><E1>6.2</E1><E2>9.8</E2><S>0</S></P0>
    <P1><E0>4.2</E0><E1>6.0</E1><E2>7.0</E2><S>1</S></P1>
    <P2><E0>5.1</E0><E1>5.9</E1><E2>7.7</E2><S>0</S></P2>
    <P3><E0>4.7</E0><E1>5.8</E1><E2>6.9</E2><S>1</S></P3>
  </ValoresDatos>
</Patrones>

<ParticionPatrones>
  <Entreno>2</Entreno>
  <Validacion>1</Validacion>
  <Testeo>1</Testeo>
</ParticionPatrones>
<ParticionBaches>
  <Entreno>2</Entreno>
  <Validacion>1</Validacion>
</ParticionBaches>
```

Figura 23 Formato patrones XML

3.4.2. Formato Red Neuronal

Para guardar en TXT, la estructura se presenta en la Figura 24, donde “DMNN:” y los demás encabezados textuales son obligatorios, así como sus posiciones en las líneas correspondientes; *NormalizaciónH* son máximos para el algoritmo *Min-Max* y promedios para *Z-score*, mientras *NormalizaciónL* son mínimos para *Min-Max* y desviaciones estándar para *Z-score*; *NormalizaciónN* en cero indica: sin normalización, en -1 indica *Z-score* y >0 es *Min-Max*.

```

DMNN: TiposDeRanas
Dimension: Entradas, Clases
3,2
Pesos
4.575,4.425,6.25,6.15,10.5,9.1,4.275,4.125,6.05,5.95,7.7,6.3
DendritasPorClase
1,1
Activas
1,1
NormalizacionH
1.,1.,1.
NormalizacionL
-1.,-1.,-1.
NormalizacionN: 0.0
NombresSalidas: RanaAgridulce, RanaMaliciosa
NombresEntradas: Longitud(cm), Grosor(cm), Peso(g)

```

Figura 24 Formato red TXT

Para guardar en XML se sigue la estructura de la Figura 25. El vector de pesos w será explicado en la pagina 41.

```

<DMNN>
  <Titulo>TiposDeRanas</Titulo>
  <Dimension>
    <Entradas>3</Entradas>
    <Clases>2</Clases>
  </Dimension>
  <NombresSalidas>
    <A0>RanaAgridulce</A0>
    <A1>RanaMaliciosa</A1>
  </NombresSalidas>
  <NombresEntradas>
    <N0>Longitud(cm)</N0>
    <N1>Grosor(cm)</N1>
    <N2>Peso(g)</N2>
  </NombresEntradas>
  <NormalizacionH>
    <H0>1.0</H0><H1>1.0</H1><H2>1.0</H2>
  </NormalizacionH>
  <NormalizacionL>
    <L0>-1.0</L0><L1>-1.0</L1><L2>-1.0</L2>
  </NormalizacionL>
  <NormalizacionN>0.0</NormalizacionN>
  <Pesos>
    <W0>4.575</W0><W1>4.425</W1><W2>6.25</W2><W3>6.15</W3>
    <W4>10.5</W4><W5>9.1</W5><W6>4.275</W6><W7>4.125</W7>
    <W8>6.05</W8><W9>5.95</W9><W10>7.7</W10><W11>6.3</W11>
  </Pesos>
  <Activas>
    <T0>1</T0><T1>1</T1>
  </Activas>
  <DendritasPorClase>
    <C0>1</C0><C1>1</C1>
  </DendritasPorClase>
</DMNN>

```

Figura 25 Formato red XML

3.4.3. Formato Matriz de Confusión

Para guardar en XML, la estructura se muestra en la Figura 26; además, esta optimizada para ser leída por el software Excel, guarda tres matrices: entreno, validación y testeo; así como el número de dendritas de la red.

```

<dataset>Matrices de Confusion DMNN: TiposDeRanas
  <record>
    <Reales>Entreno</Reales><P0 /><P1 />
  </record><record>
    <Reales>R0: RanaAgridulce</Reales>
    <P0>1</P0><P1>0</P1>
  </record><record>
    <Reales>R1: RanaMaliciosa</Reales>
    <P0>0</P0><P1>1</P1>
  </record><record>
    <Reales>Validacion</Reales><P0 /><P1 />
  </record><record>
    <Reales>R0: RanaAgridulce</Reales>
    <P0>0</P0><P1>1</P1>
  </record><record>
    <Reales>R1: RanaMaliciosa</Reales>
    <P0>0</P0><P1>1</P1>
  </record>
  <Reales>Testeo</Reales><P0 /><P1 />
  </record><record>
    <Reales>R0: RanaAgridulce</Reales>
    <P0>0</P0><P1>0</P1>
  </record><record>
    <Reales>R1: RanaMaliciosa</Reales>
    <P0>0</P0><P1>1</P1>
  </record><record>
    <Reales>Dendritas</Reales><P0 /><P1 />
    <Reales>2</Reales><P0 /><P1 />
  </record>
</dataset>

```

Figura 26 Formato matriz de confusión XML

3.4.4. Formato Resultados

Los resultados en formato TXT se muestran en la Figura 27. Aquí, cada patrón es relacionado con su valor deseado y obtenido después del proceso de aprendizaje.

```
Resultados DMNN: TiposDeRanas
Deseado, Obtenido
0,0
1,1
0,1
1,1
```

Figura 27 Formato resultados TXT

Para guardar en XML se ve la estructura en la Figura 28, optimizada para Excel.

```
<dataset>Resultados DMNN: TiposDeRanas
  <record>
    <Deseado>0</Deseado><Obtenido>0</Obtenido>
  </record>
  <record>
    <Deseado>1</Deseado><Obtenido>1</Obtenido>
  </record>
  <record>
    <Deseado>0</Deseado><Obtenido>1</Obtenido>
  </record>
  <record>
    <Deseado>1</Deseado><Obtenido>1</Obtenido>
  </record>
</dataset>
```

Figura 28 Formato resultados XML

3.4.5. Formato Pesos Sinápticos

Empaquetar la información de los pesos sinápticos w para una rápida operatividad de la red es en particular, lo más importante para la aplicación. Esto se desarrolló ingresando todos los valores flotantes en un vector. En la ecuación (21) se observa el número total de dendritas de la red, dado que cada neurona (clase) puede tener diferente cantidad de estas; entonces el número total de pesos w está dado por: $total_w = 2 \times I \times numD$, dos porque cada característica i tiene dos extremos H y L .

$$numD = \sum_{m=1}^M K_m \quad (21)$$

El ciclo anidado que genera al vector de pesos puede verse de la siguiente manera:

```
pesosW = []
for m in range(M):
    for k in range(Km[m]):
        for i in range(I):
            pesosW.append(WHmki)
            pesosW.append(WLmki)
```

Este vector resultante es necesario para ejecutar la red, hacer operaciones evolutivas diferenciales, aplicar el algoritmo de enjambres, entre otros. La Figura 29 explica gráficamente el vector donde se tienen m neuronas (2-clases para el ejemplo) que son las líneas marrón oscuro y las k dendritas de marrón claro, siendo 3 dendritas $K_m = 3$ para la primera neurona, mientras la segunda tiene $K_m = 2$; las líneas verdes son las características i por lo que el problema es de dimensión 3, luego dado que cada dendrita recibe 3 entradas, y a su vez estas entradas tienen cada una un w^H y w^L que son las líneas azul/azul-verdoso. Para este ejemplo, el vector “pesosW” es de dimensión 30.

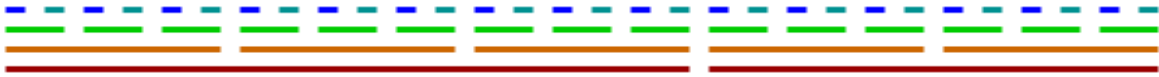


Figura 29 Vector de pesos w

La estructura del vector es además consistente, con la excepción de la variabilidad en el número de dendritas por neurona, cosa que debe tenerse en cuenta al recorrerlo usando la información en K_m , ecuación (21).

3.5. DIAGRAMA DE CLASES

Los 6 módulos *GUI* son en sí mismos clases, tal como se muestra en la Figura 30. “Menú” es la base en la *GUI* donde se abren los demás módulos; cada clase tiene sus respectivas llamadas según las acciones que el usuario ejecute en las mismas. La *GUI* “Entreno” puede instanciarse como *SGD*, *DE*, *PSO* según un valor numérico dado como parámetro; también posee una variable “estado” que se encarga de decir si está en ejecución, pausa o no-iniciado / finalizado.

Finalmente, los hilos se implementaron como clases para evitar que se bloquee el software al procesar mucha información, “Menú” cuenta con un temporizador cíclico de 1s y su señal verifica si las gráficas han cambiado y deben redibujarse, siendo útil para los procesos de entreno donde se desea ver el progreso constante. Los otros hilos refieren a los cálculos hechos para inicializar, entrenar y optimizar respectivamente, por ello, aunque la *GUI* los posee, estos llaman a la clase “Motor”, que posee los métodos y subclases para dichos procesos.

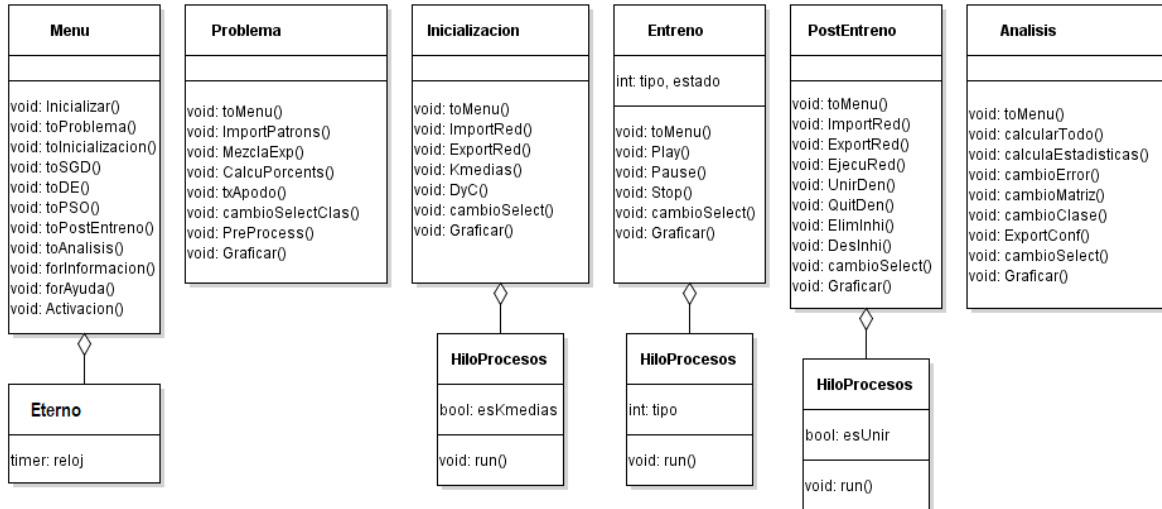


Figura 30 Diagrama de clases para las GUI's

La clase “Motor”, de la Figura 31, ejecuta las tareas más pesadas y guarda los datos de uso general: guarda los patrones del problema y su respectiva distribución entre entreno, validación y testeo, guarda el tamaño de los batches, los textos que describen al problema, la instancia a la red DMNN principal la cual es operada en todas las GUI's; también, instancia otras redes secundarias como lo son la población del algoritmo evolutivo, la información obtenida para las matrices de confusión y, finalmente mantiene los parámetros de entreno que deben sobrevivir tras los diversos manejos, instancias y eliminaciones del hilo.

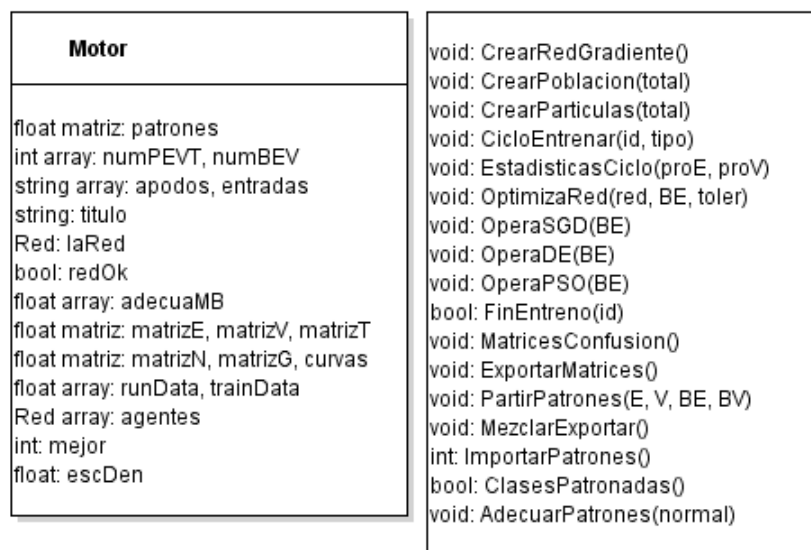


Figura 31 Diagrama de la clase Motor

La clase “Red” (ver Figura 32) guarda los pesos sinápticos de una DMNN y los parámetros necesarios para su utilización; el vector de pesos estaría entonces alojado en la variable “pesW”, “numK” sería un vector con el número de dendritas para cada neurona / clase, y “actK” es el vector del tamaño total de dendritas que además indicaría si cada una de ellas está activa o inhibida (inactiva).

