Universidad de las Ciencias Informáticas

Facultad 2

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

"Entrenamiento de un Perceptrón Multicapa utilizando agrupamiento conceptual aplicando el algoritmo RGC"

Autor: Adonis Guerrero Pi

Tutores: MSc. Maidelis Milanés Luque

Ing. Luis E. Manchón Lauzardo

La Habana, junio 2019

"Año 61 de la Revolución"

Quien nunca ha cometido un error nunca ha intentado nada nuevo...

Albert Einstein

Dedicatoria:

A los dos titanes que tengo el orgullo de llamar padres, Lily y Adonis que nunca se cansaron de salvarme en todo momento, que nunca dudaron en detener mi mano cuando estaba obrando mal, que nunca titubearon en recibir una herida por mi, que siempre me tenían una sonrisa, aunque los tiempos fueran malos, que me enseñaron que para ser mejor persona no se necesitan vanidades ni artilugios, sino un corazón puro y una mente firme.

A mi tía Any, que me enseñó que la humildad te hace más grande, a mi tío Tony que me enseñó que las cosas más importantes en la vida, no son cosas.

A mi Tesoro por ser el combustible que ha mantenido mi marcha durante todos estos años y por haber sido el apoyo incondicional que siempre necesité.

Agradecimientos:

A mis padres por haber dedicado su vida a educarnos a mi hermana y a mí,

a mi hermana por haber cuidado de mí cuando no podía hacerlo solo y ser un ejemplo de perseverancia a seguir,

a mis tíos y primos maternos por acogerme como un miembro más dentro su núcleo familiar,

a mi Tesoro por apoyarme siempre cuando lo necesito,

a mis tías por tratarme siempre como un hijo,

a Mercedes y Jorge por dejarme entrar en su hogar como un miembro más,

a mis tutores Maidelis y Luis Ernesto por guiarme en este proceso,

a la profesora Zoila por su interés, preocupación y ayuda en la realización de este trabajo,

a Omar por ser profesor, amigo y consejero,

al profesor Osmar por sus acertadas observaciones y consejos sobre este trabajo,

a los miembros del tribunal por su profesionalidad y dedicación,

a todos los profesores que de una manera u otra han contribuido a que esto hoy sea posible,

a mis amigos Yosvani y Lisardo por ser excelentes vecinos e indispensables compañeros,

y a todos los amigos que inevitablemente hice a lo largo de estos años…

Resumen

Las redes neuronales artificiales son modelos computacionales con propiedades particulares, como la capacidad de adaptarse o aprender a generalizar, agrupar u organizar datos basado en un procesamiento paralelo. Dentro de estas, uno de los modelos más utilizados para la solución de problemas es el Perceptrón Multicapa teniendo en cuenta su gran capacidad como aproximador universal. En este modelo se presentan dos problemas que son significativos tanto en la convergencia como en la ejecución de la red neuronal artificial, uno relacionado con la determinación de los pesos sinápticos iniciales y otro referente al entrenamiento de la red en sí. El presente trabajo se propone como objetivo general desarrollar una herramienta informática para el entrenamiento del Perceptrón Multicapa utilizando el algoritmo RGC. Se utilizaron como métodos científicos el analítico-sintético y el experimental, los que permitieron dirigir la investigación. Se describen las dos etapas del algoritmo RGC: determinación extensional y determinación intencional, así como el entrenamiento de la red. Se establecen comparaciones en cuanto a la cantidad de objetos clasificados correctamente entre tres configuraciones del Perceptrón Multicapa, uno sin aplicar los algoritmos conceptuales, otro aplicando el algoritmo RGC y el último aplicando algoritmo LC-Conceptual. Los resultados demuestran que la solución propuesta garantiza una mayor eficacia en la solución de problemas a la hora de entrenar una red neuronal utilizando un MLP.

Palabras clave: algoritmo RGC, entrenamiento, Perceptrón Multicapa.

Abstract

Artificial neural networks are computational models with properties such as the ability to adapt or learn to generalize, group or organize data based on parallel processing. Within these, one of the most used models for problem solving is the Multilayer Perceptron, considering its great capacity as a universal approximator. In this model, two problems are presented that are significant both in the convergence and in the execution of the artificial neural network, one related to the determination of the initial synaptic weights and another related to the training of the network itself. The present work proposes as a general objective to develop a computer tool for the training of the Multilayer Perceptron using the RGC algorithm. The analytical-synthetic and the experimental methods were used as scientific methods, which allowed directing the research. The two stages of the RGC algorithm are described: extensional determination and intentional determination, as well as network training. Comparisons are established in terms of the number of objects correctly classified among three configurations of the Multilayer Perceptron, one without applying the conceptual algorithms, another applying the RGC algorithm and the last applying the LC-Conceptual algorithm. The results show that the proposed solution guarantees greater efficiency in solving problems when training a neural network using an MLP.

Keywords: RGC algorithm, training, Multilayer Perceptron.

Índice

Índice de Figuras

Índice de Tablas

Introducción

Se calcula que los seres humanos tenemos alrededor de 100 mil millones de neuronas en nuestro cerebro y esas neuronas, que se encuentran conectadas entre sí, forman una red amplia y expandida, denominada red neuronal. Gracias al trabajo de estas neuronas, somos capaces de asimilar lo que percibimos en nuestro entorno y de adquirir, a través de esa información, el conocimiento de lo que nos rodea. Dichos conocimientos quedan atrapados en esta red, pero para ello es necesario un proceso de entrenamiento o aprendizaje (1).Las redes neuronales artificiales, surgen con el propósito de simular esta estructura biológica.

Las Redes Neuronales Artificiales (RNA) constituyen algoritmos poderosos de aprendizaje para crear bases de conocimientos que funcionan de manera similar a las redes neuronales biológicas. Entre sus principales ventajas se presentan la tolerancia a fallos, el aprendizaje adaptativo, la auto organización, el trabajo con información incompleta, entre otras. Una red neuronal artificial está compuesta por neuronas, que constituyen unidades independientes de procesamiento y/o almacenamiento de información, estas a su vez se agrupan por capas según su función en la red, y están conectadas entre sí mediante una relación ponderada por el peso sináptico que posee la relación (2).

Uno de los modelos más usados en este contexto es el Perceptrón Multicapa (MLP). Este es una generalización del Perceptrón Simple y surgió como consecuencia de las limitaciones de dicha arquitectura en lo referente al problema de la separabilidad no lineal.
Existen dos conceptos claves a analizar de manera distinta a la hora de utilizar un MLP:

1- El diseño de la arquitectura que incluye los pesos sinápticos iniciales.
2- El entrenamiento que garantice la generalización del modelo.

Estas redes requieren de muchos ejemplos y son cajas negras que no explican como la solución se alcanza, presentando entre sus mayores dificultades la selección y presentación del conjunto de entrenamiento adecuado para su aprendizaje, así como el tiempo de aprendizaje elevado cuando se incrementan la dimensionalidad de los datos de entrada, o se necesita de un grado alto en la capacidad de generalización de la red propiamente dicha.

Para resolver la problemática en el entrenamiento del MLP se han diseñado diversas estrategias entre las que se encuentran Probabilidad de Presentación Dependiente del Error (PPDE), Repetición Dependiente del Error (RDE) y Repetición Dependiente del Mayor Error (RDME), sin embargo, esto sigue siendo un tema abierto de investigación con el fin de mejorar la eficiencia en cuanto al tiempo de entrenamiento y la eficacia en la clasificación.

Los algoritmos de agrupamiento conceptual del enfoque lógico combinatorio, que además de agrupar a los objetos según el grado de semejanza que tienen entre sí, permite la descripción del grupo en función de las propiedades que lo caracterizan, teniendo en cuenta tanto las variables cualitativas como cuantitativas. Tienen como ventaja en la clasificación, que permiten la reducción de la dimensión del vector de entrada manteniendo la capacidad de ser caracterizantes y excluyentes. Además, permiten mediante el peso informacional de rasgos y conceptos establecer la importancia que existe entre los objetos que pertenecen a un mismo grupo, aspectos que pueden ayudar a alcanzar una mayor generalización de la red.

En la literatura consultada se tiene referencias del uso de los algoritmos de agrupamiento conceptual en el diseño de la red, pero no en el entrenamiento a pesar de las ventajas que estos ofrecen.

Teniendo en cuenta lo anteriormente planteado se ha identificado el siguiente **problema a resolver**: ¿Cómo contribuir a la disminución del tiempo de entrenamiento y a la generalización en un Perceptrón Multicapa?

Se define como **objeto de estudio** de la investigación: Entrenamiento del Perceptrón Multicapa.

A partir del punto de partida antes expuesto, la presente investigación define como **objetivo general** aplicar el algoritmo RGC en el entrenamiento de un MLP, para contribuir a la disminución del tiempo de entrenamiento y a la generalización, enmarcado en el **campo de acción**: el proceso de entrenamiento del Perceptrón Multicapa utilizando agrupamiento conceptual.

Para darle cumplimiento al objetivo se plantean las siguientes **tareas de la investigación**:

1 Análisis de los referentes teóricos clásicos sobre el entrenamiento en las redes neuronales artificiales, específicamente el Perceptrón Multicapa.
2 Caracterización del RGC para identificar las ventajas del mismo, así como las potencialidades que su uso puede traer al entrenamiento de un Perceptrón Multicapa.
3 Implementación del método de entrenamiento.
4 Selección de las herramientas, tecnologías, lenguajes, entre otros que permitan el desarrollo de la investigación.
5 Constatación en la práctica tanto de la disminución en el tiempo de entrenamiento como del aumento de la generalización de la red.

Para un correcto desarrollo de la investigación se utilizan los siguientes **métodos científicos:**

Métodos Teóricos:

- **Analítico-Sintético:** permite determinar las características esenciales de la apropiación del conocimiento en los modelos neuronales, específicamente el entrenamiento y la generalización del modelo. Por otra parte, permite la integración de la selección de rasgos y el agrupamiento conceptual del enfoque lógico combinatorio al proceso de entrenamiento en un MLP.
- **Modelación:** facilita representar gráficamente los elementos que componen la propuesta de solución.

Métodos Empíricos:

- **Método experimental:** la experimentación con diferentes *data sets* muestra los resultados del entrenamiento del MLP una vez aplicado el algoritmo RGC y permite la comparación de los resultados obtenidos sin el método implementado.

Para lograr una mejor organización y lectura del trabajo de diploma, este se estructuró de la siguiente manera:

Capítulo 1: Se describen los referentes teóricos relacionados con las Redes Neuronales Artificiales, específicamente el modelo Perceptrón Multicapa. Se aborda el tema sobre los algoritmos conceptuales del Reconocimiento Lógico Combinatorio de Patrones y cómo estos pueden ser usados para mejorar los tiempos de entrenamiento y la capacidad de generalización de una red. Además, se describen tecnologías, metodología y lenguajes de programación que se adecuan a las características de la herramienta computacional que se implementa como resultado de esta investigación.

Capítulo 2: Se propone un algoritmo capaz de contribuir a la disminución del tiempo de entrenamiento en un MLP, así como aumentar la capacidad de generalización del mismo. Se detallan los pasos a seguir para el desarrollo del mismo, así como los aspectos más importantes a tener en cuenta en este proceso.

Capítulo 3: Se describen los preexperimentos realizados que permiten corroborar la disminución del tiempo de entrenamiento y el aumento de la capacidad de generalización que aporta el uso del algoritmo RGC a la hora de entrenar un Perceptrón Multicapa. Se muestran los resultados obtenidos al aplicar tres configuraciones de un MLP en distintas bases de datos internacionales, la primera con el MLP sin aplicar los algoritmos conceptuales, la segunda con el MLP y el LC-Conceptual y la tercera con el MLP y el RGC.

Finalmente se muestran los principales resultados obtenidos, se realizan las recomendaciones pertinentes para futuros trabajos, así como la constancia de la bibliográfica consultada.

Capítulo 1: Fundamentación teórica de las RNAs y su entrenamiento

En el presente capítulo se exponen los elementos teóricos que sustentan el proceso de investigación y desarrollo del tema propuesto. Se analizan y caracterizan conceptos sobre las RNAs, las distintas técnicas utilizadas para mejorar aspectos particulares en el entrenamiento de un MLP, así como la selección de las herramientas a utilizar y sus características.

1.1 Redes Neuronales Artificiales

Las redes neuronales artificiales surgieron como una alternativa a la solución de los llamados “problemas del mundo real”, donde la información que se tiene del mismo es variada, poco precisa, grande y no pueden ser abordados por los métodos matemáticos convencionales. Los más habituales son los relacionados con la clasificación, estimación funcional, optimización y reconocimiento de patrones. Se pueden señalar, entre otras aplicaciones, reconocimiento del habla, reconocimiento de caracteres, visión, robótica, procesamiento de señal, predicción, economía, defensa, y muchas otras (3).

Las redes neuronales artificiales (*artificial neural networks*), imitan la estructura hardware (neurona) del cerebro con la intención de construir sistemas de procesamiento de la información paralelos, distribuidos y adaptativos, que puedan representar un conocimiento inteligente.

Las funciones del cerebro que se tratan de imitar están estrechamente relacionadas con la propiedad asociativa y la capacidad de auto organización del mismo, definiéndose:

1. La propiedad asociativa como la capacidad de recuperar un cuerpo de información complejo usando una parte pequeña del mismo como una llave para un proceso de búsqueda.
2. La auto organización significa la habilidad para adquirir conocimiento a través de un proceso de aprendizaje por tanteo que comprende organizarse y reorganizarse en respuesta a estímulos externos (4).

Una red neuronal se puede caracterizar como un modelo computacional con propiedades particulares como la capacidad de adaptarse o aprender a generalizar, agrupar u organizar datos basado en un procesamiento paralelo.

Una red neuronal es un procesador masivamente paralelo distribuido, que es propenso por naturaleza a almacenar conocimiento experimental y hacerlo disponible para su uso. Este mecanismo se parece al cerebro en dos aspectos:

1. El conocimiento es adquirido por la red a través de un proceso que se denomina aprendizaje.

2. El conocimiento se almacena mediante la modificación de la fuerza o peso sináptico de las distintas uniones entre neuronas.

Según las anteriores definiciones se puede concluir que, una red neuronal artificial es un modelo matemático que permite el procesamiento de información paralelamente, que tiene una estructura propia particular expresada en un conjunto de parámetros internos, arquitecturas y modelos, que imita algunas de las características de las redes neuronales biológicas.

1.1.1 Generalidades de las Redes Neuronales Artificiales

El elemento básico de un sistema neuronal artificial lo compone la neurona o unidad de procesamiento, que se agrupan conformando las capas, las cuáles al estar interconectadas entre sí, forman la red neuronal, que al incluírsele la regla de aprendizaje (parte algorítmica), forman el sistema neuronal completo.

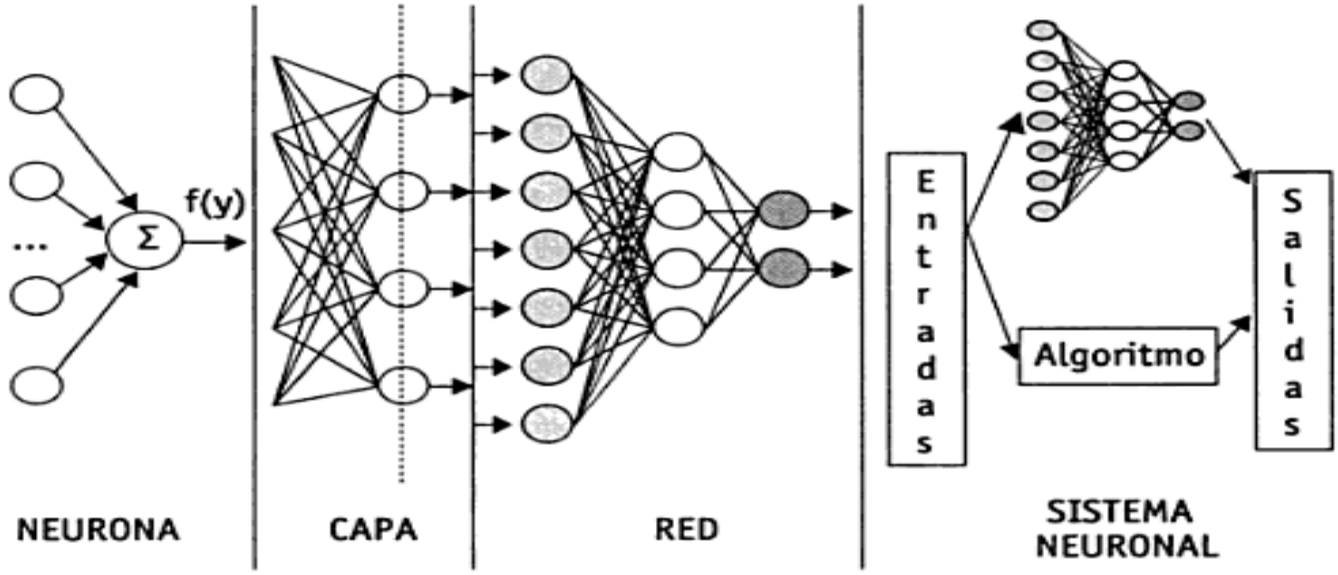

Figura 1: Estructura jerárquica de un sistema basado en redes neuronales artificiales (5)

Formalmente un Sistema Neuronal Artificial está compuesto:

- Un conjunto de procesadores elementales o neuronas artificiales.
- Un patrón de conectividad o arquitectura.
- Una dinámica de activaciones.
- Una regla o dinámica de aprendizaje.
- El entorno donde opera.

