SISTEMA AUTOMÁTICO DE ENTRENAMIENTO DE REDES NEURONALES ARTIFICIALES BASADO EN EL AJUSTE GENÉTICO DE PARÁMETROS Y VARIACIÓN DE ARQUITECTURA

EDGAR ALBERTO MÉNDEZ ORTIZ
JUAN SEBASTIÁN MARIÑO MESA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2008

SISTEMA AUTOMÁTICO DE ENTRENAMIENTO DE REDES NEURONALES ARTIFICIALES BASADO EN EL AJUSTE GENÉTICO DE PARÁMETROS Y VARIACIÓN DE ARQUITECTURA

EDGAR ALBERTO MÉNDEZ ORTIZ
JUAN SEBASTIÁN MARIÑO MESA

TRABAJO DE GRADO PRESENTADO PARA OPTAR EL TÍTULO DE INGENIERO DE SISTEMAS.

DIRECTOR:
MSC. HENRY ARGUELLO FUENTES

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2008

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS

TÍTULO: SISTEMA AUTOMÁTICO DE ENTRENAMIENTO DE REDES NEURONALES ARTIFICIALES BASADO EN EL AJUSTE GENÉTICO DE PARÁMETROS Y VARIACIÓN DE ARQUITECTURA*

AUTOR(ES): MÉNDEZ ORTIZ Edgar Alberto**
MARIÑO MESA Juan Sebastián**

PALABRAS CLAVES: REDES NEURONALES, ALGORITMOS GENÉTICOS, COMPUTACIÓN EVOLUTIVA, CAPAS OCULTAS, FUNCIONES DE ACTIVACIÓN.

DESCRIPCIÓN.

Las redes neuronales artificiales, han mostrado su impacto en diferentes áreas de la ciencia, donde se aplican a problemas de clasificación y predicción de patrones. Aunque esta técnica ha sido utilizada frecuentemente por muchos investigadores, no se tiene aun una teoría sólida que permita identificar la estructura de una red neuronal de forma adecuada; por lo tanto la selección de esta estructura se busca por medio de prueba y error, o en el mejor de los casos con la aplicación de algunas reglas para configuración; además de jugar un papel importante la experiencia adquirida por el experto.

Por lo tanto resulta de gran utilidad desarrollar un software que busque las diferentes configuraciones y los parámetros necesarios para obtener soluciones sobre las cuales el usuario analice y decida la opción que se ajuste a sus necesidades. Se utiliza entonces un algoritmo genético para hallar estos parámetros dada su capacidad como sistema de búsqueda óptima en conjuntos muestrales de gran tamaño. El sistema desarrollado busca en gran medida reducir la complejidad computacional de encontrar una arquitectura de red en un tiempo prudencial.

Este trabajo trata de forma no convencional dos típicos problemas en la optimización de redes neuronales para aprender problemas particulares, el diseño de la arquitectura (capas-neuronas) y las funciones de activación que se deben usar en cada una de estas capas. La red se entrena por métodos basados en retro-propagación que el usuario tiene la posibilidad de elegir; además se aplican técnicas para lograr una mejor generalización como son la detención temprana, la repetición del entrenamiento y el ajuste de los datos de entrenamiento a las funciones de activación usadas (normalización).

---

*Proyecto de grado

**Facultad de Ingenierías Físico-Mecánicas, Ingeniería de Sistemas e Informática, Msc. Henry Arguello Fuentes

TITLE: AUTOMATIC TRAINING SYSTEM FOR ARTIFICIAL NEURAL NETWORKS BASED ON THE GENETIC FIT OF PARAMETERS AND ARCHITECTURE VARIATION*

AUTHOR (S): MÉNDEZ ORTIZ Edgar Alberto**
MARIÑO MESA Juan Sebastián**

KEYWORDS: NEURAL NETWORKS, GENETIC ALGORITHMS, EVOLUTIONARY COMPUTATION, OPTIMIZATION.

DESCRIPTION.

The artificial neural networks have shown their impact on different areas of science, where they are applied to problems of classification and prediction of patterns. Although this technique has been used frequently by many researchers, it is not even a solid theory to identify the exact structure of a neural network properly, so the selection of this structure is being sought through trial and error, or best to implement some rules for configuration, as well as playing an important role the experience gained by the expert.

Therefore it is very useful to develop software that look different configurations and parameters required to produce solutions on which the user to analyze and decide on the option that fits your needs. It was then uses a genetic algorithm to find these parameters, given their capacity as optim search system in large size sets The search system developed largely reduce the computational complexity of finding a network architecture in a suitable timely.

This work deals in a non-conventional way two typical problems in the optimization of neural networks for learning problems, the design of the architecture (layer-neurons) and the activation functions to be used in each of these layers. The network is trained by backpropagation based methods spread that the user has a choice; also apply techniques to achieve better generalization such as the early stopping, the repetition of training and adjusting the training data for activation functions used (normalization).

---

*Work of degree

** Faculty of Physical-Mechanics Engineering's, Systems and Informatic Engineering, Msc. Henry Arguello Fuentes.

1. INTRODUCCIÓN

El diseño de la estructura de redes neuronales, es un proceso tedioso, que requiere por parte de los diseñadores un grado de experiencia de diseño/entrenamiento, junto con un proceso de prueba y error dependiente de la complejidad del problema. Estos análisis que desarrolla el experto, no se realizan de forma metódica, sino que dependen en gran parte de la suerte que se tenga en el entrenamiento y la determinación de la estructura.

En la actualidad no existe una herramienta con capacidades de determinar de forma exacta la arquitectura de una red neuronal artificial que resuelva determinado problema y su realización tiene un grado alto de dificultad alto dado que esta clase de problemas tienen infinitas soluciones. Sin embargo, se puede buscar una solución a este problema que obtenga una configuración más aproximada a la ideal, dentro un campo de búsqueda. Aunque en la teoría como en la práctica es incierta la cantidad de neuronas y capas que se requieren para solucionar un problema, existen varias estructuras de una red que ejecutan la misma tarea, cada una con diferente grado de error producido respecto al conjunto de entrenamiento. Sería entonces ideal hallar una configuración óptima para una red neuronal tomando como base las diferentes estructuras que satisfacen el mismo problema.

Cuando se habla de búsqueda de buenas soluciones en espacios de posibilidades infinitos una técnica que ha demostrado un gran potencial son los algoritmos genéticos[1]. Los algoritmos genéticos son métodos adaptativos, generalmente usados en problemas de búsqueda y optimización de parámetros, basados en la reproducción sexual y en el principio supervivencia del mas apto. Esta técnica ha demostrado comportarse de forma adecuada en la solución de dichas situaciones

[1] **Goldberg D. E.** *A comparative analysis of selection schemes used in genetic algorithms*. In Gregory Rawlins, editor. Foundations of Genetic Algorithms, pages 69-93, San Mateo, CA: Morgan Kaufmann Publishers, 1991..

problemáticas, por esta razón se considera que la utilización de un algoritmo genético dentro del proceso de diseño y entrenamiento de sistemas neuronales es una forma adecuada para abordar el problema.

1.1 Planteamiento del problema

El diseño de una arquitectura adecuada para una red neuronal es un problema de búsqueda en el campo de todas las arquitecturas posibles (problema de búsqueda en un campo infinito), donde cada punto del espacio representa un diseño posible. Considerando un criterio para organizar todas las posibilidades (como el error de entrenamiento más bajo o el menor número de capas ocultas requeridas) se obtendrá una superficie discreta de búsqueda sobre la cual el algoritmo genético se encargará de obtener el punto más alto de la superficie; el cual representará la configuración más adecuada para resolver el problema específico.

Hay muchas características sobre dicha superficie que hacen que los AG (Algoritmos Genéticos) sean buenos candidatos para la búsqueda, como por ejemplo[2]:

- La superficie es infinitamente grande ya que el número de nodos y conexiones posibles es ilimitado.

- La superficie es no diferenciable porque los cambios en el número de nodos o conexiones son discretos y pueden tener un efecto discontinuo sobre el desempeño de las redes neuronales.

[2] **Yao Xin,** *Evolving Artificial Neural Networks. School of Computer Science*. The University of Birmingham. B15 2TT 1999

- La superficie es compleja y ruidosa puesto que el mapeo desde una arquitectura hacia su desempeño es indirecto y dependiente del método de evaluación utilizado.

- La superficie es engañosa ya que arquitecturas similares pueden tener un desempeño bastante diferente.

- La superficie es multimodal porque diferentes arquitecturas pueden tener desempeños similares.

1.2 Objetivo General

Diseñar e implementar una herramienta software basada en algoritmos genéticos para la búsqueda de los parámetros y la arquitectura óptima de una red neuronal artificial.

1.2.1 Objetivos Específicos:

- Elaborar un estudio del estado del arte sobre la utilización de algoritmos genéticos para realizar la selección de una arquitectura de red neuronal.

- Implementar una técnica para la selección de los parámetros de una red usando algoritmos genéticos.

- Desarrollar una herramienta software que utilice la técnica implementada.

- Verificar el desempeño de la herramienta desarrollada mediante problemas que requieran la selección de una arquitectura determinada para una red neuronal.

1.3 Justificación.

Las redes neuronales artificiales son de gran utilidad en problemas en donde la cantidad de datos y la complejidad de los mismos dificultan la realización los cálculos por parte de un humano.

El diseño de la estructura de redes neuronales, es un proceso tedioso, que requiere por parte del usuario un grado de experiencia de diseño/entrenamiento, junto con un proceso de prueba y error dependiente de la complejidad del problema.

Estos análisis que desarrolla el experto, no se realizan de forma metódica, sino que dependen en gran parte de la suerte que se tenga en el entrenamiento y la determinación de la estructura. Con el fin de ayudar al experto humano en la búsqueda de la red que mejor se adecue a su problema y necesidades, se necesita de un método de búsqueda eficiente, como son los algoritmos genéticos.

Los algoritmos genéticos son herramientas de búsqueda y optimización. Al igual que las redes neuronales, los algoritmos genéticos están inspirados en la biología (teoría de la evolución genética de Darwin). Entre sus características más importantes se destacan la naturaleza estocástica, la capacidad de considerar simultáneamente una población de soluciones, y la adaptabilidad ante un rango amplio de problemas.

El propósito de este proyecto es combinar dos técnicas de inteligencia artificial, los

algoritmos genéticos y las redes neuronales, para crear un sistema híbrido, donde la primera se aplique para mejorar la segunda. Este trabajo se centra en el desarrollo de métodos para el diseño evolutivo de los parámetros y variación de la arquitectura de redes neuronales artificiales.

2. ALGORITMOS GENÉTICOS

"Aunque el ingenio humano puede lograr infinidad de inventos, nunca ideará ninguno mejor, más sencillo y directo que los que hace la naturaleza, ya que en sus inventos no falta nada y nada es superfluo"

Leonardo Da Vinci

2.1 Algoritmos Evolutivos

Los algoritmos evolutivos son un término empleado para describir ciertos programas o sistemas computacionales que utilizan mecanismos evolutivos en su diseño e implementación para la resolución de problemas.

En la actualidad se cuenta con diversos algoritmos evolutivos, siendo los principales: programación evolutiva, programación genética, estrategia de evolución y los algoritmos genéticos. Todos ellos comparten una base conceptual común, es decir, simular la evolución de estructuras individuales por medio de procesos de selección, mutación y reproducción. Los procesos dependen del desempeño de los individuos según el ambiente en que se desarrollan, de donde, los algoritmos evolutivos, AEs, mantienen una población que evoluciona de acuerdo a reglas de selección y otros operadores genéticos, tales como el cruce, recombinación y mutación.

En general se identifican tres tipos de algoritmos evolutivos, los que utilizan una simple función probabilística para medir las aptitudes de los individuos, los que simulan torneos de donde se seleccionan los mejores o simplemente aquellos que

en forma aleatoria seleccionan los individuos de una población para experimentar con operadores genéticos propagando el material genético.

2.2 Orígenes de los Algoritmos Evolutivos.

Con el fin de facilitar la comprensión sobre los algoritmos evolutivos y sus orígenes, se presenta a continuación un paralelo con los procesos biológicos en los cuales éstos se basan.

En primer lugar cabe notar que la evolución en la naturaleza no tiene un propósito específico, es decir, no hay evidencias que afirmen que la meta de la evolución sea crear un cierto tipo de individuo en particular. En la naturaleza, la generación de organismos biológicos parecen regirse por un proceso aleatorio, aunque se observa una selección natural de individuos que compiten en un mismo ambiente, de donde algunos son mejores que otros. Esta selección conduce a que, probablemente, aquellos más dotados sean quienes sobrevivan y propaguen su información genética a las futuras generaciones.

En la naturaleza, se dan dos tipos de procesos reproductivos, el primero es de tipo asexual, donde la información genética de los hijos es idéntica a la de los padres, y el segundo sexual, que permite la mezcla de cromosomas, es decir, la descendencia contiene información genética de ambos padres. Esta operación denominada recombinación consiste a grandes rasgos en un intercambio de trozos de dos cromosomas formando un tercero, lo que los biólogos han llamado cruce.
El proceso de recombinación sucede en un entorno donde la selección del compañero se basa principalmente en las características de aptitud de los individuos, definidas por el ambiente. Aunque también se debe considerar el efecto aleatorio, o suerte de individuos inferiormente dotados que se involucran en el proceso de cruce.

De lo anterior, se desprende que en esencia los algoritmos evolutivos son una técnica de búsqueda, optimización, con base en la selección natural de individuos según explica la teoría Darwiniana de supervivencia del más apto.

2.3 Algoritmos Genéticos

Un algoritmo genético es entonces una máquina aprendiendo a comportarse como un mecanismo evolutivo propio de la naturaleza, que se inicia con la creación de una población de individuos representados por cromosomas, que en esencia son un conjunto de cadenas análogas a las que se encuentran en el ADN.

Seguidamente, éstos son sometidos a un proceso evolutivo, donde la selección enfoca su atención en aquellos individuos con mayor grado de aptitud según la información disponible de la adaptación, mientras que la recombinación y mutación modifican la estructura de los individuos en forma heurística.

Los algoritmos genéticos son ampliamente utilizados en diversas áreas, principalmente en problemas de optimización multidimensionales donde las cadenas de caracteres de los cromosomas pueden ser utilizados para codificar los diferentes parámetros a ser optimizados.

En la práctica, se puede implementar este modelo genético computacional mediante arreglos de bits, o caracteres que representan los cromosomas, y de esta forma, una simple manipulación de un bit se traduce en la implementación de un operador genético, tal como es el cruce, la mutación u otro.

2.4 Operadores básicos utilizados en un AG.

2.4.1 Selección:

Los algoritmos de selección serán los encargados de escoger que individuos van a disponer de oportunidades de reproducirse y cuáles no.
Puesto que se trata de imitar lo que ocurre en la naturaleza, se ha de otorgar un mayor número de oportunidades de reproducción a los individuos más aptos. Por lo tanto la selección de un individuo estará relacionada con su valor de ajuste. No se debe sin embargo eliminar por completo las opciones de reproducción de los individuos menos aptos, pues en pocas generaciones la población se volverá homogénea.

2.4.1.1 Selección por ruleta

Propuesto por DeJong, es posiblemente el método más utilizado desde los orígenes de los Algoritmos Genéticos[3].

A cada uno de los individuos de la población se le asigna una parte proporcional a su ajuste de una ruleta, de tal forma que la suma de todos los porcentajes sea la unidad. Los mejores individuos recibirán una porción de la ruleta mayor que la recibida por los peores.

Generalmente la población está ordenada en base al ajuste por lo que las porciones mas grandes se encuentran al inicio de la ruleta. Para seleccionar un individuo basta con generar un número aleatorio del intervalo [0,1] y devolver el individuo situado en esa posición de la ruleta.

[3] **BLICKLE T. and THIELE L**. A comparison of selection schemes used in genetic algorithms. Technical Report 11, Computer Engineering and Comunnication Network Lab (TIK), Gloriastrasse 35, 8092 Zurich, Switzerland, 1995.

Esta posición se suele obtener recorriendo los individuos de la población y acumulando sus proporciones de ruleta hasta que la suma exceda el valor obtenido.

Es un método muy sencillo, pero ineficiente a medida que aumenta el tamaño de la población (su complejidad es $O(n^2)$). Presenta además el inconveniente de que el peor individuo puede ser seleccionado más de una vez.

En la bibliografía se suele referenciar a este método con el nombre de selección de Montecarlo.

2.4.1.2 Selección por torneo

La idea principal de este método consiste en realizar la selección en base a comparaciones directas entre individuos. Existen dos versiones de selección mediante torneo:

- Determinística
- Probabilística

En la versión determinística se selecciona al azar un número p de individuos (generalmente se escoge p = 2). De entre los individuos seleccionados se selecciona el más apto para pasarlo a la siguiente generación.

La versión probabilística únicamente se diferencia en el paso de selección del ganador del torneo. En vez de escoger siempre el mejor se genera un número aleatorio del intervalo [0,1], si es mayor que un parámetro p (fijado para todo el proceso evolutivo) se escoge el individuo más alto y en caso contrario el menos apto. Generalmente p toma valores en el rango $0,5 < p \leq 1$.

Variando el número de individuos que participan en cada torneo se puede modificar la presión de selección. Cuando participan muchos individuos en cada torneo, la presión de selección es elevada y los peores individuos apenas tienen oportunidades de reproducción. Cuando el tamaño del torneo es reducido, la presión de selección disminuye y los peores individuos tienen más oportunidades de ser seleccionados.

Un caso particular es el elitismo global. Se trata de un torneo en el que participan todos los individuos de la población con lo cual la selección se vuelve totalmente determinística. En los procesos evolutivos las hembras seleccionan a los machos mejor dotados que aseguren su descendencia, por su parte los AG involucran en su desarrollo criterios de elitismo por medio de penalizaciones para determinar su aptitud; los cromosomas con el menor número de penalizaciones serán los más aptos. Este criterio se aplica con la finalidad de mantener al mejor individuo de cada población y copiarlo a la siguiente de esta manera se asegura mantener en cada nueva población al cromosoma con mejor aptitud.

Elegir uno u otro método de selección determinara la estrategia de búsqueda del Algoritmo Genético. Si se opta por un método con una alta presión de selección se centra la búsqueda de las soluciones en un entorno próximo a las mejores soluciones actuales. Por el contrario, optando por una presión de selección menor se deja el camino abierto para la exploración de nuevas regiones del espacio de búsqueda. Existen muchos otros algoritmos de selección. Unos buscan mejorar la eficiencia computacional, otros el número de veces que los mejores o peores individuos pueden ser seleccionados. Algunos de estos algoritmos son muestreo determinístico, escalamiento sigma, selección por jerarquías, estado uniforme, sobrante estocástico, brecha generacional, etc.

2.4.2 Cruce:

Durante el proceso de fecundación, el espermatozoide y el óvulo se unen y reconstruyen en el nuevo organismo la disposición por pares de los cromosomas; la mitad de estos cromosomas procede de cada padre. Este mismo hecho simulan los algoritmos genéticos con el proceso de cruce donde dos cromosomas comparten información genética en pos de crear un nuevo individuo.

Existen multitud de algoritmos de cruce. Sin embargo los más empleados son:

- Cruce de 1 punto
- Cruce de 2 puntos
- Cruce uniforme

2.4.2.1 Cruce de 1 punto

Es la más sencilla de las técnicas de cruce. Una vez seleccionados dos individuos se cortan sus cromosomas por un punto seleccionado aleatoriamente para generar dos segmentos diferenciados en cada uno de ellos: la cabeza y la cola.

Se intercambian las colas entre los dos individuos para generar los nuevos descendientes.

De esta manera ambos descendientes heredan información genética de los padres, en la bibliografía suele referirse a este tipo de cruce con el nombre de SPX (Single Point Crossover)

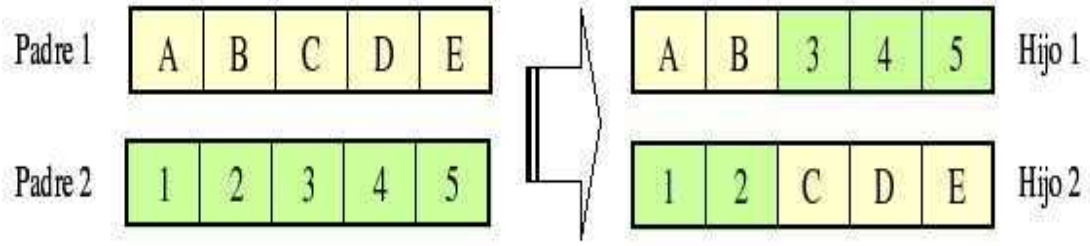

Figura 1. Cruce de 1 Punto [4]

2.4.2.2 Cruce de 2 puntos

Se trata de una generalización del cruce de 1 punto. En vez de cortar por un único punto los cromosomas de los padres como en el caso anterior se realizan dos cortes. Debería tenerse en cuenta que ninguno de estos puntos de corte coincida con el extremo de los cromosomas para garantizar que se originen tres segmentos. Para generar la descendencia se escoge el segmento central de uno de los padres y los segmentos laterales del otro padre.

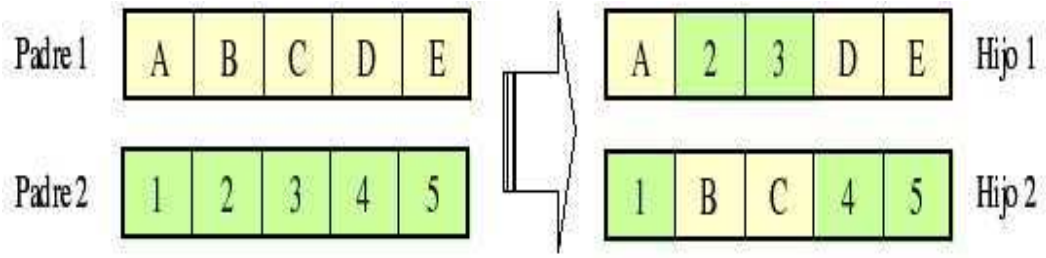

Figura 2. Cruce de 2 Puntos [3]

[4] Tomado de**: Gestal M**., Introducción a los algoritmos genéticos, departamento de tecnologías de la información y las comunicaciones, Universidade da Coruña, 1996.

Generalmente se suele referir a este tipo de cruce con las siglas DPX (Double Point Crossover).

Generalizando se pueden añadir más puntos de cruce dando lugar a algoritmos de cruce multipunto. El problema principal de añadir nuevos puntos de cruce radica en que es más fácil que los segmentos originados sean corrompibles, es decir, que por separado quizás pierdan las características de bondad que poseían conjuntamente.

2.4.2.3 Cruce Uniforme.

El cruce uniforme es una técnica completamente diferente de las vistas hasta el momento. Cada gen de la descendencia tiene las mismas probabilidades de pertenecer a uno u otro padre.

Aunque se puede implementar de muy diversas formas, la técnica implica la generación de una máscara de cruce con valores binarios. Si en una de las posiciones de la máscara hay un 1, el gen situado en esa posición en uno de los descendientes se copia del primer padre.

Si por el contrario hay un 0 el gen se copia del segundo padre. Para producir el segundo descendiente se intercambian los papeles de los padres, o bien se intercambia la interpretación de los unos y los ceros de la máscara de cruce. Se suele referir a este tipo de cruce con las siglas UPX (Uniform Point Crossover).

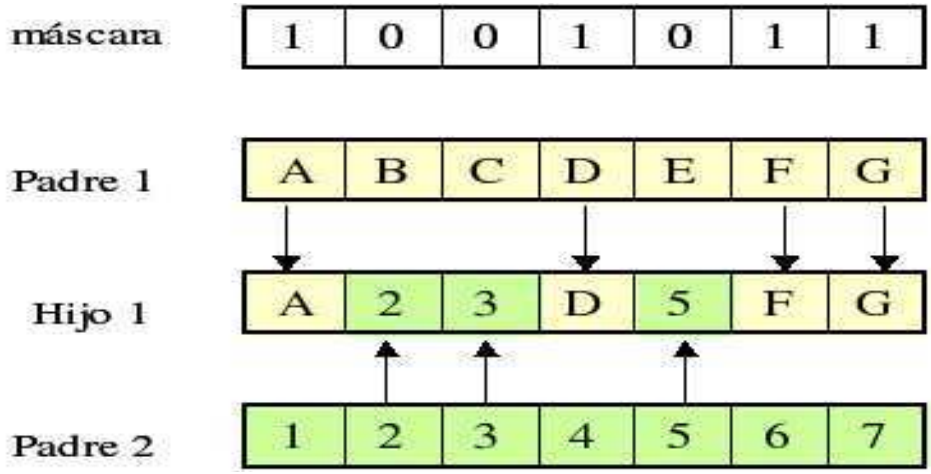

Figura 3. Cruce Uniforme [3]

2.4.3 Mutación

Las mutaciones son la materia prima de la evolución. No obstante, sin mutaciones las especies no evolucionarían. La evolución tiene lugar cuando una nueva versión de un gen, que originalmente surge por una mutación, aumenta su frecuencia y se extiende a la especie gracias a la selección natural. En los AG, las mutaciones son quienes permiten crear nuevos individuos, escapar de los mínimos locales, abastecerse de nuevo material genético, y en ciertos casos mejorar los procesos de elitismo en busca de mejores soluciones.

2.5 Fundamentos matemáticos

2.5.1 Función de adaptación

Los algoritmos genéticos son mecanismos de búsqueda que operan sobre un conjunto de códigos muy grande. Del dominio de búsqueda se toman iteradamente conjuntos de muestras y cada elemento de una muestra es calificado dependiendo de qué tan bien cumpla con los requisitos de aquello que es buscado.

Luego son seleccionados los elementos, a los que también se les llama *individuos*, que cumplan mejor con dichos requisitos. Éstos se multiplican en las siguientes muestras y son sometidos a ciertos operadores que emulan a los que funcionan en la naturaleza. Uno de los elementos esenciales involucrados en este proceso es la calificación asignada a cada individuo, a la que se le llama *grado de adaptación*.

Esta calificación es un número mayor o igual a cero, y en conjunto con la selección se convierte en una medida de la posibilidad de reproducción para cada individuo. Esto es, existe una función de adaptación que asocia a cada individuo de la muestra (población) con un número real no negativo. Mientras más grande sea el valor asignado a un individuo dado, mayor será la probabilidad de éste de ser seleccionado para formar parte de la siguiente generación de muestras.

2.5.2 Esquemas

Se ha creado una notación para los conjuntos de cadenas binarias (cromosomas) mediante cadenas compuestas de tres símbolos, a saber: {0,1,*}. Considérese un conjunto de cadenas que poseen valores comunes en ciertas posiciones. Para construir la cadena que denota este conjunto, basta colocar en las posiciones donde las cadenas coinciden el valor explícito que tienen, y en las posiciones donde los valores no coinciden se coloca un *. De esta manera por ejemplo, 1*0* denota el conjunto {1000,1001,1100,1101}. A las cadenas compuestas de 1, 0 y * se les denomina *esquemas*.

El número de posiciones con valor explícito de un esquema *H* se denomina *orden* del esquema y se denota por $o(H)$. La distancia entre la primera y la última posición explícita se denomina *longitud de definición* (*defining length*) y se denota como $\delta(H)$.

Dados una población de cadenas binarias P en un instante t y un esquema H se

definen:

- Presencia de H en P en el instante t, m(H,t): Es el número de cadenas de la población P en el instante t que encajan en el esquema H.

- Aptitud del esquema H en P en el instante t, $\overline{f}(H,t)$: Es el promedio de las aptitudes de todas las cadenas de la población que encajan en el esquema H en el instante t.

$$\overline{f}(H,t) = \frac{1}{m} \cdot \sum_{i=1}^{i=m} Aptitud(v_i) \qquad \textbf{Ec 2.1}$$

siendo $m = m(H,t)$ y
$v_i \ \ i = 1..m$ las cadenas de P que encajan en H.

- Aptitud media de la población en el instante t, $\overline{f}$: Es el promedio de las aptitudes de todas las cadenas de la población en el instante t.

$$\overline{f} = \frac{1}{n} \cdot \sum_{i=1}^{i=n} Aptitud(v_i) \qquad \textbf{Ec 2.2}$$

2.5.3 El teorema del esquema

Hasta la fecha, el modelo matemático más usual para describir el comportamiento de los algoritmos genéticos está basado en el *teorema del esquema*. Éste fue planteado originalmente por Holland[5] para el AGS (Algoritmo Genético Simple), y

[5] **Holland, J. H.,** Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, 1975

se caracteriza por utilizar selección proporcional y cruzamiento de un punto, entre otras cosas.

2.5.4 El efecto de la selección

Se asume que las cadenas son copiadas a la nueva generación con una probabilidad basada en su valor de capacidad (fitness f_i) divida por la capacidad total de la generación

$$p_i = \frac{f_i}{\sum_j f_j} \qquad \textbf{Ec 2.3}$$

Supóngase que en un instante dado de tiempo t hay m ejemplares de un esquema particular H contenido en la población P(t) (m = m(H,t)).Tomamos una población de tamaño n.

Mediante reemplazamientos a partir de la población P(t), se espera tener m(H,t+1) representantes del esquema en la población en el instante t +1 donde:

$$m(H,t+1) = m(H,t) \cdot n \cdot \frac{\overline{f}(H,t)}{\sum_j f_j} \qquad \textbf{Ec 2.4}$$

Siendo f(H) la aptitud media de las cadenas representadas por el esquema H en el instante t.

2.5.5 Ecuación de crecimiento reproductivo del esquema

$$m(H,t+1) = m(H,t) \cdot \frac{\overline{f}(H,t)}{\overline{f}} \qquad \textbf{Ec 2.5}$$

Un esquema particular crece como el porcentaje de la aptitud media del esquema respecto de la aptitud de la población.

Sea un esquema particular H que permanece por encima de la media una cantidad (c constante), entonces:

$$m(H,t+1) = m(H,t)\cdot\frac{\overline{f}+c\overline{f}}{\overline{f}} = (1+c)\cdot m(H,t)$$

$t = 0$:

$$m(H,t) = m(H,0)\cdot(1+c)^{t} \qquad \textbf{Ec 2.6}$$

La reproducción asigna un número exponencialmente creciente (decreciente) de ejemplares a los esquemas por encima (por debajo) de la media.

2.5.6 El efecto del cruce

Cuando se aplica algún tipo de cruce con cierta probabilidad p_c sobre los individuos previamente seleccionados, algunas de las cadenas se cruzarían con otras de forma tal que la cadena resultante ya no sería representante del esquema *H*, es decir, el esquema se rompería. Otras no serían seleccionadas para cruzarse y simplemente pasarían intactas a la siguiente generación y habría otras más que originalmente no eran representantes del esquema y que al cruzarse generarían cadenas de *H*. El valor esperado de cadenas representantes de *H* que han sido seleccionadas y a las que no se les aplica cruzamiento es:

$$(1-p_C)\cdot m(H,t)\cdot\frac{\overline{f}(H,t)}{\overline{f}} \qquad \textbf{Ec 2.7}$$

Siendo p_R, la la probabilidad de ruptura del esquema *H* bajo el tipo de cruce que esté siendo utilizado y *g* el número de cadenas ganadas por el esquema, el valor esperado del número de cadenas representantes de *H* que fueron seleccionadas y permanecen en el esquema después de aplicárseles cruce, es:

$$p_c\left[m(H,t)\cdot\frac{\overline{f}(H,t)}{\overline{f}}(1-p_R)+g\right] \qquad \textbf{Ec 2.8}$$

Siendo la probabilidad de ruptura del esquema *H* bajo el tipo de cruzamiento que esté siendo utilizado.

Resumiendo, el valor esperado del número de representantes del esquema *H* tras haber efectuado selección y cruzamiento es:

$$m(H,t+sel+cru)=(1-p_C)\cdot m(H,t)\cdot\frac{\overline{f}(H,t)}{\overline{f}}+p_c\left[m(H,t)\cdot\frac{\overline{f}(H,t)}{\overline{f}}(1-p_R)+g\right] \qquad \textbf{Ec 2.9}$$

donde,

t = Generación en la que se encuentra el algoritmo genético.

sel = Efecto de la selección.

cru = Efecto del cruce.

El primer término representa el aporte de las cadenas de H que no intervinieron en el cruce. El segundo término es el aporte de las cadenas de H que se cruzan y se mantienen en H más las cadenas que no eran de H, pero luego del cruce pasan a formar parte de él.

En una cadena de longitud *L* existen *L*-1 posibles puntos de corte, así que la probabilidad de romper el esquema *H* con un corte es:

$$p_R \leq \frac{\delta(H)}{L-1} \qquad \textbf{Ec 2.10}$$

Siendo δ(*H*) la distancia entre la primera y la última posición explícita en el esquema *H*.