Como puede notarse, los métodos poseen lo necesario para administrar la red, con clases hijas que agregan funcionalidades necesarias para cada tipo de entreno; por ejemplo “LaGradiente” tiene un método llamado “EjecutarSuave” que hace lo mismo que “EjecutarRed” pero en lugar de retornar el valor de salida, lo usa para calcular el gradiente descendente e internamente lo almacena en el vector “impU”. La red “Individuo” tiene las operaciones evolutivas diferenciales para crear a su descendencia, guardando los pesos en el vector “otrW”, y la red “Partícula” tiene el método que calcula un paso de la simulación física, utilizando las variables “velW” para guardar la velocidad y “besW” para guardar su mejor posición obtenida.

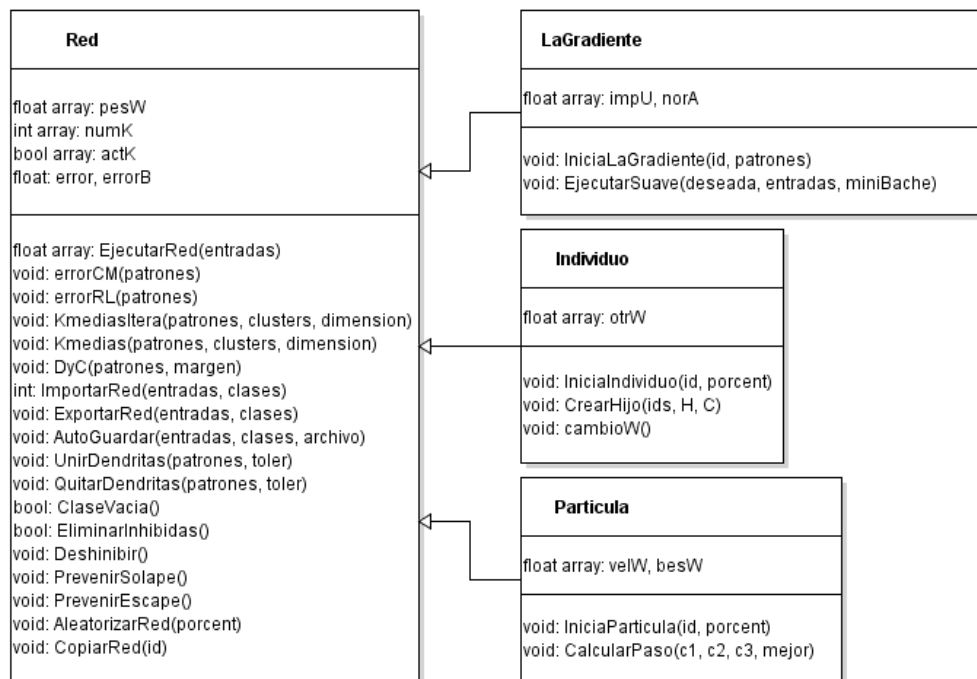


Figura 32 Diagrama de clases de la red DMNN

Finalmente, la Figura 33 muestra la relación entre todas las clases expuestas, se observa que “Motor” administra a las redes neuronales, las GUI’s secundarias están todas relacionadas con “Motor” y “Menú”.

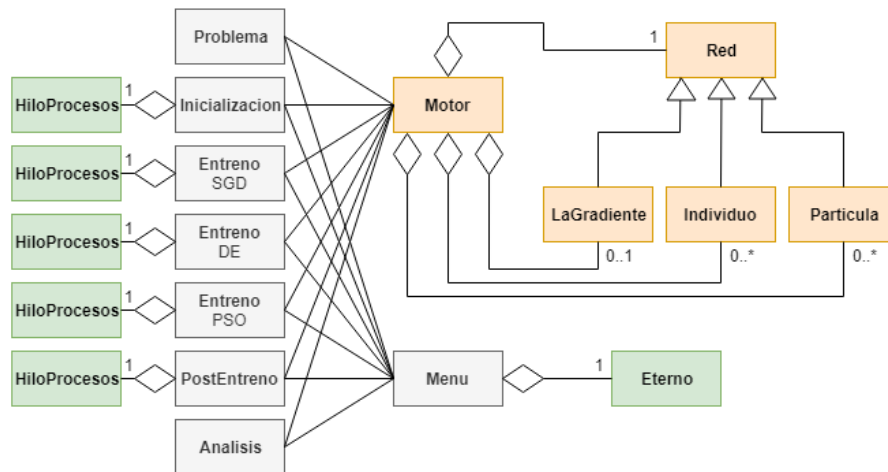


Figura 33 Diagrama de Clases

3.6. FUNCIONALIDAD DE LOS MÓDULOS

3.6.1. Menú

Se encarga de presentar el software y navegar a través de las *GUI*'s. En la Figura 34 se observa el flujo, donde se puede llegar a “Post-Entreno” y “Análisis” ya que los mecanismos de inicialización también funcionan como métodos de entrenamiento para una red DMNN.

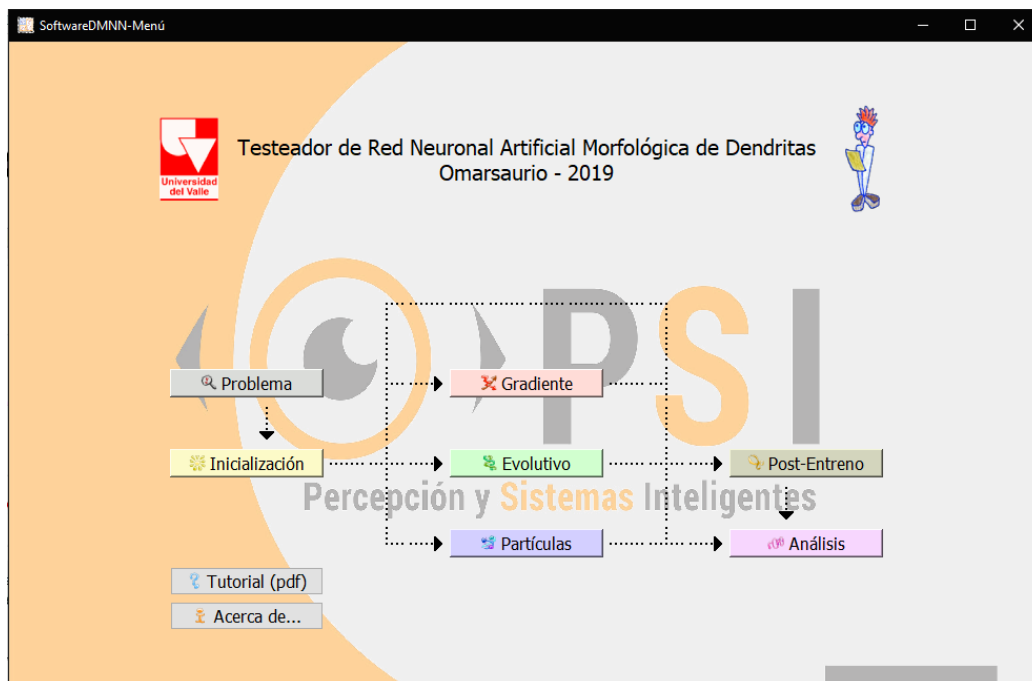


Figura 34 GUI Menú

En la Tabla 7 se ven las condiciones de disponibilidad.

Tabla 7 Desbloqueo de los módulos

Estado	Módulos Disponibles	Significado o Descripción
Código 1	Problema	No hay matriz de patrones cargada.
Código 2	Problema, Inicialización	Ya hay patrones, pero está pendiente a crearse o importarse una red.
Código 3	Todos	Hay tanto patrones como red verificada, puede procederse sin restricción.

Para este módulo se omiten los diagramas de flujo, puesto que su funcionamiento es redirigir al usuario a otra GUI al pulsar un botón, esto sólo si el botón está activo. Los dos botones en la parte inferior izquierda son para:

- Mostrar un archivo PDF de ayuda, que guíe al usuario en el manejo.
- Mostrar la información del software y su creador en una ventana emergente.

3.6.2. Problema

Este módulo *GUI* es el que administra los patrones que definen al problema (de ahí su nombre), como se ve en la Figura 35; los botones que afectan a los patrones son además deshabilitados hasta que no se hayan importado correctamente un set de estos; adicionalmente, se puede cambiar la etiqueta a cada clase.

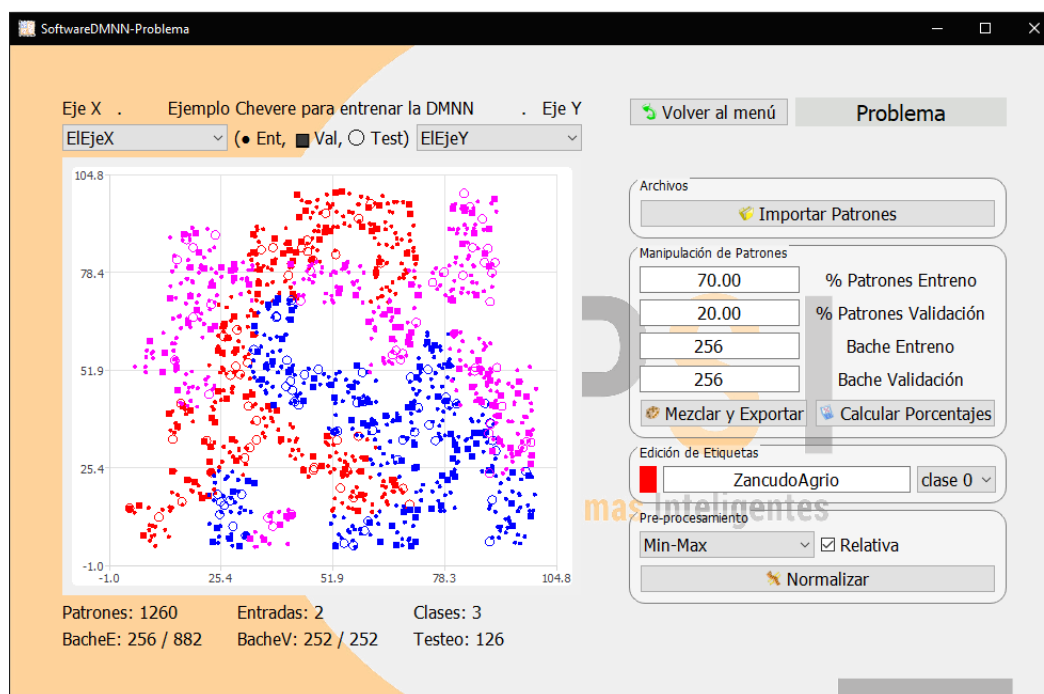


Figura 35 GUI llamada Problema

A continuación, se pueden ver los diagramas de secuencias para cada botón, en Figura 36 y Figura 37.