Donde los elementos que constituyen la unidad de procesamiento o neurona artificial son:

- Conjunto de entradas, xj(t).
- Pesos sinápticos de la neurona i, wij que representan la intensidad de interacción entre cada neurona pre-sináptica j y la neurona post-sináptica i.
- Regla de propagación σ (wij, xj(t)), que proporciona el valor del potencial post-sináptico hi(t) = σ (wij, xj(t)), de la neurona i en función de sus pesos y entradas.
- Función de activación fi(ai(t-1), hi(t)), que proporciona el estado de activación actual ai(t) = fi(ai(t-1), hi(t)) de la neurona i, en función de su estado anterior ai(t-1) y de su potencial post-sináptico actual.
- Función de salida Fi (ai(t)), que proporciona la salida actual yi(t) = Fi(ai(t)) de la neurona i en función de su estado de activación (5).

En una red neuronal se le denomina arquitectura a su topología o patrón de conexionado. Normalmente la arquitectura se clasifica atendiendo a dos criterios fundamentales: flujo de datos de la red y número de capas de la red.

Atendiendo al flujo de datos se clasifican en redes unidireccionales o *feedforward* (el flujo de datos transita desde las neuronas de la capa de entrada, hacia las neuronas de la capa de salida) o redes recurrentes, realimentadas o *backforward* (el flujo de datos puede circular en cualquier sentido).

Atendiendo al número de capas se clasifican en redes monocapas (de una sola capa) o redes multicapas (de dos o más capas).

De manera general se pueden distinguir tres tipos de capas, aunque esto varía según el modelo propuesto (6):

1. **Capa de entrada:** Es la capa encargada recibir la información de entrada al modelo.
2. **Capas ocultas:** Se dedican al procesamiento de la información para convertirla en una posible salida.
3. **Capa de salida:** Es la capa encargada de emitir la respuesta del sistema.

La arquitectura varía en dependencia del modelo neuronal al se refiere y del problema que se quiera resolver.

En las redes neuronales artificiales generalmente se definen dos modos de operación: recuerdo o ejecución y aprendizaje o entrenamiento.

Recuerdo o Ejecución: En este modo generalmente llamado estabilidad del modelo, es cuando la red ya después de pasado el entrenamiento (fijados los pesos de las conexiones), es utilizada para resolver el problema para la que fue previamente entrenada.

Aprendizaje o Entrenamiento: En este modo generalmente denominado convergencia del modelo, es cuando la red neuronal ajusta los pesos de sus conexiones a partir de una regla de aprendizaje previamente definida, de manera tal que responda adecuadamente a todos los patrones del entrenamiento.

El aprendizaje o entrenamiento en un sistema neuronal artificial se define como el proceso por el que se produce el ajuste de los parámetros libres de la red a partir de un proceso de estimulación por el entorno que rodea la red. El tipo de aprendizaje vendrá determinado por la forma en la que dichos parámetros son adaptados. En la mayor parte de las ocasiones el aprendizaje consiste simplemente en determinar un conjunto de pesos sinápticos que permita a la red realizar correctamente el tipo de procesamiento deseado (5).

El aprendizaje se puede clasificar en dos categorías; aprendizaje no adaptativo y aprendizaje adaptativo, en dependencia de la forma en que se van a calcular los pesos. En el aprendizaje no adaptativo los pesos se calculan directamente utilizando una expresión matemática que contiene la información almacenada en el conjunto de entrenamiento, mientras que en el aprendizaje adaptativo los pesos se inicializan por lo general en valores aleatorios pequeños y se van reajustando a través de una regla. Siendo este último uno de los más utilizados, aunque inicializar los pesos aleatoriamente puede constituir una dificultad en el tiempo necesario de entrenamiento.

El mecanismo de aprendizaje seleccionado depende de la cantidad de información que se tenga del problema a resolver, existiendo algoritmos para los diferentes tipos de aprendizaje: supervisado, no supervisado o híbrido.

En el Aprendizaje Supervisado le es presentada a la red neuronal un conjunto de patrones de entradas con sus respectivas salidas deseadas, y esta reajusta los pesos de sus conexiones hasta que según los patrones presentados se obtengan las salidas deseadas o una aproximación de las mismas.

En el Aprendizaje no Supervisado le es presentada a la red neuronal un conjunto de patrones de entradas, pero sin las respuestas deseadas, y esta debe ser capaz de reconocer semejanzas, parecido, similitudes o regularidades en el conjunto de patrones presentados, agrupándolos entre sí.

En el Aprendizaje Híbrido le es presentada a la red neuronal un conjunto de patrones de entradas, así como las posibles respuestas que debe proporcionar la red en un momento dado del problema, y a partir de mecanismos supervisados y no supervisados en las diferentes capas, se entrena al modelo para que de la solución.

Existen una gran variedad de modelos neuronales, estos por lo general se clasifican atendiendo a su mecanismo de aprendizaje y su flujo de datos, a continuación, se muestra una taxonomía de los principales modelos neuronales existentes basada en la figura del libro RNA y Sistemas Borrosos (5).

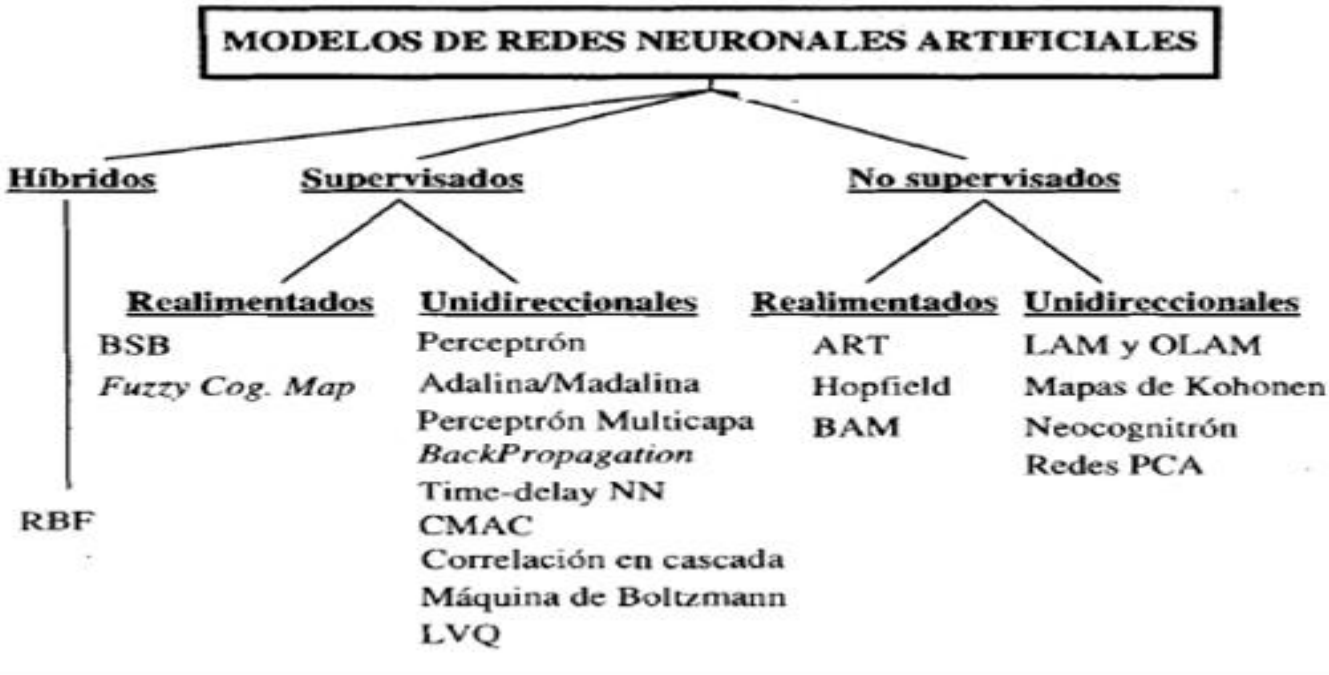

Figura 2: Modelos de Redes Neuronales Artificiales (5).

Uno de los modelos más utilizados es el Perceptrón Multicapa (MLP), debido a la facilidad de uso y la capacidad que tiene como aproximador universal.

1.2 Perceptrón Multicapa o Multi-Layer Perceptron (MLP)

El MLP surge ante la necesidad de resolver el problema de la separabilidad no lineal que presentaba el Perceptrón Simple. Minsky y Papert (7) demostraron que una solución para el tratamiento de algunos problemas de este tipo podía ser la combinación de varios perceptrones simples (aparición de las capas ocultas). Aunque estos autores no lograron encontrar un recurso que permitiera la adaptación de los pesos, pues no era posible utilizar en esta nueva propuesta la regla de aprendizaje del Perceptrón Simple.

Lo anterior permitió que Rumelhart, Hinton y Wiliams en 1986 (8), presentaran una manera de retropropagación de los errores medidos en la salida de la red hacia las neuronas ocultas, dando lugar a la llamada regla delta generalizada. A partir de este momento con la demostración dada por diferentes autores, del Perceptrón Multicapa como aproximador universal, en el sentido de que cualquier función continua en un espacio R^n puede aproximarse con un Perceptrón Multicapa, con al menos una capa oculta de neuronas,

este se convirtió en un modelo matemático útil a la hora de aproximar o interpolar relaciones no lineales entre datos de entrada y salida (5).

A pesar de las muchas facilidades que presenta el modelo descrito tiene una serie de limitaciones que se deben tener en cuenta:

1. Si la red es mal entrenada, las salidas pueden ser imprecisas.
2. Si en el proceso de entrenamiento, se alcanza un mínimo local, este se detiene, aunque no se haya conseguido la tasa de convergencia determinada.
3. El tiempo necesario de entrenamiento es directamente proporcional al número de variables identificadas en el problema.
4. Se puede caer en un sobreentrenamiento, lo que puede dificultar el nivel de generalización de la red.

1.3 Aprendizaje o entrenamiento *en el MLP*

El entrenamiento en un modelo neuronal es de gran importancia, ya que es el que va a permitir que la red funcione adecuadamente. Existen disímiles técnicas de entrenamiento para un MLP por lo que a continuación se analizan algunas de estas con el fin de poder establecer las diferencias y similitudes entre ellas.

Retropropagación o Backpropagation (BP)

La propagación hacia atrás de errores o retropropagación (BP) es un método de cálculo del gradiente utilizado en algoritmos de aprendizaje supervisado, utilizados para entrenar redes neuronales artificiales. El método emplea un ciclo propagación – adaptación de dos fases. Una vez que se ha aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas siguientes de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas.

Backpropagation (BP) trabaja en dos fases, la primera es la propagación de los valores de la variable de entrada hacia la salida, y la segunda la retropropagación del error, que permite el ajuste de los pesos de las conexiones.

En la primera fase una vez que le es presentada a la red un patrón de entrenamiento, se genera una salida, a partir de la regla de propagación, utilizando por lo general funciones de activación tipo sigmoidea. En la segunda fase se compara la salida obtenida con la deseada, calculándose una señal de error para cada salida. Las salidas del error se propagan en sentido inverso hasta la capa de entrada, minimizando el mismo mediante el descenso por el gradiente, existiendo un gradiente respecto a los pesos de la capa de salida y

otro a los de la capa oculta. Finalmente, los pesos son reajustados a través de las derivadas de las diferentes funciones de activación, teniendo en cuenta que la capa oculta solo recibe una parte de la señal total del error, en dependencia de la contribución relativa que haya aportado cada neurona a la salida (10).

BP presenta dos esquemas para su aprendizaje, aprendizaje por lotes (*batch*) y aprendizaje en serie (online). El primero se encarga de presentar todos los patrones y luego actualizar los pesos, mientras que el segundo actualiza los pesos a medida que se va presentando cada patrón.

Aunque el verdadero BP es el que se ejecuta por lotes, el aprendizaje en serie es habitualmente empleado en aquellos problemas en los que se dispone de un muy numeroso conjunto de patrones de entrenamiento (compuesto por cientos o miles de patrones), en el que habría mucha redundancia en los datos. Si se emplease en este caso el modo por lotes, el tener que procesar todos los patrones antes de actualizar los pesos demoraría considerablemente el entrenamiento (5).

En el aprendizaje en serie se debe tener presente que el orden en la presentación de los patrones debe ser aleatorio, puesto que si siempre se siguiese un mismo orden el entrenamiento estaría viciado en favor del último patrón del conjunto de entrenamiento, cuya actualización, por ser la última, siempre predominaría sobre las anteriores (9).

La ventaja principal de este algoritmo es que proporciona buenas soluciones a diferentes tipos de problemas. Entre sus desventajas se encuentran: lentitud de convergencia, sobreentrenamiento y la no garantía de alcanzar el mínimo global de la función error.

Para minimizar el efecto que pueden tener las desventajas de BP relacionadas con la rapidez del aprendizaje y la capacidad de generalización, han surgido diferentes modificaciones del algoritmo entre las que se encuentran:

- ***Levenberg-Marquardt***, también conocido como el método de mínimos cuadrados amortiguados (DLS), se utiliza para resolver problemas de mínimos cuadrados no lineales. Al igual que otros algoritmos de minimización numérica, el algoritmo de *Levenberg-Marquardt* es un procedimiento iterativo. Para iniciar una minimización, el usuario debe proporcionar una estimación inicial del vector de parámetros $\boldsymbol{\beta}$. En los casos con solo un mínimo, una conjetura estándar no informada como $\boldsymbol{\beta}^{T} = (1, 1, ..., 1)$ funcionará bien; en los casos con múltiples mínimos, el algoritmo converge al mínimo global solo si la estimación inicial ya es algo cercana a la solución final (11).
- ***Momentum*** es una leve modificación al BP que consiste en tener en cuenta los pesos previos para influir en la dirección actual del movimiento en el espacio de peso, una vez que los pesos comienzan a moverse en una dirección particular en el espacio de peso, tienden a continuar moviéndose en esa

dirección. Aplicar esta variación a una MLP puede ayudar a “saltar” un mínimo local o incluso a acelerar el proceso de aprendizaje (12).

- **Validación Cruzada (*Cross Validation*)**, en este método, se realiza el entrenamiento en el 50% del conjunto de datos dado y el 50% restante se utiliza para comprobar el entrenamiento. El principal inconveniente de este método es que se usa en el entrenamiento en el 50% del conjunto de datos, es posible que el 50% restante de la información contenga alguna información importante que estamos dejando fuera del entrenamiento. Para evitar esto existen variantes como el LOOCV (*Leave One Out Cross Validation*) y el *K-Fold Cross Validation*, que consisten en dividir el conjunto de datos en bloques e ir entrenando la mayor parte de estos y comparándolos con la menor parte, hasta que se hallan usado todos los bloques tanto para entrenar como para comparar los resultados (13).

En algunas aplicaciones veremos que las redes neuronales artificiales pueden caer en un problema que se conoce como sobreentrenamiento, en donde la red no es capaz de responder adecuadamente ante datos de entrada diferentes a los datos que se utilizaron para el proceso de aprendizaje, pero que hacen parte del problema que se quiere solucionar. La red se especializa o “memoriza” un conjunto de datos determinados con un error muy pequeño. Este sobreentrenamiento trae como consecuencia que el error de test o verificación de la red sea mucho mayor que el de entrenamiento lo cual es indeseable cuando la red está trabajando en la solución de una aplicación específica.

Una solución para eliminar el sobreentrenamiento es bajar la complejidad del modelo de red neuronal, reduciendo el número de neuronas en la capa oculta. Esta solución no siempre es la mejor, pues podemos caer en el fenómeno opuesto denominado sub-entrenamiento (14) .