Observando la ecuación 2.9:

$$m(H,t+sel+cru) = (1-p_C)\cdot m(H,t)\cdot\frac{\bar{f}(H,t)}{\bar{f}} + p_c\left[m(H,t)\cdot\frac{\bar{f}(H,t)}{\bar{f}}(1-p_R)+g\right]$$

Eliminando g se obtiene:

$$m(H,t+sel+cru) \geq m(H,t)\cdot\frac{\bar{f}(H,t)}{\bar{f}}\cdot(1-p_C\cdot p_R) \qquad \textbf{Ec 2.11}$$

En el algoritmo original de Holland se elige a un compañero, para realizar el cruce, sin predisposición. A sí que la probabilidad de que esa cadena encaje en el esquema H es P_R:

$$p_R \leq \frac{\delta(H)}{L-1}\cdot(1-P(H,t)) \qquad \textbf{Ec 2.12}$$

De esta manera se obtiene la ecuación que representa el siguiente esquema obtenido, después de realizar el cruce de los genes padre, con una probabilidad de cruce p_c:

$$P(H,t+1) \geq P(H,t) \cdot \frac{\overline{f}(H,t)}{\overline{f}} \cdot \left(1 - p_C \cdot \frac{\partial(H)}{L-1} \cdot (1 - P(H,t))\right) \quad \textbf{Ec 2.13}$$

2.5.7 El efecto de la mutación

Se supone que la mutación se aplica con probabilidad pm y que tiene el efecto de invertir un bit (cambiar un 1 por un 0 ó viceversa). Para que una cadena representante del esquema H permanezca en él tras una mutación, debe ocurrir que ninguno de los bits definidos del esquema sea invertido. La probabilidad de que un bit no sea invertido por una mutación es 1-pm, así que la probabilidad de que ninguno de los bits definidos sea invertido, suponiendo que el invertir cada bit es un evento independiente, es:

$$\mu(p_m) = (1 - p_m)^{o(H)} \quad \textbf{Ec 2.14}$$

Añadiendo a la expresión 2.13 se obtiene:

$$P(H,t+1) \geq P(H,t) \cdot \frac{\overline{f}(H,t)}{\overline{f}} \cdot \left(1 - p_C \cdot \frac{\partial(H)}{L-1} \cdot (1 - P(H,t)) \cdot \frac{\overline{f}(H,t)}{\overline{f}}\right) \cdot (1 - p_m)^{o(H)} \quad \textbf{Ec 2.15}$$

2.6 El teorema fundamental

Este resultado recibe el nombre de Teorema del esquema o Teorema Fundamental de los algoritmos genéticos:

La presencia de un esquema H en la población P de la generación del instante t en un Algoritmo Genético evoluciona estadísticamente de modo exponencial según la ecuación anterior.

Los esquemas de orden bajo adaptados por encima de la media reciben un número exponencialmente creciente de oportunidades en siguientes generaciones.

2.7 Paralelismo Implícito

La eficacia a los algoritmos genéticos se basa en que, aunque el algoritmo génetico sólo procesa n estructuras en cada generación, se puede probar que, bajo hipótesis muy generales, se procesan de modo útil al menos n^3 esquemas.

Este paralelismo implícito se consigue sin ningún dispositivo o memoria adicionales, sólo con la propia población. A pesar de la ruptura de los esquemas largos de orden alto por los operadores de cruce y mutación, los algoritmos genéticos procesan inherentemente una gran cantidad de esquemas mientras procesan una cantidad relativamente pequeña de cadenas.

3. DISEÑO DE UNA RED NEURONAL ARTIFICIAL

El interés inicial en estos sistemas de redes neuronales artificiales proviene por la creencia de que es posible mejorar el conocimiento del cerebro, el conocimiento y la percepción humana; con la esperanza de que se pudieran crear sistemas pensantes que tuvieran mejores resultados en tareas como clasificación, problemas de decisión, pronósticos y sistemas de control adaptables.

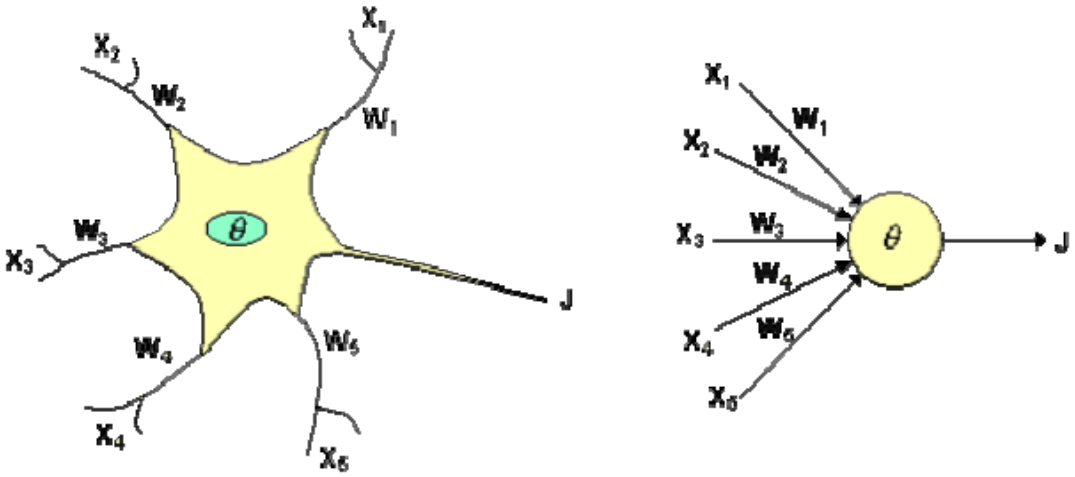

Figura 4. Modelo biológico y modelo artificial[6]

Sin embargo, el diseño de redes neuronales artificiales para aplicaciones específicas es un proceso de prueba y error, dependiendo especialmente de la experiencia previa del experto. Muchos de los algoritmos de aprendizaje utilizados en las redes neuronales buscan un ajuste adecuado de los parámetros modificables (pesos) dentro de una topología de red especificada inicialmente, guiado por unos ejemplos de entrenamiento provenientes del ambiente de la tarea. Claramente, para que este enfoque sea exitoso, el ajuste deseado de parámetros debe existir en el espacio donde se busca (el cual a su vez está

[6] Universidad Tecnológica de Pereira, tutorial de redes neuronales, 2007 , http://ohm.utp.edu.co/neuronales/main.htm

restringido por la elección de la topología de red) y el algoritmo de búsqueda usado debe ser capaz de encontrarlo[7].

A pesar de la gran actividad e investigación en esta área durante los últimos años, que ha llevado al descubrimiento de varios resultados teóricos y empíricos significativos, el diseño de las redes neuronales artificiales para aplicaciones específicas bajo un conjunto dado de restricciones de diseño (por ejemplo, aquellas dictadas por la tecnología) es, por mucho, un proceso de prueba y error, dependiendo principalmente de la experiencia previa con aplicaciones similares[8].

Estos factores hacen que sea difícil el proceso de diseño de redes neuronales. Adicionalmente, la falta de principios de diseño constituye un obstáculo de importancia en el desarrollo de sistemas de redes neuronales a gran escala para una amplia variedad de problemas prácticos. Por tal razón, las técnicas para automatizar el diseño de arquitecturas neuronales son de gran interés.

3.1 Arquitecturas de Redes Neuronales

Se denomina arquitectura a la topología, estructura o patrón de conexionado de una red neuronal. En un RNA (red neuronal artificial) los nodos se conectan por medio de sinapsis, esta estructura de conexiones sinápticas determina el comportamiento de la red. En general, las neuronas se suelen agrupar en unidades estructurales que se denominan capas. Finalmente, el conjunto de una o más capas constituye la red neuronal.

[7] **Honavar V. and L. Uhr.** Generative Learning Structures and Processes for Generalized Connectionist Networks. Information Sciences, 70:75—108,1993.
[8] **Dow R. J. y Sietsma J.** Creating Artificial Neural Networks that generalize. Neural Networks, vol. 4, no. 1, pp. 198-209, 1991.

Se distinguen tres tipos de capas:

- **De entrada:** Una capa de entrada o sensorial está compuesta por neuronas que reciben datos o señales procedentes del entorno.

- **Oculta:** es aquella que no tiene conexión directa con el contorno, es decir, que no se conecta directamente ni a órganos sensores ni a efectores.

- **De salida:** es aquella cuyas neuronas proporcionan la respuesta de la red neuronal.

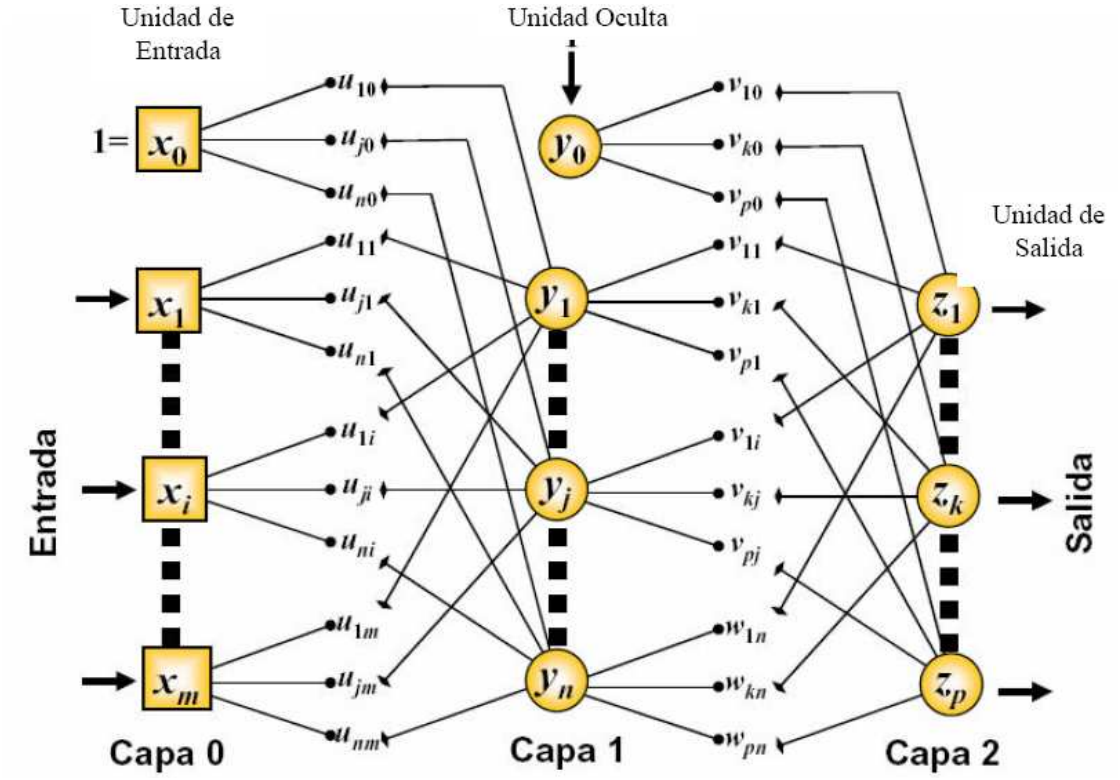

Figura 5. Estructura de una Red Neuronal Artificial[9]

[9] Tomada de: **IZAURIETA F. y SAAVEDRA C.,** Redes Neuronales Artificiales, Departamento de física, UNIVERSIDAD DE CONCEPCIÓN, Chile.

Las conexiones entre las neuronas pueden ser excitatorias o inhibitorias: un peso sináptico negativo define una conexión inhibitoria, mientras que uno positivo determina una conexión excitatoria.

Las conexiones intra-capa, también denominadas laterales, tienen lugar entre las neuronas pertenecientes a una misma capa, mientras que las conexiones inter-capa se producen entre las neuronas de las diferentes capas. Existen además conexiones realimentadas, que tienen un sentido contrario al de entrada-salida. En algunos casos puede existir realimentación incluso de una neurona consigo misma.

Atendiendo a todos estos conceptos, se puede establecer distintos tipos de arquitecturas neuronales:

- **Redes monocapa:** son aquellas compuestas por una única capa de neuronas.

- **Redes multicapa** (*layered networks*): son aquellas cuyas neuronas se organizan en varias capas.

Atendiendo al flujo de datos en la red neuronal, se puede hablar de:

- **Redes unidireccionales** (*feedforward*): la información circula en un único sentido desde las neuronas de entrada a las de salida.

- **Redes recurrentes o realimentadas** (*feedback*): la información puede circular entre las capas en cualquier sentido.

3.2 Aplicaciones de las redes neuronales artificiales.

Esta técnica de la inteligencia artificial tiene sus grandes aplicaciones en problemas enfocados al reconocimiento de patrones y generadores de los mismos. Pueden tomarse como generalizaciones a técnicas estadísticas de patrones, así como en procesamiento digital de señales, sistemas de identificación y teoría de control[10]. Se puede realizar una clasificación de las RNA, teniendo en cuenta la forma en la que se realiza el entrenamiento: entrenamiento supervisado y no supervisado. Los problemas de modelado estadístico a los cuales se aplican las redes neuronales artificiales, se pueden clasificar en tres grupos: problemas de estimación de densidad, clasificación y regresión.

Un ejemplo de problema de estimación de densidad (se realiza con el entrenamiento no supervisado), se aplicaría en un modelo de interpretación de una imagen de rayos X en la cual se trata de detectar cáncer de seno (mamografía)[11]. En esta investigación se realiza el entrenamiento de la red con imágenes de mamografías normales (casos sanos), mientras que la red se amolda a una probabilidad de densidad de la imagen p(x). Cuando se ingresa una imagen nueva, si se obtienen unos p(x) altos, significa que la imagen muestra normalidad, si los valores son bajos se tendría una posible anormalidad.

Para los problemas de clasificación y regresión (entrenamiento supervisado), se tienen valores tanto de entrada como de salida (datos de entrada y su respectiva salida a estos datos). En la clasificación se ingresa un vector X, y luego de pasar por el procesamiento de la red, se obtienen unas salidas, las cuales se clasificarán en diferentes clases, dependiendo de las características especiales que se obtengan en las salidas. Un ejemplo claro de este tipo de problemas es el del

[10] **M. Jordan, C. Bishop.** Neural Networks, MASSASHUSETTS INSTITUTE OF TECHNOLOGY – Artificial intelligence laboratory, 1996.
[11] **L. Tarassenko.** Novelty detection for the identification of masses in mammograms, Proceedings Fourth IEEE international conference on artificial neural networks, 4:442-447 , 1995.

reconocimiento dactilar. En la investigación[12] que realiza LeCun sobre reconocimiento dactilar, se tiene que la imagen (pre-procesada) que ingresa a la red, es procesada por este para hallar 10 valores (salida de la red); luego teniendo este vector, se le asigna una respectiva clase.

Con el desarrollo que ha tenido esta técnica, se han realizado estudios los cuales demuestran que las redes neuronales artificiales del tipo perceptrón multicapa podría representar cualquier función continua (o incluso discontinua[13] con varias capas ocultas), por lo cual se le denominó como un aproximador universal. La demostración visionaria de los perceptrones multicapa como aproximadores universales se puede encontrar en el trabajo de Cybenco del año 1989[14], además del trabajo de diferentes trabajos realizados por Hornik[15].

3.3 Unidades en redes neuronales

Como se había expuesto anteriormente, las redes neuronales están compuestas por nodos y uniones entre estos, cada nodo se convertiría en la representación de una neurona. Y el procedimiento que se desarrolla dentro de estos consiste en sumatorias de unos valores y la respectiva salida de una función de activación. La siguiente gráfica representa un nodo.

[12] **Y. LeCun, B. Boser, J. S. Denker, D.** Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. 1989.
[13] **S. Russell, P. Norvig.** Inteligencia Artificial Un enfoque moderno, Prentice Hall, Segunda edición, 2004.
[14] **G. Cybenko,** Approximations by superpositions of sigmoidal functions, Mathematics of Control, Signals, and Systems 2, 303--314, 1989.
[15] **H. White K. Hornik, M. Stinchcombe.** Multi-layer feed-forward networks are universal approximators. Neural Networks, pages 2:359--366, 1989.

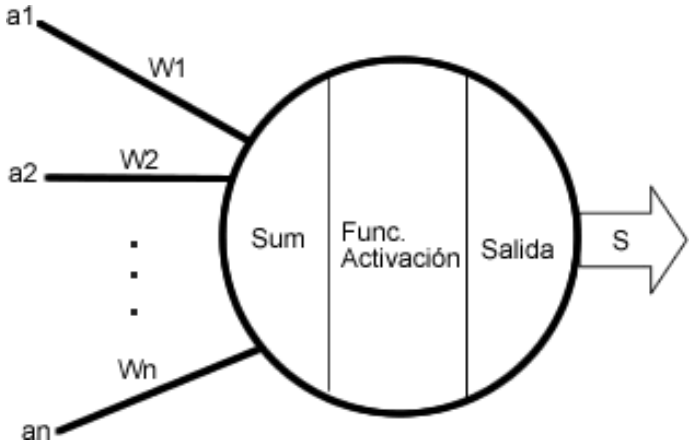

Figura 6. Estructura del nodo[16].

El funcionamiento es el siguiente:

1. Se tiene unos datos de entrada (a1,a2,..,an), los cuales son multiplicados por unos pesos (W1,W2,…,Wn) correspondientes a su línea de comunicación con el nodo, estos pesos tienen como significado lógico la importancia que va a tener dicho valor para el nodo.

2. Luego de entrar los datos multiplicados por sus pesos (a1*W1,a2*W2,…,an*Wn), estos datos son llevados a una sumatoria dentro de la neurona (∑ ai*Wi).

3. El valor de la sumatoria se evalúa en una función de activación, y la salida de esta es llevada a otros nodos o como salida del sistema.

Como conclusión final, el procedimiento realizado por el nodo es el siguiente:

$$S = G(\sum_{i=1}^{n} a_i * W_i) \qquad \textbf{Ec 3.1}$$

[16] Fuente: Autores del proyecto.

En donde G es la función de activación, ai es una de las entradas al nodo y además es multiplicado por su peso Wi, y finalmente S será la salida del nodo.

Existen diferentes formas de hallar los pesos en las conexiones de la red neuronal, entre las más utilizadas se encuentra el algoritmo *backpropagation* (propagación hacia atrás) para topologías perceptrón tanto simple como multicapa; pero también existen estudios sobre programación evolutiva y simulación recocida con muy buenos resultados[17].

3.4 Estructura de las redes neuronales artificiales

Las categorías de las redes neuronales, se pueden dividir en dos conjuntos, teniendo en cuenta su estructura. Las redes se pueden clasificar en redes acíclicas o redes con alimentación hacia adelante (conocidas también como perceptrón) y redes con retroalimentación conocidas como cíclicas o redes recurrentes. En las redes con alimentación hacia adelante los datos siguen un flujo directo a la salida, la comunicación entre los nodos de las capas adyacentes.

La característica de las redes cíclicas se encuentra en que los estados actuales de la red están supeditados a las salidas dadas con anterioridad por el sistema, en cada salida del sistema uno o varios datos son devueltos como entrada nuevamente de la red. De esta forma, ocurre una retroalimentación que producirá una memoria a corto plazo.

En las redes neuronales con alimentación hacia adelante, también conocidas como perceptrones (simples si se cuenta con una sola capa oculta y multicapa si cuenta con varias capas ocultas), se puede encontrar la siguiente estructura general.

[17] **V. W. Porto, D. B. Fogel, and L. J. Fogel.** Alternative Neural Network Training Methods, IEEE EXPERT, 1995.

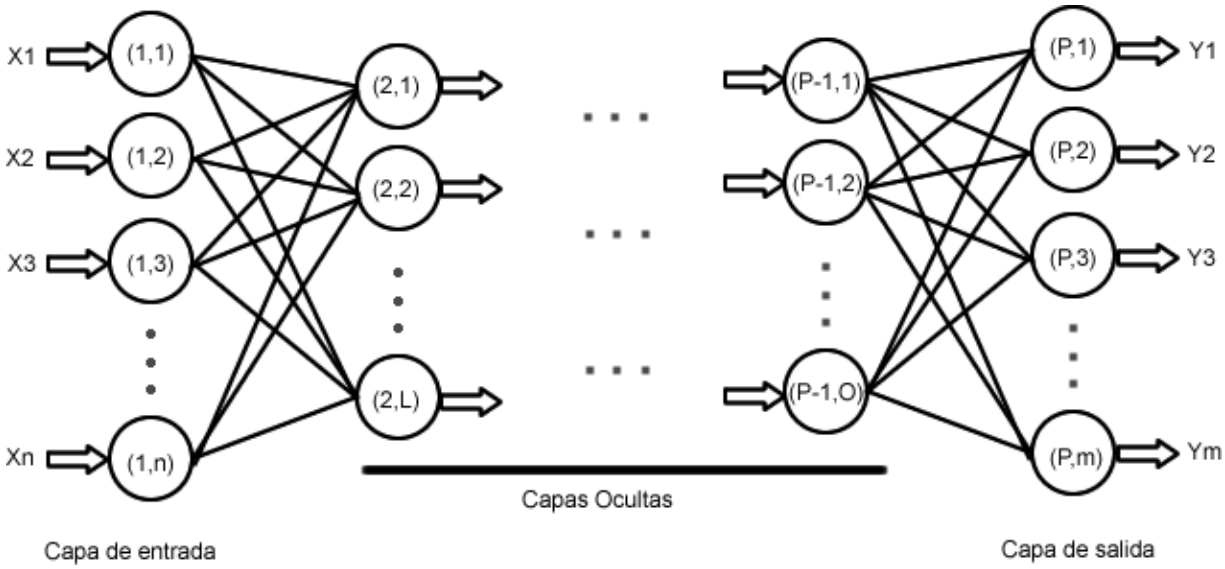

Figura 7. Estructura del perceptrón multicapa[18].

Los datos que entran a la red tienen que ser normalizados para que puedan ser evaluados en las funciones de activación; la normalización depende las características de estas funciones.

Teniendo en cuenta las estructuras tanto del nodo, como de la red neuronal, se podría hacer una formula general sobre las salidas esperadas de cada nodo. La fórmula que desarrollaba cada nodo es la siguiente (adicionando en esta ocasión subíndices para indicar un nodo en la estructura de red).

$$S_{i,j} = G_{i,j}(\sum_{k=1}^{n} a_{i-1,k} * W_{i,k,j}) \qquad \textbf{Ec 3.2}$$

$S_{i,j}$: La salida del nodo (i, j).

$G_{i,j}$: Función de activación del nodo (i, j).

$a_{i-1,k}$: Salida del nodo (i-1, k).

[18] Fuente: Autores del proyecto.

$W_{i,k,j}$: Peso configurado para la salida del nodo (i-1, k) con destino del nodo (i, j).

Esta ecuación es recursiva, puesto que los valores de entrada a un nodo dependen igualmente de la evaluación de entradas hechas por algún otro nodo de capas anteriores.

3.5 Funciones de activación

Existen diferentes funciones utilizadas en las redes neuronales artificiales y estudios referentes a las características y aplicaciones en las cuales deben ser aplicadas. En la elección de las funciones de activación (pueden ser escogidas diferentes funciones en una sola red neuronal) deben de tenerse en cuenta la velocidad de entrenamiento y la convergencia en poco ciclos; la función debe ser sencilla, tanto original como su derivada, además debe tener una amplia parte lineal[19].

Frecuentemente la función paso es utilizada en problemas de clasificación y la función lineal en distintas redes en la capa de salida. Las funciones no lineales en la capa de salida, comúnmente son utilizadas para clasificación de patrones con el fin de restringir el rango de salida. La función logística es usada en problemas de predicción, junto con la función gaussiana.

Las funciones de activación aplicadas en este proyecto son:

[19] **L. Llano, A. Hoyos, F. Arias, J. Velásquez.** Comparación del desempeño de funciones de activación en redes Feedforward para aproximar funciones de datos con y sin ruido, Universidad Nacional de Colombia Sede Medellín, 2007

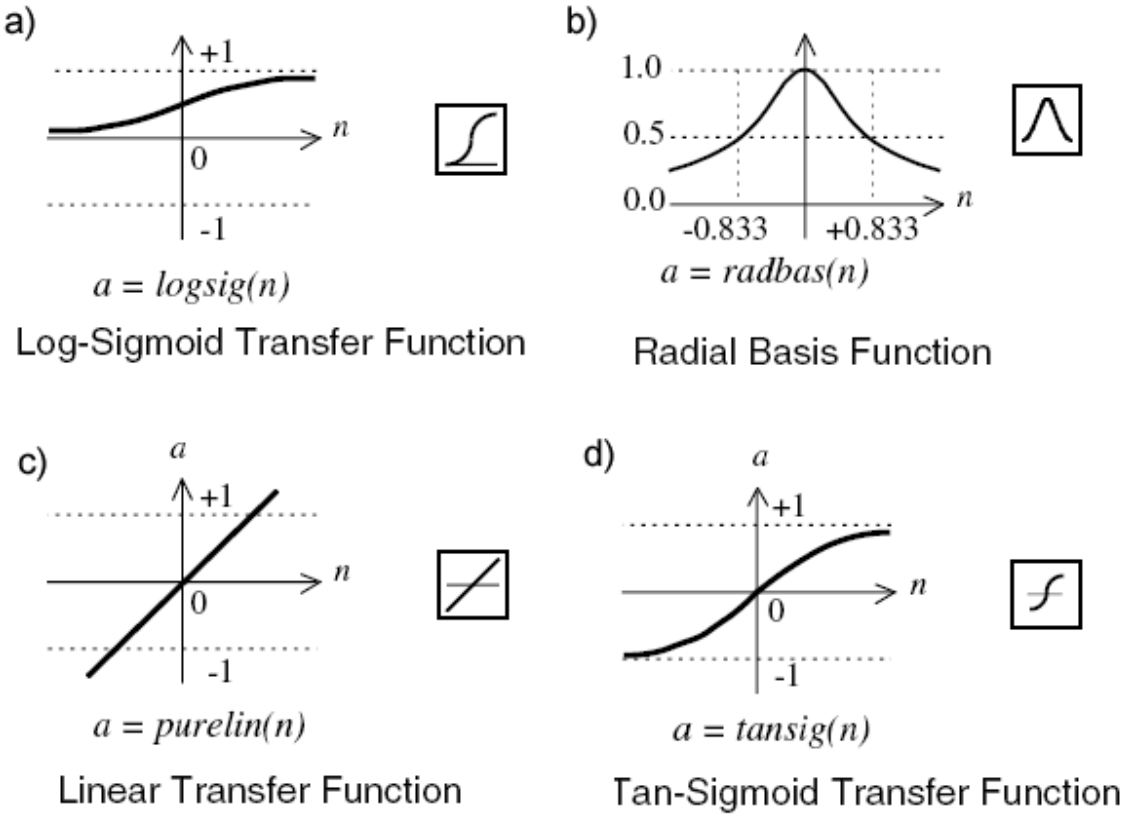

Figura 8. Funciones de activación a) Función logística o Log-Sigmoid. b) Función Gaussiana o Radias Basis. c) Función lineal. d) Función Tangente hiperbólica o tansig.[20]

3.6 Entrenamiento de las redes neuronales artificiales

El entrenamiento de una red neuronal artificial consiste en, una vez configurada la estructura de la red, ajustar los pesos de las conexiones entre las neuronas o nodos. Mientras se van ajustando estos pesos, se dice que la red está ganando conocimiento, mejorando su capacidad de predicción y/o de clasificación.

[20] Tomada de: **H. Demuth, M. Beale, M. Hagan**. Neural Network Toolbox™ User's Guide, MathWorks, inc. 2008, 471-474 .

Existen diferentes métodos para encontrar el valor de los pesos entre los nodos. Hay entrenamiento de tipo evolutivo, basadas en las teorías de evolución biológica, utilizadas para encontrar mínimos globales en espacios muestrales de gran tamaño. La simulación recocida y la propagación hacia atrás, basadas en técnicas de programación no lineal, entre otros.

El algoritmo de propagación hacia atrás (*backpropagation*), es el método de entrenamiento basado en la técnica de programación lineal llamada gradiente descendente; más utilizado para entrenamiento en las redes neuronales perceptrón, tanto simple como multicapa. El algoritmo trabaja solo en redes con propagación hacia adelante, además de necesitar las salidas reales (datos empíricos) que debe dar el sistema dependiendo de las entradas; esto es esencial para el entrenamiento.

3.6.1 Algoritmo de propagación hacia atrás (*backpropagation)*

El primer paso que se debe realizar para aplicar la técnica de propagación hacia atrás es la de inicializar los pesos en la red. Una vez se han inicializado los pesos en la red, los valores del vector de entrada pasan por la primera capa, excitando estas neuronas. Luego de procesar los datos, estas envían señales a la siguiente capa y así sucesivamente hasta llegar al final de la red. Se debe considerar alguna fórmula para hallar el error entre los valores reales de salida (los valores que empíricamente daría el experimento al tomar los datos de entrada), contra los valores obtenidos en la salida del sistema. Una vez se tiene este error, se puede dar marcha al algoritmo de entrenamiento con propagación hacia atrás. El error comúnmente utilizado es el cuadrático. Su fórmula se ve a continuación:

$$E = \frac{1}{2} Err^2 = \frac{1}{2}(y - RNA(x))^2 \qquad \textbf{Ec 3.3}$$

E: Valor del error.

y: Valor de la salida real, dado x como valor de entrada.

$RNA(x)$: La salida dada por la red neuronal con los datos x de entrada.

Cuando se comienza a trabajar el algoritmo, los pesos en la red se van modificando de atrás hacia adelante; desde la capa de salida, a la capa de entrada. El "conocimiento" se va organizando en las capas ocultas, de modo que al final se hace una correspondencia de los valores de entrada, con los resultados deseados en la salida. La forma de hallar los nuevos pesos es la siguiente:

Se halla el gradiente de la fórmula del error (cuadrático para este caso).

$$\frac{\partial E}{\partial w_j} = -Err * \frac{\partial g(\sum_{j=0}^{n} w_j * x_j)}{\partial w_j} \qquad \textbf{Ec 3.4}$$

g: Función de activación del nodo.

w_j: Peso de la conexión con el nodo j.

x_j: Salida del nodo j.

E: Error cuadrático.

La modificación de un peso se realiza con la siguiente fórmula:

$$w_j = w_j + \alpha * Err * g'\left(\sum_{j=0}^{n} w_j * x_j\right) * x_j \qquad \textbf{Ec 3.5}$$

α: Tasa de aprendizaje.

De una forma más detallada, y para perceptrones multicapa se tiene:

$$E = \frac{1}{2} Err^2 = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{r} (y_{i,j} - h_{i,j})^2$$ **Ec 3.6**

$y_{i,j}$: salida real (esperada) de la neurona j en la capa i.

$h_{i,j}$: salida de la neurona j en la capa i dada por la red.

El parámetro m es la cantidad de capas en la red. El valor r es la cantidad de nodos en la capa m.

Hay que tener en cuenta que el valor de $h_{i,j}$ es la misma función de activación en el nodo (i, j), por lo tanto:

$$h_{i,j} = G(\sum_{k=1}^{r} w_{i-1,k,j} * h_{i-1,j})$$ **Ec 3.7**

$w_{i-1,k,j}$: El peso del hilo entre la nodo k en la capa i-1 al nodo j en la capa i.

r: Número total de nodos en la capa i-1.

G: Función de activación de la neurona j en la capa i.

Luego la formula de actualización de los pesos sería:

$$w_{i,k,j} = w_{i,k,j} + \Delta w_{i,k,j}$$ **Ec 3.8**

$\Delta w_{i,k,j}$: Variación que debe tomar el peso. Y se hallaría con la formula:

$$\Delta w_{i,k,j} = -\alpha * \frac{\partial E}{\partial w_{i,k,j}} \qquad \text{Ec 3.9}$$

$\frac{\partial E}{\partial w_{i,k,j}}$: Es el gradiente del error con respecto a los pesos (w).

Ec 3.10

$$\frac{\partial E}{\partial w_{i,k,j}} = \left(y_{i,j} - h_{i,j}\right) * G_{i,j}{}'\left(w_{i,k,j} * h_{i,j}\right) * h_{i,j}$$

$G_{i,j}{}'$: Derivada de la función de activación del nodo j en la capa i.

El algoritmo se puede desarrollar haciendo cambios en los pesos a medida que se va trabajando, o guardando estos para al terminar la propagación, actualizar todos los pesos (actualización por lotes).

Cada ciclo en el cual se haga entrenamiento, con todos los ejemplos suministrados, se denomina época.

Una técnica utilizada para que el entrenamiento no se quede en un mínimo local es la de hacer un promedio entre los $\Delta w_{i,k,j}$ actuales con los de la iteración anterior, la nueva fórmula de actualización de pesos se convertiría en:

$$\Delta w_{i,k,j} = \beta * \Delta w_{i,k,j} - \alpha * \frac{\partial E}{\partial w_{i,k,j}} \qquad \text{Ec 3.11}$$

En donde β es una contante de momento. Esta técnica es la regla delta generalizada o gradiente descendiente con momentos.

Además de la variación del gradiente descendiente por momentos para que el

aprendizaje no se estanque en un mínimo local, existen otro tipo de variaciones que buscan mejorar la velocidad de convergencia. Tales algoritmos se basan en técnicas de programación en paralelo como Newton, Quasi-Newton, Levenberg-Marquardt, entre otras.