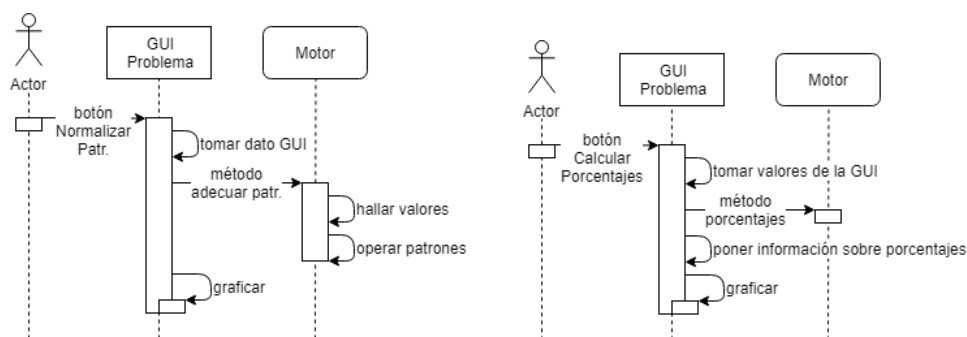


Figura 36 Diagramas de secuencia para: Normalizar Patrones / Calcular porcentajes

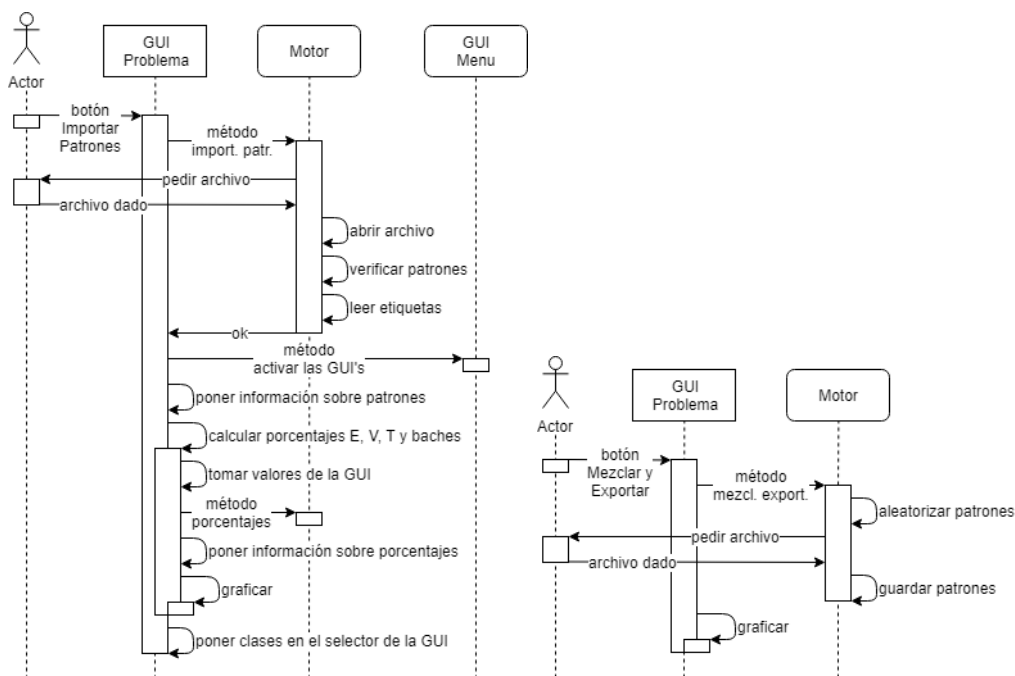


Figura 37 Diagramas de secuencia para: Importar Patrones / Mezclar y Exportar Patrones

3.6.3. Inicialización

Es donde aparece la red neuronal como tal, ya sea mediante uno de los algoritmos de inicialización o importando una red creada previamente (TXT o XML); como se ve en la Figura 38. Aquí, también se grafican los patrones de entrenamiento (de hecho, se grafican en todas las GUI's) y además se añaden las cajas (dendritas 2D) que conforman a la red neuronal y la superficie de decisión; para ello se dispone de un selector sobre la gráfica, que la redibujará de acuerdo a los requerimientos del usuario.

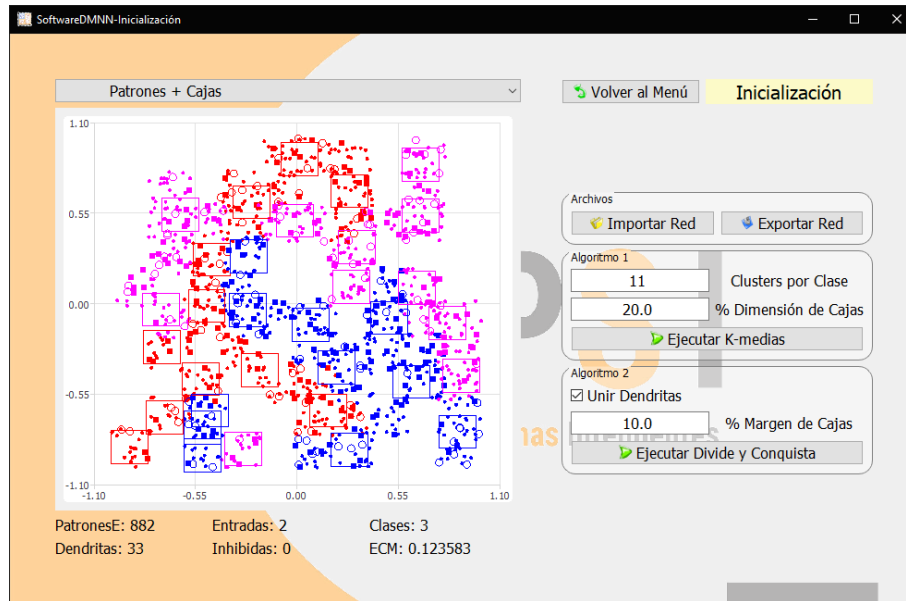


Figura 38 GUI Inicialización

Los diagramas de secuencias para cada botón, se presentan en la Figura 39 y Figura 40; el algoritmo de Divide y Conquista (D&C) puede verse en la Figura 11 y el de *Kmedias* en la Figura 9 (presentados en el Capítulo 2 – pág. 17); note que el botón de D&C tendría un diagrama de secuencia equivalente al de *Kmedias*.

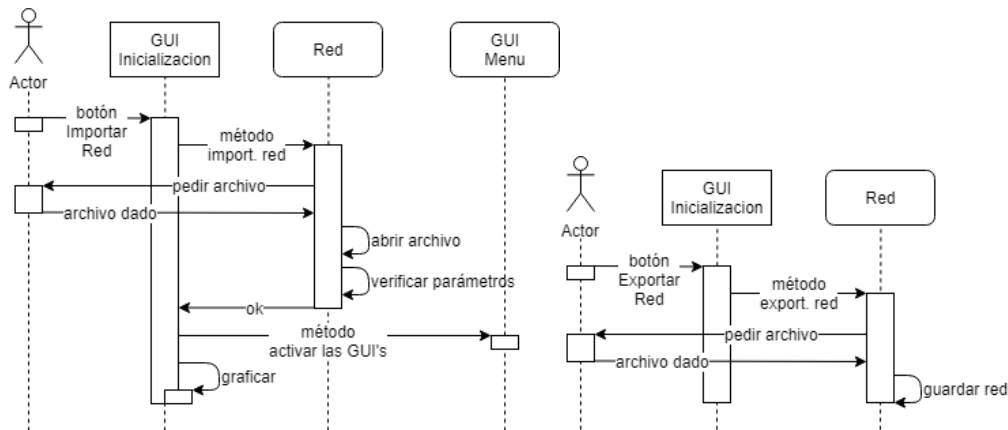


Figura 39 Diagramas de secuencia para: Importar Red / Exportar Red

Para *K-medias* se agregó la opción de *auto-tuning* al número de clústers, poniendo su valor en cero, esto funciona bajo el diagrama que se muestra en la Figura 41; el procedimiento aumenta iterativamente el número de dendritas hasta que la diferencia entre estos errores alcanza errores por debajo de un umbral definido por el usuario (0.01, es el valor por defecto), aunque esto implica tiempos de procesamiento significativos, en la medida que más dendritas deban ser generadas

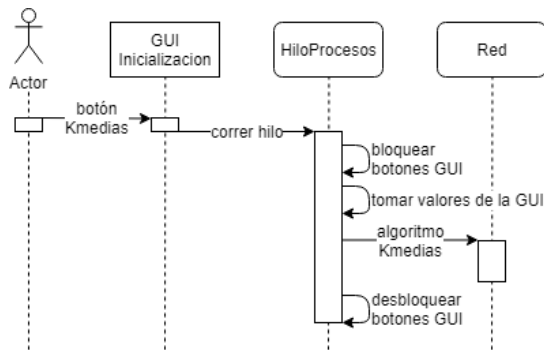


Figura 40 (izquierda) Diagrama de secuencia para inicializar por Kmedias

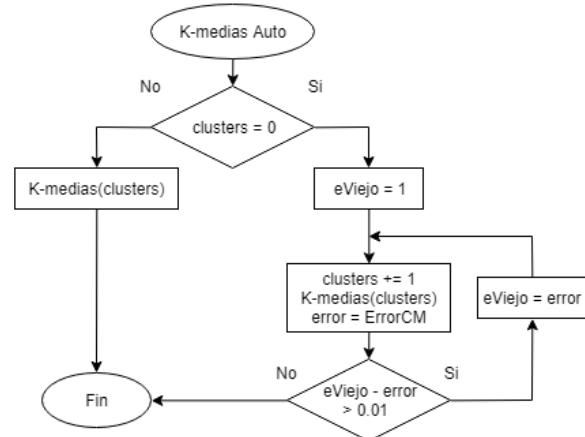


Figura 41 (derecha) Diagrama K-medias auto tuneado

3.6.4. Post-Entreno

A esta GUI se puede llegar directamente desde la inicialización, habiendo importado una red o creado una con K-medias o D&C, como se aprecia en la Figura 42, tiene las mismas opciones de importar y exportar que la GUI de inicialización.

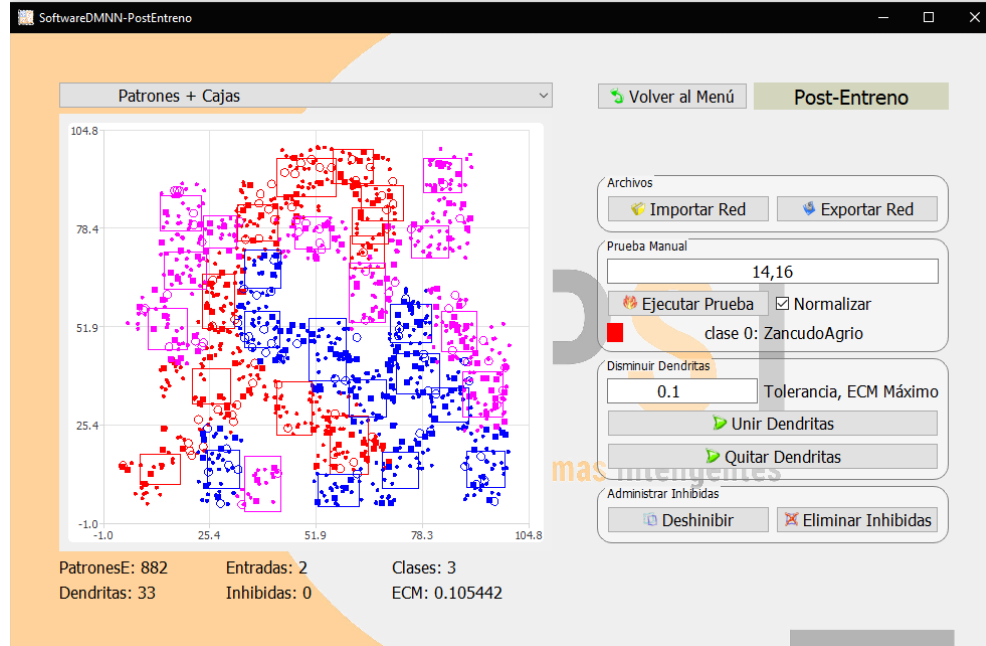


Figura 42 GUI Post-Entreno

Los botones de desinibir y eliminar inhibidas simplemente ponen el vector de booleanos de las dendritas en verdadero o eliminan los pesos que correspondan a falso;

en cuanto a ejecutar red, el usuario ingresa los valores de entrada para ver la salida. Los diagramas de secuencia se presentan en las Figura 43, Figura 44 y Figura 45.