1.4 Aplicación de técnicas de Inteligencia Artificial en el entrenamiento de un MLP

1.4.1 Algoritmos Genéticos

Los algoritmos genéticos (AG) funcionan entre el conjunto de soluciones de un problema llamado fenotipo, y el conjunto de individuos de una población natural, codificando la información de cada solución en una cadena, generalmente binaria, llamada cromosoma. Los símbolos que forman la cadena son llamados genes. Cuando la representación de los cromosomas se hace con cadenas de dígitos binarios se le conoce como genotipo. Los cromosomas evolucionan a través de iteraciones, llamadas generaciones. En cada generación, los cromosomas son evaluados usando alguna medida de aptitud. Las siguientes generaciones (nuevos cromosomas), son generadas aplicando los operadores genéticos repetidamente, siendo estos los operadores de selección, cruzamiento, mutación y reemplazo (15).

Una de las principales desventajas radica en la dificultad para definir la representación del problema. La forma para especificar las soluciones debe ser tolerante a cambios aleatorios sin producir errores fatales o resultados sin sentidos. La manera generalmente utilizada es definir a los individuos como listas de números, ya sean binarios, reales o enteros.

La dificultad para definir una función *fitness*, pues el proceso no es trivial, ya que debe permitir encontrar el mejor desempeño y que éste signifique una mejor solución para un problema determinado, una definición deficiente o inadecuada puede ser incapaz de resolver el problema o bien dar solución a un problema distinto.

El cuidado que requiere en la selección de elementos claves como; tamaño de la población, ritmo de mutación, cruzamiento y el tipo de selección, pues si el tamaño de la población es muy pequeño, es posible que el AG no sea capaz de abarcar todo el espacio de posibles soluciones o si el ritmo de mutación, o el método de selección no es el adecuado, la población puede entrar en una catástrofe de errores (16).

1.4.2 Algoritmos basados en Colonia de Hormigas

Estos algoritmos, que están basados en una colonia de hormigas artificial, son agentes computacionales que trabajan de manera conjunta para poder comunicarse a través de rastros de feromonas artificiales. Los algoritmos OCH (Optimización por Colonia de Hormigas) son programas constructivos: en cada iteración del algoritmo cada hormiga construye una solución al problema a través de un grafo. Cada arista representa las posibles opciones que el insecto puede tomar y tiene asociada el siguiente tipo de información:

- Información heurística: en ésta se mide la preferencia heurística que tienen las hormigas para moverse de un nodo a otro. El camino recorrido de un nodo a otro es una arista. Esta información no es modificada durante la ejecución del algoritmo.
- Información de los rastros de feromonas artificiales: en ésta se mide la deseabilidad aprendida en el movimiento de un nodo a otro, lo cual busca imitar la feromona real que depositan las hormigas naturales. Este tipo de información es modificada mientras que se ejecuta el algoritmo dependiendo de las soluciones encontradas por los insectos (17).

Sus principales desventajas radican:

1- No realiza un análisis de las variables relevantes.
2- Cada hormiga representa un diseño arquitectónico diferente, generando a través de inicializaciones aleatorias que engloban tanto la definición de las unidades ocultas, como los vectores de pesos y las entradas a la red.

3- Es sensible a la determinación de los parámetros iniciales: función de costo, regla para la liberación de feromonas, funciones de probabilidad para la modificación de los parámetros pesos y unidades ocultas.

1.4.3 Probabilidad de Presentación Dependiente del Error (PPDE)

Una de las vías más simples de adaptar la selección de patrones al proceso de aprendizaje es directamente tomar la probabilidad de presentación de un ejemplo proporcional a su error de salida. Los ejemplos aún son presentados en orden aleatorio, pero no con una probabilidad de presentación uniforme. La probabilidad de presentación p^u del ejemplo u es proporcional a su error ε^u. Esto permite una frecuencia de selección mayor para los ejemplos que producen mayor error. Durante el entrenamiento, las probabilidades p^u, deben ser actualizadas periódicamente debido a que la eficiencia de la red y los pesos cambian en cada iteración (18).

1.4.4 Repetición Dependiente del Error (RDE)

Esta variante presenta los ejemplos proporcionales a su error, pero a diferencia de PPDE, los patrones no son seleccionados aleatoriamente, sino de acuerdo a un esquema determinista. El procedimiento sería el siguiente: Presentar inicialmente cada ejemplo *u* a la red y guardar su error ε^u , así como el ejemplo $u_{\max}$ con el valor mayor de error $\varepsilon_{\max} = \max \varepsilon^u$. Los patrones son repetidos siguiendo el siguiente esquema:

El conjunto de ejemplos es examinado W veces, seleccionando en el i-ésimo examen todos los ejemplos *u* para los cuales $\varepsilon^u > i\varepsilon_{\max}/W$, hasta el W examen. El número de veces que un patrón es presentado depende de su error de salida. Después otra inicialización con todos los ejemplos es realizada y el procedimiento es repetido. Los valores de error ε^u son ajustados siempre que un patrón es seleccionado para el entrenamiento. Una vez que su error está por debajo de cierto criterio, no es presentado hasta la próxima inicialización (18).

1.4.5 Repetición Dependiente del Mayor Error (RDME)

La idea central de esta estrategia es presentar al entrenamiento siempre el ejemplo que produzca mayor error. En este caso los patrones no son seleccionados aleatoriamente por lo que sería una estrategia de tipo determinista. Es necesario la introducción de una estructura adicional donde mantener ordenados los ejemplos de entrenamiento o simplemente implementar un proceso de búsqueda que permita en cada iteración seleccionar el ejemplo con el mayor nivel de error. Inicialmente se realiza un ciclo BP completo para cada ejemplo *u* del conjunto de entrenamiento lo que permitiría determinar para cada uno de ellos su

error. Posteriormente se realizan M ciclos BP, presentando en cada uno de ellos el ejemplo que tenga mayor error $\varepsilon_{\max} = \max \varepsilon^u$. Después el proceso de inicialización es repetido para garantizar que todos los ejemplos sean presentados al entrenamiento por lo menos cada vez que se realiza dicha inicialización (18).

1.5 Algoritmos Conceptuales

Los algoritmos de agrupamiento conceptual se componen de dos tareas fundamentales, las cuales no tienen necesariamente que ser independientes ni realizarse en un orden determinado:

1. La estructuración o determinación extensional: se lleva a cabo el proceso de agrupar entidades, en el que se determinan grupos a partir de una colección de objetos, esto no es más que la enumeración de los objetos que componen los grupos.
2. La caracterización o determinación intencional: se determina el concepto de cada grupo de la estructuración, las propiedades que caracterizan el agrupamiento.

Los algoritmos de agrupamiento conceptual se pueden dividir en dos grandes grupos, a saber, los algoritmos incrementales y los no incrementales. Los algoritmos incrementales basan su funcionamiento en la adaptación de los agrupamientos (o conceptos) con los nuevos objetos que se le van presentando, es decir, cada vez que llega un nuevo objeto mediante una cierta estrategia éste es clasificado en los agrupamientos ya existentes o se crean nuevos agrupamientos. Por otro lado, los algoritmos no incrementales estructuran una muestra de objetos sin presuponer que éstos llegan de uno en uno.

En los trabajos de (19) y (20) se realiza un análisis crítico de diferentes algoritmos de agrupamiento conceptual, atendiendo a sus características y funcionamiento, destacándose entre sus principales resultados los siguientes:

1. La principal desventaja de los algoritmos conceptuales de tipo incremental es la dependencia del resultado (la estructuración) en función del orden de presentación de los objetos al algoritmo mientras que en los de tipo no incremental se determina el número de las agrupaciones de manera aleatoria, lo que constituye una dificultad en la vida real, debido al desconocimiento de los posibles agrupamientos en ciertos tipos de problemas.
2. Los algoritmos que no pertenecen al enfoque lógico combinatorio del reconocimiento de patrones construyen sus conceptos en función de criterios probabilísticos o estadísticos, por lo que su interpretación puede ser engorrosas para personas no especializadas en esas áreas del conocimiento.
3. Los algoritmos LC-Conceptual y RGC (ambos pertenecientes al RLCP) construyen sus conceptos en función de propiedades lógicas, basadas en los rasgos de los objetos en estudio. Además de que

no requieren que sean especificados a priori el número de agrupamientos. La dificultad que poseen estos algoritmos, es la complejidad computacional en el cálculo de los testores típicos.

La descripción de los conceptos en función de propiedades lógicas basadas en los rasgos de los objetos, puede constituir una variante para la explicación del significado que puede tener una neurona de la capa oculta en función de las entradas de la red. En la bibliografía consultada se aprecia que la selección de la cantidad de capas ocultas y sus correspondientes neuronas han sido abordadas con diferentes técnicas de la IA, sin embargo, no dejan claro el significado que puede tener una neurona de la capa oculta en función de sus entradas.

Un aspecto a tener en cuenta para afrontar esta situación, sería asociar las neuronas de la capa oculta a los conceptos que se generan en la parte intencional del algoritmo de agrupamiento. Otro elemento a revisar sería asociar la importancia de cada concepto (o neurona) a la inicialización del peso entre las conexiones de la capa oculta y la de salida.

1.5.1 Algoritmo LC-Conceptual

El algoritmo LC-Conceptual fue propuesto en el año 2001 por Shulcloper y Trinidad (19), está basado en los conceptos de la clasificación no supervisada en el enfoque lógico-combinatorio y retoma algunas ideas propuestas por Michalski para generar conceptos, interpretables por los especialistas, en términos del conjunto de rasgos original.

El algoritmo LC-Conceptual incluye dos etapas:

Etapa de estructuración extensional: se construyen los agrupamientos de objetos basándose en la semejanza entre los mismos y utilizando un criterio de agrupamiento.

Etapa de estructuración intencional o conceptual: se construyen las propiedades (conceptos) que caracterizan a cada agrupamiento de objetos utilizando el operador de refunión condicionada y los testores típicos.

Paso 1: Se determinan, los criterios de comparación por rasgos, la función de semejanza y el criterio de agrupamiento. Se estructura la muestra inicial en función de estos parámetros, siendo considerado cada grupo formado como una clase. Posteriormente se procede a calcular todos los testores típicos para cada clase.

Paso 2: Se forma una matriz de entrenamiento ME o matriz de aprendizaje MA a partir de la matriz inicial MI y de los agrupamientos o clases K_1, …, K_c, luego se calcula el conjunto de los testores típicos de ME_T = $\{t_1, \ldots, t_p\}$. Para cada clase K_i, i = 1..., c se emplea el operador de refunión condicionada para construir los l-

complejo (entiéndase por l-complejo como el producto lógico de todos los rasgos que cubren un concepto o el conjunto de todos los valores diferentes que pueden tomar los objetos de dicho concepto y a su vez construir los conceptos) (19).

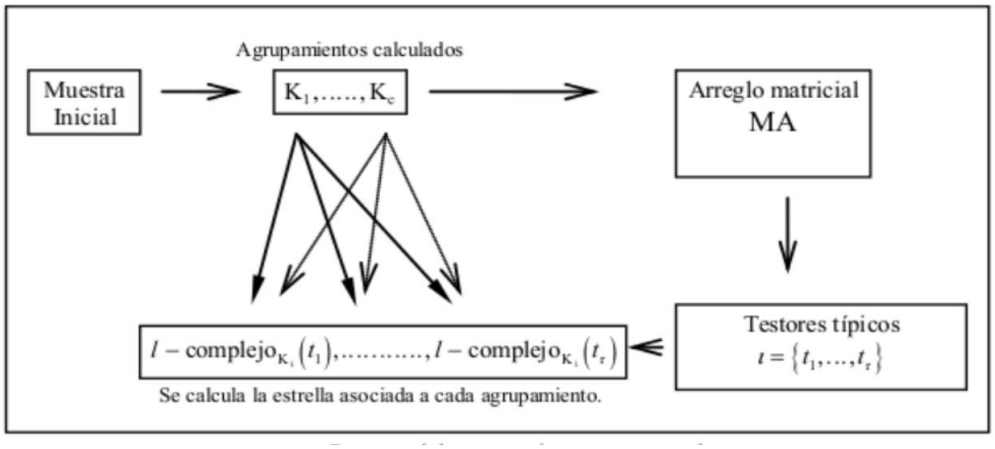

Figura 3: Proceso del agrupamiento conceptual utilizando el algoritmo LC-Conceptual. Fuente: (20)

El operador de refunión condicionada (RUC): forma los l-complejos garantizando que los l-complejos de cada agrupamiento no satisfacen a ningún objeto de otros grupos, pero no de manera independiente, sino en combinación con el resto de las variables.

Este algoritmo presenta las siguientes limitaciones:

1. Se aplica el operador de refunión condicionada, el cual garantiza que el concepto de un agrupamiento no sea satisfecho por ningún objeto de otro agrupamiento, pero entonces, no logra que este concepto cubra a todos los objetos de dicho agrupamiento.
2. No permite el empleo de otro algoritmo de reducción de rasgo que no sea el testor típico.
3. No incluye la regla de generalización como parte del proceso del algoritmo.

1.5.2 Algoritmo RGC

El algoritmo conceptual RGC fue propuesto en el año 2004 por Aurora Pons Porrata *(21)* está basado en los conceptos de la clasificación no supervisada del Reconocimiento Lógico Combinatorio de Patrones y retoma algunas de las ideas planteadas por Michalski para generar los conceptos, en función del conjunto de rasgos original.

El algoritmo RGC cuenta con dos etapas como se muestra en la figura 4:

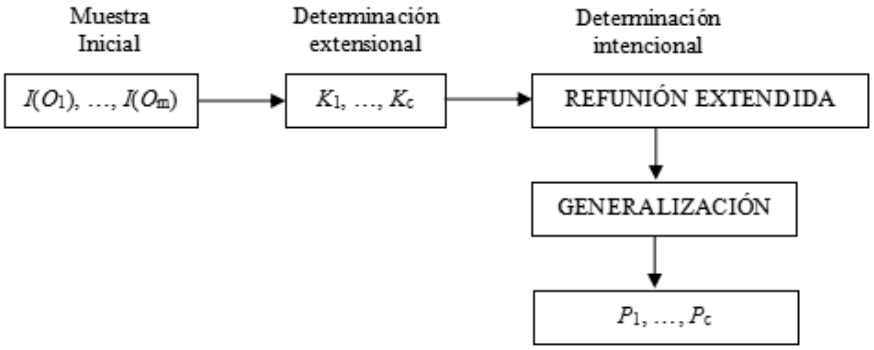

Figura 4: Proceso del agrupamiento conceptual del RGC. Fuente: (20)

I Etapa de determinación extensional: Primeramente, se realizan los agrupamientos, usando los criterios de comparación entre los rasgos, la función de semejanza entre objetos y un criterio de agrupamiento para generar la estructuración por clases. Para ello, podrá emplearse cualquier algoritmo de clasificación no supervisada como grupo de apoyo.

II Etapa de determinación intencional: Después de haber aplicado la etapa extensional se emplea el operador de refunión extendida para el cálculo de los l-complejos, se aplican las reglas de generalización y se construyen los conceptos que corresponde a cada clase K_i.

Los pasos para aplicar el algoritmo RGC son *(20)*:

1- Se construye las propiedades (conceptos) que caracterizan a cada grupo de objetos.

2- Se forma una matriz de entrenamiento a partir de MI y de los grupos K_1, …, K_c, luego se calcula el conjunto de apoyo en este caso es los testores típicos de $ME_T = \{t_1, \ldots, t_p\}$.

3- Para cada clase K_i, i = 1, …, c se calcula la estrella G_T ($K_i \setminus K_1$, ..., K_{i-1}, K_{i+1}, ..., K_c) que estará formada por l- complejos.

4- Para el cálculo de los l-complejos se emplea el operador de Refunión Extendida (RUE). Luego se aplican las reglas de generalización.

5- Los conceptos que caracterizan a cada agrupamiento K_i serán los l-complejos obtenidos en el paso anterior.

El operador de refunión extendida (RUE): forma los conceptos garantizando que los l-complejos de cada agrupamiento no satisface a ningún objeto de otros grupos; pues si un nuevo objeto no puede ser añadido a ninguno de los l-complejos existentes se crea, al menos, un nuevo l-complejo que lo cubre.

1.5.3 Comparación entre los algoritmos conceptuales

En (22) y (20) se presenta un estado del arte en la temática de los algoritmos conceptuales, se realiza una comparación entre ellos. En la presente investigación se presentan solo la comparación del LC- Conceptual y RGC, ya que el grupo de algoritmos conceptuales es lo que compete a la misma y es muy importante señalar sus diferencias para justificar la elección más adecuada para la realización del presente trabajo.

Los criterios que los autores tienen en cuenta para establecer la comparación entre los algoritmos son:

La complejidad computacional: la métrica de medición se va a establecer en fácil, medio, alta y muy alta. En este caso los dos algoritmos tienen una complejidad computacional muy alta establecido expresado en la investigación.

La estructuración: la métrica se establece en conjunto o no conjunto. El algoritmo conjunto está definido por las instrucciones ordenadas y finitas que permite realizar una tarea. El no conjunto es el que no cumple con las instrucciones ordenadas y tiene distintas salidas.