3.6.2 Propagación hacia atrás con variación en la tasa de aprendizaje.

Una forma sencilla de obtener un aumento en la velocidad de convergencia del de propagación hacia atrás (*backpropagation*), consiste en ir configurando la rata de aprendizaje a medida que el algoritmo está trabajando. La razón de esto es que con el gradiente descendiente, si se tiene una rata de aprendizaje muy alto, el algoritmo puede quedarse en un mínimo local, además que se observará una considerable oscilación.

Si la tasa de aprendizaje es muy pequeña, el algoritmo tardara mucho tiempo en encontrar el mínimo. Esta técnica puede ser utilizada tanto en propagación hacia atrás simple como con gradiente descendiente con momentos.

3.4.3 Propagación hacia atrás *resilient* (Rprop)

Las funciones comúnmente utilizadas como funciones de activación en las redes neuronales artificiales, tienen como característica que acepta valores de entrada infinitos en la función (y su salida finita); esto representa un problema cuando se realiza el gradiente de la función. El gradiente tiene magnitudes muy pequeñas, causando pequeños cambios en los pesos y conexiones; aun cuando se encuentra lejos del punto óptimo.

El algoritmo *Rprop* tiene como propósito mejorar el comportamiento indeseado del

gradiente (derivadas parciales). Solo es necesaria la dirección de la derivada para poder actualizar los pesos; la magnitud de la derivada no tiene efecto. El cambio realizado en los pesos se halla con otros parámetros por separado.

3.4.4 Algoritmo quasi-newton

Una de las variaciones al algoritmo de propagación hacia atrás, consiste en utilizar el método de Quasi-Newton, en vez de gradiente conjugado para optimizar la convergencia. En este caso se tendría este paso básico:

$$w_{i,k,j} = w_{i,k,j} - A^{-1} * \Delta w_{i,k,j} \qquad \textbf{Ec 3.12}$$

$-A^{-1}$: Matriz Hessiana de los valores actuales de pesos y conexiones.

Este algoritmo es más rápido que el gradiente conjugado. Pero requiere de mucha capacidad de cómputo para manejar la matriz Hessiana.

3.4.5 Algoritmo Levenberg-Marquardt

Este algoritmo no necesita hallar matrices hessianas; algo que aumenta la velocidad de convergencia. Cuando el error se halla por mínimos cuadrados, la matriz hessiana se puede aproximar a:

$$H = J^t * J \qquad \textbf{Ec 3.13}$$

Y el gradiente puede ser computado como:

$$g = J^t * E \qquad \text{Ec 3.14}$$

J^t: Es la transversa de la matriz Jacobiana que contiene las primeras derivadas de los errores de la red con respecto a los pesos y conexiones.

La formula de actualización de los datos sería:

$$w_{i,k,j} = w_{i,k,j} - [J^t * J + \mu * J]^{-1} * J^t * E \qquad \text{Ec 3.15}$$

μ: un numero escalar.

Si el escalar es igual a cero, la función se aproxima al método de Newton. Cuando μ es un valor grande, la función se convierte en un gradiente con tasas pequeños. Este valor debe ir decreciendo a medida que el error se reduce, y aumentando si lo mismo ocurre con el error.

3.5 Perceptrón multicapa como aproximador universal.

Estudios realizados por Cybenko sobre los perceptrones multicapa en el año 1989, dieron como resultado un teorema (Teorema de Cybenco), en el cual se demuestra que una red perceptrón multicapa, con una sola capa oculta y un número determinado de nodos en esta, sería capaz de aproximar cualquier función continua, de R^n en R. La demostración parte del teorema de Kolmogorov.

Cualquier función continua $f(x_1, x_2, x_3, \dots, x_n)$ definida en $[0\ 1]^n$, $n \geq 2$, se puede representar mediante:

$$f(x_1, x_2, x_3, \dots, x_n) = \sum_{i=1}^{2n+1} g_i\left[\sum_{j=1}^{n} \varphi_{ij}(x_j)\right] \qquad \text{Ec 3.16}$$

g_i: Funciones continuas y reales de una sola variable, escogidas adecuadamente.

φ_{ij}: Funciones continuas y monótonamente crecientes independientes de f.

La conclusión es que cualquier función de R^n en R^m puede ser expresada en términos de sumas y composiciones de funciones de una sola variable.

El teorema de Cybenko sería el siguiente:

Sea φ una función sigmoidea. Dada cualquier función continua en $[0\ 1]^n$ (o en cualquier conjunto compacto de R^n) y un $\varepsilon > 0$, existe unos vectores $w_1, w_2, w_3, \dots, w_N, \alpha\ y\ \theta$, y una función parametrizada $G(\cdot, w, \alpha, \theta)$ de $[0\ 1]^n$ en R tal que:

$$|G(x, w, \alpha, \theta) - f(x)| < \varepsilon, \qquad para\ todo\ x \in [0\ 1]^n$$

$$G(x, w, \alpha, \theta) = \sum_{i=1}^{N} \alpha_i \varphi(w_i^T x + \theta_i) \qquad \text{Ec 3.17}$$

$$w_j \in R^n.\ \alpha_i, \theta_i \in R\ .$$

Y además se tiene:

$$w = (w_1, w_2, w_3, \dots, w_N)$$
$$\alpha = (\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_N)$$
$$\theta = (\theta_1, \theta_2, \theta_3, \dots, \theta_N)$$

Hornyk, Stinchcombe y White en 1990 demostraron que estas redes también pueden aproximar la derivada de una función; incluso si esta es diferenciable a trazos.

3.6 Perceptrón multicapa como clasificador universal.

Cybenko también demostró que el perceptrón multicapa con una sola capa oculta, en la cual cada nodo tendría como función de transferencia una función sigmoidea y una unidad de salida con función de transferencia lineal, es un clasificador universal, es decir que puede aproximar con la precisión deseada cualquier función de la forma:

$$f(x) = j \; siempre \; y \; cuando \; j \; \epsilon \; P_j \qquad \text{Ec 3.18}$$

Donde f es una función de A^n en el conjunto $\{1, 2, 3, \ldots, k\}$ cerrado y acotado en R^n, y $P_1, P_2, P_3, \ldots, P_k$ es una partición de A^n en k subconjuntos disjuntos.

3.7 Algunas ideas para la elección del número de capas y de neuronas

El número de capas y el número de neuronas por capa son decisiones importantes en una red unidireccional que usa una topología *perceptrón multicapa,* aunque estas decisiones son realizadas usando la experiencia propia. No existe una regla fija que indique sus valores sino que se establecen por la intuición del diseñador de la red.

Sin embargo, existen un conjunto de reglas que se han seguido a lo largo del tiempo y que han sido seguidas por investigadores e ingenieros aplicándolas a las arquitecturas que aplican a sus problemas.

- **Regla 1:** Cuanto más aumenta la complejidad en las relaciones entre los datos de entrada y los datos de salida, entonces el número de neuronas en las capas ocultas también deberían aumentarse.

- **Regla 2:** Si el proceso que se está modelando es separable en dos etapas, entonces nueva(s) capa(s) pueden ser adicionadas. Si el proceso no es separable en varias etapas, entonces las capas adicionales simplemente permiten memorizar y no siempre es una solución general.

- **Regla 3:** El número de patrones de entrenamiento disponibles establecen una cota superior en el número de neuronas en la(s) capa(s) oculta(s). Para calcular esta cota superior primero hay que dividir el número total de neuronas de las capas de entrada y salida. Dividir el resultado de nuevo por un factor de escala entre 5 y 10. Factores de escala grandes son usados cuando existen datos con ruido. Cuando los datos de entrada tienen ruido será necesario incluso factores más elevados (20 o 50), mientras que datos en los que no exista apenas ruido se podría disminuir el factor hasta un valor de 2. Es muy importante que la(s) capa(s) oculta(s) tengan pocas neuronas ya que demasiadas neuronas podrían llevar a que los patrones de entrada fueran memorizadas de manera que ningún tipo de generalización fuera posible, haciendo que la red fuera inútil con nuevos datos de entrada.

4. REDES NEURONALES EVOLUTIVAS

La optimización de la estructura de una red neuronal puede ser clasificada de acuerdo a la meta a alcanzar. Algunos esquemas han propuesto la optimización de los pesos sinápticos, otros afirman que lo más importante es la arquitectura y otros acercamientos han incluido funciones de activación y reglas de aprendizaje. Sin embargo el área más interesante para nuevas investigaciones radica en la combinación de estos enfoques de optimización.

Para el diseñador de la red uno de los aspectos más relevantes de su trabajo es la de encontrar el número de capas ocultas, neuronas y funciones de activación con las que se debe configurar la red. El entrenamiento también es un aspecto importante para lograr una buena solución, sin embargo existen técnicas basadas en programación no lineal que han demostrado ser muy efectivas, y por tanto, esta labor es asignada a algoritmos especializados en aproximar los pesos adecuados para cada una de las conexiones. El diseñador, por lo tanto, debe concentrarse en el análisis de las topologías candidatas para optimizar el problema.

Una teoría que ha ganado terreno en el campo de la investigación de las redes neuronales artificiales es la búsqueda de pesos[21], capas ocultas[22], neuronales y funciones de activación[23] por medio de algoritmos genéticos.

Los algoritmos genéticos son métodos adaptativos, generalmente usados en problemas de búsqueda y optimización de parámetros, basados en la reproducción

[21] **T.Sasaki and M Tokoro**. Evolving learnable neural networks under changing environment with various rates of inheritance of acquired characters: Comparison between darwinian and lamarckian evolution. Artifical Life, pages 203-223, 1999.
[22] **K. Stanley and R. Miikkulainen**. Evolving neural networks through augmenting topologies, technical report ai01-290. Technical report, department of computer science, university of Texas as Austin 2001.
[23] **Xin Yao and Young Liu.** Evolving artificial neural networks Through evolutionary programming. In Evolutionary Programming, pages 257-266, 1996.

sexual y en el principio supervivencia del mas apto. Mas formalmente, y siguiendo la definición dada por Goldberg[24],

"Los Algoritmos Genéticos son algoritmos de búsqueda basados en la mecánica de selección natural y de la genética natural. Combinan la supervivencia del más apto entre estructuras de secuencias con un intercambio de información estructurado, aunque aleatorizado, para constituir así un algoritmo de búsqueda que tenga algo de las genialidades de las búsquedas humanas."

Para alcanzar la solución a un problema se parte de un conjunto inicial de individuos, llamado población, generado de manera aleatoria. Cada uno de estos individuos representa una posible solución al problema. Estos individuos evolucionarán tomando como base los esquemas propuestos por Darwin[25] sobre la selección natural, y se adaptaran en mayor medida tras el paso de cada generación a la solución requerida.

Los métodos en que se basa este trabajo son aquellos en que se utilizan específicamente en el diseño de la arquitectura de la red (capas y funciones de activación). El entrenamiento de los pesos de la red se realizará mediante métodos de aprendizaje neuronal tradicional (no evolutivo), ya que han demostrado ser muy eficientes para ese propósito. El algoritmo evolutivo utiliza el error de la red entrenada como medida de desempeño para guiar la evolución. De esta forma la búsqueda de la mejor estructura R de una red neuronal se puede ver como la maximización de la función f dada por la ecuación mostrada a continuación:

$$R = f(M_k, N_k, G_k, T_k, E_s) \qquad \textbf{Ec 4.1}$$

[24] **Goldberg D. E.** *A comparative analysis of selection schemes used in genetic algorithms*. In Gregory Rawlins, editor. Foundations of Genetic Algorithms, pages 69-93, San Mateo, CA: Morgan Kaufmann Publishers, 1991.
[25] **Darwin, C**. On the Origin of Species by Means of Natural Selection. John Murray, London, 1859.

Donde M_k representa el número de capas de la red, N_k es el número de neuronas por cada capa, G_k es la función de cada una de las capas, T_k es el algoritmo de entrenamiento y E_s son los ejemplos con que se cuenta para entrenar la red neuronal. Además cada una de estas variables está limitada a un conjunto finito de valores para acotar el espacio de búsqueda, de esta forma $M_k \in \{1, m\}$ donde m es el máximo número de capas y $N_k \in \{0, n\}$ donde n es el máximo número de neuronas por capa.

La selección de funciones de activación se realiza de acuerdo con el problema a resolver y a criterio del investigador, en ocasiones por ensayo y error. En la literatura no existe un criterio estándar para la selección de estas funciones de activación en las redes neuronales[26]. En el sistema se tienen como base cuatro funciones de activación las cuales son establecidas por el algoritmo genético buscando los mejores resultados para cada problema. De esta forma $G_k \in (logsig, radbas, purelin, tansig)$,

Por último los algoritmos de entrenamiento que se utilizarán para realizar la optimización de la estructura son:

- Traingda: Gradiente descendente con tasa de aprendizaje adaptativa con propagación hacia atrás.

- Traingdx: Gradiente descendente con momento y tasa de aprendizaje adaptativa con propagación hacia atrás

- Trainrp: Propagación hacia atrás Resilient.

- Trainlm: Propagación hacia atrás Levenberg-Marquardt.

[26] **J. Branke**. Evolutionary algorithms for neural network design and training. Technical report no. 322, University of Karlsruhe, Institute AIFB, 1995

- Trainbfg: Propagación hacia atrás BFGS quasi-Newton.

De esta forma $T_k \in (Traingda, Traingdx, Trainrp, Trainlm, Trainbfg)$. Definidos los parámetros dados por la ecuación 4.1, se procede a resolver el problema usando algoritmos genéticos (AG). La función de optimización para el algoritmo genético es la mostrada a continuación:

$$f(x) = \frac{1}{r} \sum_{i=1}^{r} f(Mx, Nx, Gx, Tx) \quad \textbf{Ec 4.2}$$

Debido a que los pesos iniciales tienen una incidencia grande en los resultados que se obtienen, este entrenamiento se repite r veces y se promedian los resultados. Es importante resaltar que si se aplica la función f dos veces sobre la misma estructura $[Mx, Nx, Gx, Tx]$, en cada una de las ocasiones se obtendrán resultados diferentes ya que los pesos iniciales serán distintos cada vez.

Con este diseño, el AG busca obtener el mejor individuo sujeto a la función de optimización dada por la ecuación 4.2. Al finalizar el proceso de evolución del AG se obtiene la configuración de red que maximiza el desempeño de la red para los datos de entrenamiento E_s.

Con la realización de esta herramienta se brindará una importante ayuda para los futuros proyectos que se deseen realizar por medio de redes neuronales ya que el proceso más complicado (hallar la arquitectura adecuada para la solución del problema) se hará de forma automatizada y fundamentada su elección.

4.1 El problema de la permutación

Es el problema más grande asociado con la codificación de las redes neuronales. Se presenta cuando dos redes neuronales las cuales son computacionalmente equivalentes contienen unidades ocultas en diferente orden.

El problema de la permutación es un tema tratado muy frecuentemente en trabajos sobre diseño de topologías de redes neuronales mediante algoritmos genéticos. Dos redes neuronales pueden ser funcionalmente equivalentes independientemente del orden de sus neuronas ocultas.

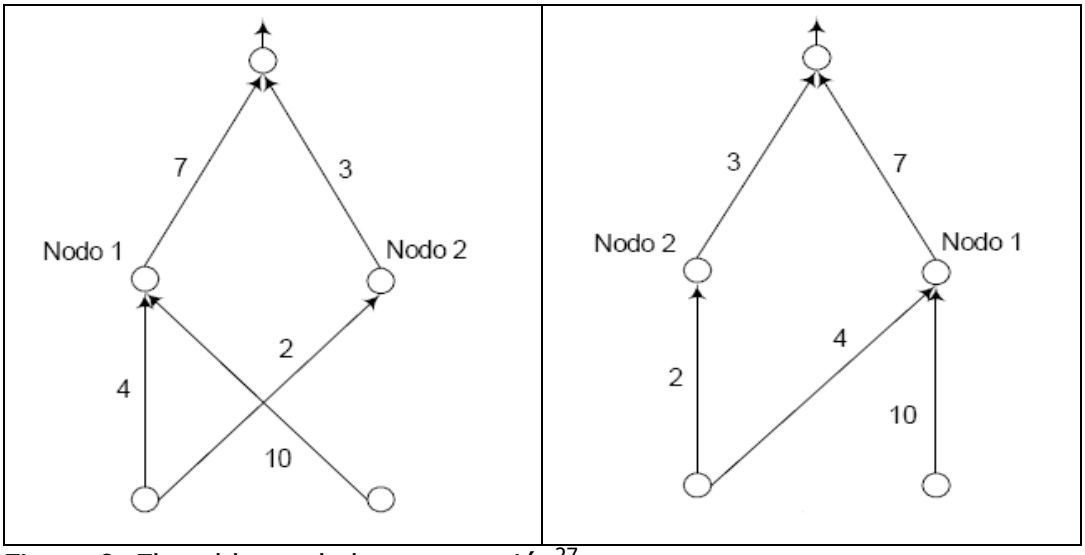

Figura 9. El problema de la permutación[27]

Dado que una red neuronal puede tener varios genotipos que la representan, la probabilidad de que el operador de cruza produzca de ellos un individuo más adaptado es baja. Si el problema de la permutación surge se piensa que la tasa de crecimiento del número de redes probables (n) es de n!, lo que impide que haya un rápida convergencia en el algoritmo genético.

[27] Fuente: **Fiszelew A.** Generación automática de redes neuronales con ajuste de parámetros basado en algoritmos genéticos, Universidad de Buenos Aires Facultad de Ingeniería, 2000.

Sin embargo se ha encontrado que el problema de la permutación no parece tener gran impacto en el desempeño del algoritmo genético.

Una solución propuesta por Hancock[28] y Whitley[29] sugiere asignar roles a cada una de las unidades ocultas para romper la simetría con entre otras redes idénticas. O el sistema de rastreo de la proveniencia histórica de los genes propuesta por Stanley y Miikkulainen[30] para indagar el origen de una neurona o conexión en particular de nuevo rompiendo la simetría producido por el problema de la permutación.

Para atenuar los efectos del problema de la permutación, se suele también utilizar un cruce de fenotipos, es decir, una cruce que trabaja sobre redes neuronales en lugar de hacerlo sobre las cadenas de genes que forman la población.

4.2 Estado del arte.

Dentro del estudio del estado del arte es importante considerar las características de las redes neuronales obtenidas mediante métodos evolutivos, además algunas de las muchas posibles soluciones para representar una red neuronal, en un formato conveniente para que los algoritmos genéticos transformen una red neuronal a un código genético o cromosoma[31].

[28] **P.J.B. Hancock**. Pruning neural nets by genetic algorithm. In I. Aleksander and J.G. Taylor, editors, Proceeding of the International Conference on Artificial Neural Networks, Brighton, pages 991-994. Elsevier, 1992

[29] **D. Whitley, T. Starweather, and C. Bogart.** Genetic algorithms and neural networks: Optimizing connections and connectivity. Parallel Computing, 347-361,1990

[30] **K. Stanley and R. Miikulainen.** Evolving neural networks through augmenting topologies, technical report ai01-290. Technical report, Department of Computer Science, University of Texas as Austin 2001

[31] **D. Curran, C. O'Riordan,** Applying Evolutionary Computation to Designing Neural Networks: A Study of the Satate of the Art, technical report NUIG-IT-111002, National University of Ireland, Galway, 2002

4.2.1 Métodos evolutivos:

La evolución de las redes neuronales puede ser clasificada de acuerdo a la meta a alcanzar. Algunos esquemas han propuesto la evolución de los pesos sinápticos, otros afirman que lo más importante es la arquitectura y otros acercamientos han incluido funciones de transferencia y reglas de aprendizaje.

4.2.1.1 Evolución de los pesos:

La evolución de los pesos implica que la arquitectura de la red debe permanecer constante, esto implica un grado de pre-procesamiento por parte del diseñador humano. La motivación para usar este método es la de corregir la falla que tienen las técnicas basadas en gradiente descendiente tales como propagación hacia atrás el cual por ejemplo puede quedar atrapado fácilmente en mínimos locales.

Muchas investigaciones han estudiado la evolución de los pesos, otras usan la capacidad de la búsqueda global para determinar el espacio de búsqueda de los pesos para el problema y luego usan propagación hacia atrás como una búsqueda local para refinar los pesos.

4.2.1.2 Evolución de la arquitectura:

El punto de vista de este método es que al encontrar una estructura adecuada, un algoritmo como propagación hacia atrás puede ser usado para encontrar los pesos correctos. Los métodos previos al enfoque evolutivo consisten en dos operaciones básicas: constructiva y destructiva.

El método constructivo inicia con una red mínima y sucesivamente agrega nodos y

conexiones hasta que la red es capaz de solucionar el problema deseado con suficiente acierto. El método destructivo en cambio comienza con una red que ya funciona y sucesivamente remueve pesos y neuronas hasta que la red ya no pueda solucionar el problema. Existen claros problemas al utilizar estos enfoques ya que se puede quedar atrapado en máximos globales. El hecho de que la superficie de búsqueda sea infinita y no diferenciable hace que los algoritmos genéticos sean un buen candidato para tener éxito.

4.2.1.3 Funciones de Transferencia:

En este enfoque se toma una proporción de las funciones de activación como fijas y se permite al algoritmo evolutivo adaptarlas a una combinación útil de acuerdo a la situación. Con frecuencia la evolución de las funciones de transferencia se usa combinada con otros enfoques como evolución de los pesos o arquitectura.

4.2.1.4 Reglas de aprendizaje:

Las reglas de aprendizaje (como tasa de aprendizaje o momento) generalmente son difíciles de determinar manualmente por esto son codificados en un gen y cada red tiene la capacidad de evolucionarlo.

4.2.1.5 Evolución simultánea:

Uno de los enfoques más interesantes de este método es la evolución simultánea de pesos y arquitectura, la mayor ventaja de esta combinación es que una red totalmente funcional puede ser hallada sin necesidad de la interacción humana. La desventaja de este método es que el nivel desempeño debe ser muy alto lo que repercute en que los tiempos de convergencia de este enfoque sean mayores.

4.2.2 Estrategias de Codificación

Un aspecto crucial para el éxito de la evolución de una red neuronal depende de la codificación usada, estas pueden ser divididas en dos grupos: codificación directa e indirecta.

4.2.2.1 Codificación Directa.

En este enfoque uno o más parámetros de la red neuronal están representados en un código genético.

4.2.2.2 Codificación de las conexiones:

La codificación de los pesos está relacionada con un mapeo de todas las conexiones de la red neuronal en un código genético. Un ejemplo de esta codificación es la usada por Miller's Innervator[32] donde cada uno de las conexiones de los nodos era representado con un bit simple.

Se obtenía un matriz derivada conteniendo un mapa completo de las conexiones de la red neuronal. Otra aproximación a la codificación de las conexiones es representar los pesos de cada conexión de la red neuronal. Algunos de los estudios han utilizado esta representación definida como cadenas binarias, números reales, fracciones enteras o cadenas binarias de tamaño variable.

[32] **P.M Todd G. F. Miller and S.U. Hedge**. Designing neural networks using genetic algorithms. In proceedings of the Third International conference on Genetic Algorithms and their Applications, pages 379-384,1989

4.2.2.3 Codificación basada en los nodos

Este método se centra en el número de neuronas que deben ser usadas. Mientras que en los enfoques de evolución de pesos se asume que una arquitectura ha sido previamente diseñada, la construcción de una arquitectura es el problema aquí y el que debe ser afrontado mediante los algoritmos genéticos. Ejemplos de esta codificación son las usadas por Schiffman[33] o el sistema similar, GANNet[34].

4.2.2.4 Codificación basada en grafos.

Esta codificación ve a la red como una malla de funciones y terminales, donde las funciones son las neuronas y las terminales son las variables de entrada. El cromosoma consiste en una lista ordenada de los nodos de la red con un índice indicador de su posición en el código genético. El modelo propuesto por Pujol[35] permite múltiples funciones de activación y tipos de red, y el número de nodos está hecho por cada elemento de la población.

4.2.2.5 Expresiones-S

Usa expresiones simbólicas de LISP donde cada red es simbolizada por un número de funciones representado nodos y terminales. Koza y Rice[36] utilizaron esta representación como un árbol de parámetros. El cruce es realizado en un subnivel del árbol garantizando que las proporciones de aprendizaje no son totalmente

[33] **W. Schiffmann, M. Joost, and R. Werner**. Application of genetic algorithms to the construction of topologies for multilayer perceptrons. In proceedings of the International Conference of Artificial Neural Networks and Genetic Algorithms pages 675-682, 1993

[34] **David W. White**. GANNet: A genetic algorithm for searching topology and weight spaces in neural network design. PhD thesis, University of Maryland College Park, 1994

[35] **Pujol J. and Poli R**. Efficient evolution of asymemetric recurrent neural networks using PDGP inspired two dimensional representation. Lecture Notes in Computer Science, 1998

[36] **John R. Koza and James P. Rice**. Genetic generation of both the weights and architecture for a neural network. In International Joint Conference on Neural Networks, IJCNN 91, volume II, pages 397-404.Washington State Convention and Trade Center, Seattle, WA, USA, pages 8-12 1991. IEEE Computer Society Press

alteradas.

4.2.2.6 Codificación basada en capas

Usa un cromosoma el cual esta subdivido en dos áreas correspondientes a la capas de la red, ejemplos de esta codificación es el mostrados en el sistema Mandischer[37].

4.2.2.7 Codificación basada en Marcas

Está inspirada en la estructura del ADN[38]. Los cromosomas están descritos como circulares de este modo el final de cada uno estará cubierto con el inicio del otro, se usan marcas para definir el principio y el final de la definición.

Esto permite mayor libertad en la definición de una red neuronal y las operaciones de cruce y mutación pueden operar sin restricciones.

4.2.3 Codificación Indirecta

Representa la red en términos de instrucciones assembly y recetas. Las motivaciones de este método son las de obtener menor tamaño y modularidad.

4.2.3.1 Matriz re-escrita

Este método comienza con una matriz de 2x2 y repetitivamente aplica reglas de generación para cada elemento no terminal hasta que todos sus elementos sean

[37] **M. Mandischer**. Representation and evolution of neural networks. In R. F. Albrecht, C.R. Reeves, and N.C Steele, editors, Artificial Neural Nets and Genetic Algorithms Proceedings of the International Conference at Innsbruck, Austria, pages 643-649. Springer, Wien and New York, 1993

[38] **Joao Carlos Figueira and Ricardo Poli**. Efficient evolution of asymmetric recurrent neural networks using PDGP-inspired two dimensional representation. Lecture Notes in Computer Science, 1998

terminales[39].

4.2.3.2 Codificación celular

Representa la red como árboles gramaticales[40]. El bloque construido por codificación celular representa un nodo en un grafo ordenado. El desarrollo comienza con una celda llamada celda ancestro, la cual está conectada a celdas de entrada y salida. La cabeza de lectura de la celda está ubicada al inicio del código celular y ejecuta varios operadores recursivamente hasta representar toda la red.

4.2.3.3 Codificación de aristas

Es un esquema similar a la codificación celular, pero esta en cambio crea el grafo de la red utilizando aristas en lugar de nodos. Mientras que la codificación celular evalúa cada gramática del árbol con una búsqueda de primero en amplitud (lo que asegura una ejecución paralela), este método lo hace con una búsqueda primero en profundidad[41].

4.2.3.4 Sistemas L

El conjunto de sistemas Lindermayer son sistemas de codificación basados en el trabajo de Lindermayer donde cada celda intercambia información con su vecino. Se usan gramáticas especializadas en el cual las reglas de producción son aplicadas en una tasa más paralela que secuencial[42].

---

[39] **P. M. Todd G. F. Miller and S. U. Hedge**. Designing neural networks using genetic algortihms. In Proceedings of the Third International Conference on Genetic Algorithms and Their Applications, pages 379-384, 1989

[40] **Frederic Gruan**. Automatic definition of modular neural networks. Adaptive Behavior, pages 151-183, 1995.

[41] **Sean Luke and Lee Spector**. Evolving graphs and networks with edgar encoding: Preliminary report. In John R. Koza, editor, Late Breaking Papersat the Genetic Programming 1996 Conference Standford University, pages 117-124, Standford University, 1996

[42] **H.M. Voigth, J. Born, and I. Santibañez-Koref**. Evolutionary structuring of artificial neural networks. Technical Report TR'02'93, TU Berlin, 1993

4.2.3.5 Codificación creciente

Está basado en la simulación del crecimiento de los pesos sinápticos[43]. Se da la posibilidad de que los nodos se dividan y muevan en dos dimensiones del ambiente. El código genético especifica el tipo de celda que está siendo representado, el número de divisiones permitido por celda, la tasa de crecimiento y el ángulo de crecimiento.

4.2.4 Estudios realizados

No son pocos los estudios sobre la configuración y/o entrenamiento de redes neuronales mediante algoritmos genéticos, de hecho este problema ha sido objeto de estudio durante muchos años sin lograrse una solución "óptima" al problema. Se han dedicado libros completos a tratar el tema, cada uno abordado desde un enfoque diferente, desde programación genética hasta lógica difusa, sin embargo los resultados a pesar de ser mejores a los de la configuración por métodos tradicionales no logran establecer un método como el paradigma a seguir a la hora de enfrentar el problema.

No obstante entre los múltiples estudios ejecutados se ha podido corroborar el excelente desempeño que muestran los algoritmos genéticos frente a los otros métodos empleados y obviamente ante al entrenamiento por prueba y error. Algunos de los estudios más sobresalientes sobre este campo son los realizados por P. Werbos (1974)[44], P.J Angeline**,** G.M Saunders and J.B Pollack (1994)[45] y J.D Shaffer (1994)[46].

---

[43] **D. Parisi A. Cangelosi and S. Nolfti**. Cell division and migration in a 'genotype' for neural networks. Network: computation in neural systems, 1994.

[44] **Werbos, P.** Beyond Regression: New tools for prediction and Analisis in the behavioral sciences, doctoral dissertation, Harvard, Cambridge, Mass, 1974.

[45] **Angeline P.J, Saunders G.M and Pollack J.B.** An evolutionary algorithm that constructs recurrent neural networks, IEEE Trans. Neural Networks, Vol 5, 1994 pp 54-65.

[46] **J.D. Schaffer,** Combinations of genetic algorithms with neural networks or fuzzy systems ,IEEE press, 1994, pp 371-382

En el ámbito iberoamericano se encuentran las investigaciones realizadas por Juan Peralta, German Gutiérrez y Araceli Sanchis (2007)[47] de la universidad Carlos III de Madrid, Abel Fiszelew (2000)[48] de la universidad de Buenos Aires y Delia Johanna Muñoz (2006)[49] de la universidad tecnológica de Pereira.

[47] **Peralta J, Gutierrez G, Sanchis A**. Design of Artificial Neural Networks based on genetic Algorithms to forecast time, CAOS Group, Computer Science Department, University Carlos III of Madrid, 2007

[48] **Fiszelew A**. Generación automática de redes neuronales con ajuste de parámetros basado en algoritmos genéticos, Universidad de Buenos Aires Facultad de Ingeniería, 2000

[49] **Muñoz J**. Diseño y entrenamiento en paralelo de redes neuronales, por medio de algoritmos genéticos desordenados y altamente recursivos. Universidad Tecnológica de Pereira, 2006

5. METODOLOGÍA

Para el desarrollo de este proyecto se tomó como modelo de vida de desarrollo software el prototipado evolutivo. Este tipo de desarrollo está enmarcado por una etapa de análisis de requerimientos, para posteriormente irlos implementando en prototipos, los cuales evolucionan durante el desarrollo del proyecto; siendo estos analizados, para luego realizar cambios en su interior y reproducir nuevamente los pasos, con la finalidad de llegar a un prototipo final que satisfaga las necesidades.

Las actividades a realizar tienen la ventaja de no ser secuenciales, luego es posible regresar a un punto anterior para alimentarse de nuevas ideas o requerimientos.

Las actividades son:

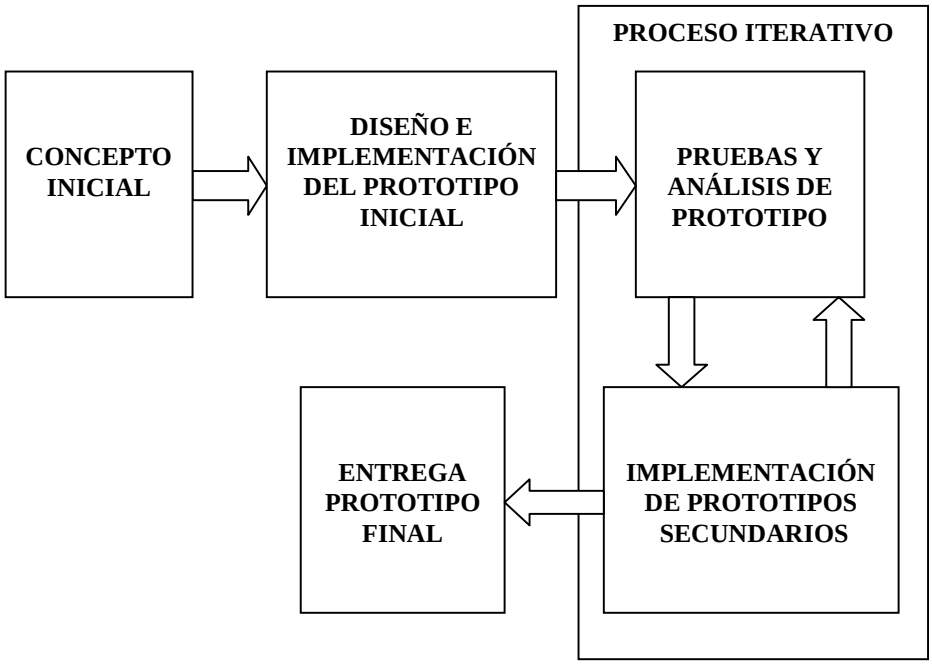

Figura 10. Modelo de prototipado evolutivo [50]

[50] Fuente: Autores del proyecto.