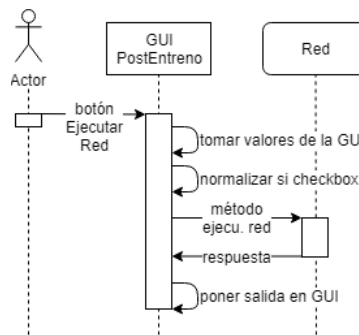


Figura 43 Diagrama de secuencia para: Ejecutar Red

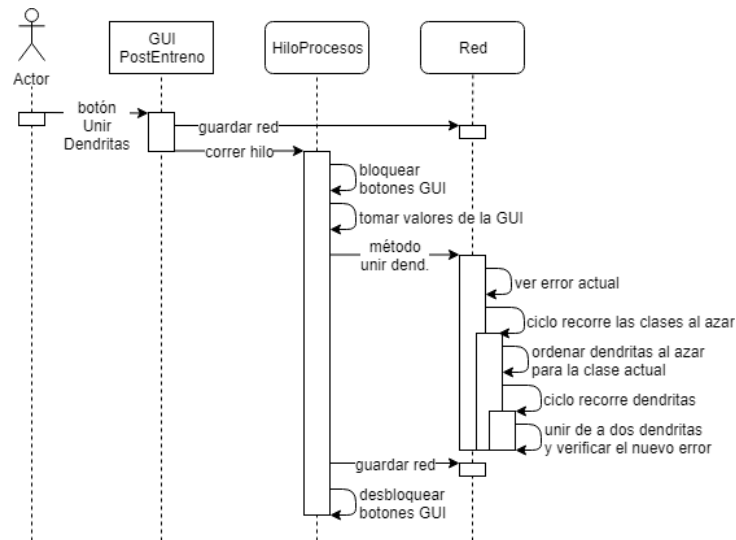


Figura 44 Diagrama de secuencia para: Unir Dendritas

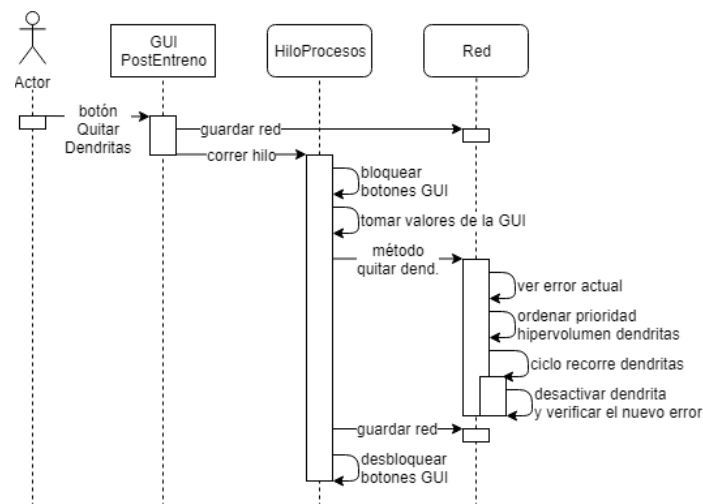


Figura 45 Diagrama de secuencia para: Quitar Dendritas

3.6.5. Entreno

Esta *GUI* tiene un parámetro que selecciona *SGD*, *DE* o *PSO*; además de otro parámetro que indica si está en ejecución, pausa o no-iniciado/finalizado (Figura 46).

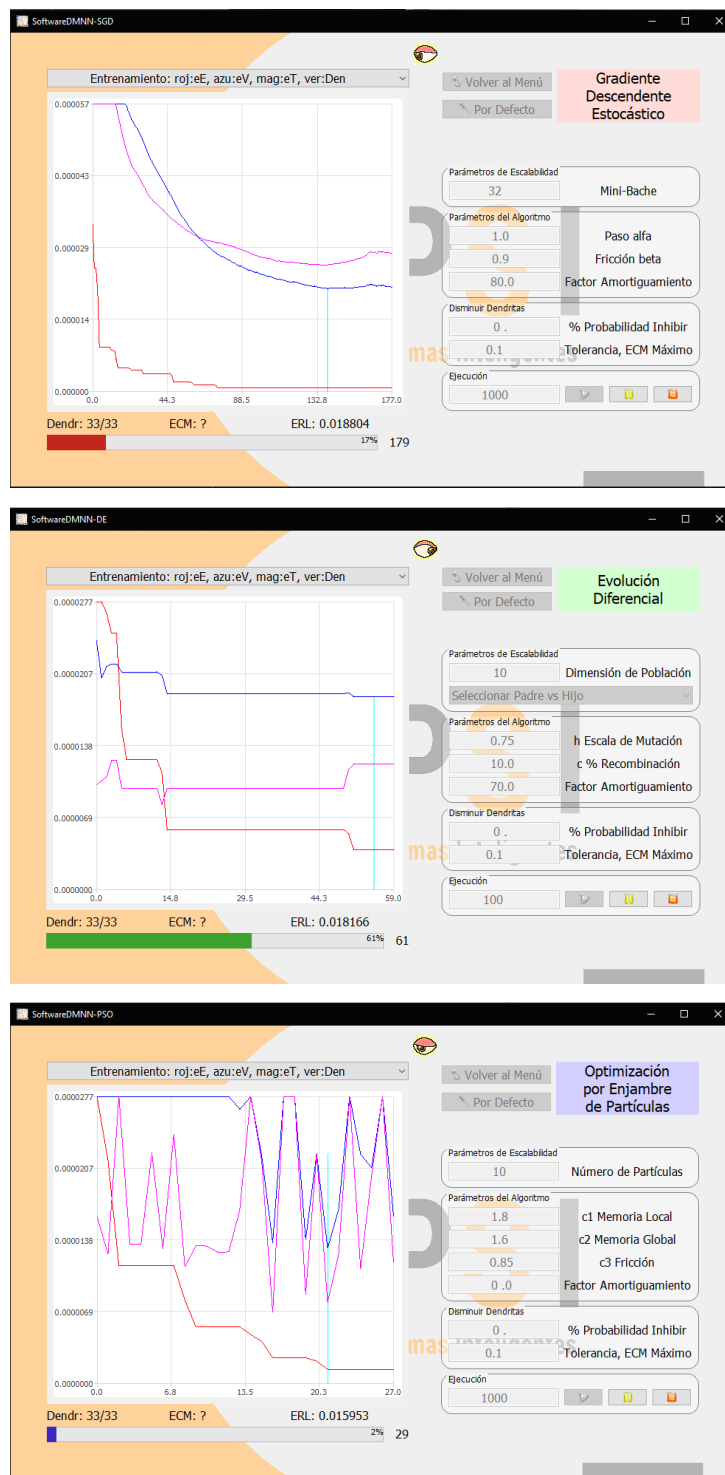


Figura 46 GUI's Entreno

En el panel de Ejecución hay tres botones para poner en marcha el entreno; la Figura 47 muestra la secuencia, donde se llama a una clase hilo para no bloquear, y los botones Pausa y Stop solo cambian la variable de estado.

Los algoritmos de modificación de los pesos son obtenidos de la Figura 19, Figura 17 y ecuación (14) en (13) para *PSO*, *DE* y *SGD* respectivamente (páginas 28, 25 y 22); cabe aclarar que durante el entreno se utiliza el error de regresión logística.

Automáticamente se hace un guardado de la red antes y después del entreno, igual como sucede con los comandos Unir y Quitar dendritas de la GUI Post-Entreno.

La gráfica y la barra de porcentaje son actualizadas cada 1s.

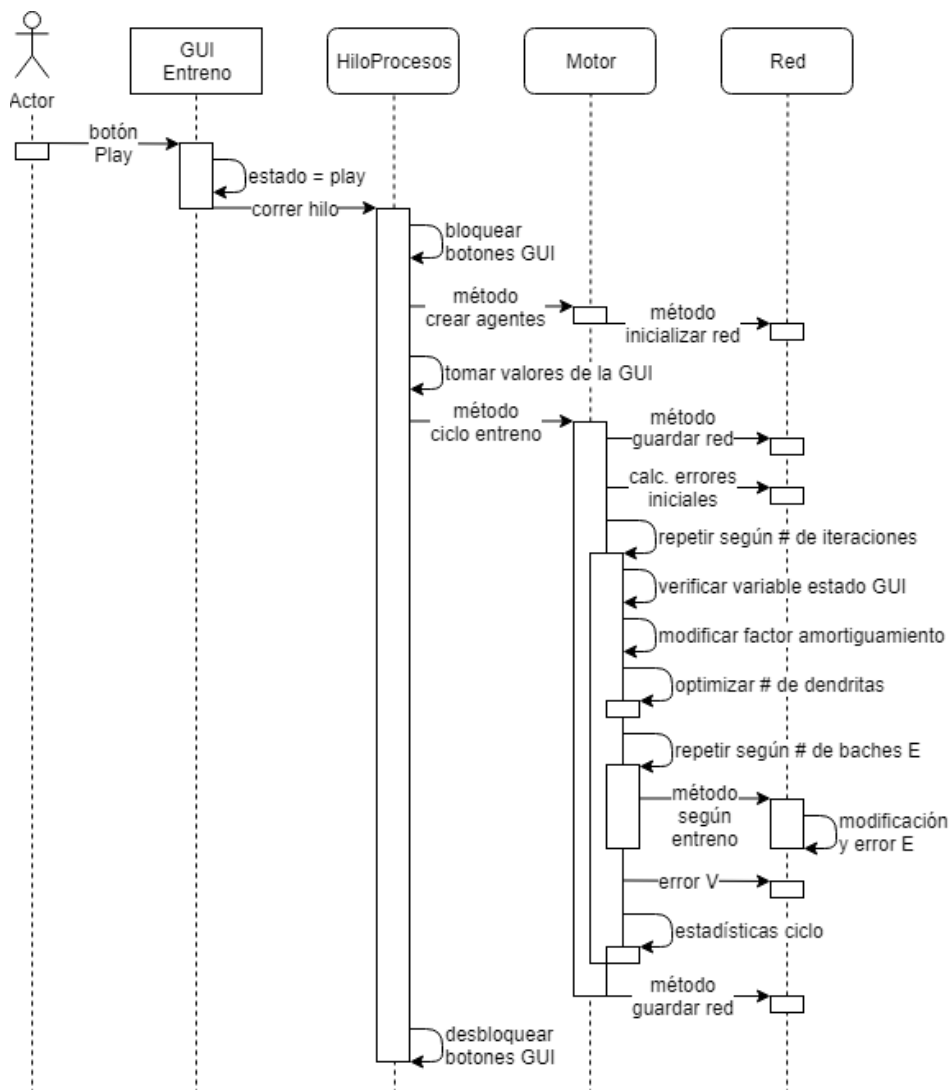


Figura 47 Diagrama de secuencia para Entrenamiento (botón Play)

La ecuación (22) genera un parámetro $0.1 \leq a \leq 1$, que decrece como lo muestra la Figura 48, esto hace que el entrenamiento pase de la exploración a la explotación, el parámetro “a” es un factor de amortiguamiento, controlado por f que es seleccionable desde la *GUI*, este se multiplicará por α , h o $c3$ según el algoritmo de optimización *SGD*, *DE*, *PSO* respectivamente.

$$a = 0.1 + 0.9. \exp \left(\frac{-\text{épocas}^2}{f_{\text{factor}} \cdot 2 \cdot \text{total_épocas}^2} \right) \quad (22)$$

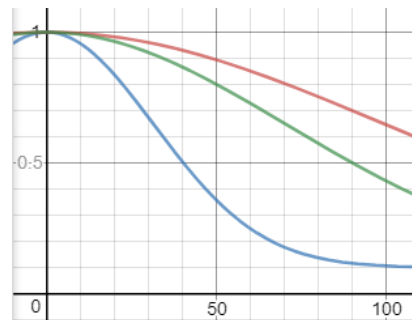


Figura 48 Campana de Gauss

3.6.6. Análisis

Este módulo *GUI* se encarga de arrojar resultados estadísticos referentes al entreno y creación de la DMNN; la Figura 49 señala que posee un botón para exportar las matrices en XML. la *GUI* automáticamente calcula las matrices de confusión y los datos estadísticos, dejando en manos del usuario el seleccionar lo que desea ver.

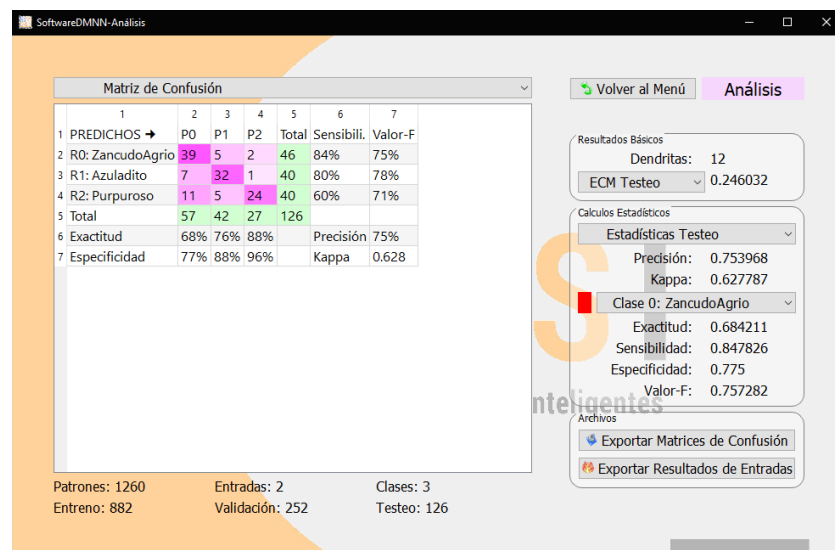


Figura 49 GUI Análisis

El diagrama de secuencia de la exportación es omitido ya que solo pide una ubicación de archivo para guardar las matrices. El primer selector muestra el error cuadrático medio o el de regresión logística para patrones de entrenamiento, validación, testeo, No-entrenamiento o Generales. De forma similar el segundo selector muestra la tabla, la *ROC* y estadísticas para dichos 5 tipos de patrones; el tercer selector, finalmente refiere a clases específicas, cuyas 4 estadísticas son mostradas debajo de él.