Tipo de atributo: se definen los tipos de atributos que puede tener los rasgos de la base de conocimiento que se puede trabajar. En (22) se considera que una de las características fundamentales, a los efectos de su investigación, radica en el tipo de atributo que admiten los algoritmos y en este sentido LC- Conceptual y RGC (basados en el Reconocimiento Lógico Combinatorio de Patrones) son los de mejores resultados pues permiten trabajar con datos mezclados e incompletos, lo cual es muy frecuente en los problemas prácticos.

Regla de generalización: se basa en tener en cuenta las reglas de generalización. En este caso según sus creadores (23) y (21) en el caso del LC-Conceptual no lo tiene incluido y el RGC sí lo tiene presente.

Excluyente: se analiza si en la forma de construir los conceptos el cual indica que no son satisfechos por ningún objeto de otra clase. Por lo que los dos algoritmos lo tienen presente cuando se construyen los conceptos.

Caracterizante: se refiere a que cubren a todos los objetos de la clase que describen. Esto ocurre en la forma de construir los l-complejos.

Sin embargo, la diferencia radica en que el RGC cumple con las propiedades de ser excluyente y caracterizante a diferencia del LC-Conceptual que solo puede ser excluyente, no garantiza que sea

caracterizante, al igual el algoritmo RGC tiene incluida las reglas de generalización y el LC-Conceptual no, como se muestra en la Tabla 1.

Tabla 1: Comparación de algoritmos. Fuente (elaboración propia)

Criterio de comparación	Algoritmos Conceptuales	
	LC-Conceptual	**RGC**
Complejidad	Muy alta	Muy alta
Estructuración	Conjunto	Conjunto
Atributos	Mezclados incompletos	Mezclados incompletos
Regla de generalización	No	Si
Excluyente	Si	Si
Caracterizante	No	Si

Considerando los algoritmos estudiados y presentados en la investigación se concluye que a pesar de las disímiles ventajas que presentan otros algoritmos que no son el RGC, se descartan para su implementación ya sea por la naturaleza aleatoria a la hora de seleccionar los rasgos, los pesos, la cantidad de neuronas o incluso el método de selección de una solución acertada. Mientras que el RGC aporta una mayor consistencia y certeza a la hora de presentar sus datos gracias a sus propiedades de ser caracterizantes, excluyentes y permitir la disminución cuantitativa del vector de entrada del entrenamiento.

1.6 Tecnologías a utilizar

1.6.1 Lenguaje de programación

El lenguaje Java, en su versión 8, está diseñado para permitir el desarrollo de aplicaciones portátiles, de elevado rendimiento, para el más amplio rango de plataformas informáticas posible. Al poner a disposición de todo el mundo, aplicaciones en entornos heterogéneos, las empresas pueden proporcionar más servicios y mejorar la productividad, las comunicaciones y colaboración del usuario final y reducir drásticamente el costo de propiedad, tanto para aplicaciones de usuario, como de empresa. Java se ha convertido en un valor impagable para los desarrolladores, ya que les permite (24):

- Escribir software en una plataforma y ejecutarla virtualmente en otra.
- Crear programas que se puedan ejecutar en un explorador y acceder a servicios Web disponibles.
- Desarrollar aplicaciones de servidor para foros en línea, almacenes, encuestas, procesamiento de formularios HTML y mucho más.
- Combinar aplicaciones o servicios que utilizan el lenguaje Java para crear aplicaciones o servicios con un gran nivel de personalización.

En la presente investigación se utilizará como lenguaje de implementación ya que posee compatibilidad con otros softwares que permiten el entrenamiento de redes neuronales y permite la importación de alguna de sus librerías.

1.6.2 Entorno de desarrollo de software

Se utiliza NetBeans 8.2: teniendo en cuenta que es una solución muy completa para programar en *Java*, aunque soporta otros lenguajes como *C/C++, JavaScript, Ruby, Groovy, Phython y PHP*. El proyecto de NetBeans está apoyado por una comunidad de desarrolladores y ofrece una amplia documentación y recursos de capacitación y consta de una gran cantidad de módulos. Además, puede ser usado tanto por programadores con poca experiencia como por expertos y permite trabajar sobre el mismo código a más de un programador. Además, puede ejecutarse en *Windows, OS X, Linux, Solaris* y otras plataformas que soportan una *Java Virtual Machine* compatible. Existe además un número importante de módulos para extender el NetBeans IDE. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

También está disponible *NetBeans_Platform*; una base modular y extensible usada como estructura de integración para crear grandes aplicaciones de escritorio. Empresas independientes asociadas, especializadas en desarrollo de software, proporcionan extensiones adicionales que se integran fácilmente en la plataforma y que pueden también utilizarse para desarrollar sus propias herramientas y soluciones (25).

Se utilizará como IDE de desarrollo ya que es ligero en cuanto a requisitos computacionales y posee alta compatibilidad con bibliotecas externas que pueden ser utilizadas para la implementación del algoritmo.

1.6.3 Herramienta MatLab 2017a

La herramienta MATLAB (abreviatura de ***MAT****rix* ***LAB****oratory*, "laboratorio de matrices") es una herramienta de software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). La herramienta es multiplataforma ya que se puede usar en varios sistemas operativos como Unix, Windows, Mac OS X y GNU/Linux (26).

La herramienta de MATLAB se desarrolla en un lenguaje de programación propio. Este lenguaje es interpretado, y puede ejecutarse tanto en el entorno interactivo, como a través de un archivo de script (archivos *.m). Este lenguaje permite operaciones de vectores y matrices, funciones, cálculo lambda, y programación orientada a objetos.

MATLAB puede llamar funciones y subrutinas escritas en C o Fortran. Se crea una función envoltorio que permite que sean pasados y devueltos tipos de datos de MATLAB. Los archivos objeto dinámicamente

cargables creados compilando esas funciones se denominan "MEX-files", aunque la extensión de nombre de archivo depende del sistema operativo y del procesador.

Esta herramienta es utilizada como asistente matemático en la mayoría de los casos, pero otras de las aplicaciones que tiene es el gran aporte que brinda a la inteligencia artificial en el área de las redes neuronales. El mismo cuenta con un *Toolbox* de Redes Neuronales Artificiales que facilita el trabajo en la presente investigación.

En esta investigación se empleará MatLab en su versión 2017a para el diseño de la arquitectura e inicialización de los pesos sinápticos del MLP.

1.6.4 Herramienta Weka 3-7-10

Weka es una plataforma de software para el aprendizaje automático y la minería de datos escrito en Java y desarrollado en la Universidad de Waikato. Es software libre distribuido bajo la licencia GNU-GPL. El paquete Weka contiene una colección de herramientas de visualización y algoritmos para análisis de datos y modelado predictivo, unidos a una interfaz gráfica de usuario para acceder fácilmente a sus funcionalidades (29).

Algunas de sus características más destacadas son:

- Está disponible libremente bajo la licencia pública general de GNU.
- Es muy portable porque está completamente implementado en Java y puede correr en casi cualquier plataforma.
- Contiene una extensa colección de técnicas para preprocesamiento de datos y modelado.
- Es fácil de utilizar por un principiante gracias a su interfaz gráfica de usuario.

Se utiliza Weka en la investigación para el entrenamiento de un MLP en la fase de preexperimentación ya que contiene herramientas de visualización y análisis de datos que son muy útiles a la hora de establecer comparaciones entre diferentes resultados.

1.7 Conclusiones parciales

Como resultado de la investigación y el análisis bibliográfico realizado durante el desarrollo del presente capítulo:

- Han sido expuestos los elementos fundamentales sobre el entrenamiento en las redes neuronales artificiales, específicamente el Perceptrón Multicapa.
- Se confeccionó un marco teórico que permite avanzar a la realización de la propuesta de solución.

- Se trataron los aspectos sobre las diferentes técnicas de Inteligencia Artificial aplicadas al entrenamiento del Perceptrón Multicapa.
- Se seleccionó las herramientas que se emplearán durante la implementación de la solución que se propone:
    - Lenguaje de programación: Java 8
    - Entorno de desarrollo: Netbeans 8.2
    - Herramienta: MatLab 2010ª
    - Herramienta: Weka 3-7-10

Luego de la realización de un estudio sobre las herramientas, los diferentes algoritmos usados en el entrenamiento de un MLP y el análisis de los elementos fundamentales que puede aportar el algoritmo RGC; quedan definidas las bases para el desarrollo un algoritmo capaz de aportar en la reducción del tiempo de entrenamiento de un MLP, así como aumentar su grado de generalización.

Capítulo 2: Propuesta de Solución

Introducción

En el presente capítulo se describirán los pasos fundamentales necesarios para el desarrollo de los algoritmos a utilizar para la disminución en el entrenamiento del Perceptrón Multicapa, así como el aumento en su capacidad de generalización, utilizando como centro el algoritmo RGC. Entre sus elementos principales se encuentran la selección de las dimensiones, la selección de rasgos, así como la asignación de los pesos sinápticos.

Se propone el desarrollo de un algoritmo que sea capaz de reducir cuantitativamente la dimensión del vector entrante, de identificar las variables relevantes mediante la selección de testores típicos, así como la asignación de los pesos sinápticos de cada una de las neuronas de la capa de entrada y finalmente con la presentación del patrón al entrenamiento hacer posible llegar a un resultado más eficiente en el entrenamiento de un MLP, así como un mayor grado de generalización de la red.

2.1 Descripción de la propuesta de solución

Se considera un Perceptrón Multicapa compuesto por una capa de entrada, una capa oculta y una capa de salida en la que sus neuronas se encuentran conectadas con conexiones pesadas entre sí. Para el entrenamiento de un MLP con estas características primeramente es necesario determinar el número de neuronas correspondientes a cada una de las capas y posteriormente los dos conjuntos de pesos sinápticos iniciales; los que se encuentran entre la capa de entrada y la capa oculta y aquellos entre la capa oculta y la capa de salida.

El cálculo de los elementos para el diseño y posterior entrenamiento de una red con las características anteriormente descritas se realiza utilizando el algoritmo conceptual RGC, teniendo en cuenta sus dos etapas. La primera etapa (determinación extensional) se encarga de agrupar los objetos por clases, mientras que su segunda etapa (determinación intencional) calcula los conceptos que caracterizan a los grupos de clases. La propuesta de solución la cual cuenta con tres fases y la fase de entrenamiento como se muestra en la figura 5.

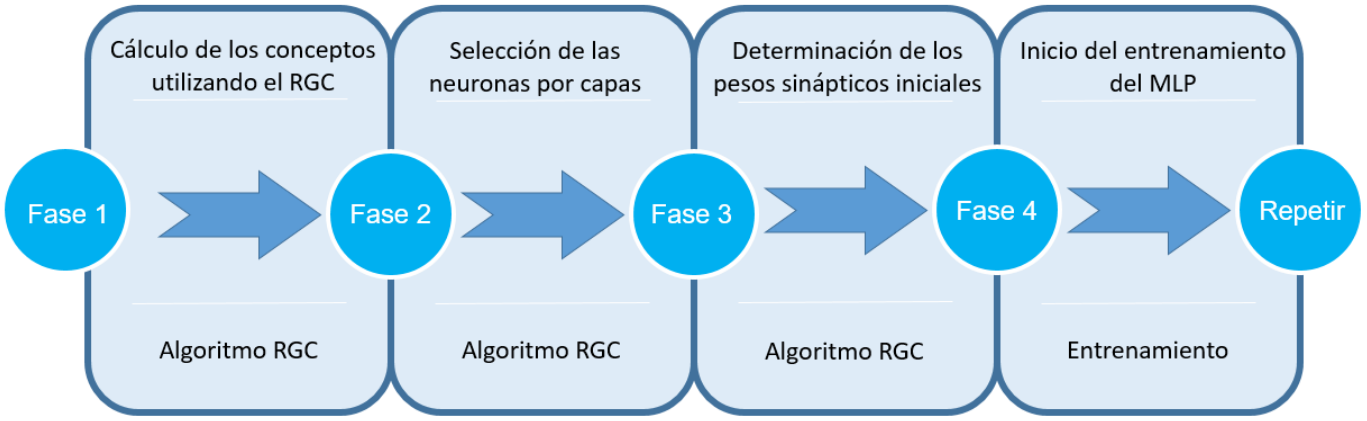

Figura 5: Fases de la solución propuesta. Fuente: (elaboración propia).

2.2 Aplicación del algoritmo RGC

La primera fase "Cálculo de los conceptos" comienza con la determinación extensional del algoritmo RGC en caso de trabajar con un problema no supervisado, agrupando los objetos en clases (algoritmo 1), luego basándose en la etapa intencional se calculan los conceptos (algoritmo 2, 3 y 4), los cuales representan las características distintivas de los objetos que conforman los grupos. Si el problema es supervisado se parte directamente de la determinación intencional (algoritmo 2, 3 y 4) puesto que ya las clases se encuentran agrupadas.

En la segunda fase "Selección de las neuronas por capa" se determina el número de neuronas pertenecientes a cada una de las capas del MLP (entrada, oculta y salida), en correspondencia con los resultados arrojados luego de aplicar el RGC.

En la tercera fase "Determinación de los pesos sinápticos iniciales" como su nombre lo indica se definen los pesos de las conexiones entre la capa de entrada y la oculta (algoritmo 1) así como entre la capa oculta y la de salida (algoritmo 2) (20).

Para esto se utilizará la siguiente simbología:

Fase 1, algoritmo 1:

MI : Matriz inicial

β : Función de semejanza

Π : Criterio de agrupamiento

ME : Matriz estructurada en c agrupamientos (K_c).

Fase 1, algoritmo 2:

ME: Matriz de Entrenamiento

TT: Conjunto de Testores Típicos

Fase 1, algoritmo 3:

TT: Conjunto de Testores Típicos

TT': Testor típico de mayor peso informacional

$\mathrm{T_i}$: Número de testores donde aparece rasgo i.

$|\mathrm{T}|$: Número de testores.

$|\mathrm{T_i}|$: Número de rasgos que forman el testor $\mathrm{T_i}$

Fase 1, algoritmo 4:

TT': Testor típico de mayor peso informacional

ME: Matriz de entrenamiento

Fase 3, algoritmo 1:

$\mathrm{T_i}$: Número de testores donde aparece rasgo i.

$|\mathrm{T}|$: Número de testores.

$|\mathrm{T_i}|$: Número de rasgos que forman el testor $\mathrm{T_i}$

N_e: Neuronas de entrada

N_o: Neuronas ocultas

$W_{0eo:}$ Vector inicial de pesos entre las neuronas de la capa de entrada y la capa oculta.

Fase 3, algoritmo 2:

N_o: Neuronas ocultas

N_s: Neuronas de salida

$W_{0os:}$ Vector inicial de pesos entre las neuronas de la capa oculta y la de salida.