5.1 Fases de desarrollo del software

5.1.1 Concepto inicial: En esta etapa se realiza el estudio de requerimientos del sistema. Estudio de las diferentes formas de codificación, mutación, cruce y selección.

5.1.2 Diseño e implementación del prototipo inicial: Se diseña un primer prototipo, sobre el cual se van a realizar las sucesivas modificaciones y adiciones.

5.1.3 Iteración de refinar el prototipo hasta que sea aceptable: En esta etapa se comienzan a realizar las entregas de prototipos hasta llegar al prototipo aceptable. Se realizan comparaciones con los resultados de utilizar fuerza bruta para hallar la mejor arquitectura.

5.1.4 Completar y entregar el prototipo: una vez encontrado el prototipo más adecuado, se realiza la entrega.

La característica más importante de este ciclo de vida y que son convenientes para el presente proyecto, es la del conocimiento teórico que se tiene en un principio de las RNA y de los algoritmos genéticos, pero no de los algoritmos mas adecuados para su integración, esta metodología permite encontrar en cada iteración las técnicas mas apropiadas para el prototipo final, además de ir adicionando módulos que, a medida que ocurra el desarrollo, se determinen que sean necesarias.

5.2 Herramienta de desarrollo

Para la realización del proyecto se debieron tomar en cuenta muchas consideraciones para escoger la herramienta de desarrollo. Primero que todo se requería crear en cada iteración redes neuronales con topologías diferentes, estas

redes deberían ser entrenadas durante un número considerable de veces con la posibilidad de elegir diferentes algoritmos de entrenamiento. Además se debería tener la posibilidad de implementar diversos tipos de funciones de activación para utilizar en cada capa oculta de las redes neuronales creadas.

Esto requería de un inmenso esfuerzo y tiempo de desarrollo para implementar un entorno completo de creación y entrenamiento de redes neuronales, siendo el principal objetivo de la tesis la aplicación de algoritmos genéticos para la búsqueda de las mejores configuraciones; no se justificaba la realización de este trabajo. La decisión tomada fue el utilizar una herramienta especializada en el manejo de las redes neuronales artificiales y que involucrara todos los parámetros que se necesitaban considerar, la mejor opción parecía usar Matlab un completo entorno que cumplía con los requerimientos necesarios. Esto sumado a que en el grupo de investigación en ingeniería biomédica (GIIB), donde se realizó el proyecto se contaba con las licencias necesarias para su desarrollo hizo que al final se usara esta herramienta para el desarrollo del proyecto.

Después de tener resuelto el problema de la creación y entrenamiento de redes neuronales; se tenía vía libre para el desarrollo de la parte correspondiente a la modificación de las configuraciones por métodos evolutivos, lo que es en si el principal problema abordado en el proyecto.

5.3 Aplicación de la metodología

5.3.1 Prototipo 1

Debido al desconocimiento inicial sobre los conceptos y los métodos que se deberían utilizar para dar solución al problema en este prototipo, las principales actividades desarrolladas fueron las encaminadas a la comprensión del problema y

el estudio de las técnicas involucradas en su solución. Para ello se consultó en libros, artículos de revistas especializadas, tesis de grado; así como también fueron de gran utilidad los conceptos iniciales sobre redes neuronales proporcionados por nuestro director de proyecto.

Era de especial importancia entender cuáles eran los tipos de redes neuronales que se deberían estudiar, de tal manera que se pudiera dar solución al mayor número de problemas posibles. Fue necesario consultar información de las redes neuronales y sus aplicaciones, así como de experiencias previas que se pudieran encontrar sobre la configuración de redes neuronales por medios no convencionales; esto para establecer un punto de partida en el desarrollo de la herramienta.

En el marco de este prototipo se desarrolló la lógica principal del algoritmo genético, el cual estaba diseñado inicialmente sólo para la determinación de las capas y neuronas ocultas que se necesitaban para dar solución a un problema específico. Este prototipo no tenía interfaz gráfica por lo cual todos los procedimientos tenían que ser manipulados directamente desde el código del proyecto lo que dificultaba la ejecución del proceso.

Después de la realización de este prototipo se hizo evidente la necesidad de implementar un método más sencillo para la carga de datos y la modificación de los parámetros de búsqueda del problema. A pesar de que en este punto el programa se encontraba en un estado muy primitivo, se tuvieron muy buenos resultados en cuanto a la determinación de las primeras configuraciones.

5.3.2 Prototipo 2

Con la realización del segundo prototipo surgieron notables mejoras algunas a nivel de funcionamiento y otras a nivel visual. La primera de estas mejoras es

respecto a la lectura de los datos, en el primer prototipo los datos debían ser introducidos directamente en la función encargada de realzar la búsqueda por medio de algoritmos genéticos, sin embargo con el desarrollo del segundo prototipo se creó el método para la carga de datos, por medio de un sencillo formato en un archivo de texto plano. En este archivo solo era necesario guardar los datos que se querían utilizar para entrenar la red neuronal, creando para cada conjunto de entrada y salida su correspondiente archivo.

En cuanto a la funcionalidad en este punto se realizo una reescritura del código, con el fin de utilizar un mejor manejo de los datos al organizar estos en estructuras que contuvieran la mayor parte de las variables utilizadas y se lograra disminuir la creación de las mismas. Así mismo se implementaron nuevos métodos para realizar el cálculo de los errores de la red neuronal, adaptándolos al número de salidas que tuviera el problema.

En este punto se estaba abarcando un espacio mayor del espectro del problema, sin embargo aun se notaban algunas deficiencias en cuanto al tiempo que se necesitaba para llegar a las soluciones del problema, el conjunto de resultados no otorgaba suficientes datos para que se pudiera decidir qué red neuronal se debería utilizar, además sólo se estaba configurando un parámetro por medio de la evolución dejando los demás estáticos.

5.3.3 Prototipo 3

En este punto todavía era mucho el trabajo que se debía realizar, lo primero que se resolvió fue el aspecto de la reducción de tiempos dado que eran quizá el factor más crítico en el problema. Para esto se implementó un método para que el algoritmo fuera "recordando" las configuraciones de red que ya había aparecido, de tal modo que si volvían a aparecer no se realizara el entrenamiento de nuevo sobre ellas. Esto redujo drásticamente los tiempos de convergencia del algoritmo.

Después de encontrada una solución al problema del tiempo era necesario considerar la modificación de más parámetros de la red neuronal, para esto fué diseñada una nueva representación genética para que incluyera la modificación de las funciones de activación dentro del proceso. La implementación de este nuevo procedimiento trajo consigo un nuevo aumento en los tiempos de ejecución debido a la consideración de nuevos espacios de búsqueda.

5.3.4 Prototipo 4

Al implementar los cambios a nivel funcional era necesario rediseñar la interfaz de usuario para que mostrara los nuevos parámetros que se estaban calculando. Este era el punto indicado también para incluir también los análisis que se deberían mostrar para que el usuario determinara el tipo de solución que más le conviniera.

En ese momento se incluyeron también los diferentes gráficos y soluciones que debería mostrar el algoritmo, así mismo se realizaron las validaciones sobre los campos de datos que se deben completar de tal modo que se redujera al mínimo la posibilidad de que se presentaran errores.

5.4 Entrada de datos

Una parte fundamental para la construcción de un buen modelo de red neuronal, corresponde a la selección y recolección de la información que lo alimentará en la fase de entrenamiento. La correcta identificación de las variables verdaderamente relevantes dentro del proceso modelado requiere de intensas labores de observación, análisis e investigación. Este trabajo previo requiere de gran conocimiento del tema, por lo tanto no puede ser asignado al programa si no que

debe ser realizado por parte del usuario. Con esto se garantiza que los datos que sean ingresados para el posterior proceso de entrenamiento logren adaptar una red neuronal que se ajuste mejor al problema.

Sin embargo la sola elección de los datos de entrenamiento, en la mayoría de los casos no es suficiente para garantizar una buena generalización de la red. Existen procedimientos que se pueden aplicar para mejorar el desempeño de la red, tanto previamente al entrenamiento como durante el proceso mismo de entrenamiento. En el proyecto se aplican estas técnicas para buscar una mejor explotación de los datos suministrados, en pos de lograr un mejor ajuste de los datos, evitando problemas como el sobre-ajuste o la pérdida de datos del conjunto de entrenamiento.

5.5 Pre-procesamiento de la información

Uno de los aspectos fundamentales para el éxito de cualquier modelo neuronal, es la calidad de los datos que se le presenten en la fase de entrenamiento; es por esto que se ha hecho necesario implementar técnicas para pre-procesar la información con el fin de aumentar la capacidad de aprendizaje de la red.

El pre-procesamiento de la información consiste de la preparación previa que se le aplican a los datos antes de ser empleados por la red, tal que puedan mostrar de forma clara y precisa la información relevante, y es este paso tan importante dentro del desarrollo de un modelo neuronal, que es posible que el mejor de los modelos fracase si se entrena con datos de baja calidad.

Existen diversos métodos y estrategias para lograr un buen pre procesamiento, entre ellas:

- Realizar una definición previa de las variables, (continuas y discretas).

- Hay que evitar el uso de valores continuos para representar conceptos simbólicos, por ejemplo la representación de los animales.

- No representar los meses del año del 1 al 12, categorizar de otra manera.

- No confundir las entradas continuas con las entradas discretas, estado civil es discreto (1 o 0), mientras que la temperatura es continua (0, 10, 15.5, 30,40.2…).

- No mezclar escalas (kilogramos con toneladas, metros con kilómetros, años con meses…).

- Evitar las variables con altas variaciones (máximos y mínimos)

- Tratar de estandarizar patrones para reducir el número de entradas

- Normalizar las variables para llevarlas a un rango entre 0 y 1.

- Entre mayor sea el número de entradas, más casos de entrenamiento requiere, lo que puede conducir a arquitecturas complejas, altos costos computacionales y difícil interpretación de resultados.

Es muy frecuente que no se tenga a disposición todo el conjunto de datos que se quiere someter a estudio, sino sólo un subconjunto de estos. Como primera medida, se debe asegurar que éste es representativo, y que no presente sesgos relevantes que puedan perjudicar el entrenamiento. Por ejemplo, si el problema es de clasificación, en el conjunto de datos deben estar representadas todas las categorías, si es de predicción, se deben incluir valores continuos de todo el rango.

El proceso de selección de los datos de relevancia, es asignado al usuario dado que solo es él quien conoce que valores son significativos para lograr una buena generalización. Se deben tener en cuenta para esta selección que los datos de entrenamiento sean los que más información proporcionen a la red neuronal.

5.6 Normalización

Una vez cargados los datos por parte del usuario, se realiza automáticamente la normalización de los datos para que el ajuste a las funciones de activación sea mayor, y haya mayor posibilidad de llegar a una mejor solución.

El método utilizado para la normalización de los datos fue tomar cada uno de los conjuntos de entrada y salida disponibles para posteriormente aplicar la fórmula escogida para el ajuste de los datos:

$$Valor\ normalizado = \frac{Valor - mínimo}{Máximo - mínimo} \qquad \textbf{Ec 5.1}$$

Donde,

$Valor$: Corresponde al dato que se va a normalizar.

$Mínimo$: Es el menor valor que se encuentra en el conjunto de datos.

$Máximo$: Es el mayor valor que se encuentra en el conjunto.

5.7 Inicialización de pesos

El método Levenberg – Marquardt minimiza una función utilizando de manera combinada el algoritmo de propagación hacia atrás y el método de la inversa de la matriz Hessiana. Se utiliza propagación hacia atrás cuando se está alejado del mínimo, y se cambia al segundo método a medida que la búsqueda se aproxima al mismo. Los métodos de entrenamiento mencionados utilizan el cálculo del gradiente para minimizar una función de error, luego, en ocasiones tienden a

quedar atrapados en mínimos locales de la función objetivo. Esta situación no es deseable, porque la red deja de aprender creyendo que ya llegó a su objetivo, esto genera grandes errores en la fase de validación.

Básicamente el mínimo alcanzado está sujeto al conjunto de pesos iniciales que se empleen en el entrenamiento. No existen métodos para seleccionar los pesos iniciales que garanticen encontrar el mínimo global, en general, dichos pesos se generan aleatoriamente, por lo que es conveniente realizar varias sesiones de entrenamiento sobre la misma red con diferentes pesos, en forma tal de comenzar el aprendizaje desde diferentes puntos de la función objetivo y así tener más posibilidad de alcanzar el mínimo global.

Algunos métodos para la inicialización de los pesos suponen el asignar una tendencia común a los datos para tener cierto control sobre los pesos utilizados. Generalmente se usa que los pesos se encuentren dentro de un valor de media o desviación estándar definida por el experto.

Para el caso de la aplicación se utiliza el procedimiento de inicialización de pesos aleatorios, directamente relacionado con el número de entrenamientos escogido para cada configuración.

5.8 El problema de la generalización

Hay que tener en cuenta en este punto que la RNA requiere alcanzar un grado de generalización, por lo tanto no debe aprender de memoria todos los casos, sino que teniendo algunos ejemplos de cada uno, sea capaz de determinar correctamente otro que no haya visto antes en base a los que le fueron mostrados durante su entrenamiento.

Entendiendo por generalización la capacidad de la red de almacenar las características que le son comunes a todos los patrones que fueron usados en la etapa de entrenamiento.

Si una red no tiene suficientes conexiones entre nodos, el algoritmo de entrenamiento puede no converger nunca; la red neuronal no es capaz de aproximar la función. Por el otro lado, en una red densamente conectada, puede ocurrir el sobre-ajuste (overfitting). El sobre-ajuste es un problema de los modelos estadísticos donde se presentan demasiados parámetros. Esto es una mala situación porque en lugar de aprender a aproximar la función presente en los datos, la red simplemente puede memorizar cada ejemplo de entrenamiento. El ruido en los datos de entrenamiento se aprende entonces como parte de la función, a menudo destruyendo la habilidad de la red para generalizar.

5.8.1 Validación Cruzada

Lo que se desea es que la red se entrene bien de forma tal que aprenda lo suficiente acerca del pasado para generalizar en el futuro. Desde esta perspectiva el proceso de aprendizaje debe elegir la parametrización de la red más acorde a este conjunto de ejemplos. Más específicamente, se puede ver al problema de selección de la red como una elección, dentro del conjunto de modelos de estructuras candidatas (parametrizaciones), de la mejor estructura de acuerdo a un cierto criterio.

Aplicando lo aconsejado por el método de validación cruzada (crossvalidation), el conjunto de datos que se escoja debe ser dividido en dos partes, una parte denominada conjunto de entrenamiento y otra llamada conjunto de validación, es común que el conjunto de validación sea aproximadamente el 10% del conjunto total. Como su nombre lo dice, el conjunto de entrenamiento es con el que la red

supervisada realiza el aprendizaje, y el conjunto de validación es con el que se simula la red para ver el grado de generalización, entre más bajo sea el error en este punto mejor ha generalizado la red.

La mejor red no es la que tenga el más bajo error durante el entrenamiento, por esto hay que tener en cuenta en qué punto el error de validación pueda empezar a crecer, separándose del error de entrenamiento, a partir de ese punto se produce un sobre ajuste y la red aprende de memoria.

5.8.2 Entrenamiento con Detención Temprana (Early Stopping)

Teniendo como meta a una buena generalización, es muy difícil darse cuenta cuándo es el mejor momento de detener el entrenamiento si solamente se está observando la curva de aprendizaje para el entrenamiento. En particular, como se mencionó anteriormente, es posible que la red termine sobre-ajustándose a los datos de entrenamiento si la sesión de entrenamiento no se detiene en el momento correcto.

Sin embargo, se puede identificar el comienzo del sobre-ajuste a través del uso de la validación cruzada, para lo cual los ejemplos de entrenamiento se separan en un subconjunto de estimación y un subconjunto de validación. El subconjunto de estimación se utiliza para entrenar a la red en el modo usual, excepto por una modificación menor: la sesión de entrenamiento se detiene periódicamente (cada tantas repeticiones), y se evalúa la red con el subconjunto de validación después de cada período de entrenamiento. Más específicamente, el proceso periódico de estimación, seguida de validación procede de la siguiente manera:

- Después del período de estimación (entrenamiento), se fijan todos los pesos y los umbrales del perceptrón multicapa, y la red opera en su modo hacia

delante. El error de validación se mide así para cada ejemplo en el conjunto de validación.

- Cuando la fase de validación se completa, la estimación (entrenamiento) se reanuda para otro período y el proceso se repite.

Este procedimiento se denomina método de entrenamiento con detención temprana y está disponible para realizar el entrenamiento de la red neuronal. Sin embargo si se tiene conocimiento del tiempo esperado de entrenamiento, o se desea usar un mayor volumen de datos para el entrenamiento, se puede especificar un número fijo de épocas durante el cual se desea realizar el entrenamiento y omitir el uso de la detención temprana.

5.9 Diagramas UML.

Del desarrollo de la metodología anterior surgen para el último prototipo los siguientes diagramas UML:

5.9.1 Diagramas de casos de uso:

CONFIGURAR TIPO ENTRENAMIENTO	
Descripción	Se configura la forma de búsqueda del error y si se desea especificar épocas de entrenamiento o detención temprana.
Conjeturas	El usuario tiene conocimientos básicos sobre redes neuronales y algoritmos genéticos.
Actor inicial	Usuario del sistema
Condiciones Previas	Ninguna.

Pasos	• Escoger el tipo de error para el entrenamiento, entre relativo y absoluto. • Escoger para el entrenamiento de la RNA entre especificar épocas o detención temprana.
Condiciones Resultantes	Se a configurado parte del sistema para realizar la búsqueda de topologías de redes.
Actor Beneficiado	Usuario del sistema.

Tabla 1. Casos de uso: Configurar tipo entrenamiento

CARGAR DATOS	
Descripción	En este punto se carga a el programa los datos para realizar el entrenamiento, verificación y prueba (este ultimo opcional).
Conjeturas	Se tiene un archivo que contiene los datos muéstrales de entradas y salidas del experimento sobre el cual se desea encontrar una RNA.
Actor inicial	Usuario del sistema.
Condiciones Previas	Configurar tipo de entrenamiento.
Pasos	1. Cargar datos de entrenamiento. 2. Cargar datos de validación. 3. Cargar datos de prueba.
Condiciones Resultantes	Se han cargado los datos para realizar el entrenamiento de la red neuronal y validación del sistema.
Actor Beneficiado	Usuario del sistema.

Tabla 2. Casos de uso: Cargar datos.

CONFIGURAR BÚSQUEDA AG(Algoritmo genético)	
Descripción	Se configura las probabilidades de los operadores sobre los algoritmos genéticos, tamaño población, máximo de capas ocultas y neuronas por capa, Numero de entrenamientos, algoritmo de entrenamiento, generaciones y tolerancia al error.
Conjeturas	Se tiene conocimientos sobre teoría de algoritmos genéticos.
Actor inicial	Usuario del sistema.
Condiciones Previas	• Configuración tipo de entrenamiento. • Carga de datos empíricos para entrenamiento y validez.
Pasos	1. Se escoge el tamaño de la población del AG. 2. Se ajusta el número máximo de capas ocultas y neuronas por capa. 3. Se digitaliza el número de entrenamientos. 4. Se da las probabilidades de mutación y cruce. 5. Se selecciona el tipo de algoritmo de entrenamiento. 6. se agrega las generaciones y tolerancia al error.
Condiciones Resultantes	El sistema queda configurado para realizar la búsqueda de soluciones.
Actor Beneficiado	Usuario del sistema.

Tabla 3. Casos de uso: Configurar búsqueda AG.

Buscar RNA (Redes neuronales Artificiales) óptimas por medio del AG (Algoritmo genético)	
Descripción	Se realiza la búsqueda de un espacio de soluciones para el problema ajustado a los valores de datos de entrenamiento y validación.
Conjeturas	Se han cargado los datos y configurado correctamente el sistema.
Actor inicial	Usuario del sistema.
Condiciones Previas	• Configuración tipo de entrenamiento. • Carga de datos empíricos para entrenamiento y validez. • Configuración de la búsqueda por AG.
Pasos	1. Escoger población inicial. 2. Realizar cruces y mutaciones. 3. Validar poblaciones nuevas. 4. Volver al punto 2 hasta alcanzar el objetivo.
Condiciones Resultantes	El sistema ha encontrado unas configuraciones de redes, que intentar dar solución al problema planteado.
Actor Beneficiado	Usuario del sistema.

Tabla 4. Casos de uso: Buscar redes neuronales artificiales óptimas por medio del algoritmo genético

MOSTRAR AYUDAS GRÁFICAS Y ESTADÍSTICAS PARA EL ANÁLISIS DE LAS CONFIGURACIONES	
Descripción	Se muestran los datos obtenidos de la búsqueda de las redes neuronales de manera ordenada y analítica, de forma que sea sencillo el análisis de los resultados.
Conjeturas	Se tiene conocimientos de rendimiento de la teoría de redes neuronales artificiales.
Actor inicial	Usuario del sistema.
Condiciones Previas	• Configuración tipo de entrenamiento. • Carga de datos empíricos para entrenamiento y validez. • Configuración de la búsqueda por AG. • Búsqueda de las RNA por medio de AG.
Pasos	1. Escoger el tipo de información grafica o estadística que se desea analizar.
Condiciones Resultantes	El sistema ha encontrado unas configuraciones de redes, que intentar dar solución al problema planteado y se muestran resultados estadísticos generales para escoger la mejor solución.
Actor Beneficiado	Usuario del sistema.

Tabla 5. Casos de uso: Mostrar ayudas gráficas y estadísticas para el análisis de las configuraciones.

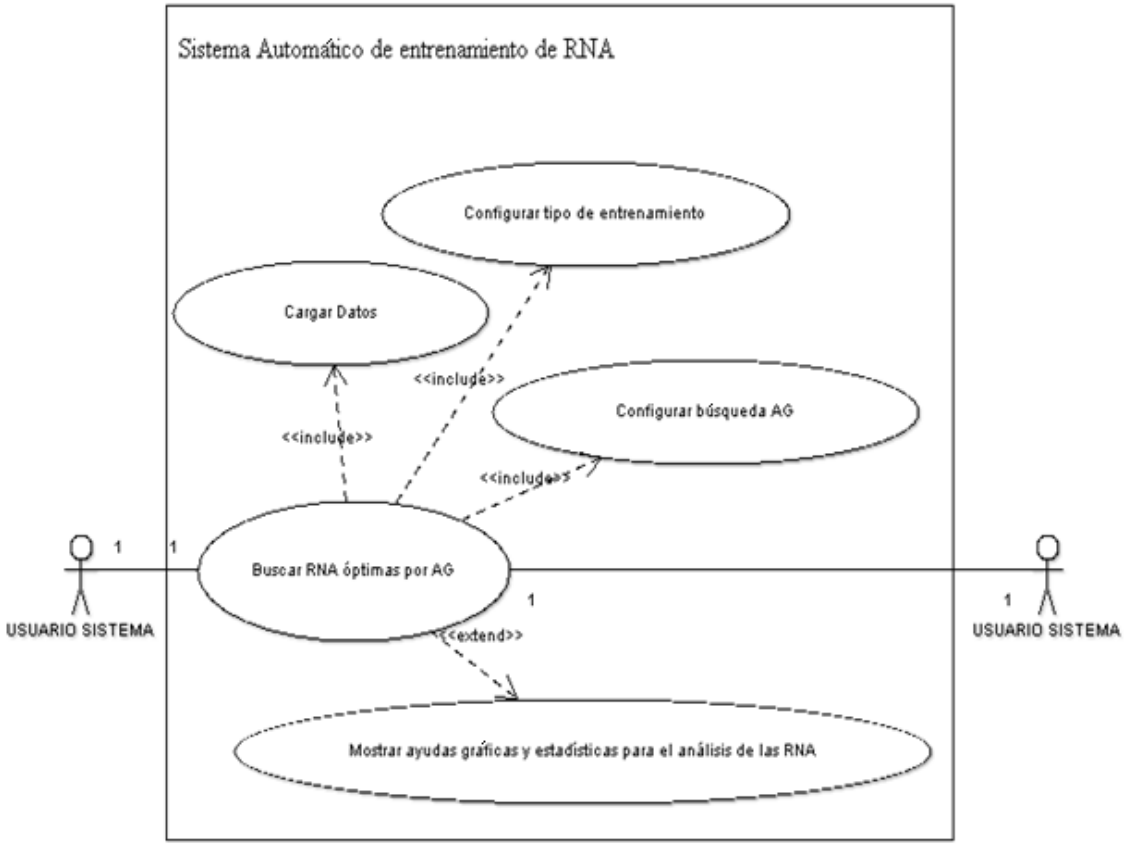

Figura 11. Diagrama de casos de uso.

5.9.2 Diagrama de actividades:

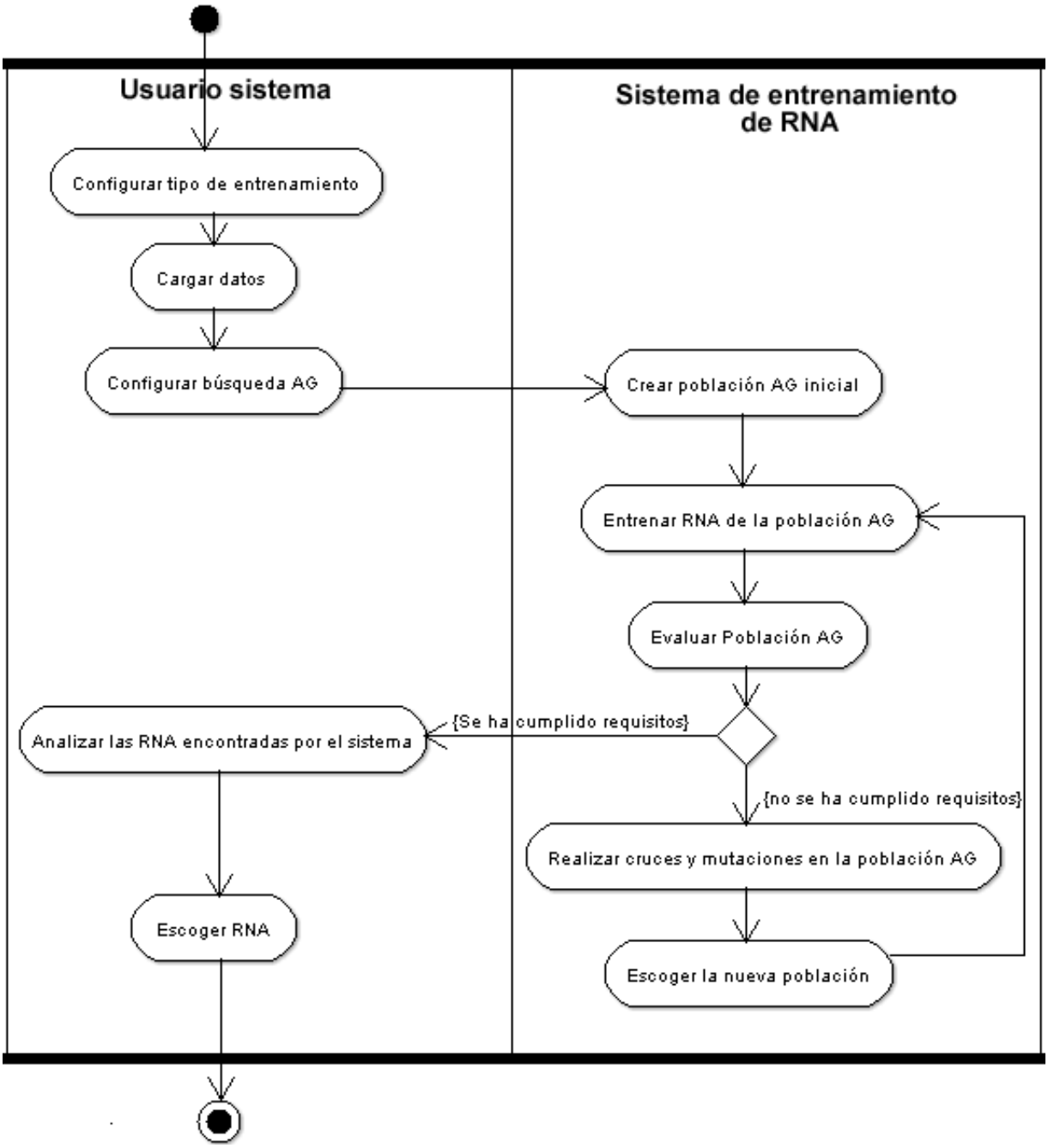

Figura 12. Diagrama de actividades.

6. RESULTADOS Y ANÁLISIS.

Para la realización de las distintas comparaciones y validaciones del comportamiento del algoritmo, se realizaron tres pruebas diferentes de las cuales se tuviera conocimiento suficiente sobre su solución para realizar comparaciones. Estas comparaciones incluyen el error promedio absoluto o el error promedio relativo según sea el caso escogido para guiar la evolución del algoritmo genético. El error promedio absoluto, (o error de la medición promedio) es la diferencia entre el valor real y el valor simulado de un determinado número de mediciones, y está dado por:

$$\varepsilon_a = \frac{\sum_{i=1}^{n} |S_i - Ss_i|}{n} \qquad \text{Ec 6.1}$$

S_i= Salida real.

Ss_i= Salida simulada.

n= Número de datos simulados.

El error relativo porcentual es un índice de la precisión de la medida, es el cociente entre el error absoluto y el valor promedio, y está dado por:

$$\varepsilon_r = \frac{\sum_{i=1}^{n} \frac{|S_i - Ss_i|}{S_i}}{n} * 100\% \qquad \text{Ec 6.2}$$

Donde:

S_i= Salida real.

Ss_i= Salida simulada.

n= Número de datos simulados.

Para determinar el ajuste de la red neuronal también se usa la tasa de ajuste obtenida durante la evaluación con los datos de prueba. La tasa de ajuste es un indicador que muestra el porcentaje de semejanza entre los datos reales y los valores simulados ya sea por la red neuronal artificial hallada o por las redes usadas para las comparaciones, está basado en el error relativo promedio obtenido durante la etapa de predicción y está dado por:

$$Ta = 100 - \frac{\sum_{i=1}^{n} \frac{|S_i - Ss_i|}{S_i}}{n} * 100\% \qquad \textbf{Ec 6.3}$$

Donde:

S_i= Salida real.

Ss_i= Salida simulada.

n= Número de datos simulados.

Para el ordenamiento de las mejores soluciones obtenidas se usan dos criterios que representan el desempeño de cada configuración, al ser simulada con los datos de prueba. El primero de estos indicadores es el número de errores cometidos por la red neuronal, este tipo de error está basado en la tolerancia establecida para la salida. Si el error absoluto de cada simulación dado por la ecuación 6.1, es mayor que la tolerancia establecida, se cuenta como un error en

salida; de una manera más formal está definido por la siguiente ecuación:

$$Si\ error_{simulado} > tolerancia \rightarrow error \qquad \textbf{Ec 6.4}$$

Donde,

$error_{simulado}$= Es el error absoluto de la simulación (ecuación 6.1).

$tolerancia$= Es la máxima diferencia permitida entre la salida simulada y la real.

El segundo criterio para el ordenamiento de las mejores soluciones es el peor cometido, este valor corresponde al máximo error producido por la red neuronal al ser simulada con los datos de prueba. Esta medida está representada por la ecuación 6.5.

$$Peor\ error = \text{máx}\ (S_i) \qquad \textbf{Ec 6.5}$$

Siendo S_i todos los errores cometidos por la salida de la red al ser simulada con los datos de prueba y máx una función para hallar el máximo de estos valores.

Adicionalmente se encuentran una serie de gráficas que ilustran el comportamiento del problema con la continua modificación de los parámetros, de tal forma que se puedan observar algunas tendencias o características que puedan servir como base de conocimiento a la hora de escoger la topología de red neuronal artificial que más le convenga al usuario. Para la realización de todas las pruebas, se utilizó un equipo dedicado solamente a esta tarea (excepto por las simulaciones de la sección 6.3.2), las características hardware incluían, procesador Intel Centrino dúo a 1.7 GHz y 1Gb de memoria Ram.

6.1 Prueba 1.

La siguiente prueba se realizó sobre un problema de redes neuronales artificiales, que se encontró en el proyecto de pregrado para optar por el título de ingeniero de sistemas de la Universidad Industrial de Santander, en el año 2007, llamado "Herramienta software para la identificación de sistemas dinámicos no lineales basado en modelos de redes neuronales artificiales"[51], en el cual se trata de demostrar que esta técnica de inteligencia artificial se puede aplicar a problemas de tipo dinámicos no lineales. Con el fin de demostrarlo se estudia el comportamiento de los émbolos en un motor de combustión interna.