La Figura 50 muestra la región de convergencia (*ROC*) para 3 clases; esto se ha agregado al software para dotarlo de más herramientas, pero se aclara que la métrica no evaluará los resultados de la pregunta problema.

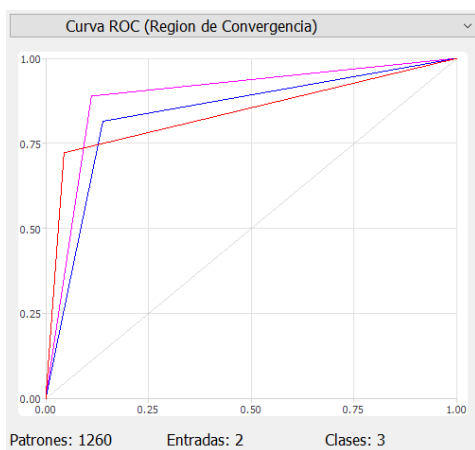


Figura 50 Ejemplo ROC

Adicionalmente, el botón inferior (Figura 49) permite exportar en archivo TXT o XML el listado de salidas deseadas vs obtenidas para todo el set de datos; esto es útil si se tiene un set de datos para indagar por las salidas de la red, no necesariamente debe tener salidas deseadas. Por lo tanto, se puede indagar sobre un nuevo set (fila por fila), a diferencia de la matriz de confusión que empaqueta todas las salidas en pocas celdas, con este método podrá conocer la salida específica para una entrada dada; en otras palabras, es la versión masiva de la funcionalidad “Prueba” de la *GUI* “Post-Entreno”.

3.7.OBSERVACIONES DEL APLICATIVO SOFTWARE

Llevar a cabo el desarrollo mediante el protocolo *RUP* garantizó un adecuado flujo de trabajo. los algoritmos explorados durante la etapa del marco teórico fueron convertidos en bloques, que el usuario de la aplicación podrá operar siguiendo un flujo lógico.

El lenguaje de programación *Python* ha demostrado su versatilidad y facilidad de uso junto con la *API PyQt*, que brinda las herramientas de *GUI* de *Qt Creator* con lo que se desarrolló la interfaz de usuario del aplicativo, no se requirió el *IDE* de *Qt Creator*. Por otra parte, la *API numpy* fue indispensable, pues brinda las operaciones matemáticas matriciales con la velocidad de ejecución apropiadas para desarrollos científicos.

Adicionalmente, el uso de formatos de importación y exportación *TXT* y *XML*, brinda eficiencia al permitir la fácil legibilidad y edición del primero, y la robustez computacional del segundo. Finalmente, la aplicación (*EXE*) ocupa aproximadamente 60Mb, y su código será liberado al público durante el presente año.

CAPÍTULO 4

4. ANÁLISIS DE RESULTADOS

4.1. INTRODUCCIÓN

En este capítulo se pone en marcha el software creado, resolviendo en primer lugar problemas de clasificación de patrones. Estos *datasets* provienen de bases de datos libres que han sido ampliamente usados en entrenamiento de sistemas clasificadores; también se usaron dos *datasets* artificiales y uno proveniente de un problema real planteado. Posteriormente, se expondrán los resultados del test de usabilidad, y se mostrará la prueba de integración resumida.

4.2. RESULTADOS DATASETS

Los *datasets* provenientes de Internet son: “Iris”, “Wine”, “Glass”, “Mammographic Mass” y “Letter Recognition” (Dua19), luego los *datasets* “Espiral2” y “Ejemplito” han sido creados artificialmente para probar la red, el primero es la doble espiral cuya complejidad es altamente reconocida en el ambiente científico, y su solución se aprecia en la Figura 2 (página 9), mientras el segundo es una distribución irregular bidimensional, usada para ensayar durante la elaboración del software. Por último, se generó un *dataset* de un juego llamado “Payaso Lander”. La Tabla 8 muestra la información referente a cada *dataset*.

En la Tabla 9 se aprecia para cada *dataset*, una solución general a la cual se llegó, la finalidad de esta tabla es mostrar la capacidad de la red DMNN. Los valores resaltados en naranja son mayores al 80%, los rojos mayores al 90%.

Tabla 8 Datasets para testear la red DMNN

Dataset	Total	Dimensión	Clases
Iris	150	4	3
Wine	178	13	3
Glass	214	9	6
Mammographic Mass	961	5	2
Letter Recognition	20000	16	26
Espiral2	2028	2	2
Ejemplito	1260	2	3
PayasoLander	2032	5	6

Tanto para esta como otras tablas, “den”, “preciE” y “preciT” refieren al número de dendritas y las precisiones de entreno y prueba (*test*) respectivamente.

Tabla 9 Óptima solución hallada

Dataset	#den	%preciE	%preciT
Iris	5	98.33	96.67
Wine	5	88.03	80.56
Glass	23	95.32	60.47
Mammographic Mass	10	78.31	71.72
Letter Recognition	254	91.81	82.98
Espiral2	40	99.82	97.78
Ejemplito	39	97.02	90.48
PayasoLander	68	40.10	35.29

En la Tabla 10 se tienen los resultados de inicializar la red con el K-medias automático, con cajas del 10%, y el Divide y Conquista con margen fija en 10%; el K-medias puede variar mucho entre una ejecución y otra, por lo que se probó 3 veces y se conservó la mejor; además, se implementó la auto-sintonización para tener una aproximación global de la eficiencia. Los valores resaltados en naranja son la menor cantidad de dendritas usadas para el *dataset*, luego los rojos son el mayor porcentaje de precisión de entreno y en azul el mayor porcentaje de precisión en la prueba. Para “*Letter Recognition*”, D&C tomó mucho tiempo (3 días) sin arrojar solución.

Tabla 10 Comparación algoritmos de inicialización

Dataset	K-medias			D&C		
	#den	%preciE	%preciT	#den	%preciE	%preciT
Iris	12	96.67	93.33	12	97.50	96.67
Wine	31	73.94	72.22	24	96.48	77.78
Glass	40	78.82	63.64	44	84.73	63.64
Mammographic Mass	21	71.76	68.04	96	84.84	73.20
Letter Recognition	299	75.83	71.69
Espiral2	40	94.64	94.09	57	98.83	97.29
Ejemplito	45	91.37	88.49	60	94.94	86.51
PayasoLander	70	34.03	32.35	436	52.84	24.02

En la Figura 51, se ejecutó nuevamente el algoritmo de K-medias, con el problema “*Mammographic Mass*”, en este caso el Valor-F de la matriz de confusión, dice que

las mediciones referentes a masas benignas son más fiables que las de masas malignas, en un 3% de diferencia para dicha medida; aunque la sensibilidad de ambas es un 75%, lo que significa que esa es la capacidad de selección de los verdaderos, la exactitud difiere; siendo más fiable (en un 6%), el resultado benigno respecto al maligno; se eligió mostrar la matriz de confusión de este problema, por ser uno de los que denotó (Tabla 10) más dificultad; demostrando aun así las ventajas de la inicialización no aleatoria.

Mamografía						
1	2	3	4	5	6	
1 PREDICHOS →	P0	P1	Total	Sensibili.	Valor-F	
2 R0: Benign	391	125	516	75%	77%	
3 R1: Malignant	108	337	445	75%	74%	
4 Total	499	462	961			
5 Exactitud	78%	72%		Precisión	75%	
6 Especificidad	75%	75%		Kappa	0.514	

Figura 51 Matriz de confusión para el problema Mammographic Mass

La Tabla 11 muestra los resultados obtenidos de inicializar con *K-medias* cada *dataset*, y posteriormente aplicarle cada uno de los tres algoritmos de entrenamiento paralelamente; por lo que los tres algoritmos han hecho cambios sobre la misma red *DMNN* base. Para esta prueba se escogieron parámetros de inicialización que no llevaran al sistema clasificador a un error muy bajo de partida, y no se disminuyeron dendritas durante el entreno. Los valores en naranja son mejoras respecto a *K-medias*, en rojo el mejor resultado de entrenamiento y en azul el de prueba.

Tabla 11 Comparación algoritmos de entreno

Dataset	K-medias			SGD		DE		PSO	
	#den	%preciE	%preciT	%preciE	%preciT	%preciE	%preciT	%preciE	%preciT
Iris	9	95.00	96.67	98.33	96.67	97.50	96.67	96.67	100.0
Wine	41	85.21	83.33	86.62	83.33	85.92	83.33	86.62	83.33
Glass	29	71.92	36.36	95.57	63.64	72.91	45.45	73.40	45.45
Mammogra. Mass	10	73.96	72.16	73.96	72.16	73.73	73.20	76.39	77.32
Letter Recognition	130	64.15	61.97	83.80	75.66	64.09	61.76	64.00	62.03
Espiral2	40	95.50	93.60	97.35	94.83	95.62	94.09	95.56	94.83
Ejemplito	15	79.76	77.78	89.38	84.52	80.26	78.17	81.05	78.57
PayasoLander	106	36.54	29.90	40.54	29.90	36.65	30.39	37.25	29.90

Para hacer el análisis correspondiente, referente a las métricas implicadas en el estudio (matriz de confusión), se ejecutó nuevamente el entrenamiento *SGD* y *PSO* para el problema Iris, Figura 52, por ser este muy común en la literatura y tener 3 clases, de las cuales una es linealmente separable y las otras estar entrelazadas.

iris SGD							
	1	2	3	4	5	6	7
1 PREDICHOS →	P0	P1	P2	Total	Sensibili.	Valor-F	
2 R0: Setosa	41	0	0	41	100%	100%	
3 R1: Versicolor	0	36	1	37	97%	97%	
4 R2: Virginica	0	1	41	42	97%	97%	
5 Total	41	37	42	120			
6 Exactitud	100%	97%	97%		Precisión	98%	
7 Especificidad	100%	98%	98%		Kappa	0.975	

iris PSO							
	1	2	3	4	5	6	7
1 PREDICHOS →	P0	P1	P2	Total	Sensibili.	Valor-F	
2 R0: Setosa	50	0	0	50	100%	100%	
3 R1: Versicolor	0	48	2	50	96%	96%	
4 R2: Virginica	0	2	48	50	96%	96%	
5 Total	50	50	50	150			
6 Exactitud	100%	96%	96%		Precisión	97%	
7 Especificidad	100%	98%	98%		Kappa	0.96	

Figura 52 Matrices de confusión para el problema Iris

Puede verse entonces como para dicho problema, los datos estadísticos de la matriz arrojan valores cercanos a 1, lo que implica casi un 100% para cada concepto; en el caso de la “Setosa”, la clasificación es, según las métricas infalible, cosa que era esperada según el estado del arte para este problema; en cuanto a las otras dos plantas, “Versicolor” y “Virginica”, cabe resaltar que las medidas de exactitud y sensibilidad se encuentran ambas cercanas a 1, por lo que el sistema es robusto.

La Figura 53 muestra las matrices de confusión para el problema “Glass”, donde según lo visto en la Tabla 11, fue más dificultoso para los algoritmos heurísticos; aquí se puede notar como *SGD* logró mediciones estadísticas, no solo mayores sino además similares; por su parte *PSO* tuvo buena clasificación para el vidrio tipo 4 y 6, pero pésima para los demás.