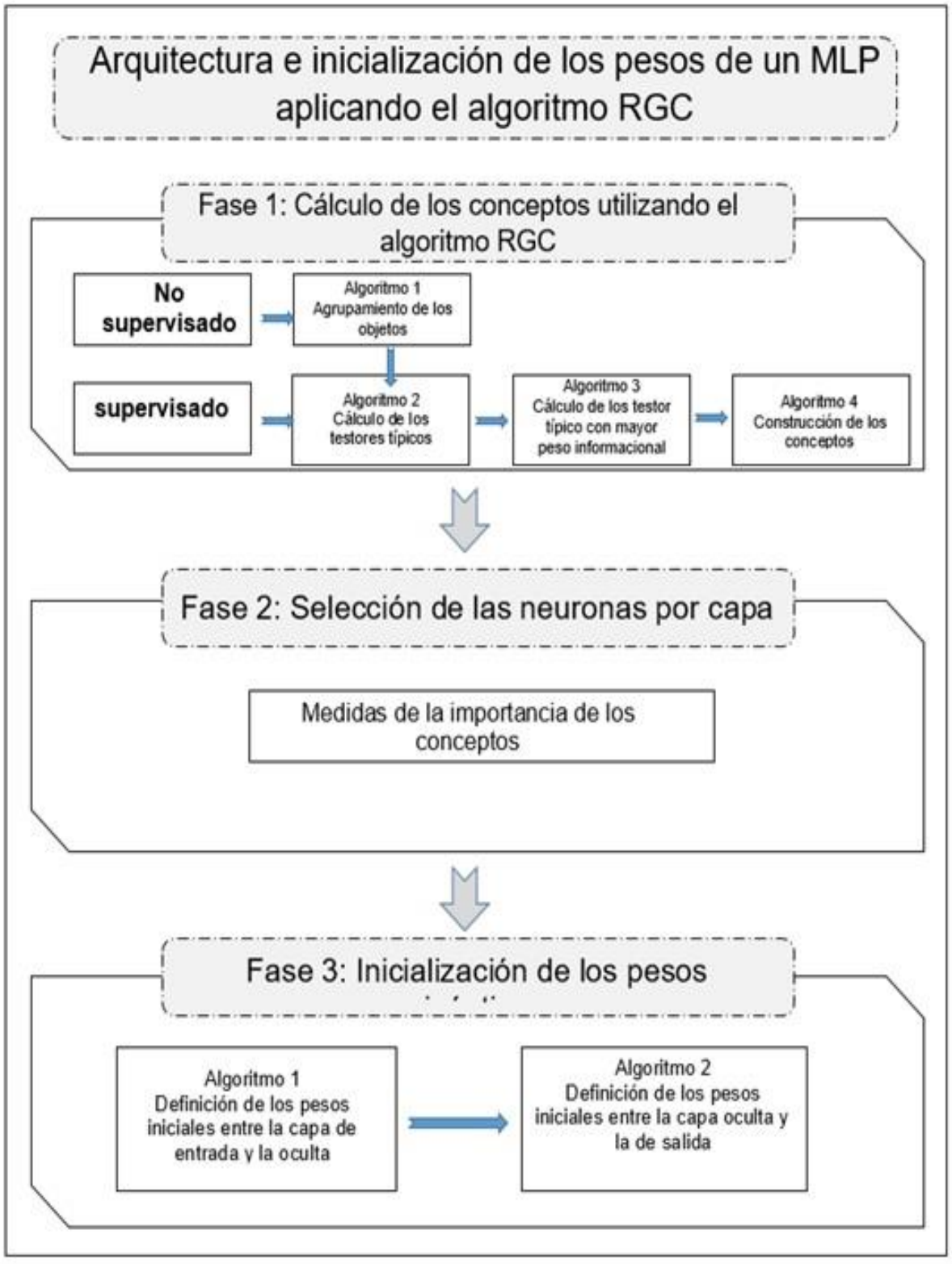

Figura 6: Fases del algoritmo RGC (Fuente: elaboración propia)

Fase 1: Cálculo de los conceptos utilizando el RGC

En esta fase se aplica el algoritmo RGC para tomar el testor típico de mayor peso informacional y el número de conceptos formados.

Objetivo: Calcular los testores típicos y seleccionar el de mayor peso informacional y los conceptos asociados a cada clase.

Independientemente del tipo de problema a resolver lo constituye el planteamiento formal del mismo, con lo que se persigue precisar el objetivo, así como determinar las fuentes de información con las cuales se trabaja. La presencia del especialista del área de aplicación es de vital importancia, pues sus criterios deciden en gran medida varios de los aspectos a tener en cuenta, tales como: cuáles son los rasgos y su importancia, cómo se comparan las variables y los objetos, determinar la forma en que se representan los objetos de estudio, si se tienen clases, si son disjuntas o no, duras o difusas. Para este proceso se propone adoptar la metodología descrita en (19).

Del enfoque lógico-combinatorio para problemas de clasificación sin aprendizaje, se conoce que los objetos pueden ser representados en una matriz inicial (MI), $MI = \{I(O_1), I(O_2), \dots, IO_m\}$ conjunto de descripciones de los objetos $O_1, O_2, \dots, O_m$ de un universo U, dadas como $I(O_n) = (x_1, (O_i), \dots, x_n(O_n))$, como se puede observar en la figura 7. Para cada x_i se tiene asociado un conjunto de valores admisibles M_i $i = 1, \dots, n$, consecuentemente el espacio de representación inicial de los objetos, no es otra cosa que $M_1 \times M_2 \times , \dots, \times M_n$ el producto cartesiano de los conjuntos admisibles de valores de los rasgos $x_1, .. x_n$. Sobre M_i se define un criterio de comparación de valores:
C_i: donde G es un conjunto dado$(G = [0,1], G = \{0,1\}, G = \{1, \dots, k\}\ y\ otros\ )$.

	x_1	x_2	...	x_n
O_1	$x_1\ (O_1)$	$x_2\ (O_1)$	...	$x_n\ (O_1)$
O_2	$x_1\ (O_2)$	$x_2\ (O_2)$	...	$x_n\ (O_2)$
⋮		⋮		
O_m	$x_1\ (O_m)$	$x_2\ (O_m)$	...	$x_n\ (O_m)$

Figura 7: Matriz inicial (Fuente: elaboración propia)

Cada rasgo constituye una variable aleatoria con valores discretos (nominales u ordinales) o continuos, y se admite la ausencia de información, representada por el símbolo *. En dependencia de la naturaleza del rasgo pueden utilizarse diferentes criterios de comparación tales como los mostrados en las ecuaciones 2 a 5 o pueden construirse nuevas funciones que no necesariamente son booleanas.

Ecuación 1: Criterio de comparación igualdad estricta

$$C_s\left(X_s(O_i), X_s(O_j)\right) = \begin{cases} 1 \quad si\ X_s(O_i) = X_s(O_j)\ \vee\ X_s(O_i) = *\vee\ X_s(O_j) = * \\ 0\ en\ otro\ caso \end{cases}$$

Ecuación 2: Intervalos de valores semejantes

$$C_s\left(X_s(O_i), X_s(O_j)\right) = \begin{cases} 1 \quad si\ X_s(O_i), X_s(O_j)\ \in\ \left[a_p, a_{p+1}\right] \vee\ X_s(O_i) = *\vee\ X_s(O_j) = * \\ 0\ en\ otro\ caso \end{cases}$$

Ecuación 3: Umbral de error admisible de semejanza

$$C_s\left(X_s(O_i), X_s(O_j)\right) = \begin{cases} 1 \quad si\ \left|X_s(O_i) - X_s(O_j)\right| \le \varepsilon_s\ \vee\ X_s(O_i) = *\vee\ X_s(O_j) = * \\ 0\ en\ otro\ caso \end{cases}$$

Ecuación 4: Conjunto de valores semejantes

$$C_s\left(X_s(O_i), X_s(O_j)\right) = \begin{cases} 1 \quad si\ X_s(O_i), X_s(O_j)\ \in\ A_p\ \vee\ \ X_s(O_i) = *\vee\ X_s(O_j) = * \\ 0 \quad en\ otro\ caso \end{cases}$$

Entre las descripciones de objetos se define una función de semejanza:

$$\beta: (M_1 \times, \dots, \times M_n)^2 \longrightarrow \beta$$

A partir de MI y β se puede construir una matriz que refleje las relaciones de semejanza entre todos los objetos sujetos a estudio. A esta matriz se le llama matriz de semejanza (MS) y es:

$$MS = \left|\left|\beta(I(O_i), I(O_n))\right|\right|_{m \times n}$$

MS es simétrica y $\beta(I(O_i), I(O_n) = 1, i = 1, \dots, m.$

La función de Semejanza β determina una medida numérica del grado de similitud de un objeto con respecto al otro teniendo en cuanta las similitudes entre los rasgos. Esta función unida a la MI son usadas para hallar la matriz de semejanza.

Para determinar la similitud entre rasgos, es importante considerar la naturaleza de los mismos, en dependencia de esta, se utilizan diferentes criterios de comparación (27).

Umbral de Semejanza: El umbral es la cota inferior de los posibles valores de semejanza existentes para cada uno de los casos recuperados con respecto al nuevo caso. Su función consiste en restringir el intervalo en caso de que se desee lograr una mayor precisión en la obtención de los casos más semejantes.

La magnitud $\beta_0 \in \Delta$ (Δ=[0,1], Δ={0,1}, Δ={1,..,k}, y otros) se denomina umbral de semejanza y puede ser calculada, por ejemplo, según las ecuaciones 6, 7 y 8 (28).

Ecuaciones 6 ,7 y 8 Cálculo del umbral de semejanza:

$$a)\ \beta_0 = \frac{2}{\mathrm{m}(\mathrm{m}-1)} \sum_{\mathrm{i}=1}^{\mathrm{m}-1} \sum_{\mathrm{j}=\mathrm{i}+1}^{\mathrm{m}} \beta(\mathrm{I}(\mathrm{O_i}), \mathrm{I}(\mathrm{O_j}))$$

$$b)\ \beta_0 = \max_{\substack{i=1\ldots m-1 \\ i \neq j}} \left\{ \min_{j=i+1\ldots m} \{\beta(I(O_i), I(O_j))\} \right\}$$

$$c)\ \beta_0 = \min_{\substack{i=1\ldots m-1 \\ i \neq j}} \left\{ \max_{j=i+1\ldots m} \{\beta(I(O_i), I(O_j))\} \right\}$$

El criterio de agrupamiento unido a la función de semejanza y a la existencia de otros objetos, es la razón por la cual un objeto va a pertenecer a un agrupamiento o por qué dos objetos pertenecerán a una misma agrupación. De esta manera se puede apreciar que la selección del criterio a usar, es determinante en la calidad de la solución del problema de clasificación no supervisado.

La definición del criterio de agrupamiento debe estar basada en el conocimiento que se tenga sobre el problema en concreto que se está tratando, para poder definir así el tipo de comportamiento entre los objetos a partir de sus semejanzas, que resulte significativo, según el problema en particular. Por tanto, al seleccionar algún criterio de agrupamiento, dado un conjunto de objetos y la función de semejanza, se ha definido indirectamente, la familia de agrupaciones, es decir, la estructura del universo ha sido conformada.

El planteamiento formal de la estructuración de universos, de la clasificación no supervisada, consiste en encontrar un criterio de agrupamiento que responda a los intereses del problema en cuestión.

β_0-semejantes

Dos descripciones (objetos) $I(O_n), I(O_j)$ se denominan β_0 -semejantes si $\beta(I(O_i), I(O_j)) \geq \beta_0$ .Sea un espacio de representación $\Phi = \{\Xi, \beta\}$, sea dado un conjunto de descripciones de objetos $MI =$

$\{I(O_1), I(O_2), \ldots, IO_m\}$, una función de semejanza $\beta: MI \times MI \longrightarrow \Delta$, y un umbral $\beta_0 \in \Delta$ que define la β_0-semejanza entre los elementos de MI.

Criterio agrupacional β_0-conexo

Se dice que $C \subseteq MI, C \neq \emptyset$ es una componente β_0-conexa si y sólo si:

a) $\forall\, O_i\, O_j \in C \in\ O_{i_1}, \ldots, O_{i_q} \in C\ [O_i = O_{i_1} \wedge O_j = O_{i_q} \wedge\ \forall\, p \in \{1, \ldots, q-1\}\, \beta\ (O_{i_p}, \ldots, O_{i_{p+1}}) \geq \beta_0]$

b) $\forall O_i \in MI\left[\left(O_j \in C, \beta\left(O_i, O_j\right) \geq \beta_0\right) \Rightarrow O_i \in C\right]$

c) Todo elemento β_0-aislado es una componente β_0-conexa (degenerada).

La condición a) expresa que para cualquier par de elementos de C existe una sucesión de elementos en C, que empieza en O_i y termina en O_j tales que uno es β_0-semejante al siguiente, b) significa que no existe fuera de C un elemento β_0-semejante a un elemento de C.

Criterio agrupacional β_0-compacto

Se dice que $\mathcal{B}$ (F 9) cMI, $\mathcal{B} \neq$o es un conjunto β_0-compacto si y sólo si:

a) $\forall \mathrm{O_j} \in \mathrm{MI}[\mathrm{O_i} \in \mathcal{B} \wedge \max\limits_{\substack{O_t \in MI \\ O_t \neq O_i}} \{\beta(O_i, O_t)\} = \beta\left(O_i, O_j\right) \geq \beta_0\ ] \Rightarrow O_i \in \mathcal{B}$

b) $[\max\limits_{\substack{O_t \in MI \\ O_t \neq O_i}} \{\beta(O_P, O_i)\} = \beta(O_P, O_t) \geq \beta_0 \wedge O_t \in \mathcal{B}] \Rightarrow O_i \in \mathcal{B}$

c) $|\mathcal{B}|$ es la mínima

d) Todo elemento β_0-aislado constituye un conjunto β_0-compacto (degenerado).

La condición a) expresa que todo elemento de $\mathcal{B}$ tiene en $\mathcal{B}$ al elemento que más se le parece que es β_0-semejante con él. La condición b) significa que no existe fuera de $\mathcal{B}$ un elemento cuyo elemento más parecido que sea β_0-semejante esté en $\mathcal{B}$, la tercera condición c) especifica que $\mathcal{B}$ debe ser el conjunto más pequeño de cardinalidad mayor que 1 (20).

El algoritmo 1 realiza el agrupamiento de los objetos semejantes, a partir de la MI se obtiene la MS y posteriormente se utiliza un umbral de semejanza $\beta_0 \in \Delta$ y un criterio de agrupamiento para conformar la ME.

Algoritmo 1: "Agrupamiento de los objetos"

Entrada: MI, β, Π

$C_s\left(X_s(O_i), X_s\left(O_j\right)\right)$ // Funciones de comparación por rasgos

Salida: ME

Paso 1: Construir la matriz de semejanza utilizando la función de semejanza β.

Paso 2: Calcular el umbral de semejanza utilizando un criterio β_0.

Paso 3: Agrupar siguiendo un criterio agrupacional Π (28).

Como dato de salida del algoritmo 1 se obtiene la ME, la cual está estructurada en c agrupamientos (K_c) de objetos similares, culminando así la primera etapa con la determinación extensional.

Para la construcción de los conceptos, es necesario seleccionar en primer lugar los rasgos determinantes en el problema, esto es realizado mediante el cálculo de los testores, lo que permite la reducción eficiente de la cantidad de rasgos que describen los objetos. A partir de la Matriz de Entrenamiento (ME) se obtiene la Matriz de Diferencias (MD) y posteriormente la Matriz Básica (MB), mediante la cual se hallan los testores. Se aplica un algoritmo para obtener los testores típicos, luego se obtienen los l-complejos de cada agrupamiento con el operador refunión extendida (30), después el operador Generalización (GEN) y finalmente se forman los conceptos para cada clase perteneciente al problema.

La ME es una matriz estructurada en m clases de objetos similares. La MD una matriz booleana que se obtiene de la ME comparando los respectivos valores de los rasgos en objetos de clases diferentes por medio de los criterios de comparación de valores de las variables (31).

La MB es una matriz formada únicamente por filas básicas de la MD, las cuales constituyen el conjunto de testores.

Se entiende por fila básica según (19).

La fila i_t es básica sí y sólo sí en MD no existe fila i_p alguna que sea subfila de i_t.

Sea i_p, i_t filas de MD Se dice que i_p es subfila de $_{it}$ sí y sólo sí:

a) $\forall\forall j\ (ai_tj = 0 \longrightarrow ai_pj = 0)$

b) $\forall\exists j_0\ (ai_tj_0 = 1 \wedge ai_pj_0 = 0)$

Algoritmo 2: "Cálculo de los testores típicos"

Entrada: ME

Salida: TT

Paso 1: Calcular la matriz de diferencias.

Paso 2: Calcular la matriz básica.

Paso 3: Aplicar algoritmo para el cálculo de los TT.

El número de testores típicos para ciertos problemas puede ser muy grande. Esto da como resultado que cada clase podría tener asociado una gran cantidad de conceptos o propiedades, por lo cual, el algoritmo 3 calcula la utilidad de cada testor típico utilizando las propuestas descritas en (27) y (32) para seleccionar el mejor.

Algoritmo 3. Cálculo del testor típico de mayor peso informacional.

Entrada: TT

Salida: TT'

Paso 1: Calcular el peso ε_i de los rasgos X_i que aparecen en la familia de testores típicos según:

$$\varepsilon(x_i) = \alpha F(x_i) + \gamma L(x_i)$$

α>O, β>O y α+β=1. α y β parámetros que ponderan la participación o influencia de frecuencia de aparición y la longitud de los testores típicos respectivamente.

$$\alpha = \beta = 0.5 \qquad p(x_i) = \frac{T_i}{|T|} \qquad L(X_i) = \frac{\sum_{t \in T} \frac{1}{|T_i|}}{|T|}$$

Paso 2: Seleccionar los testores típicos de menor longitud.

Paso 3: Calcular el peso de los testores típicos de menor longitud según:

$$\Psi(t_i) = \sum_{x \in I} \frac{\varepsilon(x)}{|TT|}$$

Paso 4: Ordenar descendentemente los testores típicos según su peso.

Paso 5: Seleccionar el testor típico de mayor peso informacional (Ψ), a partir de un umbral previamente definido.

La importancia de utilizar el testor típico de mayor peso informacional radica en que el mismo no contiene todos los rasgos, sino los más relevantes al problema en cuestión, por tanto cuando se haga referencia al conjunto de rasgos, debe entenderse que son sólo aquellos rasgos presentes en el testor típico con que se trabaje (33).