En la investigación original antes mencionada, las mejores redes encontradas fueron las siguientes:

Nombre Red neuronal	Configuración Capas ocultas		Función de activación capa	Error de la red
	Capas	Neuronas por Capa		
Red 1a	Capa oculta 1	4	Sigmoidal	2,47%
	Salida	4	Tangente-hiperbólica	
Red 2a	Capa oculta 1	160	Gaussiana	2,45%
	salida	4	Sigmoidal	

Tabla 6. Mejores configuraciones del problema original.

Las redes fueron entrenadas con técnicas diferentes. La configuración de red número 1a se entrenó con la técnica de propagación hacia atrás

[51] **U. CHINCHILLA, G. PÉREZ.** Herramienta software para la identificación de sistemas dinámicos no lineales, Universidad Industrial de Santander, Facultad Ingenierías Físico Mecánicas, 2007.

(*Backpropagation*), mientras la red dos se entrenó como una red de base radial.

Para realizar la búsqueda, los parámetros utilizados para la búsqueda son los mostrados en la tabla a continuación y fueron establecidos de acuerdo a la complejidad del problema.

Para este ejemplo, la medida encargada de guiar la evolución del algoritmo genético fue el error promedio absoluto. Se estableció una tolerancia al error de simulación de 0.01, lo que representa aproximadamente un 1% en el error relativo; medida que se usaba para demostrar el desempeño de las configuraciones en la aplicación original.

Parámetros	Simulación
Número máximo de capas	3
Número máximo de neuronas	10
Tolerancia al error	0.01
Generaciones	20
Algoritmo de entrenamiento	Trainlm
Número de entrenamientos	60
Tamaño de la población	20
Probabilidad de cruce	0.8
Probabilidad de mutación	0.1

Tabla 7. Parámetros utilizados en la simulación.

Colocando a prueba el software desarrollado en la presente investigación y tomando los datos que se suministran en anterior proyecto de grado sobre sistemas dinámicos no lineales, se produjeron las redes mostradas en la siguiente tabla. Para esta simulación el tiempo de búsqueda fue de una hora con 12 minutos, se probaron 1140 redes, de las cuales 280 fueron redes diferentes, y las mejores configuraciones de redes en la última generación, fueron las siguientes:

Nombre red neuronal	Configuración Capas ocultas		Función de activación capa
	Capas	Neuronas por Capa	
Red 1b	Capa oculta 1	4	Purelin
	Capa oculta 2	8	Logsig
	Salida	4	Logsig
Red 2b	Capa oculta 1	4	Logsig
	salida	4	Purelin
Red 3b	Capa oculta 1	3	Logsig
	Capa oculta 2	10	Purelin
	Salida	4	Logsig
Red 4b	Capa oculta 1	7	Tansig
	Capa oculta 2	2	Radbas
	Salida	4	Radbas
Red 5b	Capa oculta 1	9	Radbas
	Capa oculta 2	6	Logsig
	Salida	4	Tansig

Tabla 8. Mejores configuraciones encontradas.

Un análisis más detallado del desempeño de cada configuración es la indicada en la siguiente tabla, estos datos también son obtenidos de los resultados mostrados por el sistema automático de configuración.

Nombre Red Neuronal	Número de errores[52]	Peor error[53]	Error promedio[54]	Tasa de ajuste[55] (%)
Red 1b	1	0.99361	5.1815	80
Red 2b	3	0.78159	4.3599	80
Red 3b	3	1	10.1931	60
Red 4b	3	1	13.5327	40
Red 5b	4	0.99084	13.8224	60

Tabla 9. Desempeño de las mejores configuraciones obtenidas.

Se muestran adicionalmente, las gráficas para las tres mejores soluciones al problema de identificación de sistemas dinámicos no lineales, los cuales son de gran utilidad para la toma de decisiones acerca de la red más conveniente. Para este ejemplo, al tenerse a disposición un número reducido de datos, solo se usaron cinco (5) para el conjunto de prueba, los cuales nunca hicieron parte del entrenamiento de la red.

[52] Calculado con la ecuación 6.4
[53] Calculado con la ecuación 6.5
[54] Calculado con la ecuación 6.1
[55] Calculado con la ecuación 6.3

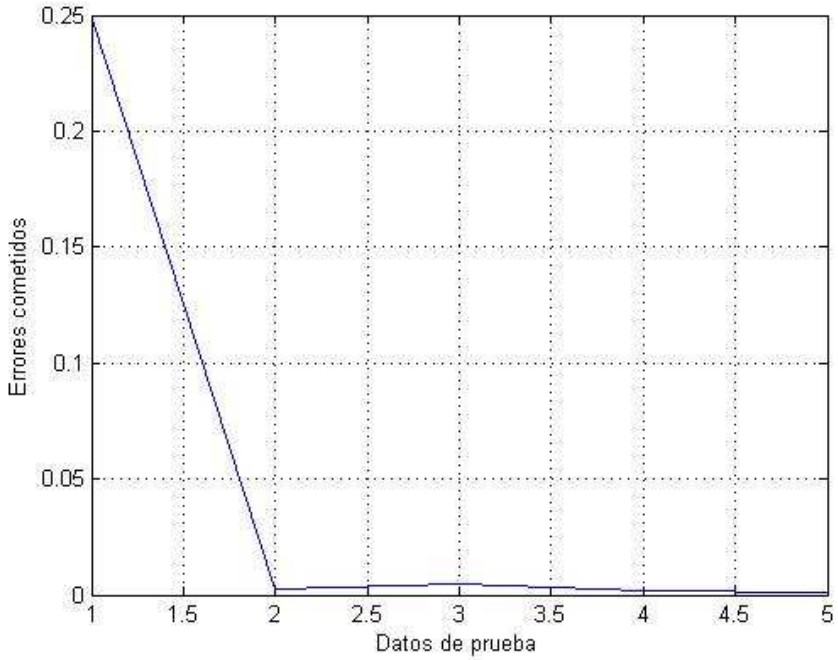

Figura 13. Errores cometidos[56] versus datos prueba en la Red 1b[57].

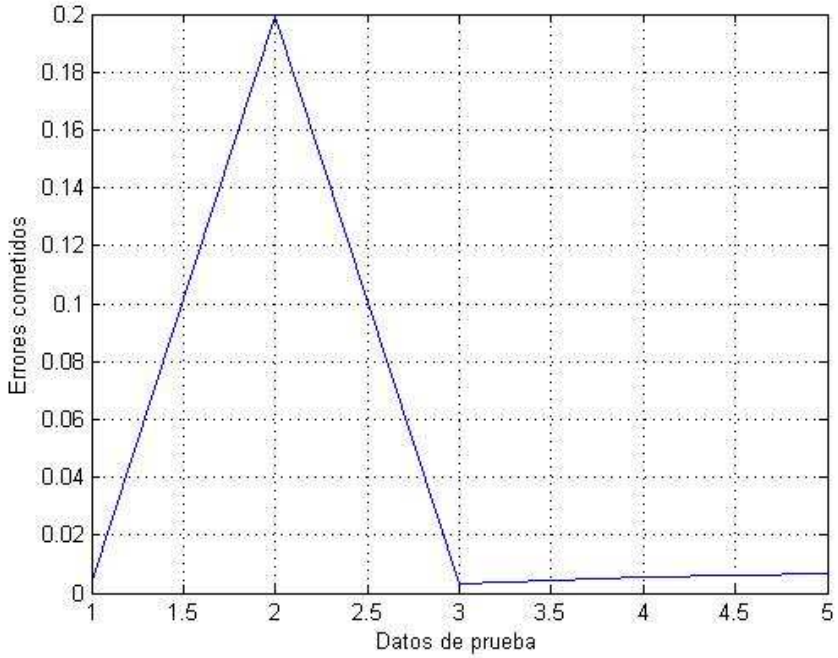

Figura 14. Errores cometidos versus datos prueba en la Red 2b[58].

[56] Calculado con la ecuación 6.1.
[57] Configuración mostrada en la tabla 9.

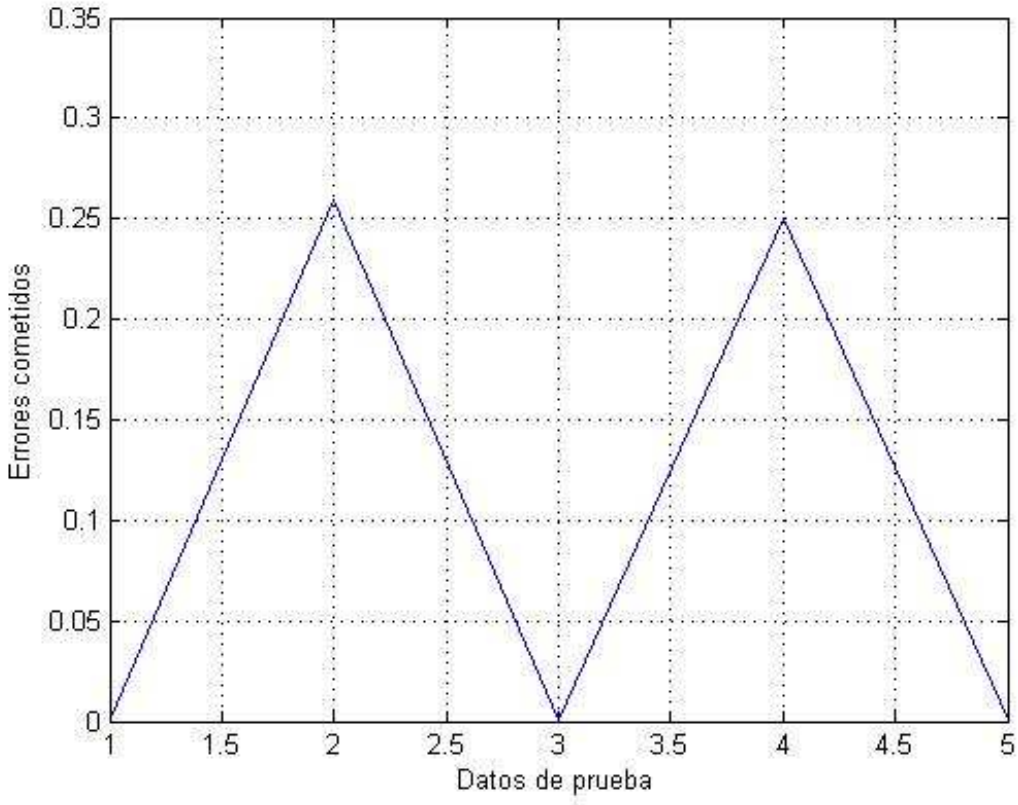

Figura 15. Errores cometidos[59] versus datos prueba en la Red 3b[60].

Las gráficas generales sobre el procedimiento del software para llegar a estas redes finales, sirven de guía para observar la variación del comportamiento del algoritmo genético al agregar o quitar neuronas o capas ocultas para la resolución del problema. Estas gráficas son mostradas a continuación:

[58] Configuración mostrada en la tabla 9.
[59] Calculado con ecuación 6.1.
[60] Configuración mostrada en la tabla 9.

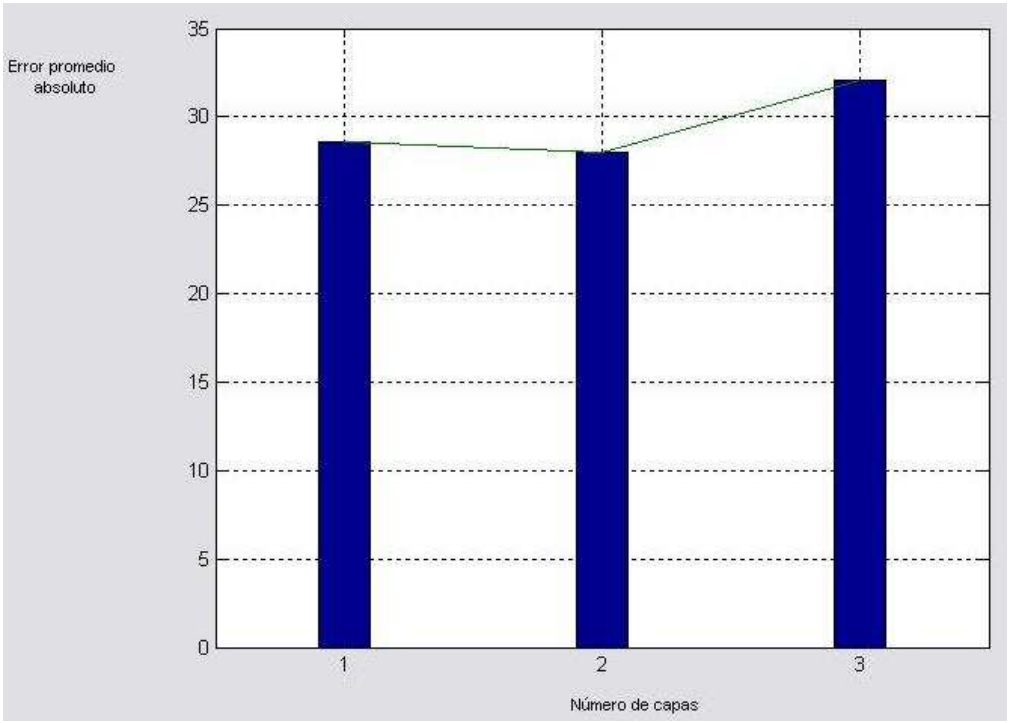

Figura 16. Error promedio[61] absoluto versus número de capas.

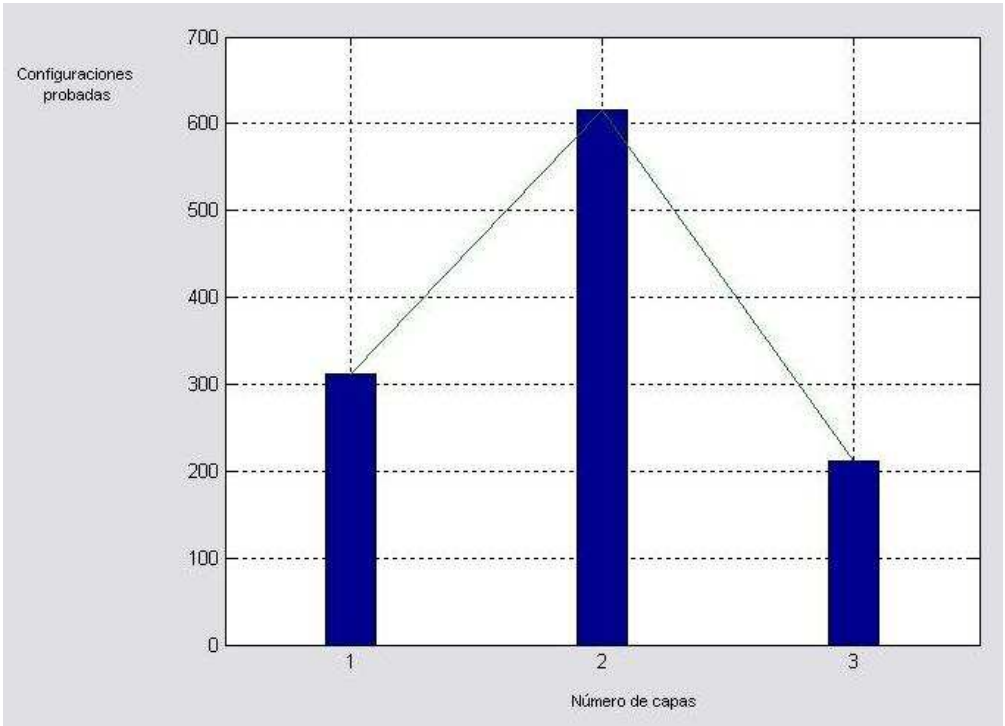

Figura 17. Configuraciones probadas versus número de capas.

[61] Calculado con la ecuación 6.1.

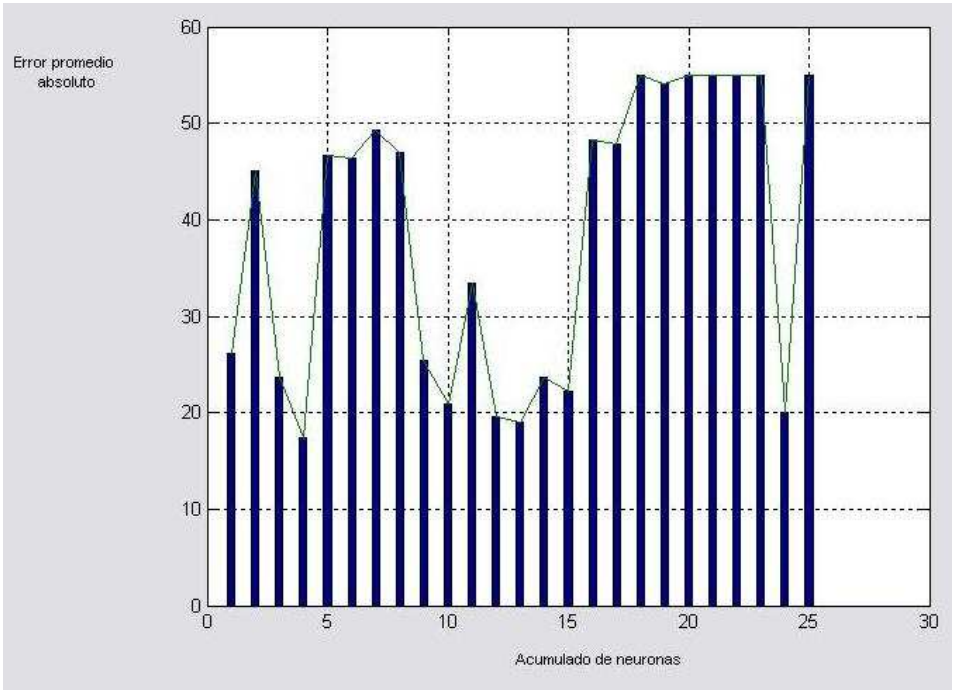

Figura 18. Error Promedio absoluto[62] versus acumulado de neuronas.

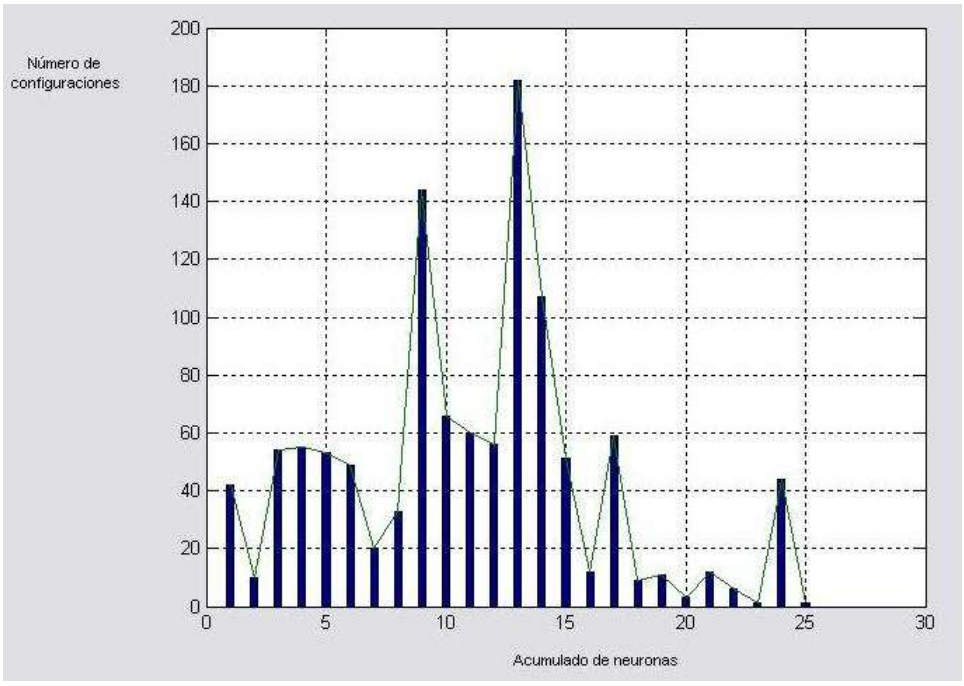

Figura 19. Número de configuraciones versus acumulado de neuronas.

[62] Calculado con la ecuación 6.1.

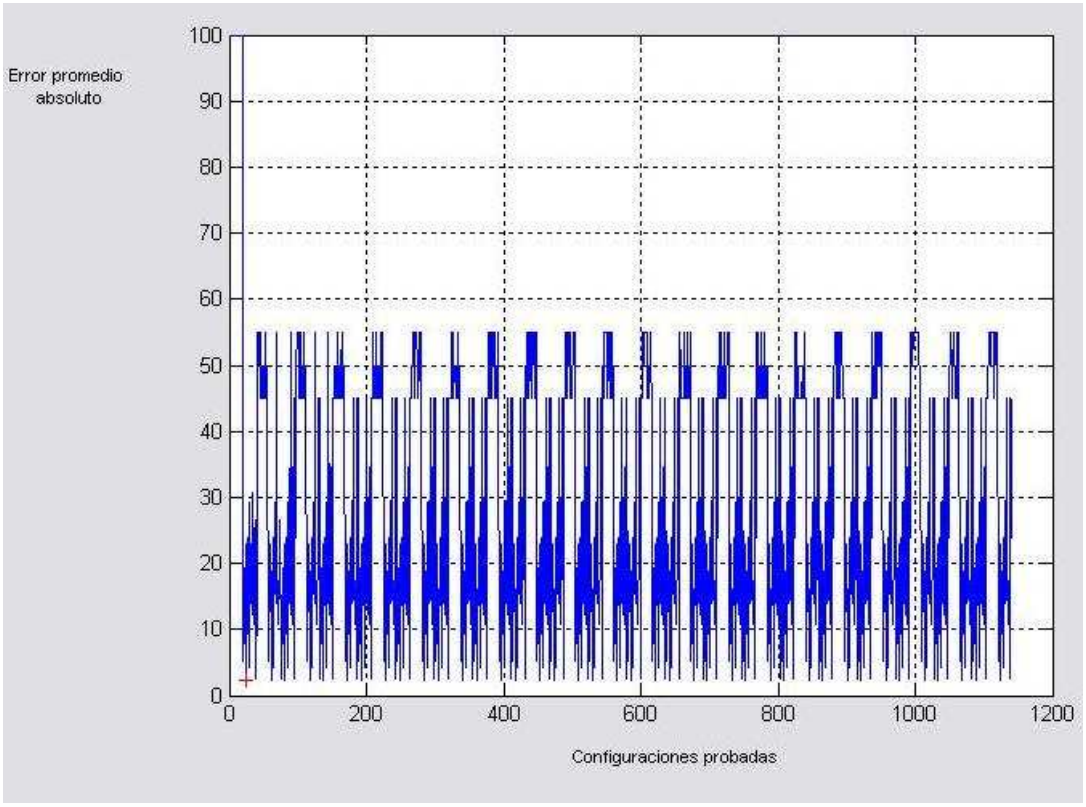

Figura 20. Error promedio absoluto[63] versus configuraciones probadas.

En el proyecto de pregrado donde se realizó la prueba original, no se muestra la ecuación utilizada para calcular los errores cometidos por la red. Pero de igual manera se encontró que, dentro de las soluciones[64] encontradas por el software WÖTAN Genetics[65], se encuentra la red 2b, que se muestra como una variación a la red escogida en el proyecto de sistemas no lineales; utilizando el mismo número de capas y neuronas ocultas pero con la diferente función de activación en la capa de salida. Respecto a la red alterna (Red 2a), no se pueden realizar comparaciones, puesto que en el presente proyecto no se implementó arquitectura de red neuronal de base radial.

Las soluciones encontradas por el software se asemejan a las encontradas en el

[63] Calculado con la ecuación 6.1.
[64] Solución mostrada en la tabla número 8.
[65] Este software es el desarrollado en el proyecto actual.

proyecto de prueba, por la aproximación que tiene estas redes para diagnosticar el comportamiento de los émbolos en un motor de combustión interna. Si bien, en el proyecto anterior se probaron técnicas diferentes de entrenamiento y configuración, no se consideró la variación de las funciones de activación las cuales son de importancia en el entrenamiento. Al variar dichas funciones, se obtuvieron topologías similares, pero con una mayor tasa de aptitud.

Las gráficas que genera el software ayudan a tomar mejores decisiones respecto a que red se acopla al problema, permiten observar la evolución de las redes y el ajuste por medio de agregar neuronas y capas ocultas, así como el cambio de las funciones de activación.

6.2 Prueba 2.

Los datos de esta prueba fueron tomados de la tesis de grado: "Determinación del patrón de flujo multifásico en tuberías de recolección de petróleo emulsionado, a partir de los datos históricos de producción apoyado en una aplicación con redes neuronales artificiales"[66] presentada para optar por el título de ingeniero de sistemas de la Universidad Industrial de Santander en el año 2007.

El objetivo principal de este proyecto era el de determinar el patrón de flujo multifásico promedio presente en una tubería a partir de los datos históricos de producción, con el fin de calcular la caída de presión, esto se realizaba mediante una herramienta basada en redes neuronales artificiales. Al igual que todos los problemas de este tipo, el principal inconveniente radicaba en la dificultad de decidir que configuración de red neuronal utilizar con el fin de solucionar el problema en cuestión.

[66] **J. FLÓREZ**, **F. PORRAS**, Determinación del patrón de flujo multifásico en tuberías de recolección de petróleo emulsionado, a partir de los datos históricos de producción apoyado en una aplicación con redes neuronales artificiales, Universidad Industrial de Santander, Facultad Ingenierías Físico Mecánicas, 2007.

Dentro de la tesis se encontraban definidos claramente los datos de entrenamiento y validación usados en el entrenamiento de la red neuronal, lo que hizo posible que las pruebas con el sistema configurador pudieran ser realizados con el mismo conjunto de datos que en la aplicación original. Después de realizar los correspondientes análisis encontraron que la configuración que mejor resolvía el problema era utilizando 4 (cuatro) neuronas en una sola capa oculta, la cual arrojaba un error de simulacion de 4.93%.

Este problema utiliza tres variables de entrada (3) llamadas velocidad superficial del aceite, velocidad superficial del gas y velocidad superficial del agua. A partir de estos datos se debe clasificar la muestra en cuatro (4) diferentes grupos (salidas) que representan el patrón de flujo llamados segregado, transición, intermitente y distribuido.

Para la verificación de la configuración obtenida en la tesis se decidió ajustar un espacio de búsqueda amplio considerando el grado de dificulad, y el volumen de datos del que se tenía a disposción, de tal forma que se explotara la posibilidad de nuevas topologías de red neuronal para dar solución al problema. Para esta simulación los parámetros utilizados fueron:

Número máximo de capas: 3
Número máximo de neuronas: 10
Tolerancia al error: 0.01
Generaciones: 25
Algoritmo de entrenamiento: Trainlm
Número de entrenamientos: 45
Tamaño de la población: 20
Probabilidad de cruce: 0.8
Probabilidad de mutación: 0.1

El espacio de búsqueda se definió de tres capas, así se tuviera conocimiento de la facilidad de resolución del problema; con el fin de explorar otras posibilidades que pudieran conducir a mejorías en el desempeño en el reconocimiento del patrón. Esto fue apoyado con el ya comprobado potencial de entrenamiento del algoritmo de Levenberg-Maquardt (Trainlm), y el número de repeticiones del entrenamiento que garantizaba la explotación de cada solución factible contemplada. Las mejores dos configuraciones obtenidas en esta prueba son las mostradas a continuación.

Las configuraciones obtenidas se muestran entre llaves cuadradas donde un cada elemento representa una capa oculta y el número representa el número de neuronas utilizadas en cada una de estas.

Configuración	Número de errores[67]	Peor error[68]	Error promedio[69]	Tasa de ajuste[70] (%)	Funciones de activación capas ocultas	Funciones de activación capas de salida
[9 7 5][71]	0	0.009450	0.30441	100	Radbas Purelin Tansig	Purelin
[9]	2	0.63237	0.54109	96.296	Radbas	Logsig

Tabla 10. Mejores configuraciones halladas en la simulación.

Para esta simulación se logró una configuración con tres (3) capas ocultas con lo cual se lograba una total adaptación a los datos de prueba que fueron utilizados en

[67] Calculado con la ecuación 6.4.
[68] Calculado con la ecuación 6.5.
[69] Calculado con la ecuación 6.1.
[70] Calculado con la ecuación 6.3.
[71] Para más detalles en la representación de la red, consulte sección 9.5 del anexo.

la evaluación, los cuales no fueron conocidos por la red neuronal en la fase de entrenamiento. Además también se obtuvo una topología que resolvía el problema usando una (1) sola capa oculta, que aunque produjo algunas fallas en la salida de acuerdo a la tolerancia de error establecida, también logró un excelente ajuste en la solución del problema. En la siguiente tabla se muestra una comparativa entre los errores obtenidos por cada una de las configuraciones y el tiempo de simulación al utilizar los 247 datos que se tienen a disposición.

Configuración	Error de simulación	Tiempo de simulación
[4] [72]	4.93%.	-
[9 7 5]	0 %	0.012168
[9]	3.704%	0.007095

Tabla 11. Comparación entre las configuraciones halladas y configuración original

Para esta simulación el número total de configuraciones diferentes probadas fue de 383, el número total de redes fue de 1420, y el tiempo total empleado en la búsqueda fue de 2.1 horas. Los gráficos de los errores producidos en la simulación con los datos de prueba son los mostrados a continuación:

[72] Configuración original utilizada en la tesis de donde fueron obtenidos los datos para la prueba. Para mas detalles en la representación de la red, consulte sección 9.5 del anexo.

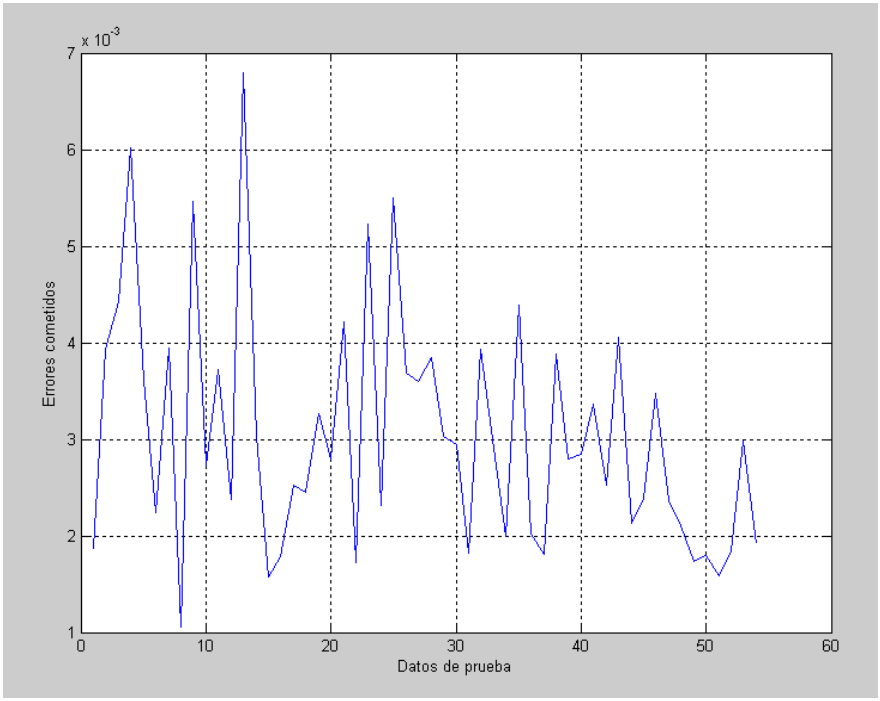

Figura 21. Errores de simulación[73] de la configuración [9 7 5].

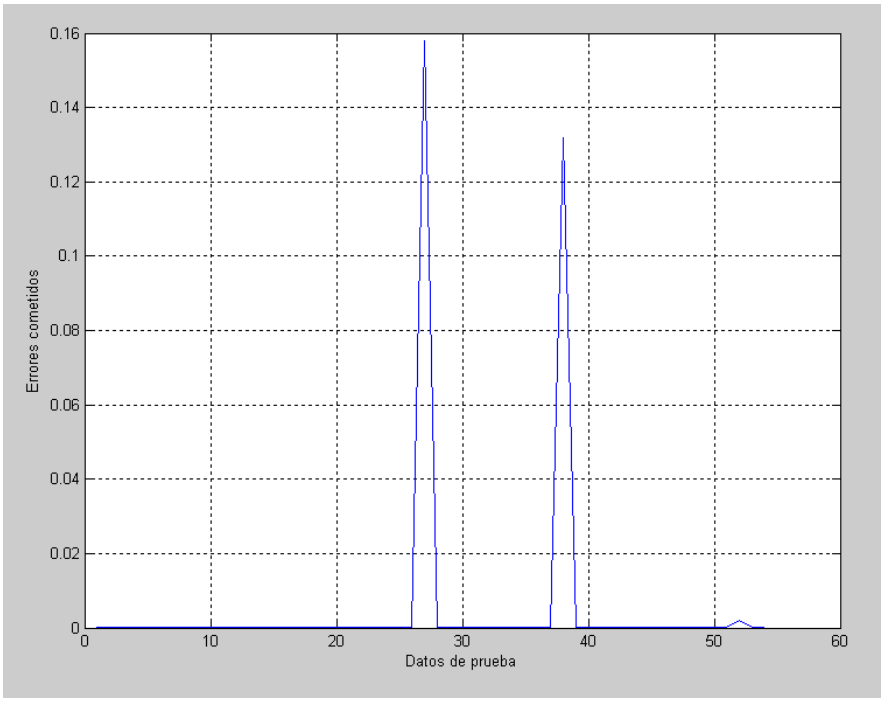

Figura 22. Errores de simulación de la configuración [9]

[73] Calculado con la ecuación 6.1.