Glass SGD

	1	2	3	4	5	6	7	8	9	10
1	PREDICHOS →	P0	P1	P2	P3	P4	P5	Total	Sensibili.	Valor-F
2	R0: Tipo1	66	4	0	0	0	0	70	94%	94%
3	R1: Tipo2	2	71	1	1	1	0	76	93%	92%
4	R2: Tipo3	0	2	15	0	0	0	17	88%	90%
5	R3: Tipo4	1	0	0	12	0	0	13	92%	92%
6	R4: Tipo5	1	0	0	0	8	0	9	88%	88%
7	R5: Tipo6	0	0	0	0	0	29	29	100%	100%
8	Total	70	77	16	13	9	29	214		
9	Exactitud	94%	92%	93%	92%	88%	100%		Precisión	93%
10	Especificidad	97%	95%	99%	99%	99%	100%		Kappa	0.917

Glass PSO

	1	2	3	4	5	6	7	8	9	10
1	PREDICHOS →	P0	P1	P2	P3	P4	P5	Total	Sensibili.	Valor-F
2	R0: Tipo1	54	8	8	0	0	0	70	77%	71%
3	R1: Tipo2	17	48	9	0	2	0	76	63%	71%
4	R2: Tipo3	7	1	9	0	0	0	17	52%	40%
5	R3: Tipo4	0	0	1	11	0	1	13	84%	91%
6	R4: Tipo5	2	1	0	0	5	1	9	55%	62%
7	R5: Tipo6	2	0	0	0	0	27	29	93%	93%
8	Total	82	58	27	11	7	29	214		
9	Exactitud	65%	82%	33%	100%	71%	93%		Precisión	71%
10	Especificidad	80%	92%	90%	100%	99%	98%		Kappa	0.624

Figura 53 Matrices de confusión para el problema Glass

4.3. PROBLEMA “PAYASOLANDER”

Se escogió un problema real para probar las capacidades de la red, este se trata de un videojuego, que está inspirado en el clásico *Space-Lander*, donde se pone a un agente con influencias de cálculos físicos a aterrizar en un sitio designado.

En la Figura 54 se puede observar la dinámica del agente, donde las componentes del viento como disturbio fueron puestas a cero. Los comandos del juego son: inclinar a izquierda o derecha e impulsar en la dirección de inclinación; esto supone 6 clases de salida para la red: *GiroD*, *Nulo*, *GiroI*, *ImpGiroD*, *Impulso*, *ImpGiroI*; y las 5 entradas serían: *PosiciónX*, *PosiciónY*, *VelocidadX*, *VelocidadY*, *Giro* (inclinación).


```

// físicas giro
ag = cg * m_giro
vg = limitar(vg * m_fg + ag * dt, -m_vg, m_vg)
pg = limitar(pg + vg * dt, -m_pg, m_pg)
// físicas movimiento
ax = -sen(pg) * cp * m_propulsion
ay = m_gravedad - cos(pg) * cp * m_propulsion
vx = limitar(vx * m_fv + vv + ax * dt, -m_vxy, m_vxy)
vy = limitar(vy * m_fv + ay * dt, -m_vxy, m_vxy)
px = limitar(px + vx * dt, -m_px, m_px)
py = limitar(py + vy * dt, -m_py, 0)

```




Figura 54 Físicas PayasoLanderANN

Los resultados obtenidos se aprecian en la Tabla 12. Estos están ordenados según el éxito en el aterrizaje, esta tabla es una recopilación de las mejores redes creadas para resolver el problema (aproximadamente 36), incluyendo aquellas entrenadas mediante los tres algoritmos ya mencionados, y optimizadas mediante unión y eliminación de dendritas.

“Maniobra” es un contador que suma a los éxitos, las fallas que estuvieron cerca de cumplir el objetivo, por ejemplo, caer próximo a la plataforma, con mucha velocidad sobre la misma o en un ángulo inadecuado.

Tabla 12 Resultados 30 vuelos PayasoLander

Algoritmos	#den	%preciE	%preciT	%Éxitos	%Maniobra
K-medias	94	34.20	32.32	36.67	70.00
K-medias + DE	106	36.65	30.39	26.67	56.67
K-medias	106	36.54	29.90	23.33	63.33
K-medias	208	42.56	28.92	20.00	80.00
D&C	834	87.31	21.08	10.00	36.67

La matriz de confusión (ver Figura 55), corresponde a la mejor red entrenada, muestra datos estadísticos para todo el set de datos, se nota en primera instancia la ausencia de la diagonal mayor (como en la Figura 51), de ahí que la precisión y el factor Kappa sean muy bajos, 29% y 0.089 respectivamente. Este problema es simétrico, y combina 3 clases (giro derecha, No giro, giro izquierda) con 2 clases (No impulso, impulso), de ahí se obtienen las dos matrices de la derecha, entonces, la precisión verdadera es del 51% para control direccional y 56% para impulsos. Luego el sistema es mejor detectando “en qué momento No debe girar”.

	1	2	3	4	5	6	7	8	9	10
1	PREDICHOS →	P0	P1	P2	P3	P4	P5	Total	Sensibili.	Valor-F
2	R0: GiroD	18	82	4	39	31	4	178	10%	12%
3	R1: Nula	27	237	60	53	170	61	608	38%	36%
4	R2: Girol	3	72	39	4	44	36	198	19%	21%
5	R3: ImpGiroD	25	79	8	53	61	9	235	22%	23%
6	R4: Impulso	35	155	36	55	197	76	554	35%	34%
7	R5: ImpGirol	12	74	25	21	79	48	259	18%	19%
8	Total	120	699	172	225	582	234	2032		
9	Exactitud	15%	33%	22%	23%	33%	20%		Precisión	29%
10	Especificidad	94%	67%	92%	90%	73%	89%		Kappa	0.089

real.	GD	N	GI		sens.
rGD	135	253	25	413	33%
rN	170	759	233	1162	65%
rGI	40	269	148	457	32%
	345	1281	406	2032	
exac.	39%	59%	36%	preci.	51%

real.	N	I		sens.
rN	542	442	984	55%
rl	449	599	1048	57%
	991	1041	2032	
exac.	55%	58%	preci.	56%

Figura 55 Matriz de confusión para PayasoLander

En los **anexos** se encuentra el ejecutable, el pseudocódigo del videojuego y la tabla ampliada, con varias redes puestas a prueba.

4.4. PRUEBA DE USABILIDAD

Para medir la aceptación del software por parte de un nuevo usuario, se realizó una prueba donde se le pide a la persona, que realice una secuencia de tareas como la realizada en la prueba de integración, tras lo cual responde una encuesta. Los resultados se presentan en la Figura 56.

De un total de 24 encuestados, 20 son del curso de redes neuronales artificiales de la Universidad del Valle, por lo que son jóvenes que comienzan a aprender sobre dicho tema, con la edad y género promedios de la población universitaria; uno de los directores de este trabajo (Wilfredo) realizó la prueba, quien claramente es alguien inmerso en el área; finalmente los otros 3 son jóvenes universitarios ajenos al tema.

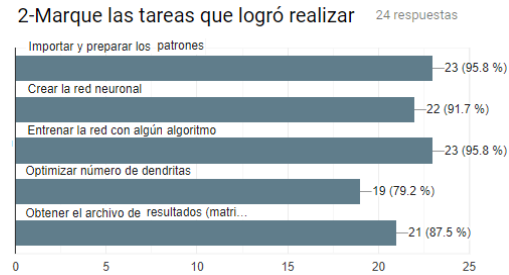
Para el caso de los 20 estudiantes de Ing. Electrónica, la prueba y encuesta fueron realizadas durante el horario de una clase, con presencia del ejecutor de este trabajo y del director Wilfredo, quienes dieron una clase corta sobre el tema y atendieron a estudiantes, que llegasen a encontrarse muy desubicados con la aplicación.

El cuestionario contaba con un apartado sobre opiniones amplias, estas han sido integradas a los puntos a tener en cuenta para trabajo futuro (página 72).

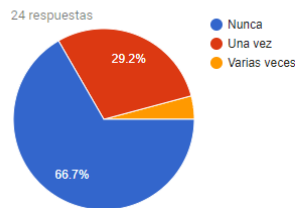
1-¿Nivel de conocimiento sobre problemas de clasificación?



2-Marque las tareas que logró realizar



3-¿Se presentó algún error que requirió re-ejecutar el software?



4-¿Utilizó el tutorial (pdf) que incluye el software?



5-¿Qué tan intuitiva fue la GUI llamada Problema?



6-¿Qué tan intuitiva fue la GUI llamada Inicialización?



7-¿Qué tan intuitiva fue alguna de las GUI: Gradiente, Evolutivo, Partículas?



8-¿Qué tan intuitiva fue la GUI llamada Post-Entreno?



9-¿Qué tan intuitiva fue la GUI llamada Análisis?



10-¿Cómo calificarías al software?

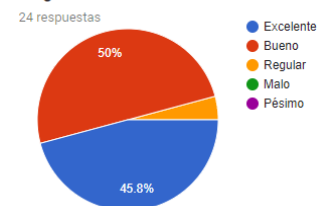


Figura 56 Test de usabilidad

El test en formato *PDF* con el hipervínculo al cuestionario de *Google Drive* se puede ver en los **anexos**.

4.5. PRUEBA DE INTEGRACIÓN

En la Tabla 13 se aprecia un resumen de las pruebas de integración, la secuencia básica de funcionamiento del software fue exitosa; en los **anexos**, se encuentra el archivo del protocolo *RUP* con las tablas ampliadas, incluyendo pantallazos. Aquí, se han omitido las ventanas emergentes de éxito.

Tabla 13 Resumen prueba de integración generalizada

Resumen prueba de integración generalizada		
Requerimientos	Todos los elementales (E).	
Sistemas Necesarios	Computador portátil, SoftwareDMNN.exe	
Objetivo de la Prueba	Demostrar que el aplicativo es capaz de importar un set de datos y manipularlo, luego que es capaz de crear y administrar una red neuronal DMNN, después se procede a ajustar sus pesos entrenándola, se pasa a la optimización de número de dendritas y finalmente se observan los resultados en matrices de confusión.	
Datos de la Prueba	Dataset: vinos.txt	
Prueba		
Procedimiento	Resultados Esperados	Válido
1. Iniciar el aplicativo.	1. Ver la GUI menú	Si
2. Pulsar Problema.	2. Ingresar a la GUI problema.	Si
3. Pulsar Importar Patrones.	3. Muestra ventana emergente para abrir.	Si
4. Seleccionar el archivo vinos.txt y pulsar Abrir.	4. Mostrará datos en gráfica, y sus valores en la GUI.	Si
5. Digitar valores en las cajas de partición y bache.	5. El botón correspondiente se resalta.	Si
6. Pulsar Calcular Porcentajes.	6. Se mostrarán los nuevos valores en la GUI.	Si
7. Pulsar Mezclar y Exportar.	7. Aleatorizado, abre ventana emergente de guardado.	Si
8. Guardar los patrones como vinos.xml.	8. Obtención del archivo vinos.xml.	Si
9. Pulsar Normalizar.	9. Ver escala en gráfica, la función se auto bloquea.	Si
10. Pulsar Menú.	10. Volver a la GUI menú.	Si
11. Pulsar Inicialización.	11. Ingresar a la GUI inicialización.	Si
12. Digitar parámetros para K-medias.	12. El botón correspondiente se resalta.	Si
13. Pulsar Ejecutar K-medias.	13. Ciclo de inicialización, bloquea GUI, luego grafica.	Si
14. Pulsar Exportar Red.	14. Muestra ventana emergente para guardar.	Si
15. Guardar la red como red.xml.	15. Obtención del archivo red.xml.	Si
16. Digitar parámetros para D&C.	16. El botón correspondiente se resalta.	Si
17. Pulsar Ejecutar Divide y Conquista.	17. Ciclo de inicialización, bloquea GUI, luego grafica.	Si
18. Pulsar Importar Red.	18. Muestra ventana emergente para abrir.	Si
19. Seleccionar el archivo red.xml y pulsar Abrir.	19. Grafica la red con los datos abiertos.	Si
20. Pulsar Menú.	20. Volver a la GUI menú.	Si
21. Pulsar Gradiente.	21. Ingresar a la GUI SGD.	Si
22. Digitar parámetros del SGD.	22. Nada.	Si
23. Pulsar Correr (símbolo Play).	23. Ciclo de entreno, bloquea GUI, actualiza gráfica.	Si
24. Pulsar Parar (símbolo Stop).	24. Desbloqueo de la GUI.	Si
25. Pulsar Menú.	25. Volver a la GUI menú.	Si
26. Pulsar Evolutivo.	26. Ingresar a la GUI DE.	Si
27. Digitar parámetros del DE.	27. Nada.	Si

28. Pulsar Correr (símbolo Play).	28. Ciclo de entreno, bloquea GUI, actualiza gráfica.	Si
29. Pulsar Parar (símbolo Stop).	29. Desbloqueo de la GUI.	Si
30. Pulsar Menú.	30. Volver a la GUI menú.	Si
31. Pulsar Partículas.	31. Ingresar a la GUI PSO.	Si
32. Digitar parámetros del PSO.	32. Nada.	Si
33. Pulsar Correr (símbolo Play).	33. Ciclo de entreno, bloquea GUI, actualiza gráfica.	Si
34. Pulsar Parar (símbolo Stop).	34. Desbloqueo de la GUI.	Si
35. Pulsar Menú.	35. Volver a la GUI menú.	Si
36. Pulsar Post-Entreno.	36. Ingresar a la GUI post-entreno.	Si
37. Digitar valor de tolerancia.	37. Los botones correspondientes se resaltan.	Si
38. Pulsar Quitar Dendritas.	38. Ciclo de optimización, bloquea GUI, luego grafica.	Si
39. Pulsar Unir Dendritas.	39. Ciclo de optimización, bloquea GUI, luego grafica.	Si
40. Pulsar Eliminar Inhibidas.	40. Elimina inhibidas, se pone a 0 su display.	Si
41. Pulsar Menú.	41. Volver a la GUI menú.	Si
42. Pulsar Análisis.	42. Ingresar a la GUI análisis y ver las estadísticas.	Si
43. Pulsar Exportar Matrices de Confusión.	43. Muestra ventana emergente para guardar.	Si
44. Guardar las matrices como resultados.xml.	44. Obtención del archivo resultados.xml.	Si
45. Cerrar el aplicativo.	45. Cierre total de la aplicación.	Si

Se adjunta un video en **anexos** de toda la secuencia de la prueba, constatando el funcionamiento correcto de la herramienta software.