En el algoritmo 4, se obtienen los l-complejos de cada agrupamiento con el operador Refunión Extendida (34) luego se aplica el operador Generalización (GEN) y finalmente se forman los conceptos para cada clase perteneciente al problema.

Algoritmo 4. Construcción de los conceptos

Entrada: TT', ME

Salida: l-complejos //conceptos para cada clase

Paso 1: Calcular l-complejos

Paso 2: Calcular la estrella G_τ ($K_i \backslash K_1$, ..., K_{i-1}, K_{i+1}, ..., K_c) para cada clase K_i, i = 1,…, *c.*

Paso 3: Aplicar reglas de generalización en dependencia del tipo de variable.

La complejidad computacional de este algoritmo depende de la naturaleza de las variables que describen a los objetos, de los criterios de comparación por rasgos y la función de semejanza entre objetos seleccionados, del criterio de agrupamiento que se utilice y de las características que posean los objetos de un mismo agrupamiento. El operador de mayor complejidad computacional es el de refunión extendida. En el caso peor la complejidad es exponencial. No obstante, los pasos del operador de refunión extendida pudieran optimizarse explotando las características propias de la función de semejanza y del criterio de agrupamiento empleados (34).

Fase 2: Selección de las neuronas por capa

Esta fase corresponde a la estructura física del modelo en función de la cantidad de unidades de procesamiento (neuronas) correspondientes a las capas de entrada, oculta y salida.

Objetivo: Seleccionar el número de neuronas de la capa de entrada y la capa oculta.

Aspectos a examinar:

- Capa de entrada: las neuronas coinciden con los rasgos que componen los conceptos construidos en función de los testores típicos utilizados.
- Capa oculta: las neuronas están asociadas a los conceptos construidos.

- Capa de salida: las neuronas coinciden con el número de clases definidas inicialmente en el problema.

Fase 3: Determinación de los pesos sinápticos iniciales

En esta etapa se establecen los pesos sinápticos iniciales del modelo.

Objetivo: Seleccionar el conjunto de pesos iniciales utilizando métricas definidas como una heurística en función de la importancia informacional de rasgos y conceptos.

Para la inicialización de los pesos se proponen dos algoritmos, el primero referente a los pesos entre las neuronas de la capa de entrada y las correspondientes a la capa oculta, mientras que el segundo define los pesos iniciales entre la capa oculta y la de salida. Esta división está dada debido a que entre las dos primeras capas el que rige el proceso es el rasgo, y por ende se utiliza el peso informacional del mismo en relación con la asociación interpretativa que tiene con los conceptos que definen las neuronas de la capa oculta. Mientras que entre las neuronas de la capa oculta y las de salida la relación está entre los conceptos y las clases, siendo dirigidos por la importancia informacional dada por los conceptos.

Para la determinación de los pesos sinápticos iniciales entre las dos primeras capas el algoritmo 5 utiliza la métrica definida por (27), donde la importancia informacional del rasgo $\varepsilon(x_i)$ se calcula según la ecuación $\varepsilon(x_i) = \alpha F(x_i) + \gamma L(x_i)$ α>O, β>O y α+β=1. α y β parámetros que ponderan la participación o influencia de frecuencia de aparición y la longitud de los testores típicos respectivamente.

$$\alpha = \beta = 0.5 \qquad p(x_i) = \frac{T_i}{|T|} \qquad L(X_i) = \frac{\sum_{t \in T} \frac{1}{|T_i|}}{|T|}$$

T_i : número de testores donde aparece rasgo i.

$|T|$: número de testores.

$|T_i|$: número de rasgos que forman el testor T_i

Algoritmo 1. Definición de $W_{0e/o}$ entre las neuronas de entrada y las ocultas

---

Entrada: N_e, N_o

Salida: W_{0eo}

Paso 1: Calcular el peso de cada rasgo presente en N_e aplicando la ecuación:

$$\varepsilon(x_i) = \alpha F(x_i) + \gamma L(x_i)$$

Paso 2: Inicializar los pesos sinápticos entre la capa de entrada y la capa oculta.

$W_{0eo} = [W_{011}, \ldots, W_{0eo}]$

La inicialización de los pesos sinápticos entre la capa oculta y la de salida es determinada por el algoritmo 6, a partir de las métricas propuestas en (35) que fija la importancia informacional de los conceptos identificados en las neuronas de la capa oculta. Para determinar el peso informacional de los subconceptos y los conceptos se incorpora la semántica del significado de los valores de los rasgos a las ecuaciones con similares propósitos descritas en (27).

El peso informacional de un subconcepto $PI(b_i)$: Se define utilizando la ecuación del peso informacional de un objeto incorporándole la medida denominada importancia informacional del valor del rasgo y se calcula:

$$PI(b_i) = \frac{1}{|k_j|} \cdot \sum_{j=1}^{n} a_j^i \cdot \varepsilon(x_i) \cdot t_{xi}(v_i)$$

Cuando se comparan dos rasgos con distintos valores, la medida $\iota_r(v_i)$ se calcula como la media de ambos.

En la ecuación anterior el subíndice i recorre la cantidad de valores de los rasgos, j la cantidad de clases y $a_j^i = |\{b_t \in k_j | b \; y \; b_t \; son \; semejantes \; en \; el \; rasgo \; x\}|$

Con los valores calculados a nivel de cada subconcepto se aplica la regla del máximo peso para calcular el peso informacional de un concepto.

El peso informacional del concepto $PI\ (P_i)$: Se determina a partir del subconcepto de mayor peso informacional y se calcula:

$$PI(P_i) = \max_{i=1\ldots\gamma} \{PI_j(b)\}$$

Algoritmo 2. Definición de $W_{0o/s}$ entra las neuronas ocultas y las de salidas

---

Entrada: N_o

N_s

Salida: W_{0os}

Paso 1: Determinar los subconceptos asociados a cada concepto

Paso 2: Calcular el peso informacional de cada subconcepto:

$$PI(b_i) = \frac{1}{|k_j|} \cdot \sum_{j=1}^{n} a_j^i \cdot \varepsilon(x_i) \cdot t_{xi}(v_i)$$

Paso 3: Calcular el peso informacional de cada concepto:

$$PI(P_i) = \max_{i=1\ldots\gamma} \{PI_j(b)\}$$

Paso 4: Inicializar los pesos sinápticos entre la capa oculta y la capa de salida.

$W_{0os} = [W_{011}, \ldots, W_{0os}]$

Culminada la segunda fase, la red neuronal se encuentra lista para ser entrenada.

2.3 Entrenamiento del MLP

La definición del MLP propuesto estará en concordancia con la definición de redes neuronales artificiales en término de grafos dado en (36). La figura 8 define el diseño arquitectónico del MLP como un grafo dirigido con las siguientes propiedades:

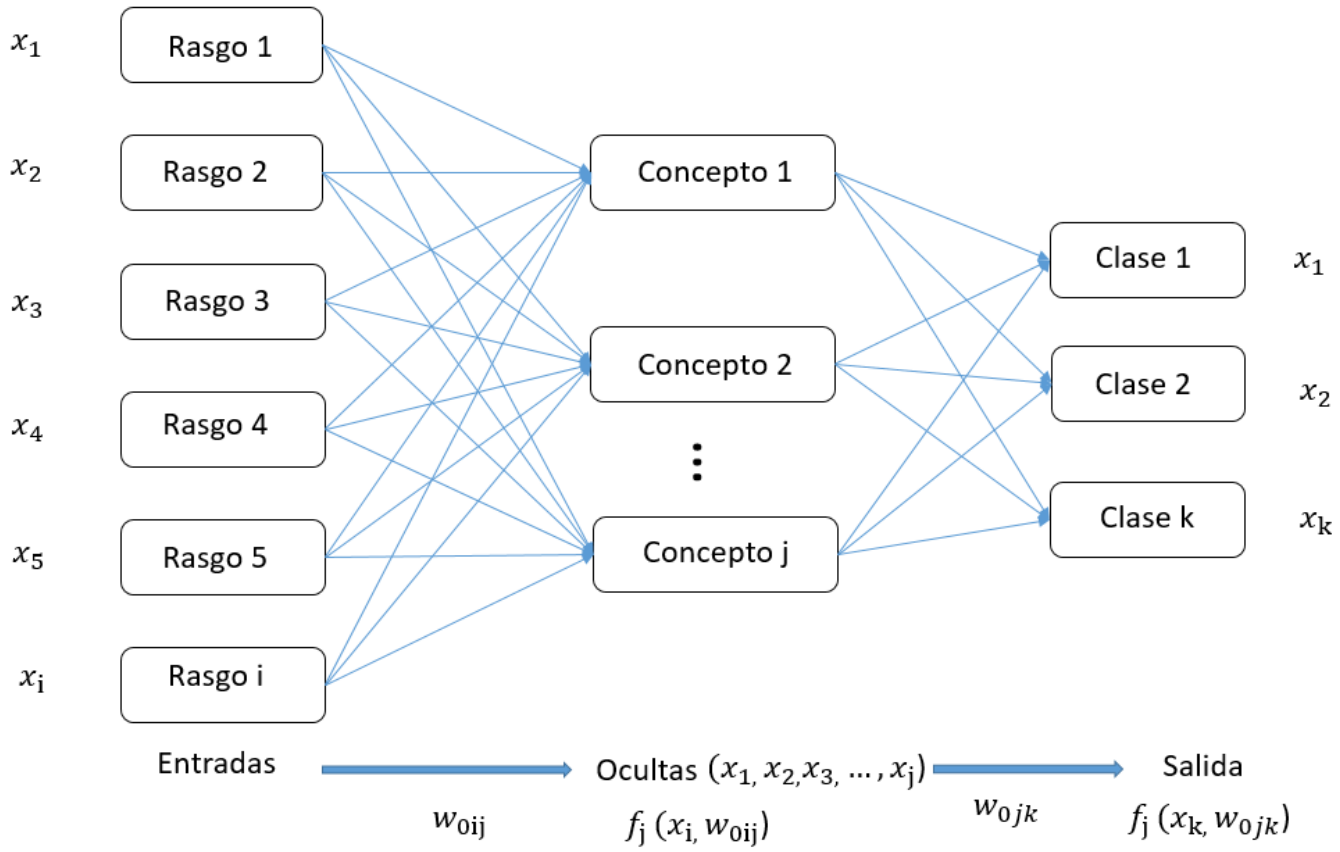

Figura 8: Diseño de la arquitectura e inicialización de los pesos del MLP aplicando la propuesta de solución (Fuente: elaboración propia)

a) A cada nodo i de entrada se la asocia una variable de estado x_i tal que $x_i \in$ {conjunto de rasgos presentes en los conceptos}
b) A cada nodo j oculto se le asocia una variable de estado x_j tal que $x_j \in$ {conjunto conceptos}
c) A cada nodo k de salida se le asocia una variable de estado x_k tal que $x_k \in$ {conjunto de clases definidas en el problema}
d) A cada conexión ij de los nodos i y j se le asocia su correspondiente W_{0ij}.
e) A cada conexión jk de los nodos j y k se le asocia su correspondiente W_{0jk}.

Para cada nodo j y k se define una función f_j (x_i, W_{0ij}) y f_j (x_k, W_{0jk}) respectivamente, que dependerá de los pesos de sus conexiones y de los estados de los nodos i y j a ellos conectados (el Backpropagation exige que la función debe ser diferenciable, y el tipo lo define el rango de valores en los que se presentan las entradas y salidas según la naturaleza del problema).

Una vez concluidas las etapas de la aplicación del algoritmo RGC se procede al entrenamiento adaptando el BP a las nuevas entradas.

Pasos y fórmulas a utilizar para aplicar el algoritmo de entrenamiento (37):

Paso1: Se inicializan los pesos de la red según los resultados del algoritmo RGC.

Paso 2: Se presenta un patrón de entrada: Xp=xp1, xp2, ..., xpn y se especifica la salida deseada: d1, d2, ..., dM (si se utiliza como clasificador, todas las salidas deseadas serán 0, salvo una, que la de la clase a la que pertenece el patrón de entrada).

Paso 3: Se calcula la salida actual de la red, para ello se presentan las entradas y se van calculando la salida que presenta cada capa hasta llegar a la capa de salida, y1, y2, ..., yM.

Luego

• Se calculan las entradas netas para las neuronas ocultas procedentes de las neuronas de entrada.

• Para una neurona j oculta:

$$net_{pj}^{h} = \sum_{i=1}^{N} w_{ij}^{h} x_{pi} + \Theta_{j}^{h}$$

donde el índice h se refiere a magnitudes de la capa oculta (hidden), el subíndice p, al p-ésimo vector de entrenamiento, y j a la j-ésima neurona oculta.

• Se calculan las salidas de las neuronas ocultas:

$$y_{pj} = f_{j}^{h}(net_{pj}^{h})$$

• Se realizan los mismos cálculos para obtener las salidas de las neuronas de salida (capa o: *output*)

$$net_{pk}^{o} = \sum_{j=1}^{L} w_{jk}^{o} y_{pj} + \Theta_{k}^{o}$$

$$y_{pk} = f_{k}^{o}(net_{pk}^{o})$$

Paso 4: Calcula los términos de error para todas las neuronas.

Si la neurona k es una neurona de la capa de salida, el valor de delta es:

$$\delta_{pk}^{o} = (d_{pk} - y_{pk}) f_k^{o'}(net_{pk}^{o})$$

La función f debe ser derivable, para ello existen dos funciones que pueden servir: la función lineal (fk(netjk)=netjk) y la función sigmoidal:

$$f_k(net_{jk}) = \frac{1}{1 + e^{-net_{jk}}}$$

La elección de la función de salida depende de la forma de representar los datos: si se desean salidas binarias, se emplea la función sigmoidal, en otro caso es tan aplicable una como la otra.

Para la función lineal tenemos $dx\, f_k^0 = 1$, mientras que la derivada de una función sigmoidal es:

$$f_k^{o'} = f_k^{o}(1 - f_k^{o}) = y_{pk}(1 - y_{pk})$$

por lo que el término de error para las neuronas de salida queda:

$$\delta_{pk}^{o} = d_{pk} - y_{pk}$$

para la salida lineal, y

$$\delta_{pk}^{o} = (d_{pk} - y_{pk}) y_{pk}(1 - y_{pk})$$

para la salida sigmoidal.

Si la neurona j no es de salida se tiene:

$$\delta_{pj}^{h} = f_j^{h'}(netJ_{pj}^{h}) \sum_k \delta_{pk}^{o} w_{jk}^{o}$$

Donde observamos que el error en las capas ocultas depende de todos los términos de error de la capa de salida. De aquí surge el término de propagación hacia atrás. Para la función sigmoidal:

$$\delta_{pj}^{h} = x_{pi}(1 - x_{pi})\sum_{k}\delta_{pk}^{o} w_{jk}^{o}$$

Donde k se refiere a todas las neuronas de la capa superior a la de la neurona j. Así, el error que se produce en una neurona oculta es proporcional a la suma de los errores conocidos que se producen en las neuronas a las que está conectada la salida de ésta, multiplicados por el peso de la conexión.

Paso 5: Actualización de los pesos.

Se utiliza el algoritmo recursivo, comenzando por las neuronas de salida y trabajando hacia atrás hasta llegar a la capa de entrada, ajustando los pesos de la forma siguiente:

Para la capa de salida:

$$w_{jk}^{o}(t+1) = w_{jk}^{o}(t) + \Delta w_{jk}^{o}(t+1);$$
$$\Delta w_{jk}^{o}(t+1) = \alpha\delta_{pk}^{o} y_{pj}$$

y para los pesos de las neuronas de la capa oculta:

$$w_{ij}^{h}(t+1) = w_{ij}^{h}(t) + \Delta w_{ij}^{h}(t+1);$$
$$\Delta w_{ij}^{h}(t+1) = \alpha\delta_{pj}^{h} x_{pi}$$

En ambos casos, para acelerar el proceso de aprendizaje, se puede añadir un término momento de valor:

$$\beta(w_{jk}^{o}(t) - w_{jk}^{o}(t-1))$$

en el caso de la neurona de salida, y de una neurona oculta:

$$\beta(w_{ij}^{h}(t) - w_{ij}^{h}(t-1))$$

Paso 6: El proceso se repite hasta que el término de error resulta aceptablemente pequeño para cada uno de los patrones aprendidos.

$$E_{p} = \frac{1}{2}\sum_{k=1}^{M}\delta_{pk}^{2}$$

2.4 Conclusiones parciales

Al aplicar el algoritmo RGC en el entrenamiento de un Perceptrón Multicapa queda evidenciado que:

- Mediante la selección de los testores típicos se logra reducir la dimensión de los rasgos iniciales lo que converge en un menor tiempo de entrenamiento.
- El cálculo de los conceptos permite la definición intencional de la cantidad de neuronas en la capa oculta de la red que junto a la selección supervisada de las neuronas en la capa de entrada elimina la aleatoriedad presente en un entrenamiento estándar de un Perceptrón Multicapa.
- La importancia informacional de rasgos y conceptos proporciona una heurística para calcular los pesos sinápticos iniciales de forma determinística.