De forma adicional también se encuentran diferentes gráficos que apoyan la labor de búsqueda del algoritmo genético, en las cuales se muestran los errores producidos al variar el número de capas y de neuronas utilizados en la resolución del problema. En estas se puede observar como los niveles de error promedio generados al evaluar las redes estuvieron muy cercanos, sin embargo fueron menos las configuraciones probadas con dos (2) capas ocultas dada la convergencia del algoritmo genético hacia las soluciones con una (1) y tres (3) capas ocultas que producían mejores resultados.

Otra gráfica de interés es la de la variación de los errores a través de las generaciones del algoritmo genético, donde además se indica la posición donde apareció la mejor solución.

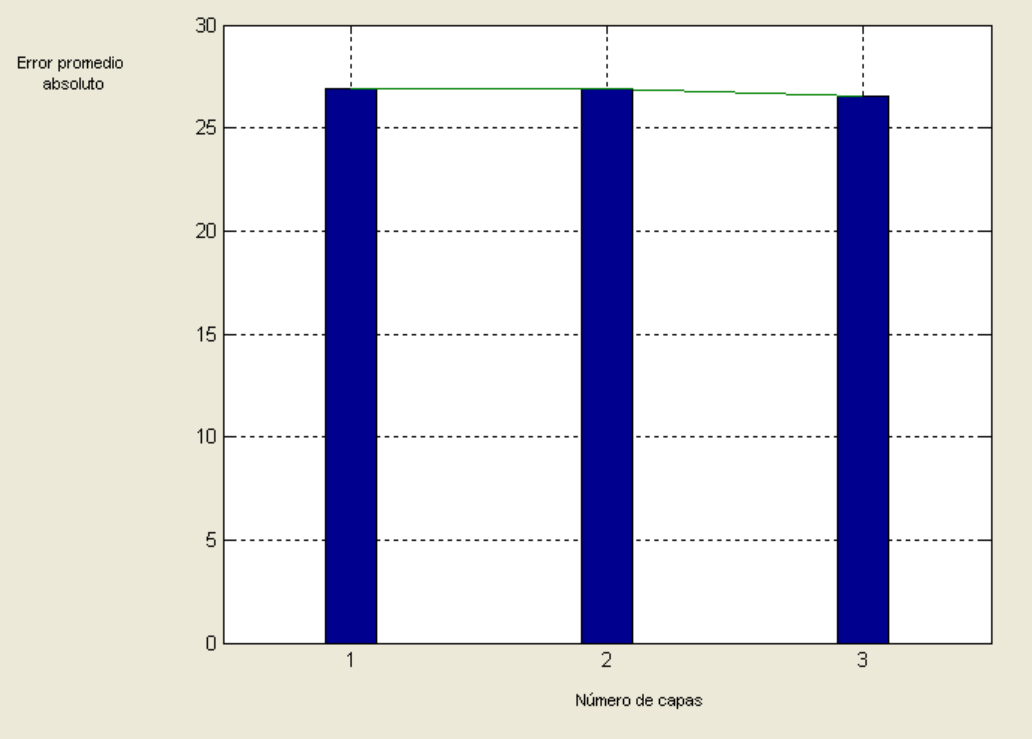

Figura 23. Errores[74] vs número de capas ocultas.

[74] Calculado con la ecuación 6.1.

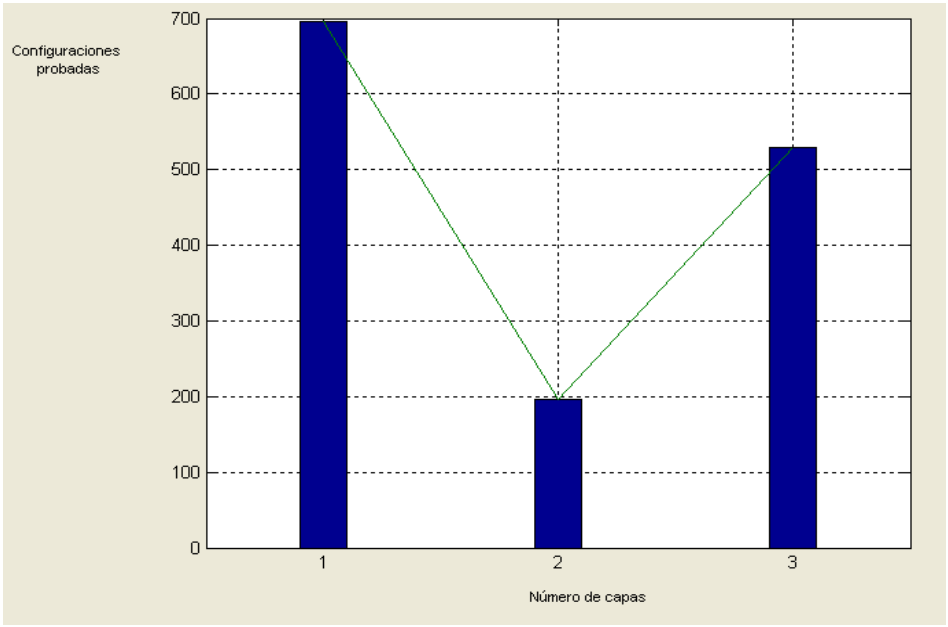

Figura 24. Redes probadas vs número de capas

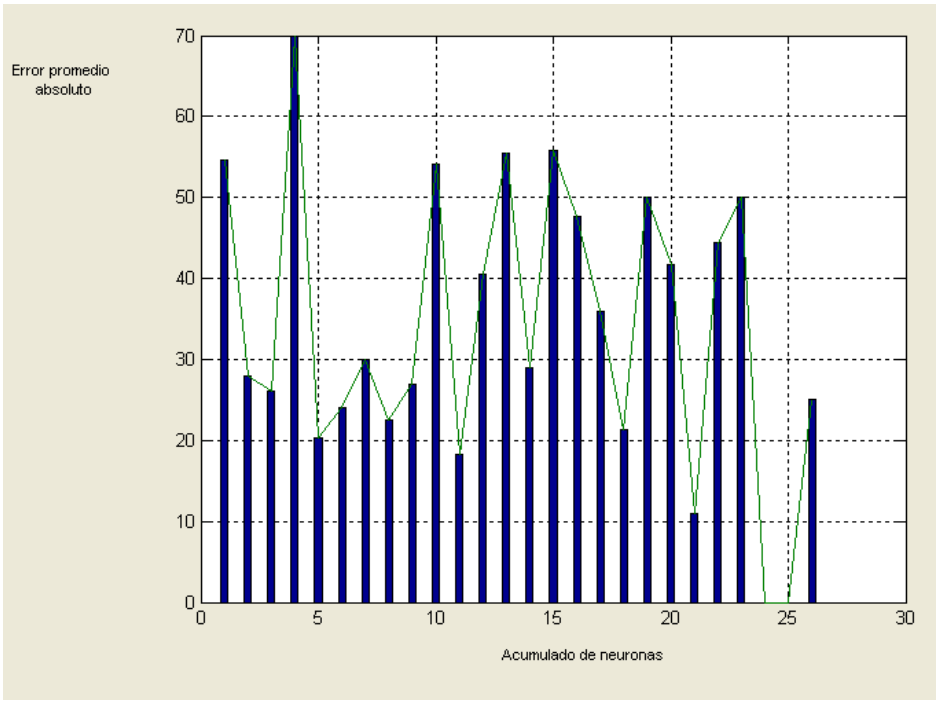

Figura 25. Errores[75] vs número de neuronas por red.

[75] Calculado con la ecuación 6.1.

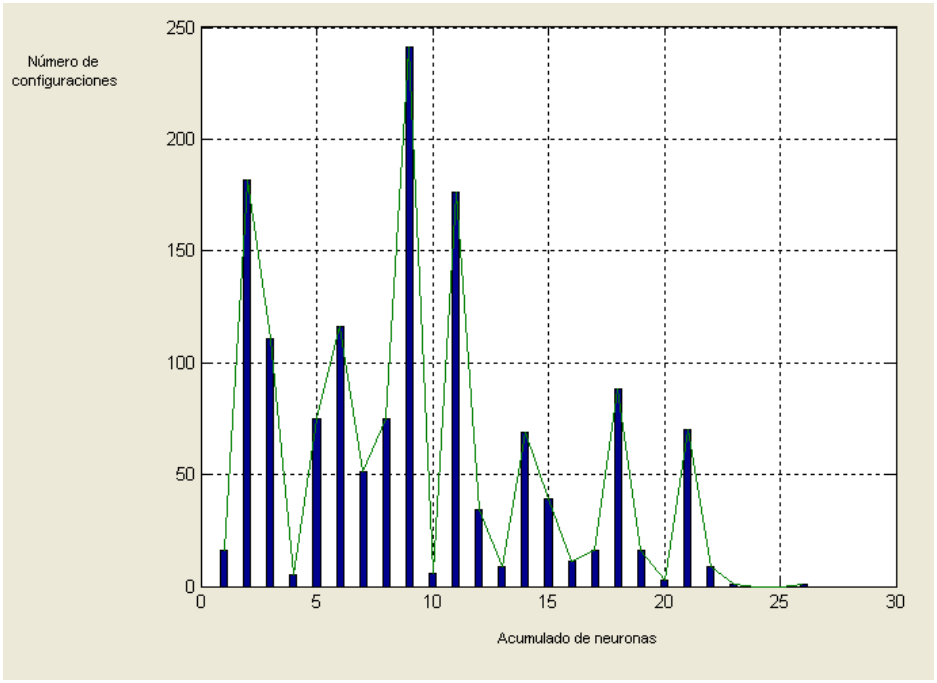

Figura 26. Redes probadas vs número de neuronas por configuración.

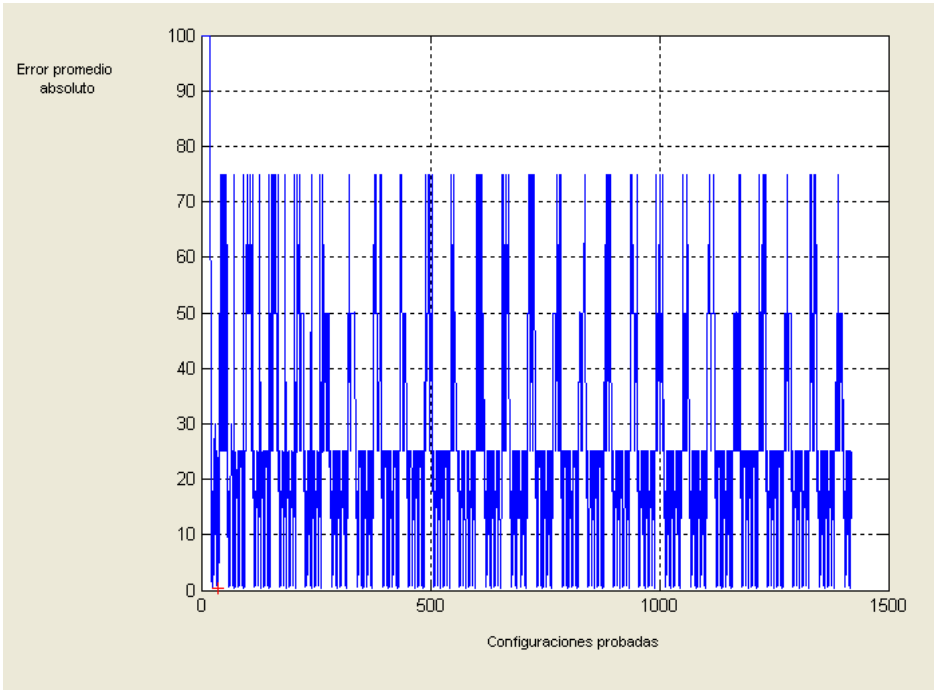

Figura 27. Variación del error[76] en el algortimo genético.

[76] Calculado con la Ecuación 6.1.

En conclusión para esta prueba se logró una reducción el nivel de error obtenido a costo de aumentar el número de capas y neuronas utilizado, lo cual no causa un gran aumento en el tiempo necesario para simular una solución. La mejor solución obtenida por el software WÖTAN Genetics logró reducir el error de simulación a un cero por ciento con la tolerancia establecida, pero se tuvo que aumentar el número de capas utilizadas a tres. La segunda mejor solución obtuvo un error de 3.704%, pero el número de neuronas se aumentó a nueve (9), cuando en la solución original se usaban cuatro (4), ajustando un error de 4.93%. Debido al tipo de problema y al acertado espacio de búsqueda, que permitió una rápida convergencia del algoritmo, el tiempo requerido para llegar a una solución fue de 2.1 horas; con buenos resultados.

6.3 Prueba 3.

Esta prueba fue desarrollada con los datos de un sistema para el reconocimiento de rostros. La configuración de una red neuronal que aprenda todos los casos de entrenamiento es de gran importancia para el óptimo funcionamiento del sistema, en este problema cada una de las imágenes de la base de datos se somete a un procesamiento de tal manera que los datos puedan ser comprendidos por la red neuronal.

Después de aplicados los filtros a las imágenes que se tienen en la base de datos, cada una de estas queda convertida en una matriz de 49 filas por 200 columnas las cuales representan los datos de entrada para el entrenamiento de la red neuronal. De igual manera los datos de salida se deben procesar aplicando el mismo filtro, mediante el cual se obtiene un vector de 1 fila por 200 columnas. Cada uno de estos los elementos que conforman la columna son los utilizados para conformar los diferentes ejemplos de entrenamiento y los datos de prueba de la red neuronal artificial.

Los datos de entrenamiento para este problema fueron obtenidos de una aplicación realizada en el grupo de investigación en ingeniería biomédica GIIB, los cuales servirían para realizar una nueva validación al comportamiento del sistema de configuración.

Adicionalmente se quiso agregar otro tipo de prueba del comportamiento del sistema configurador por medio de la prueba por parte de algunos usuarios. Las personas elegidas para realizar las pruebas al programa fueron alumnos de últimos semestres de ingeniería de sistemas que estaban cursando la materia de inteligencia artificial, y que por tanto tuvieran conocimientos sobre el tema que se estaba tratando.

Para estas pruebas a los voluntarios se les entregó el programa configurador y el conjunto de datos utilizados para realizar el entrenamiento, los parámetros de la búsqueda debían ser escogidos por ellos mismos para buscar diferentes comportamientos en la resolución del problema. Sin embargo se estableció que se usara el error promedio absoluto[77] para guiar la evolución, con una tolerancia en la salida de 0.1, el mismo que era usado en la aplicación original.

Los resultados obtenidos por los estudiantes, en los cuales se utilizan diferentes técnicas para llegar a una solución son los mostrados a continuación. En cada uno de estos se observan las cinco mejores configuraciones obtenidas así como las características de cada una de estas. Adicionalmente se muestra el comportamiento gráfico de la simulación de la mejor red obtenida por ellos, evaluada con los datos de prueba, los cuales no hicieron parte del entrenamiento de la red. Cada una de estas pruebas se realizó sobre diversas máquinas, con distintas características, lo que hizo que se produjeran diversos comportamientos en los tiempos de simulación.

[77] Mostrado en la ecuación 6.1

Las configuraciones obtenidas se muestran entre llaves cuadradas donde un cada elemento representa una capa oculta y el número representa el número de capas utilizadas en cada una de estas.

6.3.1 Parámetros utilizados en las búsquedas:

Para la realización de estas pruebas se usaron los siguientes parámetros en el algoritmo genético.

Parámetros	Simulación 1	Simulación 2	Simulación 3	Simulación 4	Simulación 5
Número máximo de capas	3	1	2	2	2
Número máximo de neuronas	20	30	30	30	30
Tolerancia al error	0.1	0.1	0.1	0.1	0.1
Generaciones	20	20	20	20	25
Algoritmo de entrenamiento	Traingda	Traingdx	Trainrp	Trainlm	Trainbfg
Número de entrenamientos	30	35	60	50	40
Tamaño de la población	20	20	20	20	20
Probabilidad de cruce	0.8	0.8	0.8	0.8	0.8
Probabilidad de mutación	0.1	0.1	0.1	0.1	0.1

Tabla 12. Parámetros utilizados en las simulaciones.

6.3.2 Configuraciones obtenidas.

6.3.2.1 Simulación 1.

En este ejemplo se usó detención temprana para el entrenamiento de las redes. Los resultados para esta búsqueda son los mostrados en la tabla. El número de redes diferentes probadas fue 482, el número total de redes probadas fue 1140 y el tiempo empleado fue de 60 minutos.

Configuración	Número de errores[78]	Peor error[79]	Error promedio[80]	Tasa de ajuste[81] (%)	Funciones de activación capas ocultas	Funciones de activación capas de salida
[9 10][82]	18	0.972	19.6172	64	Purelin Purelin	Radbas
[11]	20	0.71508	12.8386	60	Radbas	Tansig
[11]	23	0.67659	12.9421	54	Tansig	Tansig
[11 13]	23	0.99	21.1854	54	Radbas Radbas	Radbas
[11 13]	23	0.99	21.1854	54	Tansig Radbas	Radbas

Tabla 13. Mejores configuraciones halladas en la simulación 1.

Al observar los resultados obtenidos, se nota que al utilizar estos parámetros de búsqueda no se logra tener un buen nivel de acierto , esto se justifica

[78] Calculado con la ecuación 6.4.
[79] Calculado con la ecuación 6.5.
[80] Calculado con la ecuación 6.1.
[81] Calculado con la ecuación 6.3.
[82] Para más información sobre la representación, consulte el anexo en la sección 9.5.

principalmente por lo amplio del espacio de búsqueda (3 capas) y el reducido número de generaciones y entrenamientos utilizado. Esto causó que el espacio de búsqueda no haya sido explorado en profundidad y que las configuraciones que pudieran resolver el problema no tuvieran la posibilidad de demostrar todo su potencial.

Algunos de los comportamientos de la búsqueda de las soluciones se pueden observar en las gráficas generadas por el programa, lo cual brinda información sobre el comportamiento del problema y la posibilidad de corregir los parámetros para futuras búsquedas, o simplemente para en base a esta información realizar una configuración manual.

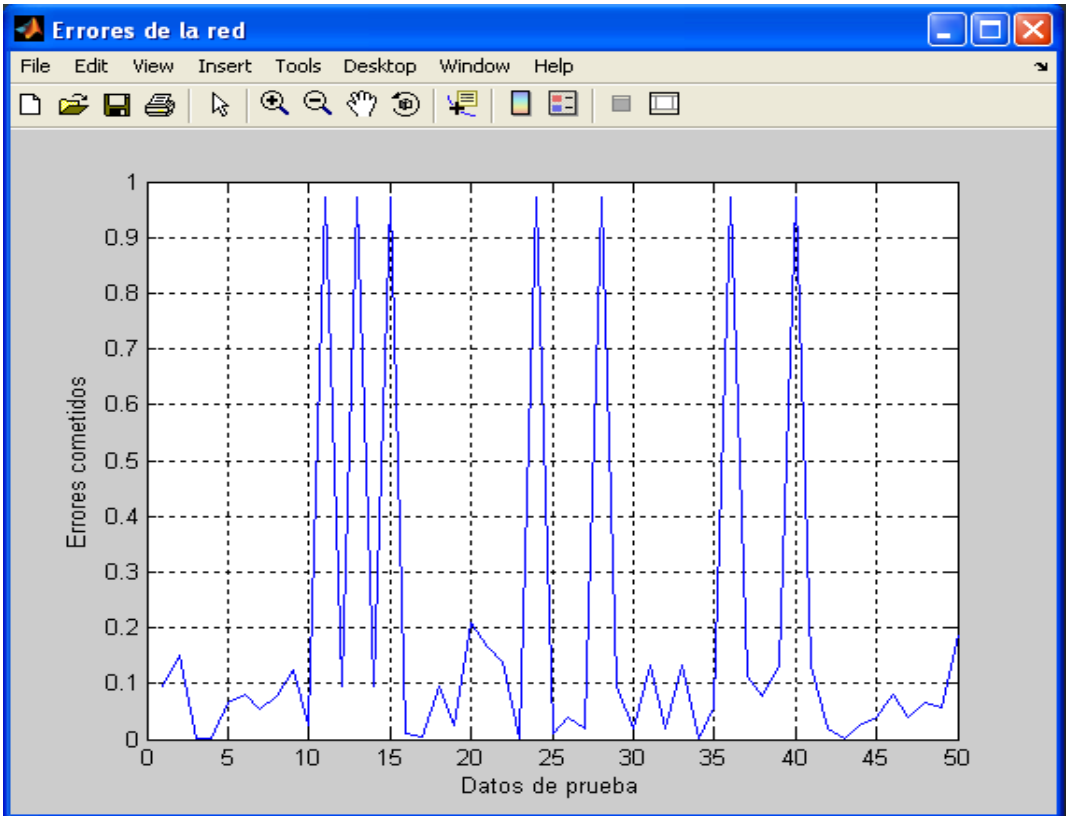

Figura 28. Errores de simulación[83] de la mejor configuración obtenida [9 10]

[83] Calculado con la ecuación 6.1.

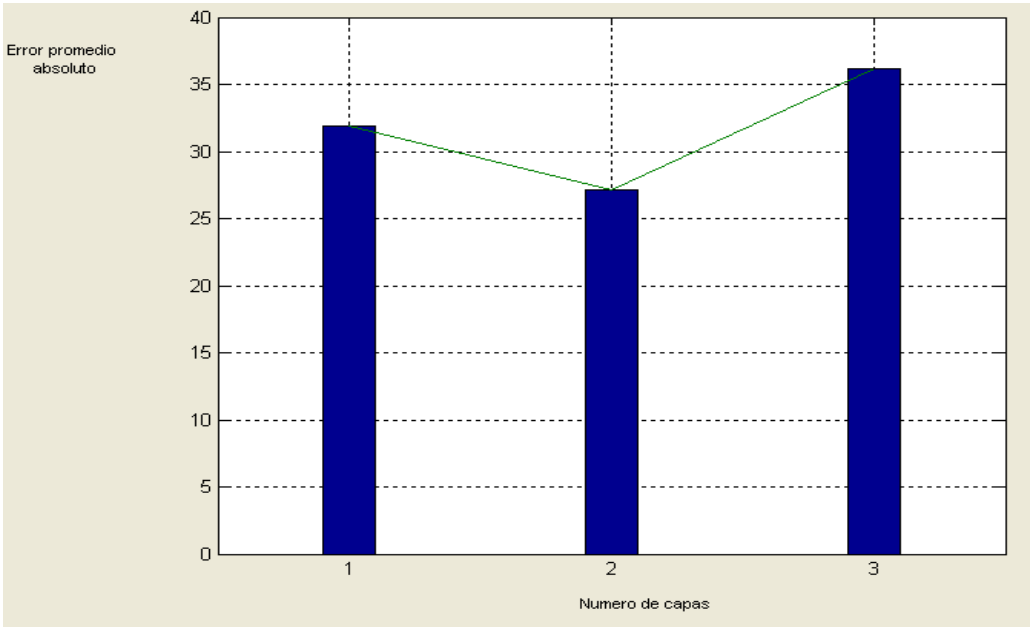

Figura 29. Errores[84] vs número de capas ocultas.

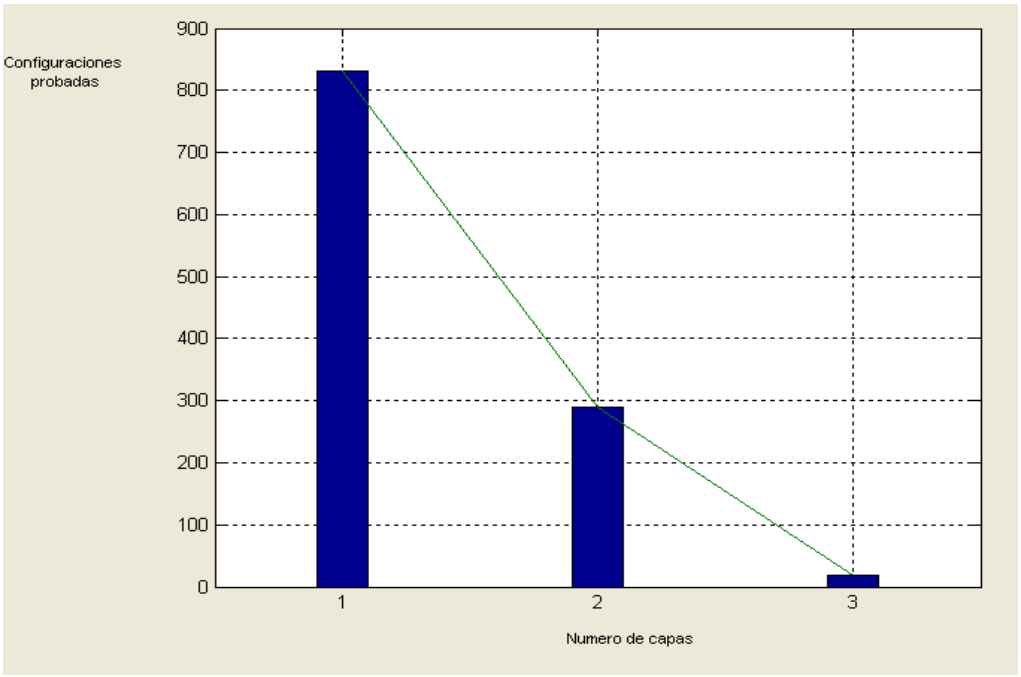

Figura 30. Redes probadas vs número de capas

[84] Calculado con la ecuación 6.1.

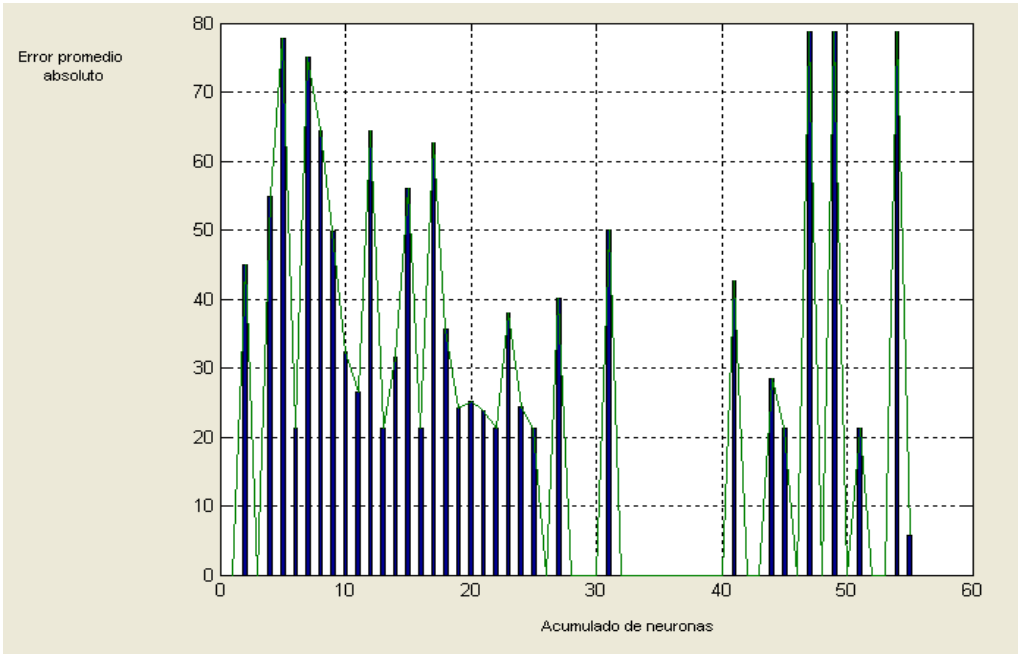

Figura 31. Errores[85] vs número de neuronas por red

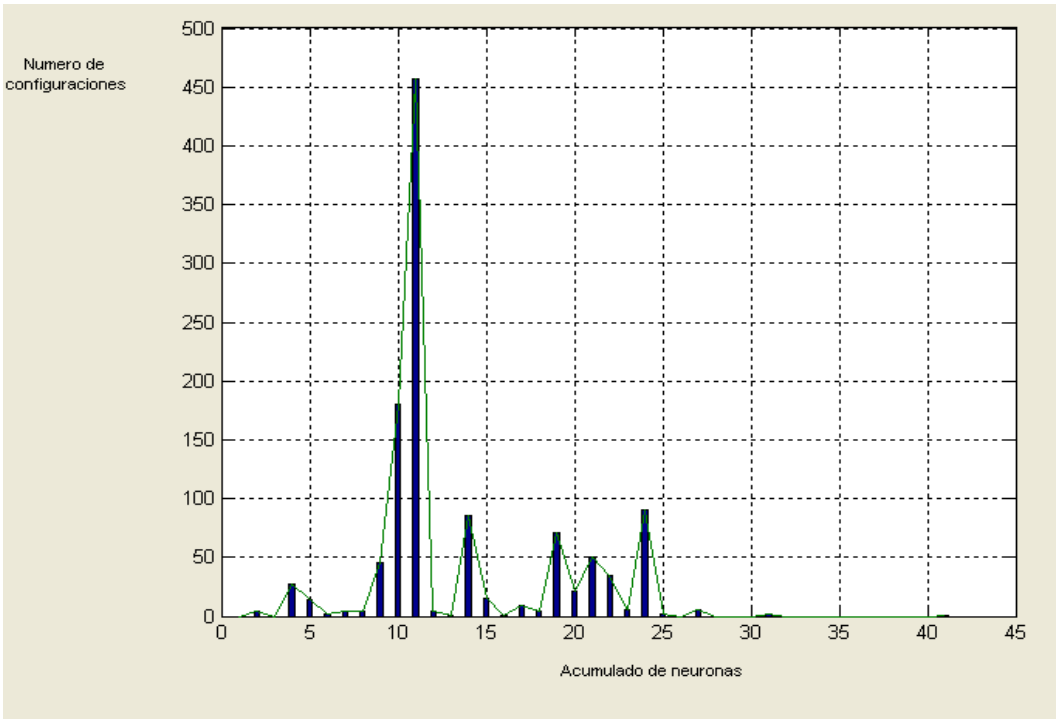

Figura 32. Redes probadas vs número de neuronas por configuración

[85] Calculado con la ecuación 6.1.

6.3.2.2 Simulación 2.

Parámetros de búsqueda:

Configuración	Número de errores	Peor error	Error promedio	Tasa de ajuste (%)	Funciones de activación capas ocultas	Funciones de activación capas de salida
[29][86]	2	0.13937	4.6945	96	Tansig	Radbas
[24]	2	0.13965	3.4954	96	Radbas	Radbas
[27]	2	0.1914	5.1887	96	Tansig	Radbas
[12]	4	0.36679	4.5884	92	Radbas	Radbas
[23]	4	0.81592	5.6848	92	Logsig	Radbas

Tabla 14. Mejores configuraciones halladas en la simulación 2.

Para este caso el entrenamiento se realizó especificando el número de épocas durante el cual se realizaría el entrenamiento y se omitió el conjunto de datos de prueba para agregarlo al conjunto de entrenamiento. El número de épocas elegido para el entrenamiento de la red fue de 1000. Debido a la gran cantidad de entrenamientos que realizo el algoritmo, el tiempo empleado para llegar a la solución fue muy alto, aún usando un método de entrenamiento de rápida convergencia. Sin embargo, aunque se haya tardado mucho en obtener una solución, las configuraciones obtenidas para este caso son más que aceptables al ser evaluadas con los datos de prueba. En esta simulación el número de redes diferentes probadas fue de 33, el número total de redes probadas fue 132 y el tiempo total empleado fue de 18 horas.

[86] Para más información respecto a la representación, consulte el anexo en la sección 9.5.

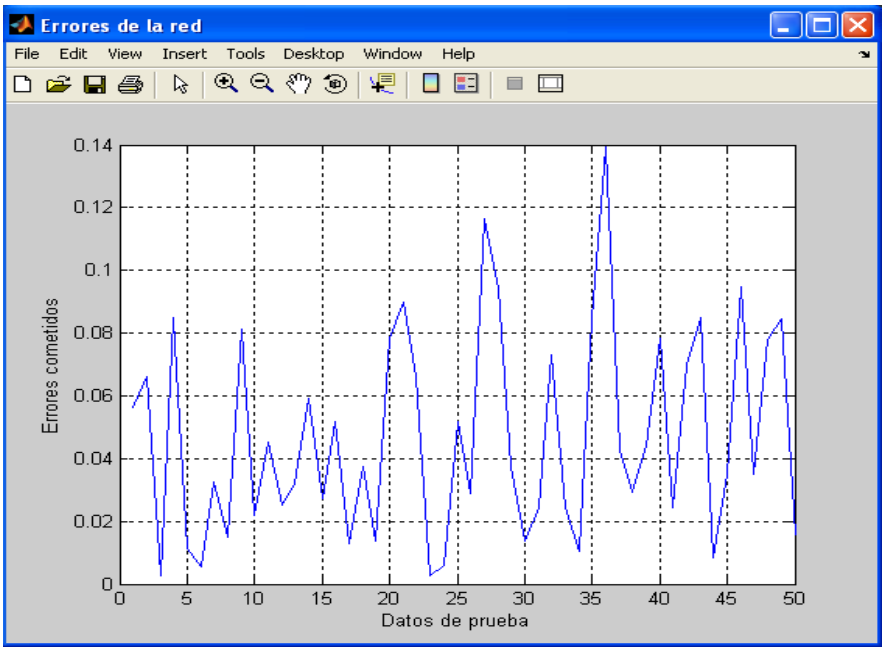

Figura 33. Errores[87] de simulación de la mejor configuración obtenida [29].