4.6. ANÁLISIS DE RESULTADOS

4.6.1. Análisis de Resultados Generales

A partir de la Tabla 8 y Tabla 9, se evidencia que el sistema clasificador funcionó de una manera adecuada; los *datasets* artificiales obtuvieron resultados superiores al 90% para entreno y testeo, teniendo en cuenta que estos dos problemas son de dimensionalidad 2, por lo que su solución puede verse intuitivamente.

En el caso de la espiral, Figura 2 se tiene una forma geométricamente bien definida, donde las superficies de decisión no necesitan superponerse; aun así, el estado del arte afirma que causa grandes inconvenientes a clasificadores como MLP, SVM o RBN. El otro caso se ve en la Figura 38, hay cruce de las clases, sin embargo, las formas de estas están claramente definidas y son separables, tolerando un bajo error.

En cuanto a los 5 *datasets* reales, “Iris” obtuvo la mejor calificación en entrenamiento y prueba, siendo las precisiones superiores al 90%, para “Wine” y “Letter Recognition” fueron superiores al 80%, en cuanto a “Glass” y “Mammographic Mass”, se tienen resultados mayores a 90% y 70% en entrenamiento respectivamente, mientras que valores en prueba solo mayores a 60% y 70%. El problema “Mammographic Mass” está fuertemente discretizado, lo que hace más complejo para la estructura de las hiper-cajas romper ciertas formas alineadas complejizando el entrenamiento.

En cuanto a “Glass”, cuenta con una forma geométrica no muy definida, las hiper-superficies están muy entrelazadas según lo visto en recortes bidimensionales, esto hace que un algoritmo agresivo, como el D&C pueda forzar a bajos errores de entreno, pero la generalización se perderá; por ello, la precisión en la fase de prueba tiende a quedar baja. “Wine” y “Letter Recognition” son de mediana complejidad, pudiéndose llevar a soluciones aceptables y con buena correlación entre entreno y prueba, el último tiene también una marcada discretización, lo que aparentemente limita el entreno.

En general un problema de clasificación, con clases bien definidas en el hiper-espacio puede ser solucionado con la red *DMNN* usando el aplicativo creado.

4.6.2. Análisis de Algoritmos de Inicialización

La Tabla 10 compara al algoritmo inicializador K-medias con Divide y Conquista (D&C). Tal como se explicó previamente, un margen del 10% fue usado para el algoritmo D&C, mientras que se manejaron tamaños de caja al 10% con auto-sintonización para K-medias; ambos algoritmos logran definir el número de dendritas eficazmente; sin embargo, es difícil realizar una comparación equitativa dada la naturaleza como cada algoritmo enfrenta los problemas. Mientras el auto-tuneo de K-medias genera una red minimalista, D&C busca maximizar la precisión respecto al conjunto de entrenamiento. Es por ello que, de acuerdo a los resultados, D&C supera en precisión en fase de entrenamiento para 7 de 8 *datasets*, pero sobreexplota la geometría del problema, dando resultados con baja capacidad de generalización y poca movilidad en el momento del entreno.

Otra observación es que la velocidad de K-medias es muy superior a D&C en ejecución, como ejemplo, el problema “*Wine*” se tomó 69s inicializando con D&C, para crear 24 dendritas, mientras esa misma cantidad fue alcanzada por K-medias en 1s y posteriormente alcanzando 39 dendritas en modo auto-sintonizado en solo 2s. Lo anterior efectuado en un procesador *Intel Core i5 - 7th gen*, sistema operativo *Windows 10*, sin optimización del algoritmo por *GPU*.

El problema “Letter Recognition” fue abortado para D&C luego de aproximadamente 3 días en ejecución; esto sucede porque cada hiper-caja es dividida en 2^I hiper-cajas. Este problema tiene una alta dimensionalidad que aumenta exponencialmente el tiempo y la memoria requeridos; por otra parte, además de tener que calcular la cantidad de patrones que una hiper-caja tiene dentro, debe hacer un segundo ciclo de eliminación de hiper-cajas.

En definitiva, K-medias otorga de entrada una baja cantidad de dendritas, mayor capacidad de generalización, más maleabilidad al entrenamiento y bajo consumo de recursos computacionales; por lo que se utilizó en mayor medida.

4.6.3. Análisis de Algoritmos de Entrenamiento

La Tabla 11 se creó a partir de los *datasets* inicializados con K-medias, para luego aplicar los tres algoritmos de entrenamiento en iguales condiciones, cada uno fue ejecutado 3 veces y se tomó la mejor solución, dada la naturaleza estocástica de los mismos.

Como se observa, el gradiente descendente estocástico (*SGD*) logró los mejores resultados, seguido de la optimización por enjambre de partículas (*PSO*), es un desafío seleccionar para los algoritmos heurísticos los parámetros adecuados, en este caso se han utilizado los valores recomendados en el estado del arte, independientemente del problema afrontado.

Un caso notorio es el de “Mammographic Mass”, donde *SGD* se vio totalmente estancado debido a la alta discretización de los patrones, mientras *SGD* y evolución diferencial (*DE*) lograron buscar otras soluciones, *DE* sacrificó error de entreno y ganando en generalización, mientras *PSO* logró salir del mínimo local.

Hay valores en los algoritmos que muestran una desmejora en el entreno, esto es debido a que el entreno se realiza con el error de regresión logística, mientras la mayoría de estadísticas usa el error cuadrático medio (*ECM*).

El comportamiento observado, para *SGD* es un descenso constante del error hasta estancarse en el mínimo local, para *DE* el descenso es más accidentado, aleatorio, pero constante y no siempre muy abrupto, hasta alcanzar un mínimo local, para *PSO* el descenso es muy aleatorio y suele tener fuertes cambios, hasta llegar a un mínimo local con mejores condiciones. Esto causa que *SGD* sea más confiable, mientras *PSO* tiene comportamientos donde se queda estancado, aunque logra obtener soluciones donde el gradiente descendente no logra llegar, partiendo de las mismas condiciones iniciales; un ejemplo de esto se manifiesta en el problema “Iris” de la Tabla 11.

En cuanto a la capacidad de generalización, se puede hacer un análisis viendo las diferencias entre entreno y testeo (varianza) para cada caso: K-medias ofrece una buena generalización, *SGD* tiene una mayor varianza, seguido de *PSO*, mientras *DE* tiene una menor varianza, suponiendo mejor índice de generalización el algoritmo *DE*.

La validación funciona igual para *DE* y *PSO*, pero estos pueden hacer “saltos” aleatorios en el hiper-espacio, que empeoran el error, pero que mejoran la validación, lo que hace prevalecer estrategias generalizadoras. Luego los movimientos de las partículas son más bruscos, así que la solución depende de dónde caiga la partícula en cada paso.

Otro detalle es que *DE* y *PSO* consumen más recursos y tiempo de computo, en general se ha comprobado que *PSO* es un algoritmo adecuado para entrenar la red *DMNN*, que cuenta con buena capacidad de generalización y logra afinar los pesos sinápticos para todos los *datasets* empleados.

4.6.4. Análisis de problema “PayasoLander”

Como se vio en la Tabla 9, la precisión fue pésima, un 40.1% en entreno y 35.29% en validación, aun así ese valor fue de los mayores obtenidos, y su ejecución no

alcanzó a entrar en la Tabla 12, esta refiere a 30 vuelos realizados para cada solución propuesta; donde la calificación de un vuelo es: éxito, casi lo logró, o falla total.

Puede verse la importancia de la capacidad de generalización, ya que la tabla está ordenada según los éxitos, máximo 36.67%, y esto coincide con los porcentajes de precisión de testeo; contrariamente una mayor precisión de entreno no necesariamente garantiza un buen vuelo. Además, el número elevado de dendritas implica una superficie de decisión más compleja, con menor error de entreno, pero cercano al sobre-entrenamiento.

Aunque la red con 106 dendritas se trató de entrenar con los tres algoritmos varias veces, los resultados solo mejoraron para *DE* tal como se observa en la Tabla 12 y corroborando la mayor capacidad de generalización.

El software creado en *Game Maker Studio*, representó un dificultoso problema de optimización, obtener un set de patrones adecuado es sin dudas la principal causa de fallos, los datos salen de jugadas manuales naturalmente muy ruidosas, y lo adecuado para afrontar este problema podría ser una simulación evolutiva multi-agente.

4.6.5. Análisis Test de Usabilidad

De la Figura 56 se puede decir que, la mayor parte de los encuestados, el 87.5% están involucrados en el tema de las *ANN* o problemas de clasificación, lo que es beneficioso puesto que los resultados más bajos provienen de usuarios ajenos al tema, el 12.5%.

Todas las tareas fueron exitosamente realizadas por un 90.0% de los encuestados (en promedio), siendo la optimización del número de dendritas la más fallida, esto quizá por el desconocimiento de la novedosa red neuronal y el concepto de dendrita.

Si bien el software ha demostrado estar listo para uso, aún posee un 33.3% de cierres en su ejecución, según la encuesta; se ha notado durante la operación de *datasets* que múltiples *clics* en momentos críticos pueden ocasionar cierres; esto sin lugar a dudas requiere más trabajo en cuanto a corrección de errores.

El tutorial/guía en *PDF* que viene con el software, ha demostrado ser crucial para los nuevos usuarios, pues el 50% lo marcó como indispensable y al 33.3% le resolvió alguna duda.

El 85.86% (en promedio) de las respuestas referentes a la intuitividad de las *GUI's* fue positiva, siendo la más intuitiva Análisis, seguida de Entreno, Inicialización, Problema y Post-Entreno; la primera, con 66.7% no tiene muchos comandos, básicamente muestra las estadísticas, mientras la última, con 41.7% requiere seguramente un replanteamiento; estos 2 porcentajes son la calificación “Muy fácil de entender”.

Finalmente, la calificación general de los usuarios es muy favorable, mostrando un 50% para “Bueno” y 45.8% para “Excelente”.

4.7. CONCLUSIONES

Una vez recopilados los 8 sets de datos, se utilizó la herramienta creada para procesarlos, estos registros se hicieron en *Excel*.

El problema elegido para testear la *DMNN* fue construido con éxito, se tomaron los datos suficientes para entrenar una red, que diera resultados aceptables (36.67% éxitos) al momento de su ejecución en tiempo real.

Desarrollar la prueba de integración y el test de usabilidad, este último con 24 usuarios reales y desconocedores de la herramienta, da al proyecto una primera aproximación de evaluación objetiva, requiriéndose mayor muestra poblacional; demostrando aun así el cumplimiento de sus funciones y la intuitividad necesaria para ser puesta en marcha; aun así, es de anotar que los usuarios no experimentados en el área, un 12.5%, dieron calificaciones más bajas puesto que no entendían qué hacer. El público idóneo claramente son los conocedores de las *ANN* y de problemas de clasificación, equivalente al 87.5% de los encuestados.

La herramienta implementada cumple plenamente con el propósito de servir como apoyo a la docencia en el curso de redes neuronales en la etapa final donde se introducen arquitecturas avanzadas y novedosas de redes neuronales artificiales. Durante el semestre 1 de 2019, se realizó una prueba de uso con los estudiantes matriculados.

CAPÍTULO 5

5. ¿A QUÉ SE LLEGÓ?

5.1. CONCLUSIONES GENERALES

La red *DMNN* y sus algoritmos de entrenamiento, entre los cuales resaltan *SGD*, *DE*, *PSO*, fueron correctamente implementados en un aplicativo software, utilizando lenguaje *Python* y la *API* de *Qt Creator* para hacer la *GUI*, se cuenta en el mismo con las métricas de desempeño estandarizadas, es decir, matrices de confusión, cuya documentación se realizó usando el protocolo *RUP*.