Capítulo 3: Preexperimentación y resultados.

En el presente capítulo se exponen los resultados preexperimentales al aplicar la propuesta de solución. Se explican las técnicas y criterios a considerar para realizar los mismos, posteriormente se detallan las características de las bases de datos utilizadas, y finalmente los resultados alcanzados al aplicar la propuesta de solución sobre las mismas.

3.1 Descripción de los preexperimentos

Para el análisis de la propuesta del diseño de un MLP se utilizaron 6 bases de datos de reconocimiento internacional, disponibles en el repositorio para aprendizaje automatizado de la Universidad de Irvine, California (38). En la selección se consideran conjuntos de datos con variadas características como: la presencia de rasgos numéricos y no numéricos, la ausencia de información, y diversos en cuanto a la cantidad de rasgos y de objetos.

Se compararon tres diseños de redes MLP: el primero donde se seleccionan las neuronas de entrada directamente proporcional a las variables definidas en el problema, las neuronas de las capas ocultas se escogieron inicialmente teniendo en cuenta la cantidad de variables de entrada + cantidad de salidas / 2 y los pesos iniciales en valores aleatorios, denominado MLP1. El segundo MLP se crea aplicando el algoritmo LC-Conceptual mientras que el tercero se forma aplicando el algoritmo conceptual RGC reflejado en la propuesta de solución.

Para el entrenamiento y prueba de los diseños, se aplicó el método validación cruzada (*k-fold cross validation*). Dicho método consiste en dividir los datos en dos partes; una parte se utiliza como conjunto de entrenamiento para determinar los parámetros del clasificador neuronal y la otra parte, llamada conjunto de validación, se utiliza para estimar el error de generalización, es decir, la tasa de clasificación incorrecta del clasificador con datos diferentes a los utilizados en el proceso de entrenamiento. Se selecciona el 85% de los datos para entrenar la red y el 15% restante para la simulación.

3.2 Descripción de las bases de datos utilizadas

Para los preexperimentos se escogieron 6 bases de datos, 3 supervisadas y 3 no supervisadas con diferentes características en cuanto a cantidad de rasgos. La selección se realiza en función de analizar los resultados del diseño propuesto en problemas con diferentes dominios en las variables de entrada, como se muestra en las tablas 2 y 3.

Tabla 2: Bases de datos supervisadas. Fuente (elaboración propia)

Base de Datos	Cantidad de objetos	Cantidad de rasgos	Cantidad de clases
Wine	178	13	3
Annealing	798	16	38
Glass	214	10	7

Tabla 3: Bases de datos no supervisadas. Fuente (elaboración propia)

Base de Datos	Cantidad de objetos	Cantidad de rasgos
Tae	151	6
Sonar	207	60
Liver-disorders	345	8

3.3 Preexperimentación

Con la experimentación con diferentes bases de datos se puede llegar hasta cierto punto de certeza en la eficacia de la propuesta planteada. El análisis de esta se realiza en función del tiempo de entrenamiento aplicando el algoritmo RGC con respecto al modelo sin aplicar el agrupamiento conceptual y las diferencias entre ambos.

3.3.1 Preexperimento 1

Se realizó el preexperimento 1 que se basa en la comparación de la eficacia del Perceptrón Multicapa sin aplicar el agrupamiento conceptual (MLP1) y el Perceptrón Multicapa aplicando el algoritmo RGC (MLP-RGC) para bases de datos supervisadas. Se evidencia en la tabla 4 que el comportamiento de los tiempos de entrenamiento (T.E.), mostrándose en todos los casos, una reducción en el MLP-RGC. En la figura 9 se muestra en qué porcentaje mejoran los tiempos de entrenamiento el MLP-RGC en cuanto al MLP1.

Objetivo: Constatar la disminución del tiempo de entrenamiento y el aumento en la eficiencia de la clasificación de variables en bases de datos supervisadas en los modelos MLP1 y MLP-RGC.

Tabla 4: Tiempo de entrenamiento del preexperimento 1 para bases de datos supervisadas (Fuente: elaboración propia)

Base de Datos	Cantidad de variables en MLP 1	Cantidad de variables en MLP-RGC	T.E. MLP 1 (s)	T.E. MLP-RGC(s)
Wine	13	3	0.70	0.38
Annealing	38	11	0.84	0.71
Glass	10	4	0.60	0.49

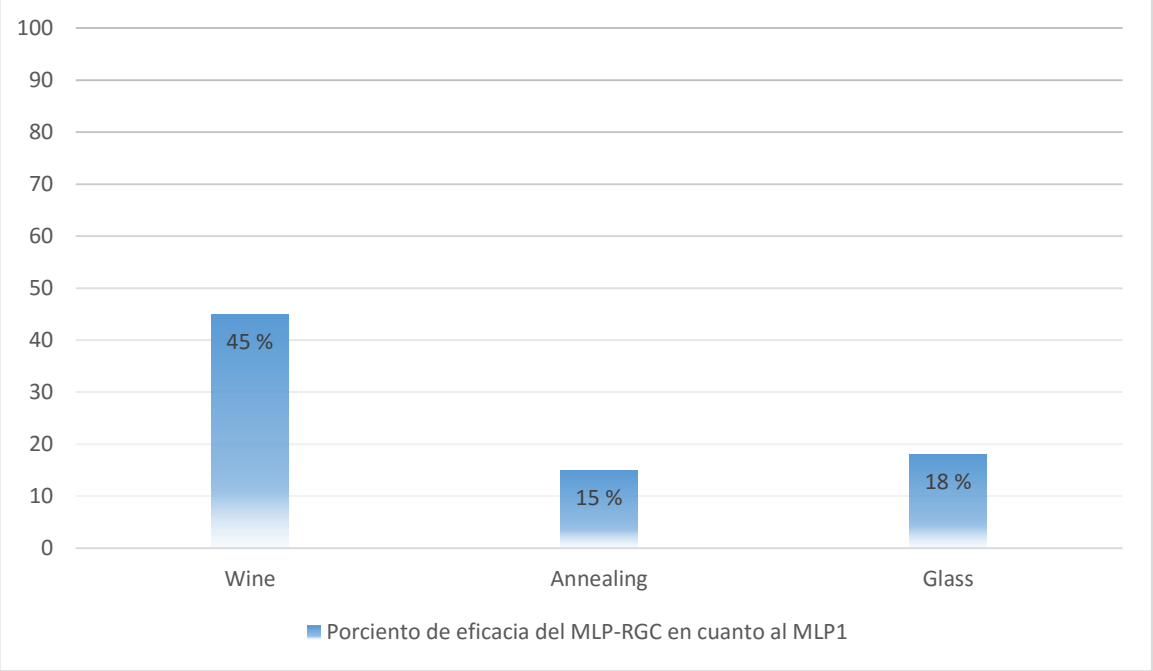

Figura 9: Eficacia del MLP-RGC en cuanto al MLP1. Fuente (elaboración propia)

En la tabla 5 y figura 10, se muestra un aumento en la eficiencia en la clasificación de variables del modelo MLP-RGC en comparación al MLP1 en bases de datos supervisadas lo que se decanta en un aumento en la capacidad de generalización de la red.

Tabla 5: Eficiencia de la clasificación de variables en preexperimento 1. Fuente (elaboración propia)

Base de Datos	Cantidad de variables en MLP 1	Cantidad de variables en MLP-RGC	Eficiencia MLP 1 (%)	Eficiencia MLP-RGC (%)
Wine	13	3	95.02	100
Annealing	38	11	98.51	98.99
Glass	10	4	98.32	100

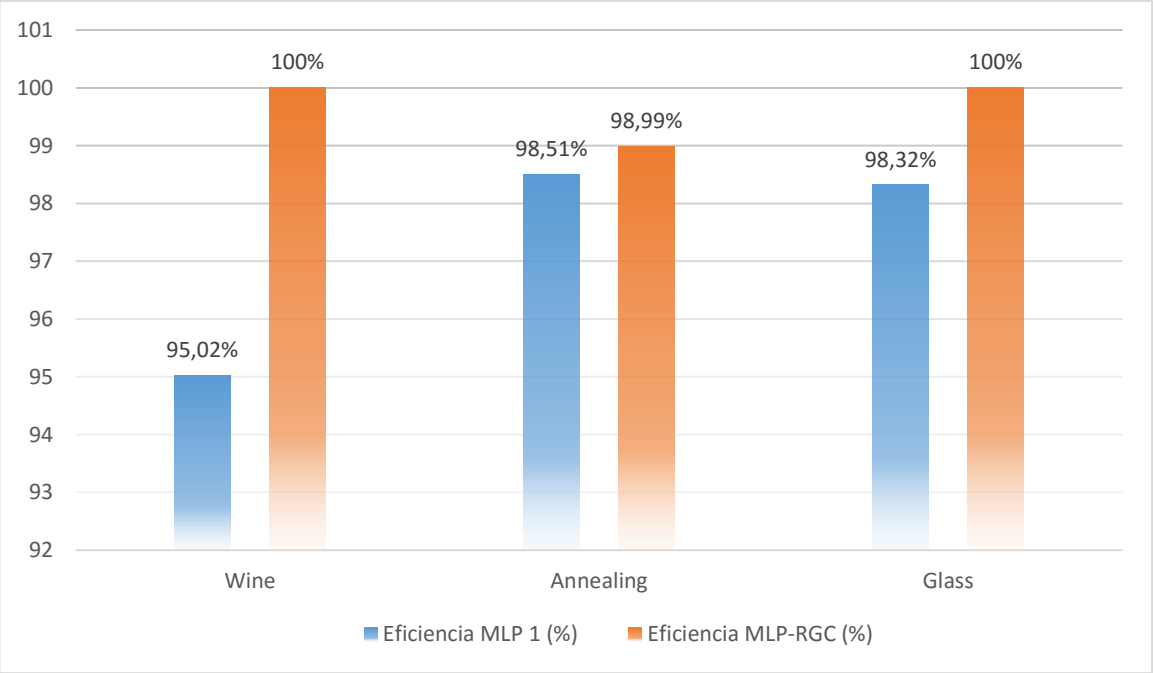

Figura 10: Eficiencia de la clasificación de variables en preexperimento 1. Fuente (elaboración propia)

3.3.2 Preexperimento 2

El segundo preexperimento se basa en la comparación de la eficacia del Perceptrón Multicapa sin aplicar el agrupamiento conceptual (MLP1) y el Perceptrón Multicapa aplicando el algoritmo RGC (MLP-RGC) para bases de datos no supervisadas. Se observa en la tabla 6 que el comportamiento en los tiempos de entrenamiento, disminuyen en el MLP-RGC en los tres casos, específicamente se denota una significativa reducción del tiempo de entrenamiento en el caso de la base de datos Sonar. En la figura 11 se muestra en que porcentaje aumenta la eficacia del modelo MLP-RGC con respecto al MLP1.

Objetivo: Constatar la disminución del tiempo de entrenamiento y el aumento en la eficiencia de la clasificación de variables en bases de datos no supervisadas en los modelos MLP1 y MLP-RGC.

Tabla 6: Datos del preexperimento 2 para bases de datos no supervisadas (Fuente: elaboración propia)

Base de Datos	Cantidad de variables en MLP 1	Cantidad de variables en MLP-RGC	T.E. MLP 1 (s)	T.E. MLP-RGC(s)
Tae	6	4	0.30	0.19
Sonar	60	3	0.91	0.21
Liver-disorders	8	3	0.34	0.17

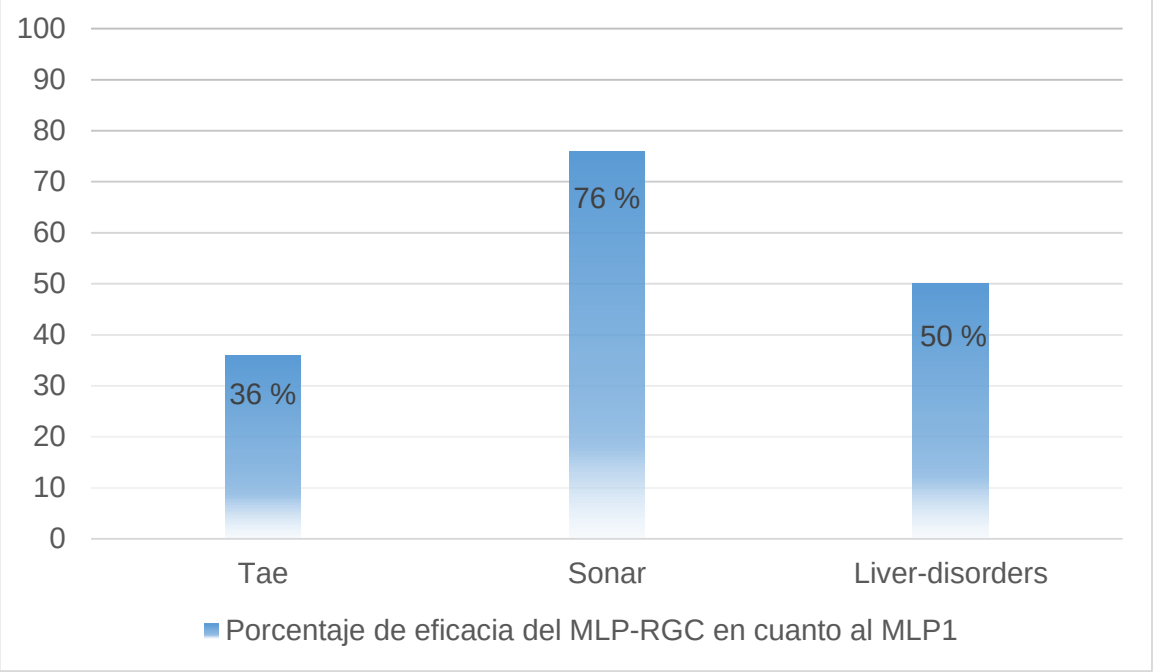

Figura 11: Eficacia del MLP-RGC en cuanto al MLP1. Fuente (elaboración propia)

En la tabla 7 y figura 12 se puede constatar la diferencia en la eficiencia de la clasificación de variables entre los modelos MLP1 y MLP-RGC para bases de datos no supervisadas y como en los tres casos el modelo MLP-RGC es superior al MLP1, especialmente en la base de datos Sonar.

Tabla 7: Eficiencia de la clasificación de los algoritmos MLP1 y MLP-RGC para base de datos no supervisadas. Fuente: (elaboración propia)

Base de Datos	Cantidad de variables en MLP 1	Cantidad de variables en MLP-RGC	Eficiencia MLP 1 %	Eficiencia MLP-RGC %
Tae	6	4	92.50	97.33
Sonar	60	3	78.33	92.14
Liver-disorders	8	3	98.24	99.45

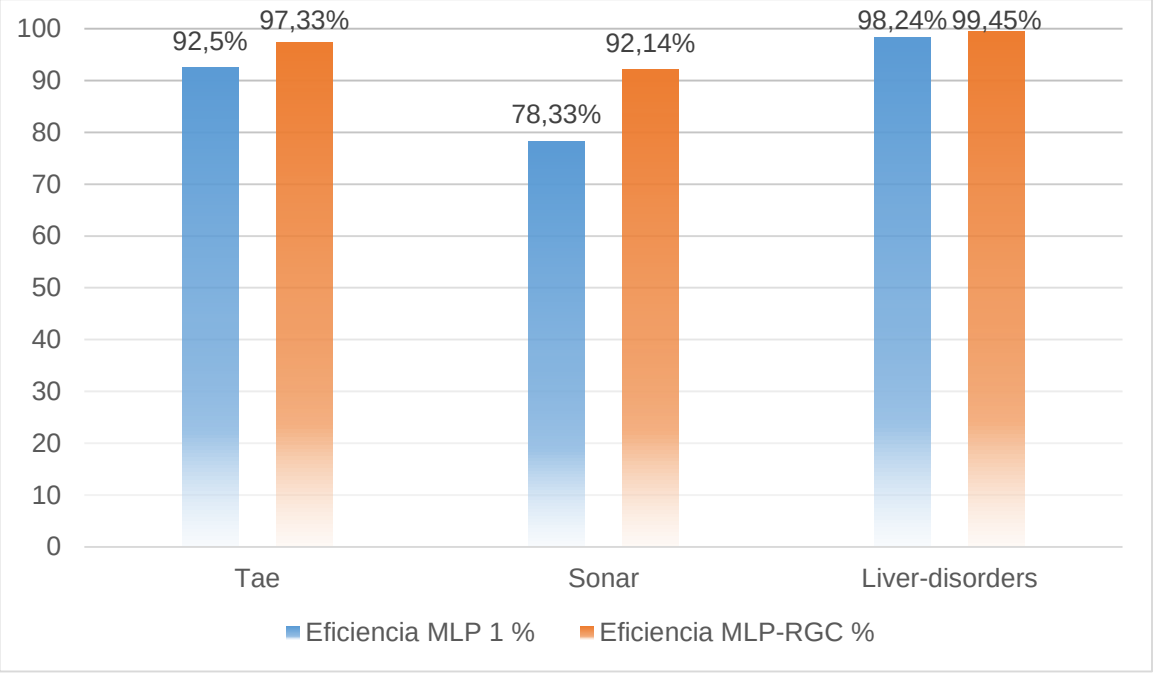

Figura 12: Eficiencia en la clasificación del MLP1 y el MLP-RGC en bases de datos no supervisadas. Fuente: (elaboración propia)

3.3.3 Preexperimento 3

En el tercer preexperimento se compara la eficacia del Perceptrón Multicapa aplicando el algoritmo LC-Conceptual (MLP-LC) y el Perceptrón Multicapa aplicando el algoritmo RGC (MLP-RGC) para bases de datos supervisadas y no supervisadas. En la tabla 8 se visualiza que el comportamiento en los tiempos de

entrenamiento, se comporta mejor en el MLP-RGC en tres de las seis bases de datos de prueba y aunque la diferencia en tiempos de entrenamiento es ínfima, esta sería mayor en caso de tener bases de datos mucho más grandes.