6.3.2.3 Simulación 3.

Parámetros de búsqueda:

Configuración	Número de errores[88]	Peor error[89]	Error promedio[90]	Tasa de ajuste[91] (%)	Funciones de activación capas ocultas	Funciones de activación capas de salida
[30 21][92]	0	0.078334	2.0031	100	Tansig Radbas	Radbas

[87] Calculado con la ecuación 6.1.
[88] Calculado con la ecuación 6.4.
[89] Calculado con la ecuación 6.5.
[90] Calculado con la ecuación 6.1.
[91] Calculado con la ecuación 6.3.
[92] Para mayor información sobre la configuración, consulte el anexo, sección 9.5.

[29 1]	4	0.31821	4.7532	92	Radbas Radbas	Radbas
[17 10]	4	0.6685	6.1312	92	Tansig Tansig	Tansig
[8 17]	9	0.89638	16.8921	82	Purelin Radbas	Tansig
[5]	10	0.98672	9.098	80	Radbas	Logsig

Tabla 15. Mejores configuraciones halladas en la simulación 3.

En esta prueba se puede observar como la mejor solución obtenida tiene un grado de ajuste ideal al ser simulada con los datos de prueba, esto fue posible al realizar un entrenamiento de la red neuronal durante un número significativo de repeticiones, además con el mecanismo de detención temprana utilizado y el algoritmo de entrenamiento de rápida convergencia se logró un tiempo de solución muy bajo.

En esta simulación el número de redes diferentes probadas fue de 257, el número total de redes probadas fue de 1140 y el tiempo total empleado fue de 2.3 horas.

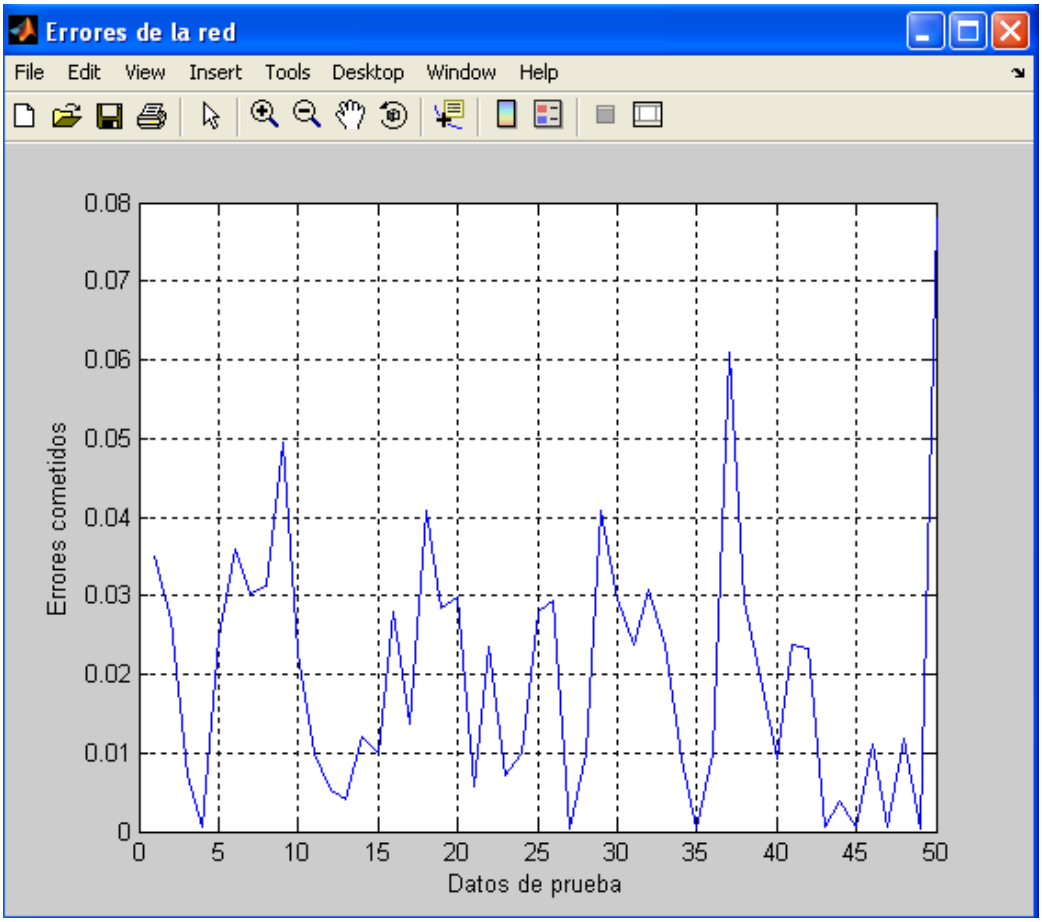

Figura 34. Errores[93] de simulación de la mejor configuración obtenida [30 21].

6.3.2.4 Simulación 4.

Parámetros de búsqueda:

Configuración	Número de errores	Peor error	Error promedio	Tasa de ajuste (%)	Funciones de activación capas ocultas	Funciones de activación capas de salida
[6 1]	0	0.023656	0.36326	100	Logsig	Tansig

[93] Calculado con la ecuación 6.1.

					Purelin	
[24 11]	0	0.028893	1.229	100	Radbas Radbas	Radbas
[4 15]	0	0.032383	0.76508	100	Tansig Tansig	Radbas
[14 6]	0	0.034888	2.0689	100	Radbas Radbas	Purelin
[2]	0	0.035028	0.98123	100	Radbas	Logsig

Tabla 16. Mejores configuraciones halladas en la simulación 4.

En esta prueba se puede observar como fueron mejorados los resultados obtenidos, tanto en el error producido en la simulación de los datos, como en el número de neuronas utilizadas para dar solución al problema. La calidad de los resultados se debe al número de entrenamientos utilizado y al algoritmo, el cual si bien se tarda un poco más en realizar el entrenamiento, los resultados generalmente son muy buenos. El número de redes diferentes probadas fue de 161, el número total de redes probadas fue de 1140 y el tiempo total empleado fue de 12 horas. Un aspecto que influyó en el mayor tiempo de simulación fueron las bajas prestaciones de memoria Ram y procesamiento de la máquina donde fue realizada la prueba (Procesador a 800 MHz con 256 Mb de RAM), limitadas con respecto a la carga que sufre el equipo al realizar todos los entrenamientos.

En este caso como los errores están por debajo de la tolerancia escogida, la configuración elegida para realizar las comparaciones es la que utiliza una sola capa oculta con dos neuronas, puesto que tiene menor complejidad que las demás soluciones obtenidas.

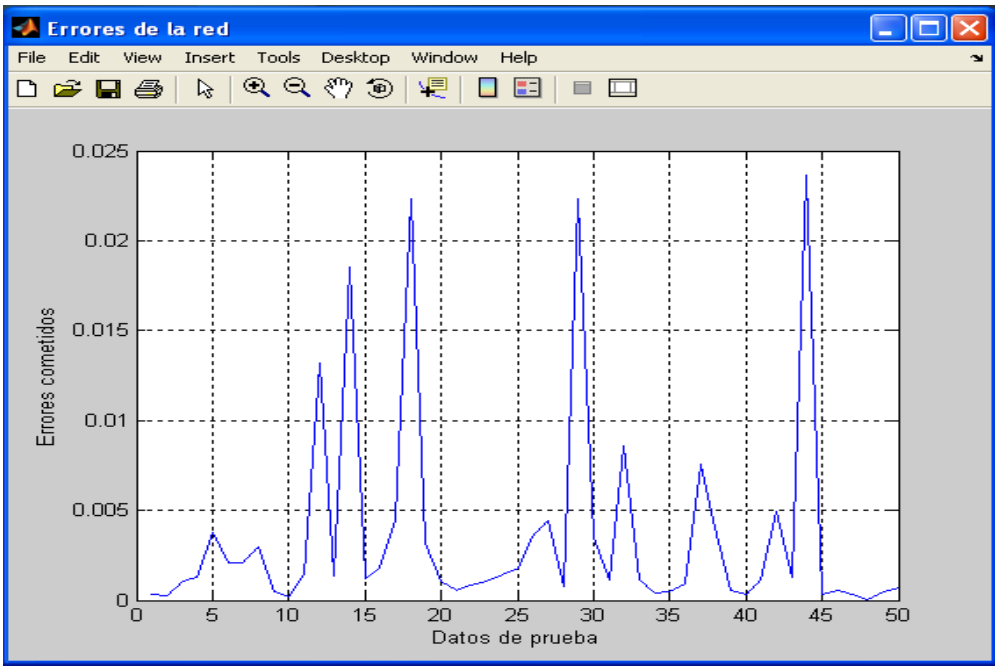

Figura 35. Errores[94] de simulación de la mejor configuración obtenida [6 1]

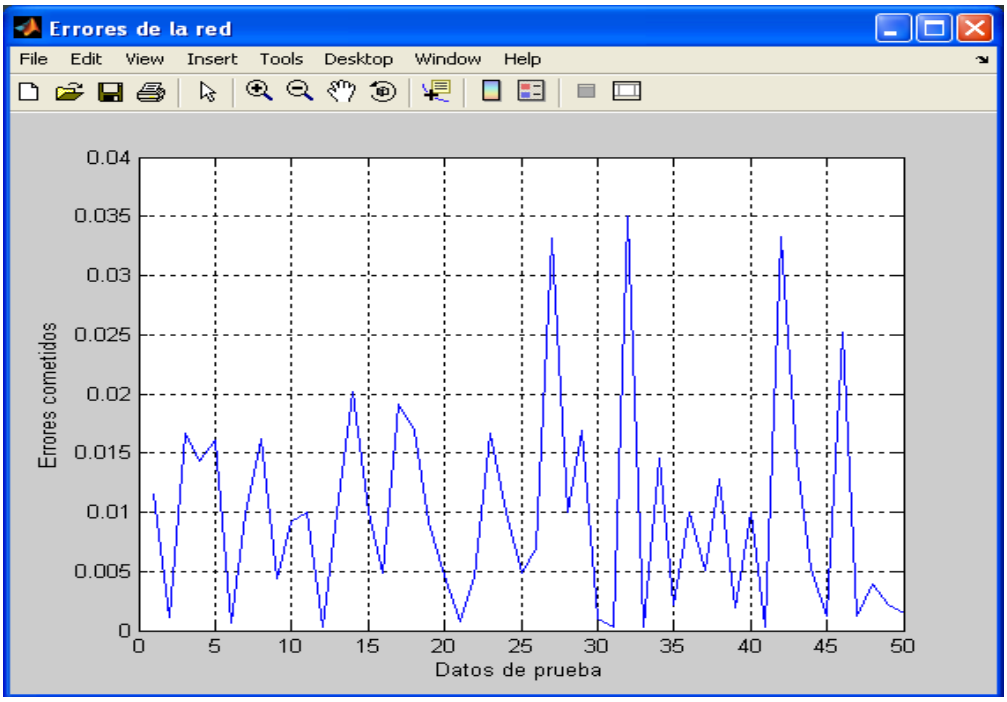

Figura 36. Errores[95] de simulación de configuración [2].

[94] Calculado con la ecuación 6.1.
[95] Calculado con la ecuación 6.1.

6.3.2.5 Simulación 5.

Los parámetros establecidos para la búsqueda son los mostrados a continuación:

Configuración	Número de errores	Peor error	Error promedio	Tasa de ajuste (%)	Funciones de activación capas ocultas	Funciones de activación capas de salida
[5 1]	0	0.073241	3.3208	100	Purelin Logsig	Tansig
[9 3]	0	0.086744	3.4948	100	Purelin Tansig	Tansig
[5 2]	0	0.089004	3.7553	100	Purelin Logsig	Radbas
[9 2]	0	0.091995	3.5159	100	Radbas Logsig	Radbas
[10 5]	1	0.11189	3.4419	98	Purelin Purelin	Logsig

Tabla 17. Mejores configuraciones halladas en la simulación 5.

En esta última prueba se obtuvieron buenos resultados al ser simulada con los datos de prueba, e igualmente se lograron soluciones usando un número reducido de neuronas por capa. Las redes fueron entrenadas con el algoritmo cuasi-newton (trianbfg), el número de redes diferentes probadas fue de 279, el número total de redes probadas fue de 1140 y el tiempo total empleado fue de 4.5 horas.

En esta simulación de nota una gran diferencia en el tiempo de simulación, con respecto a la prueba anterior. Esto se explica en el algoritmo de más rápida convergencia utilizado y las mejores características hardware del equipo donde se

realizó la prueba.

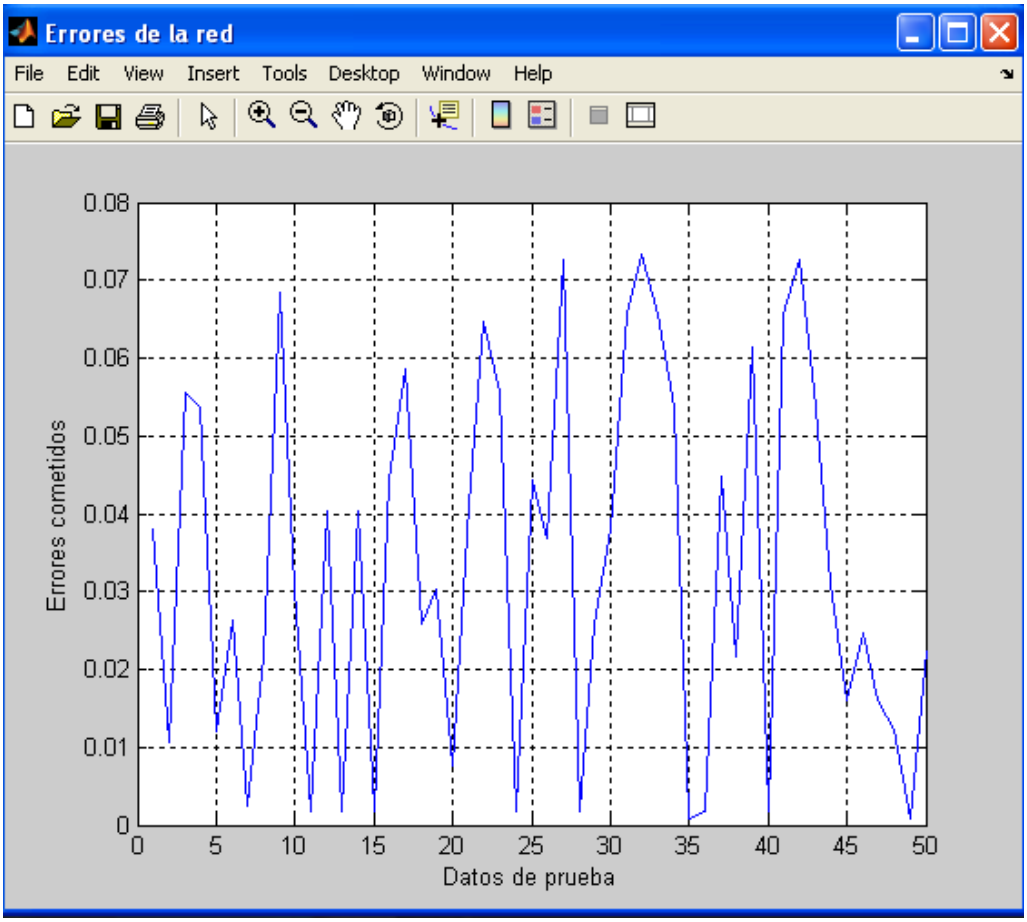

Figura 37. Errores[96] de simulación de la mejor configuración obtenida [5 1]

6.3.3 Comparación redes halladas por los estudiantes de inteligencia artificial.

Después de tener las diferentes configuraciones se debe escoger la que haya presentado mejores resultados para posteriormente ser comparada con la utilizada por el sistema original y la hallada por los autores del proyecto[97]. Para esto se simula cada una de las configuraciones con la totalidad de los datos del problema y se procede a calcular el porcentaje de ajuste de cada una de de estas topologías.

[96] Calculado con la ecuación 6.1.
[97] Todos los estudiantes utilizaron en software WÖTAN Genetics para la realización de las pruebas.

Las redes escogidas para las comparaciones son las mostradas en la tabla a continuación:

Nombre	Número de Capas	Capa	Número de neuronas	Función de activación	Tasa de ajuste[98] (%)
Red 1	3	Oculta 1	9	Purelin	57.5
		Oculta 2	10	Purelin	
		Salida	1	Radbas	
Red 2	2	Oculta 1	29	Tansig	91.4
		Salida	1	Radbas	
Red 3	3	Oculta 1	30	Tansig	95.6
		Oculta 2	21	Radbas	
		Salida	1	Radbas	
Red 4	3	Oculta 1	2	Logsig	99.9
		Salida	1	Tansig	
Red 5	3	Oculta 1	5	Purelin	97.4
		Oculta 2	1	Logsig	
		Salida	1	Tansig	

Tabla 18. Mejores soluciones obtenidas en cada simulación.

La configuración que mostró mejores resultados (red 4) fue la entrenada por medio del algoritmo trainlm y es la escogida para ser comparada con las otras configuraciones.

Al utilizar el error promedio de simulación[99], como medio de comparación del desempeño de las configuraciones, se puede ver más claramente la topología que obtuvo los mejores resultados; como es mostrado a continuación:

[98] Calculado con la ecuación 6.3.
[99] Ecuación 6.1.

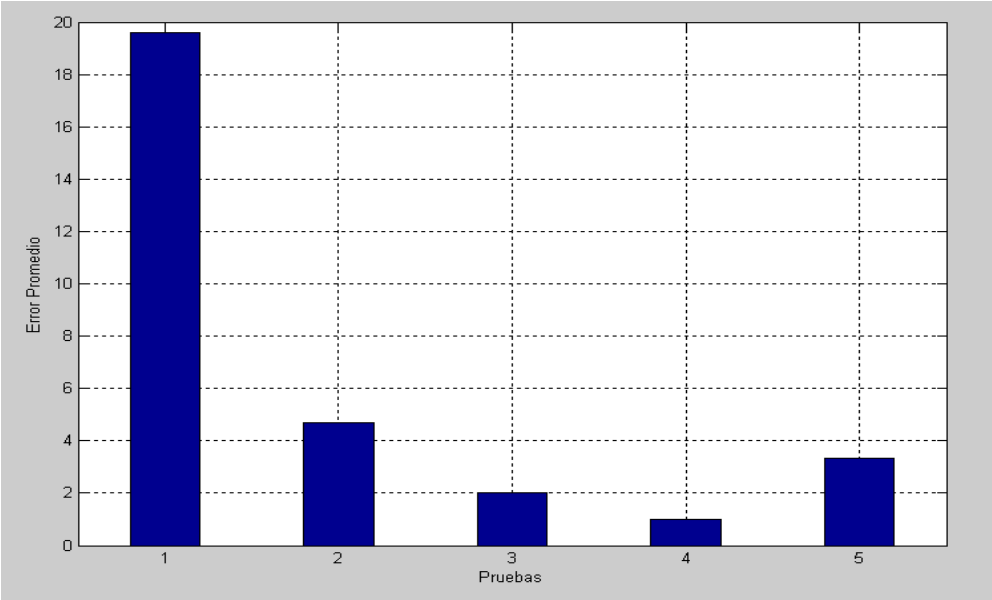

Figura 38. Error promedio[100] de las mejores configuraciones obtenidas en cada simulación.

La gráfica de los errores producidos por esta configuración, al ser simulada con la totalidad de los datos, es la mostrada a continuación:

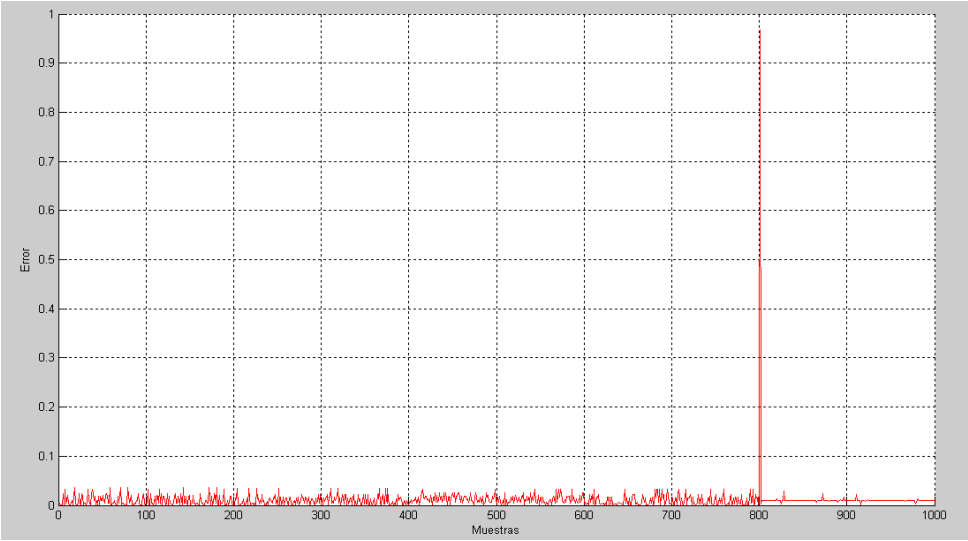

Figura 39. Errores[101] de simulación de red 4.

[100] Calculado con la ecuación 6.1.
[101] Calculado con la ecuación 6.1.

6.3.4 Configuración hallada por los autores del proyecto:

Después de tener el suficiente conocimiento sobre el comportamiento del problema, se decidió ajustar un espacio de búsqueda basado en los resultados obtenidos por los estudiantes de inteligencia artificial[102], al utilizar el software desarrollado. En este punto el número de neuronas y capas aproximadas para solucionar el problema eran conocidas, y se decidió utilizar un algoritmo que explotara el espacio de búsqueda de la mejor manera. Los parámetros utilizados fueron:

Número máximo de capas: 2
Número máximo de neuronas: 30
Tolerancia al error: 0,1
Algoritmo de entrenamiento: Trainlm
Numero de entrenamientos: 60
Numero de generaciones: 20
Tamaño de la población: 20
Probabilidad de mutación: 0.1
Probabilidad de cruce: 0.8

Configuración	Número de errores [103]	Peor error[104]	Error promedio[105]	Tasa de ajuste[106] (%)	Funciones de activación capas ocultas	Funciones de activación capas de salida
[6]	0	0.01497	0.48541	100	Radbas	Tansig

[102] Resultados mostrados en la sección 6.3.3.
[103] Calculado con la ecuación 6.4.
[104] Calculado con la ecuación 6.5.
[105] Calculado con la ecuación 6.1.
[106] Calculado con la ecuación 6.3.

[21]	0	0.02430	0.7097	100	Tansig	Logsig
[24]	0	0.03291	0.65773	100	Radbas	Radbas
[23]	0	0.03482	1.0594	100	Logsig	Radbas
[29]	0	0.03562	0.92956	100	Tansig	Radbas

Tabla 19. Mejores configuraciones obtenidas en la simulación por los autores.

Al ver las mejores configuraciones obtenidas se puede observar la tendencia de resolver el problema con una sola capa oculta. Los resultados de acuerdo al número de entrenamientos utilizados, lograron ser refinados a tal punto que en todos los casos produjeron errores menores a los establecidos en la tolerancia. Para esta prueba el número de redes diferentes probadas fue de 120, el número total de redes probadas fue de 1250 y el tiempo total empleado fue de 12 horas.

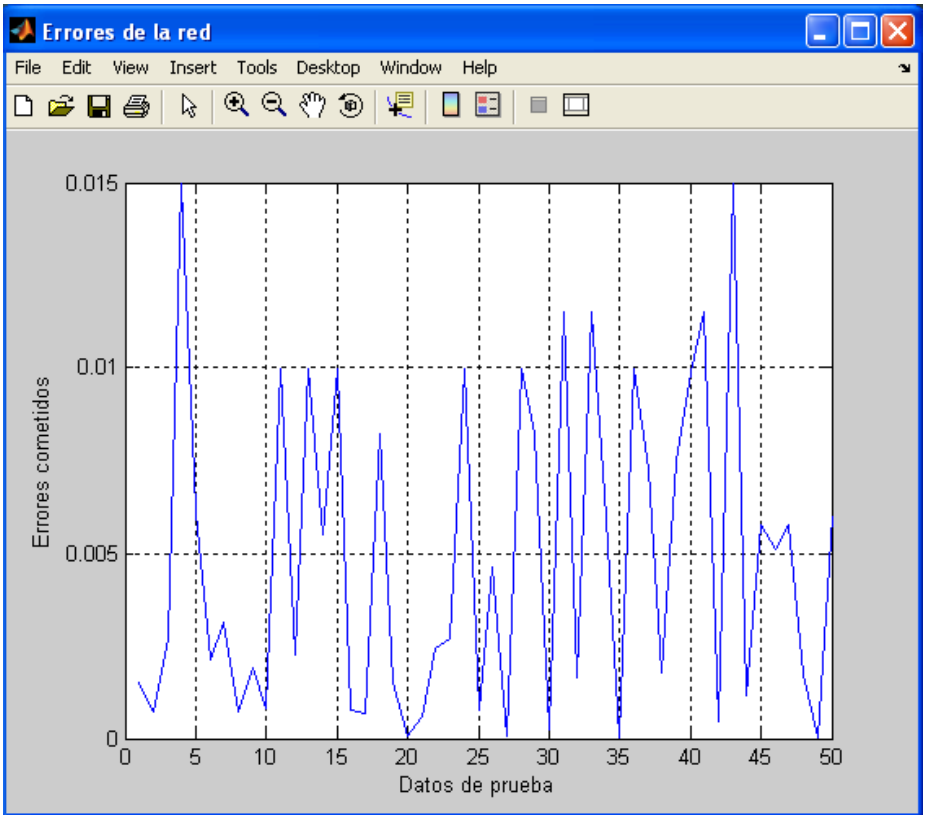

Figura 40. Errores[107] de simulación de la mejor configuración obtenida [6].

[107] Calculado con la ecuación 6.1.

6.3.5 Comparación de la configuración original usada en el programa, la hallada por los estudiantes y la hallada por los autores.

En este punto se tiene conocimiento de las mejores configuraciones en cada punto, halladas por diferentes métodos y con diferentes niveles de conocimiento del problema. El siguiente paso que se va a realizar es una comparación de las tres configuraciones, para esto se evaluarán con la totalidad de los datos del problema para calcular el error cometido. Además se medirán los tiempos de simulación para determinar la configuración que produzca los resultados más rápidamente.

En este caso la configuración hallada por los estudiantes de inteligencia artificial se denominara como red 1, la original del programa red 2, y la encontrada por los autores del proyecto red 3.

Nombre	Número de Capas	Capa	Número de neuronas	Función de activación	Tasa de ajuste[108] (%)	Tiempo de simulación. (segundos)
Red 1	2	Oculta 1	2	Logsig	90.40	0.008305
		Salida	1	Tansig		
Red 2	2	Oculta 1	20	Tansig	99.9	0.015592
		Salida	1	Tansig		
Red 3	2	Oculta 1	6	Radbas	100	0.010988
		Salida	1	Tansig		

Tabla 20. Comparación mejores configuraciones.

[108] Calculado con la ecuación 6.3.

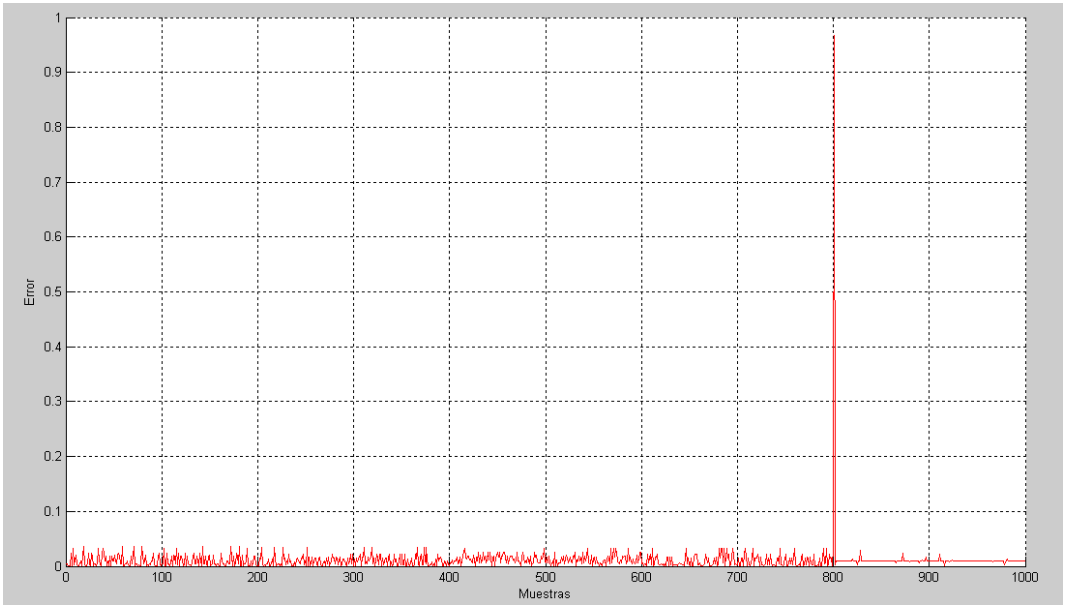

Figura 41. Errores[109] de simulación de red 1.

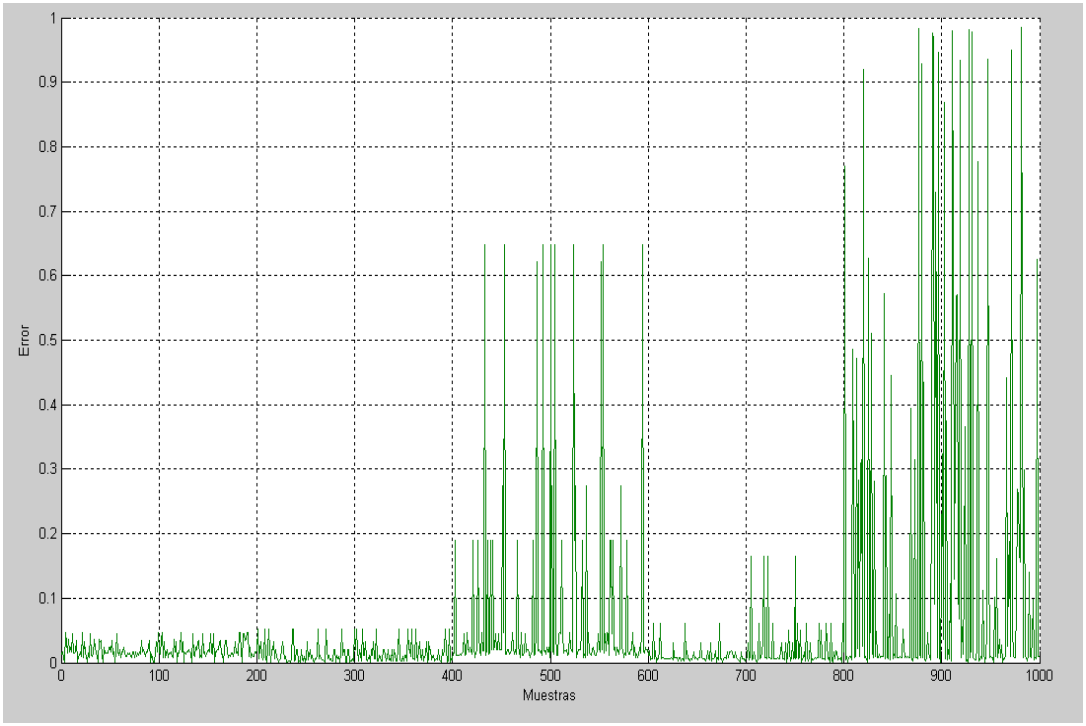

Figura 42. Errores[110] de simulación de red 2.

[109] Calculado con la ecuación 6.1.
[110] Calculado con la ecuación 6.1.