Fueron puestos a prueba 5 *datasets* reales, 2 artificiales y 1 problema puntual, que involucra dinámicas físicas y fue programado paralelamente al aplicativo.

Como resultados, se afirmó la superioridad del inicializador K-medias sobre D&C, otorgando de entrada una baja cantidad de dendritas, mayor capacidad de generalización, bajo consumo de recursos computacionales y mejor maleabilidad para la sintonización de la red (ver Análisis de Algoritmos de Inicialización, página 66).

Luego se encontró que *SGD* en efecto es el algoritmo que garantiza la mejor solución para la mayoría de los casos, garantizando un descenso continuo del error hasta alcanzar el mínimo local, por su parte *DE* fue el que tuvo el menor rendimiento.

El algoritmo *PSO* está en un punto intermedio en cuanto a generalización y disminución del error, logró optimizar todos los problemas, encontrando en algunos casos soluciones mejores a las de *SGD*, esto es un evento esporádico, por lo que no se puede generalizar, además que requiere parámetros bien seleccionados para no destruir la estructura de decisión.

A pesar de que *DE* y *PSO* consumen más recursos computacionales, son estrategias adecuadas para la red *DMNN*, en especial *PSO* puede sacar provecho en problemas fuertemente discretizados, o donde hay múltiples soluciones.

Para todo lo anterior, se realizó la búsqueda bibliográfica concerniente a la red *DMNN* y los algoritmos involucrados en su funcionamiento, con esto se hace un

aporte en la adquisición del conocimiento por parte del grupo investigativo PSI, y se pone en marcha las implementaciones derivadas de este trabajo.

Un importante fruto de este trabajo, sin lugar a dudas es el aplicativo llamado “**SoftwareDMNN**” que será puesto como código abierto a la comunidad, y el cual puede ser usado para resolver diversidad de problemas de clasificación, tanto a nivel de investigación como en la enseñanza de la red *DMNN* misma.

El tutorial en *PDF* que posee el software además contiene una resumida y amena explicación de todos los temas vistos en este trabajo de grado, por lo que dicho documento contribuye a la expansión del conocimiento, de un tema que es poco conocido en comparación a otros más ampliamente difundidos.

Como producto secundario, se tiene el software “PayasoLanderANN”, el cual es un videojuego, que puede manejarse manualmente o ejecutar redes *DMNN* y *MLP* de una capa oculta, además de muestrear las variables cruciales para crear *datasets*.

5.2. TRABAJOS FUTUROS

En el entrenamiento de redes clásicas como la *MLP* con gradiente descendente, la función de error viene acompañada del factor de regularización L_2 (Caicedo & Lopez, 2013), que es el promedio de los pesos w al cuadrado, con el fin de mejorar los efectos de generalización de la red; desde la perspectiva de las *DMNN* el efecto de regularización podría estar orientado a reducir el número de dendritas en todas las neuronas, tal como se muestra en la ecuación (23) lo cual se presenta como una opción para trabajos futuros.

$$\varepsilon^{DMNN} = \varepsilon + \lambda \cdot \frac{1}{total_dendritas} \cdot \sum_{m=1}^M \sum_{k=1}^{Km} b_{mk} \quad (23)$$

En cuanto a los algoritmos *DE* y *PSO*, hace falta estrategias para encontrar los parámetros óptimos a cada problema, aún queda debate abierto sobre su eficiencia y está en pie el hacer más pruebas, con más *datasets* y mediciones estadísticas adecuadas para soluciones provenientes de comportamientos heurísticos.

Existen variantes de los algoritmos empleados, tanto de gradiente descendente, como evolutivos / genéticos y de enjambre de partículas; se plantea entonces hacer una aplicación de estas variantes, para cerciorarse de la eficacia de las mismas al entrenar la *DMNN*; también hay otros algoritmos heurísticos bio-inspirados, que podrían aportar ventajas, entre los cuales se hallan: el forrajeo de bacterias, algoritmo de búsqueda de abejas, hormigas y luciérnagas.

La red fue planteada para este estudio, con coordenadas globales (ver página 11), también es posible tratar un extremo de la hiper-caja como global y el otro referenciado al primero (posición vs tamaño), de este modo también es posible tomar el extremo posicional como centro de la hiper-caja, de esta forma es, según se hipotetiza, viable que el algoritmo *PSO*, pueda tener una mayor eficacia; al mover principalmente la posición de la hiper-caja como si fuese toda una partícula.

Se plantea el estudio de la aplicación de las *DMNN* en problemas de regresión.

Con el fin de mejorar la robustez de la herramienta desarrollada se propone incluir en su implementación optimización de hilos, ejecución, excepciones, estructura e intuitividad y comodidad de la GUI, con el fin de precisar el trabajo futuro se plantea:

- Mostrar progreso estimado para las inicializaciones y optimizaciones de dendritas, también puede ser una estimación de tiempo.
- Permitir que el usuario interrumpa las inicializaciones y optimizaciones de dendritas, en cuyo caso se alcanzarán algunos resultados.
- Importar y exportar red deberían ser funciones globales en lugar de estar solo en dos *GUI's*.
- Crear un sistema de bases de datos que administre usuarios y sus trabajos, para así acceder eficientemente a las redes creadas.
- Implementar otro tipo de algoritmos e incluso *ANN* al aplicativo.

6. REFERENCIAS

- A. Serani, M. D. (2014). On the Use of Synchronous and Asynchronous Single-Objective Deterministic Particle Swarm Optimization in Ship Design Problem. *1st International Conference on Engineering and Applied Sciences Optimization*. Kos Island, Greece.
- Aggarwal, C. C., & Reddy, C. K. (2013). *Data Clustering: Algorithms and Applications*. Chapman & Hall / CRC Press.
- Arce, F., Zamora, E., Sossa, H., & Barron, R. (2018). Differential evolution training algorithm for dendrite morphological neural networks. *Applied Soft Computing*, vol. 68, pp. 303-313.
- Arthur, D., & Vassilvitskii, S. (2007). *K-means++: the Advantages of Careful Seeding*. New Orleans, USA: Symposium on Discrete Algorithms.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. New York, USA: Springer.
- Caicedo, E. F., & Lopez, J. A. (2013). *Una Aproximación Práctica a las Redes Neuronales Artificiales*. Cali, Colombia: Universidad del Valle.
- Carlos A. Coello Coello, D. A. (2007). *Evolutionary Algorithms for Solving Multi Objective Problems*. US: Springer, Genetic and Evolutionary Computation.
- Cecilia Sosa, G. L. (2012). *Evolución Diferencial con Factor de Mutación Dinámico*. Argentina: RedUNCI.
- Corchado E., A. A. (2008). *Hybrid Artificial Intelligence Systems*. Berlin, Heidelberg: Springer, Lecture Notes in Artificial Intelligence.
- Davidson, J. L., & Hummer, F. (1993). Morphology Neural Networks: An Introduction With Applications. En *Circuits, Systems and Signal Processing* (págs. vol. 12, pp. 177-210). Springer.

- Davidson, J. L., & Ritter, G. X. (1990). Theory of Morphological Neural Networks. *Digital Optical Computing II* (págs. vol. 1215, pp. 378-389). Los Angeles, CA, United States: SPIE.
- Diederik P. Kingma, J. B. (2014). *Adam: A Method for Stochastic Optimization*. Cornell University.
- Eiben, A., & Smith, J. (2003). *Introduction to Evolutionary Computing*. Berlin Heidelberg: Springer.
- Eric Bonabeau, G. T. (2000). *Swarm Smarts*. US: Scientific Amer, Inc.
- Eric Bonabeau, M. D. (1999). *Swarm Intelligence: From Natural To Artificial Systems*. New York, NY: Oxford Univ. Press.
- Forbes, N. (2005). *Imitation of Life: How Biology is Inspiring Computing*. MIT Press.
- Gaia, T. (2012). Dendritic structural plasticity. *Developmental Neurology*, vol. 72, pp. 73-86.
- Herbert Robbins, S. M. (1951). A Stochastic Approximation Method. En *Ann. Math. Statist.* (págs. vol. 22, no. 3, pp. 400-407). JSTOR.
- Herbert, R., & Sutton, M. (1951). *A Stochastic Approximation Method*. Project Euclid.
- Humberto Sossa, E. G. (2014). Efficient training for dendrite morphological neural networks. En *Neurocomputing* (págs. vol. 131, pp. 132-142). Vallejo, Mexico: Elsevier.
- IBM Software Group. (2003). *Rational Unified Process: A Best Practices Approach*. USA: IBM corporation.
- James Kennedy, R. C. (1995). Particle Swarm Optimization. *International Conference on -neural Networks*. Perth, WA, Australia, Australia: IEEE.

- John Duchi, E. H. (2010). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *The Journal of Machine Learning Research*, vol. 12, pp. 257-269.
- Joseph N. Wilson, G. X. (2000). *Handbook of Computer Vision Algorithms in Image Algebra*. NY, USA: CRC Press, 2 edition.
- Kenneth V. Price, R. M. (2005). *Differential Evolution, A Practical Approach To Global Optimization*. Berlin Heidelberg: Springer.
- Kiranyaz, S., Ince, T., & Gabbouj, M. (2014). *Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition*. Berlin Heidelberg: Springer.
- Raducanu, B., & Grana, M. (2000). Morphological Neural Networks for Robust Visual Processing in Mobile Robotics. *International Joint Conference on Neural Networks* (págs. vol. 6, pp. 140-143). Como, Italy, Italy: IEEE.
- Ritter, G. X., & Sussner, P. (1996). An Introduction to Morphological Neural Networks. *International Conference on Pattern Recognition* (págs. vol. 4, pp. 709-717). Vienna, Austria: IEEE.
- Ritter, G. X., Lancu, L., & Urcid, G. (2003). Lattice Algebra Approach to Single Neuron Computation. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 14, pp. 282-295.
- Ritter, G. X., Lancu, L., & Urcid, G. (2003). Morphological Perceptrons With Dendritic Structure. *The 12th IEEE International Conference on Fuzzy Systems*, vol. 2, pp. 1296-1301.
- Sofia Visa, B. R. (2011). Confusion Matrix-based Feature Selection. *CEUR Workshop Proceedings*, vol. 710, pp. 120-127.
- Storn, R., & Price, K. (1997). Differential Evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, vol. 11, issue. 4, pp. 341-359.

- Swets, J. (1988). *Measuring the accuracy of diagnostic systems*. USA: American Association for the Advancement of Science.
- Trelea, I. C. (2003). The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection. vol. 85.
- Vincent Morard, E. D. (2000). *Mathematical Morphology and Its Applications to Image and Signal Processing*. US: Springer, Computational Imaging and Vision.
- Weise, T. (2009). *Global Optimization Algorithms Theory and Applications*. Institute of Applied Optimization.
- Won, Y., & Gader, P. D. (1996). Comparison of Linear and Morphological Shared-Weight Neural Networks. *Electronic Imaging: Science and Technology*. San Jose, CA, United States: SPIE.
- Yang Liu, K. M. (2002). *Biomimicry of Social Foraging Bacteria for Distributed Optimization: Models, Principles, and Emergent Behaviors*. US: J. Optimization Theory and Applications.
- Yang, X.-S. (2010). *Engineering Optimization - An Introduction With Metaheuristic Applications*. Cambridge, United Kingdom: Wiley.
- Zamora, E., & Sossa, H. (2017). Dendrite Morphological Neurons Trained by Stochastic Gradient Descendent. *Neurocomputing*, vol. 260, pp. 420-431.
- Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. vol. 1212.

7. ANEXOS

En el CD entregado se encuentra la siguiente estructura de carpetas, allí verá los diferentes archivos mencionados a lo largo del documento; la carpeta “SoftwareDMNN” contiene el código fuente y *Assets* del aplicativo.

 LibroDMNN.pdf

▼  Anexos

 SoftwareDMNN_v7.exe

 ➤  SoftwareDMNN

 ▼  PayasoANN

 PayasoLanderANN_v131.exe

 PayasoLanderPseudo.pdf

 ▼  InformacionDMNN

 TestSoftwareDMNN.pdf


 RUP_DMNN.pdf

 TutorialDMNN.pdf

 AnteproyectoDMNN.pdf

 PresentacionAnteproyectoDMNN.pdf

 VideoPruebaDMNN.mp4

 ➤  PapersArticulos

 ➤  Figuras

 ▼  ProyectoClaseANN

 DMNNclaseANN_v12.exe

 ExposicionClaseANN.pdf

 InformeClaseANN.pdf

 ➤  PruebasDMNN