Objetivo: Constatar la disminución del tiempo de entrenamiento y el aumento en la eficiencia de la clasificación de variables en bases de datos supervisadas y no supervisadas en los modelos MLP-LC y MLP-RGC.

Tabla 8: Eficacia del entrenamiento para bases de datos supervisadas y no supervisadas (Fuente: elaboración propia)

Base de Datos	Cantidad de variables en MLP-LC	Cantidad de variables en MLP-RGC	T.E. MLP-LC (s)	T.E. MLP-RGC(s)
Wine	3	3	0.09	0.08
Annealing	11	11	0.20	0.17
Glass	4	4	0.10	0.10
Tae	4	4	0.11	0.11
Sonar	3	3	0.10	0.09
Liver-disorders	3	3	0.10	0.10

En la tabla 9 y figura 13 se muestra la eficiencia en la clasificación de variables de los modelos MLP-LC y MLP-RGC, destacándose el MLP-RGC con una leve ventaja sobre el MLP-LC en las seis bases de datos utilizadas.

Tabla 9: Eficiencia en la clasificación de variables para bases de datos supervisadas y no supervisadas (Fuente: elaboración propia)

Base de Datos	Cantidad de variables en MLP-LC	Cantidad de variables en MLP-RGC	Eficiencia MLP-LC %	Eficiencia MLP-RGC %
Wine	3	3	95.60	100
Annealing	11	11	84.43	98.99
Glass	4	4	99.32	100
Tae	4	4	96.24	97.33
Sonar	3	3	90.23	92.14
Liver-disorders	3	3	90.34	99.45

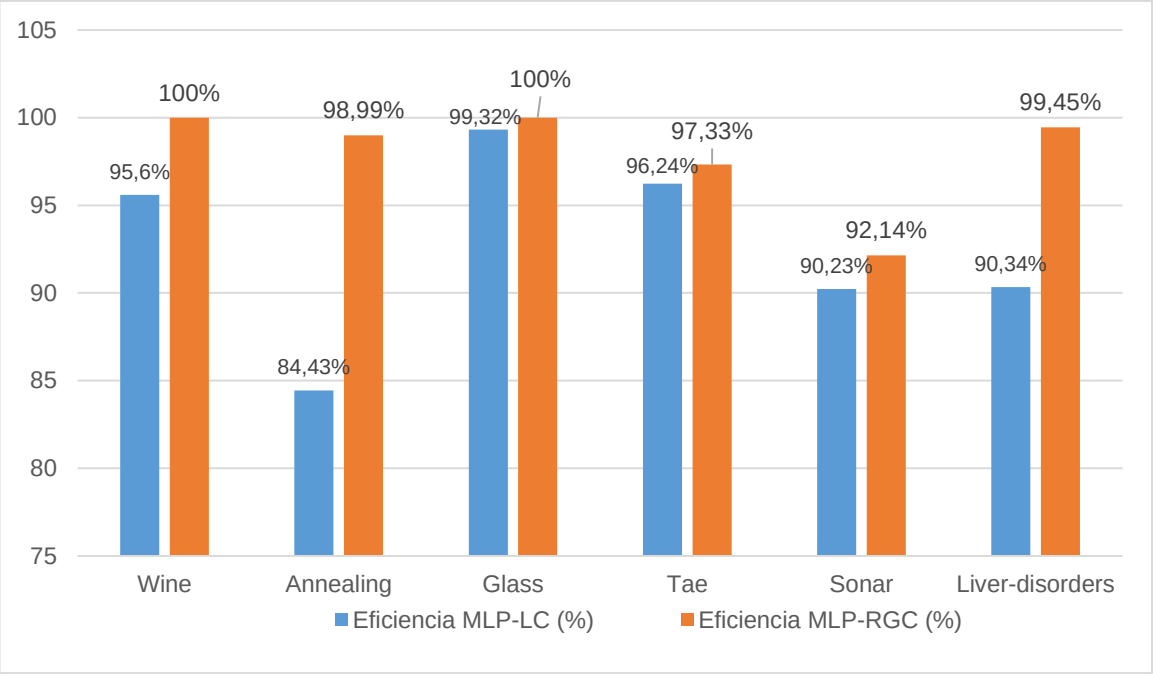

Figura 13: Eficiencia en la clasificación de variables en bases de datos supervisadas y no supervisadas (Fuente: elaboración propia)

3.4 Conclusiones parciales

En las bases de datos utilizadas se evidencia que, al aplicar el algoritmo RGC:

- La eficacia del modelo MLP-RGC es superior a los otros modelos propuestos.
- La eficacia disminuye proporcionalmente con el aumento de variables a clasificar.
- Los tiempos de entrenamiento son ligeramente menores una vez aplicado el algoritmo RGC ya que este disminuye la cantidad de variables de entrada, indicando esto, que a mayor sea la base de datos a utilizar mayor será la diferencia en la reducción del tiempo de entrenamiento entre los diferentes modelos.
- La eficiencia en la clasificación de variables en el modelo MLP-RGC es superior a los otros modelos planteados.

Conclusiones

Al finalizar esta investigación se llega a la conclusión de que se le dio solución al problema de investigación cumpliendo con el objetivo propuesto al principio de la misma:

- Se planteó un algoritmo capaz de reducir la dimensión del vector de entrada de una red neuronal artificial, capaz de obtener un grado de generalización superior a los estudiados en este trabajo.
- Se demostró mediante preexperimentos que la aplicación del algoritmo RGC para el entrenamiento de un MLP, disminuye el tiempo de entrenamiento de la red.
- Se evidenció como el uso de algoritmos conceptuales pueden suponer una solución alternativa a muchos de los problemas que están presentes hoy en el entrenamiento de redes neuronales artificiales, gracias a su escasez de aleatoriedad en la selección de objetos y rasgos relevantes.

Recomendaciones

1- Realizar preexperimentos utilizando lo propuesto en el presente trabajo, con data sets de mayor envergadura que permitan corroborar un comportamiento similar en los resultados.

2- Implementar lo propuesto en el presente trabajo en otros lenguajes de programación para explotar las facilidades que ofrecen unos más que otros y poder explotar las ventajas que puedan ofrecer los mismos a la hora de la obtención de los resultados.

Referencias bibliográficas

1. VIVAS, Hevert. *Optimización en el Entrenamiento del Perceptrón Multicapa*. Tesis de Maestría en Ciencias Matemáticas. Popoyán: Universidad del Cauca, 2014.

2. DOMINGUEZ. Redes Neuronales en java, Herramienta para redes Neuronales ** JavaBrain **. In: .

3. IZAURIETA, Fernando y SAAVEDRA, Carlos. Redes neuronales artificiales. In: *Departamento de Física, Universidad de Concepción Chile*. 2000,

4. BELLO, Rafael. *Curso Introductorio a las Redes Neuronales Artificiales*. Universidad Central de Las Villas: s.n., 1993.

5. MARTÍN-DEL-BRÍO, Bonifacio y SANZ, Alfredo. *Redes neuronales y sistemas borrosos / B. Martín del Brío, A. Sanz Molina ; pról. de Lotfi A. Zadeh.* S.l.: s.n., 2006. ISBN 978-84-7897-743-7.

6. SALAS, Rodrigo. Redes neuronales artificiales. In: *Universidad de Valparaıso. Departamento de Computación*. 2004, Vol. 1.

7. MINSKY, Marvin y PAPERT, Seymour A. *Perceptrons: An Introduction to Computational Geometry*. S.l.: MIT Press, 2017. ISBN 978-0-262-53477-2.

8. RUMELHART, David E, HINTON, Geoffrey E y WILLIAMS, Ronald J. Learning representations by back-propagating errors. In: *nature*. 1986, Vol. 323, no. 6088, pp. 533.

9. BISHOP, Christopher M. *Mixture density networks*. S.l. 1994.

10. HECHT-NIELSEN, Robert. Theory of the backpropagation neural network. In: *Neural networks for perception*. S.l.: Elsevier, 1992. pp. 65-93.

11. MORÉ, Jorge J. The Levenberg-Marquardt algorithm: implementation and theory. In: *Numerical analysis*. S.l.: Springer, 1978. pp. 105-116.

12. SUT, Necdet y MUSTAFA, Senocak. Assessment of the performances of multilayer perceptron neural networks in comparison with recurrent neural networks and two statistical methods for diagnosing coronary artery disease. In: *Blackwell Publishing Ltd.* 2007, Vol. 24, no. 3.

13. KOHAVI, Ron. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: *Ijcai*. S.l.: Montreal, Canada, 1995. pp. 1137-1145.

14. CAICEDO B, Eduardo Francisco y LÓPEZ S, Jesús Alfonso. *Una aproximación práctica a las redes neuronales artificiales* [online]. 1. S.l.: Programa Editorial Universidad del Valle, 2009. ISBN 978-958-765-510-0. Available from: http://revistas.univalle.edu.co/omp/index.php/programaeditorial/catalog/view/64/28/298-1.

15. DEB, Kalyanmoy, PRATAP, Amrit, AGARWAL, Sameer y MEYARIVAN, TAMT. A fast and elitist multiobjective genetic algorithm: NSGA-II. In: *IEEE transactions on evolutionary computation*. 2002, Vol. 6, no. 2, pp. 182-197.

16. MUNOZ, Wenceslao Palma. ESTIMACIÓN DE COSTOS PARA LA FABRICACIÓN DE TUBERÍAS, UTILIZANDO REDES NEURONALES POLINOMIALES CON ALGORITMOS GENÉTICOS. In: . 2014, pp. 77.

17. DORIGO, Marco y BIRATTARI, Mauro. *Ant colony optimization*. S.l.: Springer, 2010. ISBN 0-387-30768-0.

18. DÍAZ-SARDIÑAS, Adolfo y BELLO-PÉREZ, Rafael. Estrategias pedagógicas para la presentación de patrones al entrenamiento de redes neuronales de tipo MLP que utilizan backpropagation como algoritmo de aprendizaje. In: *Revista Facultad de Ingeniería*. 2000, no. 21, pp. 92-101.

19. SHULCLOPER, J. Ruíz, ARENAS, A. Guzmán y TRINIDAD, José Francisco Martínez. Enfoque lógico combinatorio al reconocimiento de patrones. In: *Serie avances en reconocimiento de patrones. Edit. Instituto Politécnico Nacional. México*. 1999, pp. 59-75.

20. GONZÁLEZ, Yunia Reyes, ARCEO, Alfonso Claro, SÁNCHEZ, Natalia Martínez y HERNÁNDEZ, Antonio. Agrupamiento conceptual lógico combinatorio: una alternativa para la toma de decisiones. In: *Inteligencia Artificial*. 2016, Vol. 19, no. 57, pp. 82-96.

21. PORRATA, Aurora Pons. *DESARROLLO DE ALGORITMOS PARA LA ESTRUCTURACIÓN DINÁMICA DE INFORMACIÓN Y SU APLICACIÓN A LA DETECCIÓN DE SUCESOS*. Trabajo de Tesis en opción al grado científico de Doctor en Ciencias Técnicas. S.l.: s.n., 2004.

22. SUÁREZ, Airel Pérez y PAGOLA, José E. Medina. A clustering algorithm based on generalized stars. In: *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. S.l.: Springer, 2007. pp. 248-262.

23. MARTÍNEZ-TRINIDAD, José Fco y SÁNCHEZ-DÍAZ, Guillermo. LC: a conceptual clustering algorithm. In: *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. S.l.: Springer, 2001. pp. 117-127.

24. FAN, Xuelie. *55 New Features in Java SE 8* [online]. 2018. S.l.: s.n. Available from: https://www.oracle.com/technetwork/cn/community/developer-day/2-55-new-features-java-se-8-2202551-zhs.pdf.

25. NetBeans IDE 8.2 Release Notes. In: [online]. Available from: https://netbeans.org/community/releases/82/relnotes.html#new.

26. MATLAB - MathWorks. In: [online]. Available from: https://www.mathworks.com/products/matlab.html. The official home of MATLAB software. MATLAB is the easiest and most productive software environment for engineers and scientists. Try, buy, and learn MATLAB.

27. RUIZ-SHULCLOPER, J. Reconocimiento lógico combinatorio de patrones: teoría y aplicaciones (Tesis en opción al grado científico de Doctor en Ciencias). In: . 2009,

28. RUIZ SHULCLOPER, José. Acerca del surgimiento del Reconocimiento de Patrones en Cuba. In: . 2013, Vol. 7, no. 2, pp. 169–192.

29. Weka 3 - Data Mining with Open Source Machine Learning Software in Java. In: [online]. Available from: https://www.cs.waikato.ac.nz/ml/weka/documentation.html.

30. MARTÍNEZ-TRINIDAD, J.F y RUIZ-SHULCLOPER, José. Algoritmo LC Conceptual Duro. In: *IV Simposio Iberoamericano de Reconocimiento dePatrones*. La Habana, Cuba: s.n., 1999.

31. SHULCLOPER, José Ruiz, ALBA-CABRERA, Eduardo, LAZO-CORTÉS, Manuel S. y GRUPO DE RECONOCIMIENTO DE PTRONES CUBA-MÉXICO. Introducción al Reconocimiento de Patrones. Grupo de Reconocimiento de Patrones Cuba-México. In: . 9 noviembre 1995,

32. REYES GONZÁLEZ, Yunia. *Modelo para la adaptación de las soluciones en un Sistema Basado en Casos utilizando el agrupamiento conceptual*. Tesis de Maestría en Informática Aplicada. S.l.: Universidad de las Ciencias Informáticas, 2014.

33. REYES-GONZÁLEZ, Yunia, ARCEO, A.C., MARTÍNEZ-SÁNCHEZ, Natalia y HERNÁNDEZ-DOMINGUEZ, Antonio. Agrupamiento conceptual lógico combinatorio: Una alternativa para la toma de decisiones. In: *INTELIGENCIA ARTIFICIAL*. 2016, Vol. 19, pp. 82-96.

34. PONS-PORRATA, Aurora. *DESARROLLO DE ALGORITMOS PARA LA ESTRUCTURACIÓN DINÁMICA DE INFORMACIÓN Y SU APLICACIÓN A LA DETECCIÓN DE SUCESOS.* Trabajo de Tesis en opción al grado científico de Doctor en Ciencias Técnicas. S.l.: s.n., 2004.

35. REYES-GONZÁLEZ, Y., RODRÍGUEZ-VALLEJO, L., MARTÍNEZ-SÁNCHEZ, N. y YERO-OSES, E. A. Métricas para la validación de los conceptos en el Reconocimiento Lógico Combinatorio de Patrones. In: . 2016,

36. DEL BRÍO, Bonifacio Martín y MOLINA, Alfredo Sanz. *Redes neuronales y sistemas difusos Bonifacio Martín del BRio, Alfredo Sanz Molina*. S.l.: Alfaomega Ra-Ma, 2001. ISBN 970-15-0733-9.

37. BERZAL, Fernando. Backpropagation. In: [online]. Universidad de Granada, Departamento de Ciencias de la Computación e IA. 2018. Available from: https://elvex.ugr.es/decsai/computational-intelligence/.

38. MERZ, C. J. y MURPHY, P. M. *UCI Repository of Machine Learning Databases. Irvine, University of California*. S.l.: s.n., 1998.