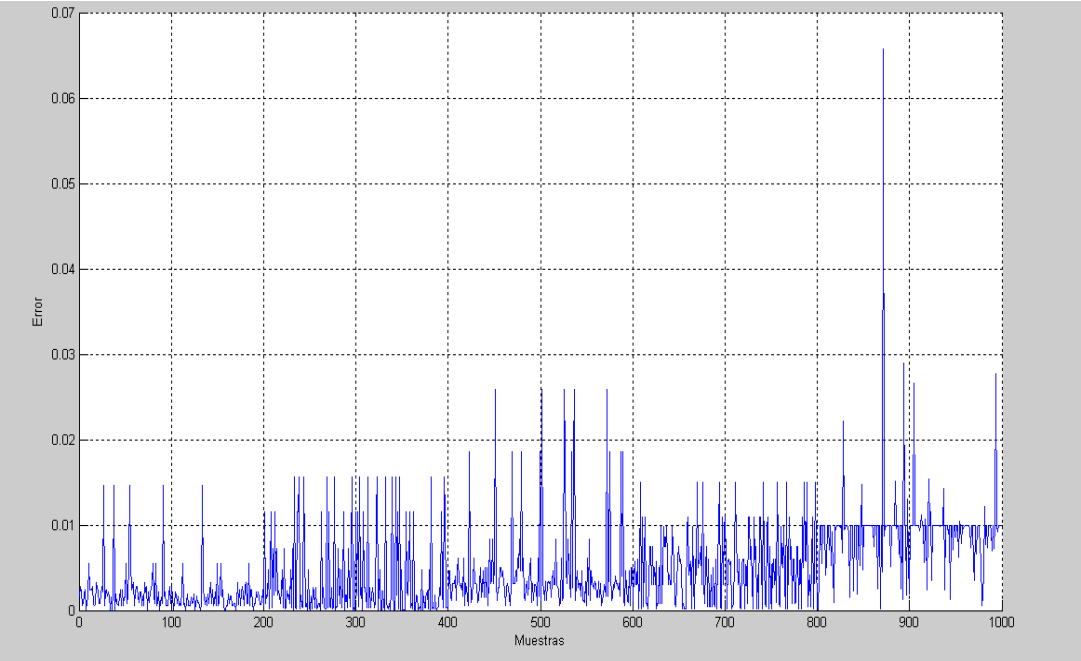

Figura 43. Errores[111] de simulación de red 3.

Después de analizar el comportamiento de las mejores configuraciones halladas en cada uno de las pruebas realizadas, se puede observar como la configuración obtenida por los autores del proyecto es la que produce mejores resultados en cuanto niveles de error, al ser simulada con la totalidad de los datos que se tiene disponibles. El siguiente mejor resultado obtenido es el de la configuración hallada por los estudiantes de inteligencia artificial, usando la herramienta desarrollada, la cual también produce los menores tiempos de simulación, esto debido al menor número de capas que se usan para dar solución al problema.

En general, al realizar las comparaciones de las mejores soluciones halladas por el configurador, se puede observar como el nivel de error que producía la red original, pudo ser reducido. Otra observación importante que se puede realizar sobre el comportamiento del sistema configurador, es la posibilidad de que posteriormente pudieran aparecer nuevos datos para ser agregados a la base conocimiento de la red neuronal; en ese caso lo único que debe hacerse es

[111] Calculado con la ecuación 6.1.

agregar estos datos al conjunto de entrenamiento, con la gran ventaja que ya se tiene una base de conocimiento sólida para determinar el espacio de búsqueda donde pudiera estar ubicada la mejor configuración de red neuronal para dar solución al problema.

7. CONCLUSIONES.

Es de notable importancia la aplicación de una herramienta que permita encontrar la topología de una red neuronal de forma automática, dada la cantidad de variables que afectan el comportamiento de determinado problema. Los métodos tradicionales de configuración están estrechamente ligados a la suerte que se tenga al realizar la búsqueda además del conocimiento que tenga el experto sobre el comportamiento del problema, adicionalmente la utilización de determinada configuración no logra tener una justificación sólida que indique porque se usó.

Con la aplicación del sistema automático se logran mejorar algunos de los problemas que existen en la configuración tradicional. Por una parte las configuraciones probadas se generan de manera automática y son guiadas por el comportamiento del algoritmo genético, y por otro lado se presenta una justificación que indica la variación del problema al cambiar algunos de los parámetros que intervienen, mostrando así el comportamiento del problema, y una base de conocimiento para decidir que configuración usar.

En la configuración de redes neuronales artificiales existen muchos problemas en los cuales no se logra una solución óptima aplicando una sola técnica básica de configuración, cada una de estas presenta sus ventajas y desventajas. El potencial del algoritmo híbrido radica en la combinación de dos enfoques usados en la configuración de redes neuronales artificiales (capas-neuronas, funciones de activación) modificados de forma evolutiva de tal manera que se compensen sus debilidades y se potencien sus fortalezas.

Los resultados obtenidos mediante la aplicación del algoritmo, han demostrado ser muy satisfactorios, superando los resultados obtenidos al realizar una configuración a prueba y error, donde el éxito está determinado en gran parte por el azar. Uno de los factores críticos en la configuración/entrenamiento de redes

neuronales es el tiempo que se requiere para realizar la búsqueda y llegar a un resultado que cumpla las condiciones que fueron establecidas. Esto fue un aspecto que fue considerado de manera especial en la realización del algoritmo y que logró ser controlado para que las búsquedas se produjeran en un tiempo razonable.

Otra observación importante que se puede realizar sobre el comportamiento del sistema configurador, es la posibilidad de que posteriormente pudieran aparecer nuevos datos para ser agregados a la base conocimiento de la red neuronal. En ese caso lo único que debe hacerse es agregar estos datos al conjunto de entrenamiento; con la gran ventaja que ya se tiene un conocimiento previo del problema que permita determinar el espacio de búsqueda donde pudiera estar ubicada la mejor configuración de red neuronal para dar solución al problema.

En conclusión, la aplicación del algoritmo muestra una mejora visible en cuanto al desempeño de las soluciones obtenidas con respecto a técnicas clásicas de configuración, además de presentar un análisis de la variación del problema al variar sus parámetros más influyentes. Además se logró una disminución en los tiempos de búsqueda del algoritmo, logrando un muy buen desempeño para un algoritmo encargado también de realizar el entrenamiento de las configuraciones de manera secuencial.

8. RECOMENDACIONES.

Los tiempos de entrenamiento son sin duda los que implican mayor carga computacional y por tanto los que retardan el tiempo requerido para producir una solución. En aras de disminuir estos tiempos sería muy útil desarrollar una versión paralela del algoritmo, donde se pudieran lograr mejores soluciones utilizando una menor cantidad de tiempo.

Sin duda se han mostrado las bondades de las soluciones encontradas modificando dos (2) de los parámetros importantes de la red neuronal artificial, sin embargo, son muchos más los parámetros que se pueden modificar para lograr explorar un espacio de búsqueda aun mayor. En el desarrollo de este algoritmo no se contempló la modificación de más parámetros dado el aumento de los tiempos que representaría, sin embargo seria de gran utilidad realizar una nueva versión incluyendo la modificación genética de más parámetros.

En esta versión, el entrenamiento de las configuraciones se realiza mediante los algoritmos tradicionales dada su comprobada efectividad. La realización del entrenamiento mediante un enfoque evolutivo podría ser una gran mejoría en el desempeño en cuanto a soluciones obtenidas, sin embargo el costo computacional de realizar este tipo de entrenamiento solo sería viable mediante una versión paralelizada del algoritmo.

Dados las grandes demandas computacionales al realizar el entrenamiento de un gran número de configuraciones en cada generación, se recomienda utilizar para la ejecución un equipo con buenas prestaciones, en cuanto a memoria y procesador se refiere, y dedicado a la actividad de búsqueda de las configuraciones.

9. BIBLIOGRAFÍA

[1] Angeline P.J, Saunders G.M and Pollack J.B. An evolutionary algorithm that constructs recurrent neural networks, IEEE Trans. Neural Networks, Vol 5, 1994 pp 54-65.

[2] Blickle T. and Thiele L. A comparison of selection schemes used in genetic algorithms. Technical Report 11, Computer Engineering and Comunnication Network Lab (TIK), Gloriastrasse 35, 8092 Zurich, Switzerland, 1995.

[3] D. Curran, C. O'Riordan, Applying Evolutionary Computation to Designing Neural Networks: A Study of the Satate of the Art, technical report NUIG-IT-111002, National University of Ireland, Galway, 2002.

[4] D. Whitley, T. Starweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. Parallel Computing, 347-361,1990.

[5] Darwin, C. On the Origin of Species by Means of Natural Selection. John Murray, London, 1859.

[6] David W. White. GANNet: A genetic algorithm for searching topology and weight spaces in neural network design. PhD thesis, University of Maryland College Park, 1994.

[7] Dow R. J. y Sietsma J. *Creating Artificial Neural Networks that generalize*. Neural Networks, vol. 4, no. 1, pp. 198-209, 1991.

[8] Fiszelew A. Generación automática de redes neuronales con ajuste de parámetros basado en algoritmos genéticos, Universidad de Buenos Aires Facultad de Ingeniería, 2000.

[9] G. Cybenko, Approximations by superpositions of sigmoidal functions, Mathematics of Control, Signals, and Systems 2, 303--314, 1989.

[10] Gestal M., Introducción a los algoritmos genéticos, departamento de tecnlogías de la información y las comunicaciones, Universidade da Coruña, 1996.

[11] Goldberg D. E. *A comparative analysis of selection schemes used in genetic algorithms*. In Gregory Rawlins, editor. Foundations of Genetic Algorithms, pages 69-93, San Mateo, CA: Morgan Kaufmann Publishers, 1991.

[12] H. White K. Hornik, M. Stinchcombe. Multi-layer feed-forward networks are universal approximators. Neural Networks, pages 2:359--366, 1989.

[13] Holland, J. H. *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, 1975

[14] Honavar V. and L. Uhr. *Generative Learning Structures and Processes for Generalized Connectionist Networks*. Information Sciences, 70:75—108,1993.

[15] J.D. Schaffer, Combinations of genetic algorithms with neural networks or fuzzy systems ,IEEE press, 1994, pp 371-382.

[16] J. Flórez, F. Porras, Determinación del patrón de flujo multifásico en tuberías de recolección de petróleo emulsionado, a partir de los datos históricos de producción apoyado en una aplicación con redes neuronales artificiales, Universidad Industrial de Santander, Facultad Ingenierías Físico Mecánicas, 2007.

[17] K. Stanley and R. Miikulainen. Evolving neural networks through augmenting topologies, technical report ai01-290. Technical report, Department of Computer Science, University of Texas as Austin 2001.

[18] Koza, John R. and Rice James P.. Genetic generation of both the weights and architecture for a neural network. In International Joint Conference on Neural Networks, IJCNN 91, volume II, pages 397-404.Washington State Convention and Trade Center, Seattle, WA, USA, pages 8-12 1991. IEEE Computer Society Press.

[19] L. Tarassenko. Novelty detection for the identification of masses in mammograms, Proceedings Fourth IEEE international conference on artificial neural networks, 4:442-447 , 1995.

[20] L. Llano, A. Hoyos, F. Arias, J. Velásquez. Comparación del desempeño de funciones de activación en redes Feedforward para aproximar funciones de datos con y sin ruido, Universidad Nacional de Colombia Sede Medellín, 2007.

[21] M. Jordan, C. Bishop. Neural Networks, MASSASHUSETTS INSTITUTE OF TECHNOLOGY – Artificial intelligence laboratory, 1996.

[22] M. Mandischer. Representation and evolution of neural networks. In R. F. Albrecht, C.R. Reeves, and N.C Steele, editors, Artificial Neural Nets and Genetic Algorithms Proceedings of the International Conference at Innsbruck, Austria, pages 643-649. Springer, Wien and New York, 1993.

[23] Muñoz J. Diseño y entrenamiento en paralelo de redes neuronales, por medio de algoritmos genéticos desordenados y altamente recursivos. Universidad Tecnológica de Pereira, 2006.

[24] P.J.B. Hancock. Pruning neural nets by genetic algorithm. In I. Aleksander and J.G. Taylor, editors, Proceeding of the International Conference on Artificial Neural Networks, Brighton, pages 991-994. Elsevier, 1992.

[25] P.M Todd G. F. Miller and S.U. Hedge. Designing neural networks using genetic algorithms. In proceedings of the Third International conference on Genetic Algorithms and their Applications, pages 379-384, 1989.

[26] Peralta J, Gutierrez G, Sanchis A. Design of Artificial Neural Networks based on genetic Algorithms to forecast time, CAOS Group, Computer Science Department, University Carlos III of Madrid, 2007.

[27] Pujol J. and Poli R. Efficient evolution of asymemetric recurrent neural networks using PDGP inspired two dimensional representation. Lecture Notes in Computer Science, 1998

[28] S. Russell, P. Norvig. Inteligencia Artificial Un enfoque moderno, Prentice Hall, Segunda edición, 2004.

[29] U. Chinchilla, G. Pérez. Herramienta software para la identificación de sistemas dinámicos no lineales, Universidad Industrial de Santander, Facultad Ingenierías Físico Mecánicas, 2007.

[30] V. W. Porto, D. B. Fogel, and L. J. Fogel. Alternative Neural Network Training Methods, IEEE EXPERT, 1995.

[31] W. Schiffmann, M. Joost, and R. Werner. Application of genetic algorithms to the construction of topologies for multilayer perceptrons. In proceedings of the International Conference of Artificial Neural Networks and Genetic Algorithms pages 675-682, 1993.

[32] Werbos, P. Beyond Regression: New tools for prediction and Analisis in the behavioral sciences, doctoral dissertation, Harvard, Cambridge, Mass, 1974.

[33] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. 1989.

[34] Yao Xin, *Evolving Artificial Neural Networks. School of Computer Science*. The University of Birmingham. B15 2TT 1999.

10. ANEXO 1: MANUAL DE USUARIO

10.1 Introducción

Wötan Genetics es una herramienta computacional para la configuración de redes neuronales artificiales por medio de algoritmos genéticos, para problemas específicos de clasificación y aproximación. Son varios los procesos que realiza el sistema durante el estudio de los datos de un problema particular, los cuales serán explicados con más detalle a continuación.

10.2 Selección del tipo de error y tipo de entrenamiento

Lo primero que se ve al iniciar el programa es un menú de selección de los parámetros con los cuales se va a guiar la búsqueda, dependiendo de la selección de estos parámetros será el comportamiento del algoritmo genético y la forma de cargar los datos usada.

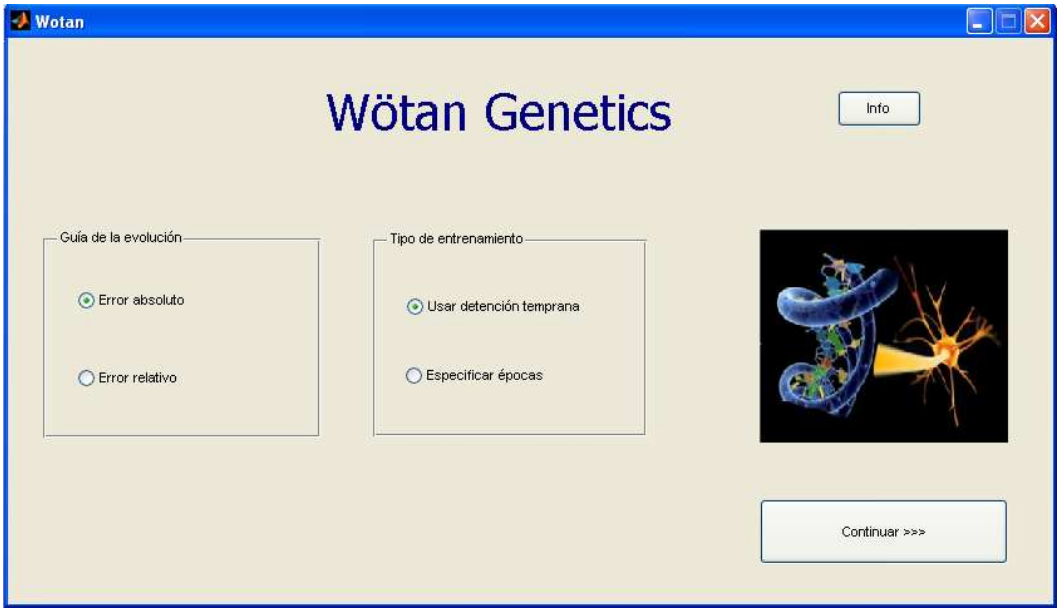

Figura 44. Ventana inicial del programa.

El tipo de error le indica al algoritmo genético la manera cómo va a evaluar el desempeño de una configuración, las opciones que se muestran para guiar la evolución son mediante el error absoluto o un error relativo porcentual. La selección del tipo de error a usar depende de la naturaleza de los datos de entrenamiento a usar y/o de la conveniencia del usuario de usar uno u otro. Se recomienda utilizar el error absoluto cuando alguna de las salidas de los datos de entrenamiento toma el valor de cero puesto que se produciría un error en el cálculo del error relativo al dividirse por este valor.

Al usar el error relativo se debe tener en cuenta que el error tolerado que se escoge no es una diferencia real como en el absoluto sino que es un porcentaje de error permitido entre la salida obtenida y la obtenida.

Los tipos de entrenamiento se refieren a los tiempos en los que se va a realizar el entrenamiento de la red neuronal, para esto se tiene la opción de escoger entrenamiento con detención temprana lo que indica que se deben tener seleccionados los conjuntos de datos necesarios para este tipo de entrenamiento. Al especificar el número de épocas en las que se va a realizar el entrenamiento se debe tener un conocimiento sobre el comportamiento del problema para poder escoger un valor adecuado.

Esta opción es recomendada cuando se desean tomar mayor cantidad de datos para realizar en el entrenamiento de la red o cuando se tiene la noción sobre el número de épocas necesarias para realizar un buen entrenamiento.

10.3 Carga de datos

La carga de datos se realiza de acuerdo al tipo de entrenamiento escogido, si se

decidió usar detención temprana se deben especificar los conjuntos de entrenamiento y validación; el conjunto de prueba es opcional y es usado para la evaluación del error de la configuración, si no se especifica se usará el conjunto de datos de validación.

Por otro lado si se decidiera especificar las épocas solo se tendría que cargar el conjunto de datos de entrenamiento de la red, en este caso es recomendado escoger un grupo de datos para realizar las pruebas a la red ya que si se usan los mismos datos de entrenamiento no se garantiza una buena generalización de la red. Adicionalmente se debe escoger el número de épocas durante la cual se va a realizar e entrenamiento de la red.

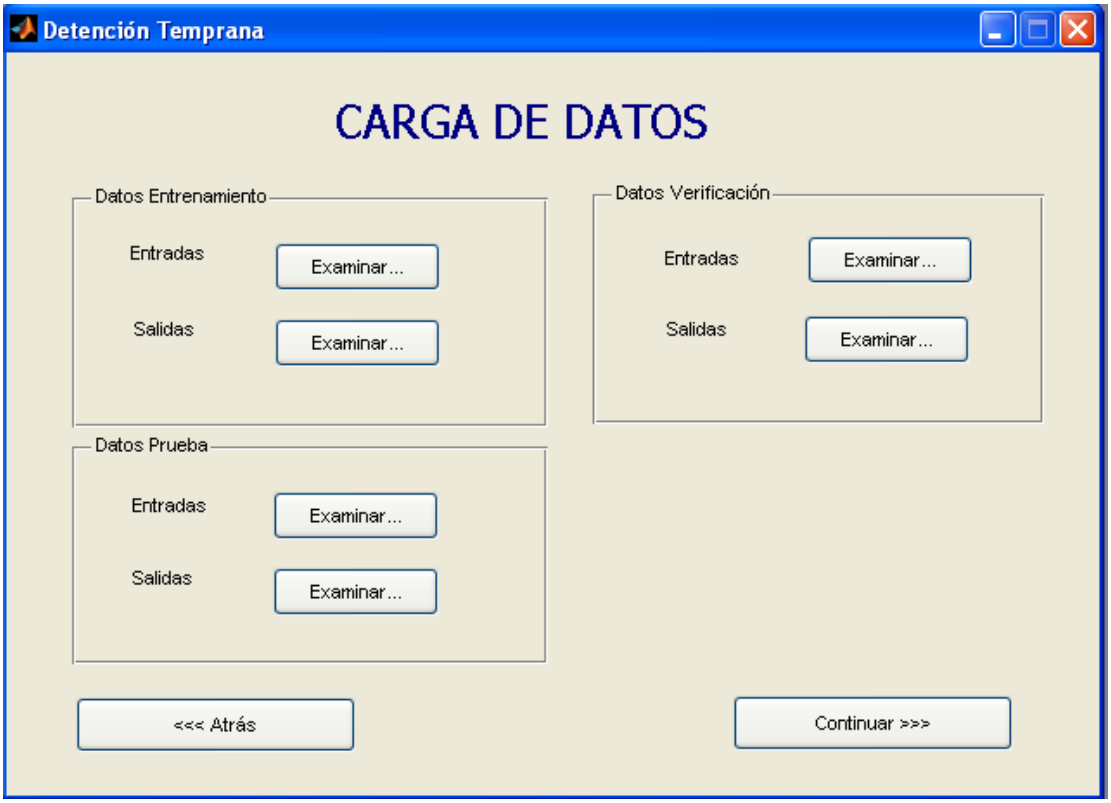

Figura 45. Carga de datos para detención temprana.

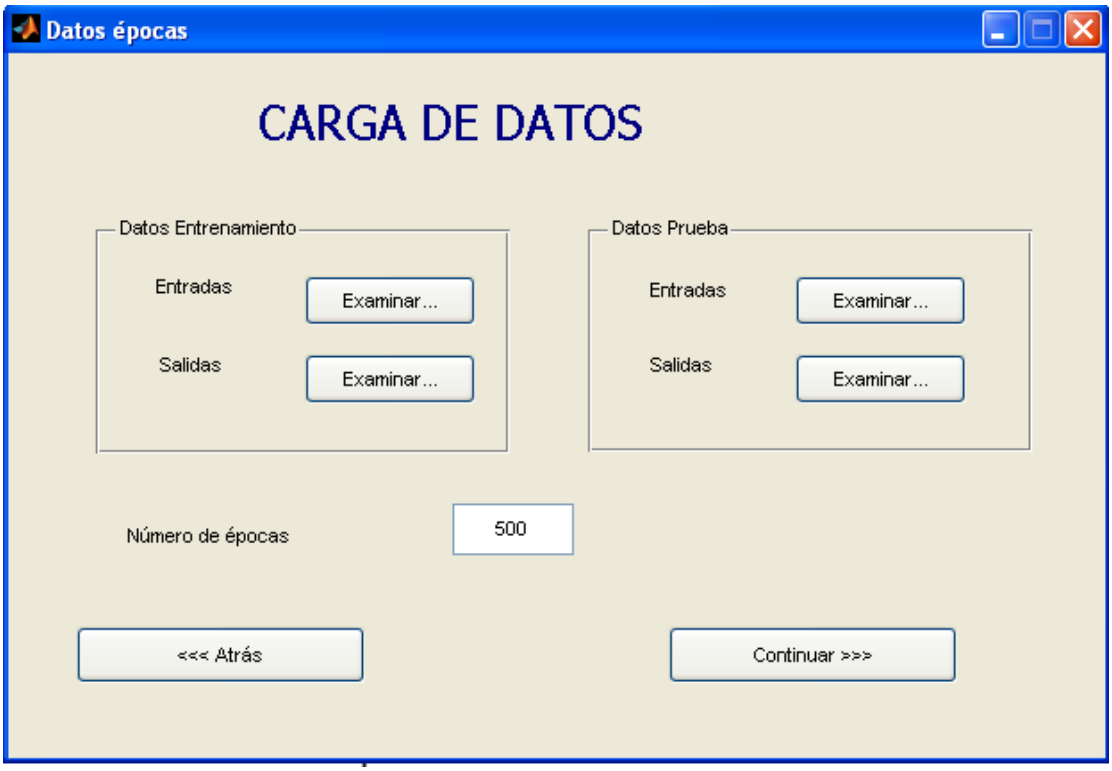

Figura 46. Carga de datos especificando épocas.

Para la carga de datos se definió un sencillo formato en el cual cada una de los conjuntos de entrenamiento se guarda en una archivo de texto plano con extensión .dat y mediante el programa se realiza la carga de cada uno de estos conjuntos. Para esto cada una de las entradas o salidas del problema se ubica en una columna y los distintos ejemplos que se van a utilizar se ordenan en las filas.

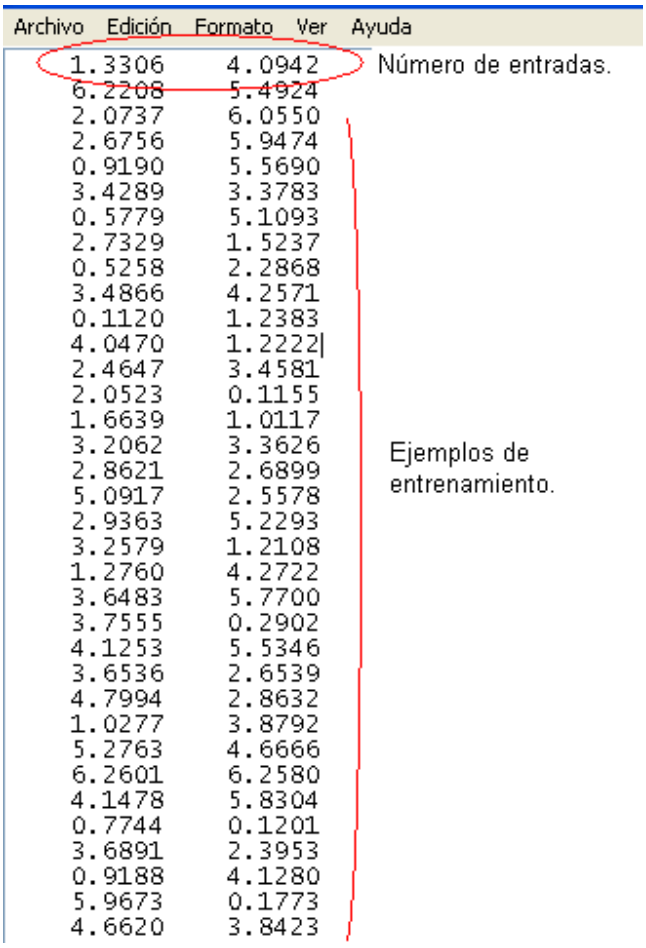

Figura 47. Datos para un problema con dos entradas.

Cada uno de los conjuntos debe ser guardado en un archivo por separado para posteriormente ser cargados al programa.

10.4 Carga de Parámetros

El siguiente paso consiste en especificar el espacio de búsqueda del algoritmo genético y los demás datos que se van a usar en el problema, para esto se tiene:

- Tamaño de la población: Cantidad inicial de soluciones posibles consideradas, por defecto se utiliza el valor de 20 el cual es un valor

estándar usado en algoritmos genéticos. El empleo de 20 individuos sirve para acelerar el desarrollo de los experimentos sin afectar a los resultados.

- Probabilidad de mutación: Determinación de la probabilidad que tienen los individuos de mutar un gen en cada ciclo del algoritmo genético, esto se hace para buscar diversidad. La probabilidad de mutación suele ser muy baja, por lo general entre el 0.5% y el 2 %.
- Probabilidad de cruce: Indica la probabilidad que tienen los individuos que sean seleccionados para jugar el papel de padres, y posteriormente ser recombinados.
- Máximo de capas: Indica el máximo número de capas que se va a utilizar en la busque de las configuraciones.
- Máximo de neuronas por capa: Selección del límite superior de neuronas que se van a probar para buscar la solución el problema.
- Número de entrenamientos: Selección del número de veces que va a ser entrenada cada configuración, se recomienda usar una cantidad significativa de entrenamientos para garantizar que cada configuración logre explotar todo su potencial.
- Algoritmo de entrenamiento: Selección del algoritmo con el cual se realizara el entrenamiento de cada una de las configuraciones, en la sección 3.4.1 se pueden ver con más detalle los algoritmos de entrenamiento implementados.
- Generaciones: Número de repeticiones durante las cuales se va a realizar la búsqueda de las configuraciones.
- Tolerancia al error: Máximo valor de error permitido en la salida de la red para ser considerado una falla.

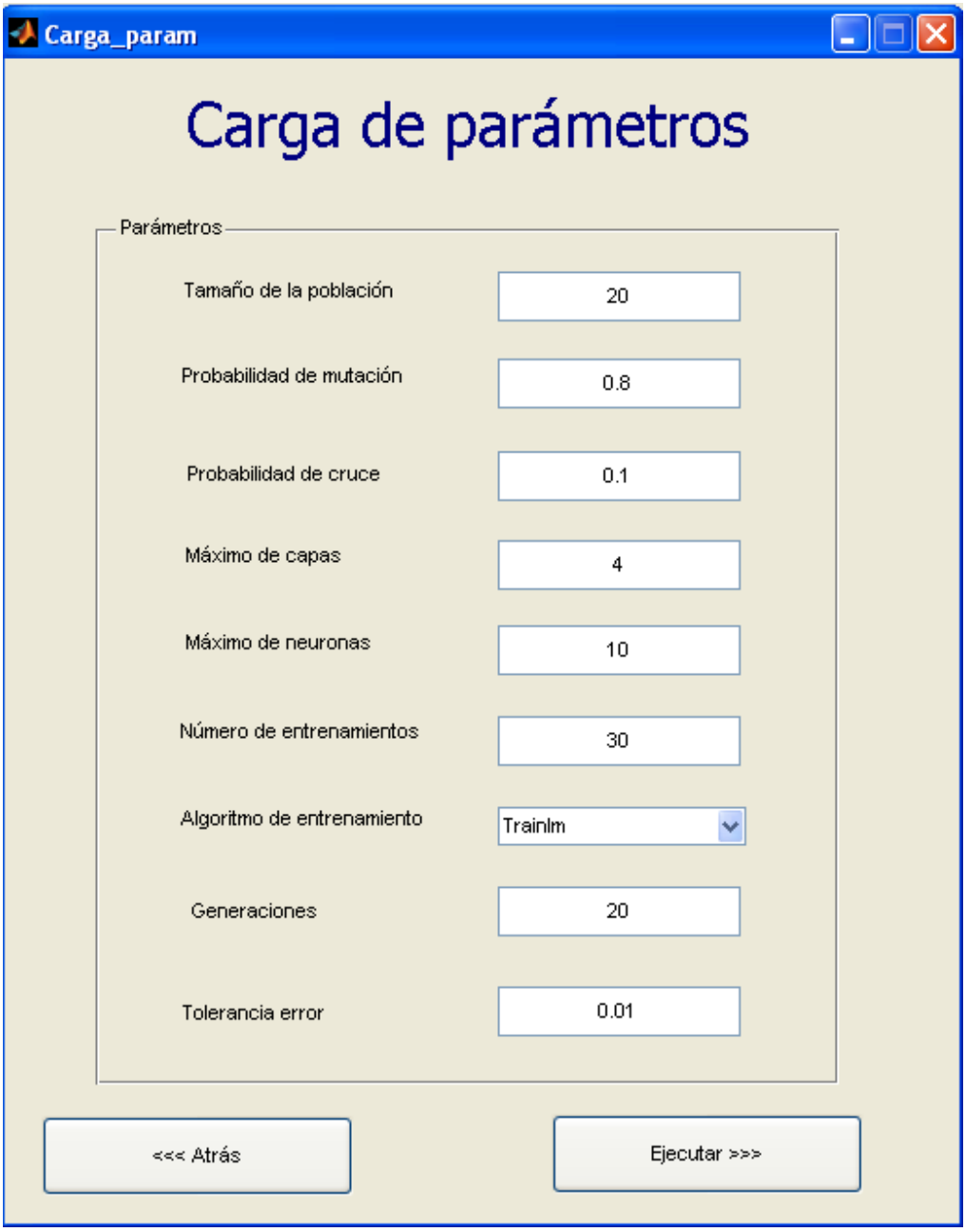

Figura 48. Carga de parámetros para un problema.

Después de seleccionar cada uno de los parámetros se procede a realizar la ejecución del algoritmo donde se probaran las posibilidades para luego mostrar los resultados correspondientes.

10.5 Visualización de soluciones

Luego del proceso de búsqueda se muestran las soluciones encontradas, ordenadas de acuerdo al número de errores cometidos al evaluar cada configuración con los datos de prueba. En la parte izquierda (Numeral 1.) se muestran las configuraciones finales encontradas en el algoritmo genético de acuerdo un arreglo donde cada posición representa una capa oculta usada, mientras que el número representa las neuronas ocultas utilizadas en cada capa. Así pues la configuración [3 5 7], representaría una configuración con tres capas ocultas cada una con 3, 5 y 7 neuronas en cada capa respectivamente.

Al seleccionar cada una de estas configuraciones, se muestran algunos detalles sobre el comportamiento de las mismas. Las opciones se describirán a continuación:

1. Configuración de las capas ocultas: Soluciones encontradas por el algoritmo.
2. Número de errores: Muestra el número de fallas cometidas al simular la red teniendo como referencia la tolerancia al error previamente seleccionada.
3. Peor error: Indica el máximo error cometido al simular la configuración hallada.
4. Error promedio: Indica la media de los errores cometidos en la salida de la red simulada.
5. Varianza de los errores: Dispersión de los errores cometidos por la red neuronal para determinar la uniformidad de los mismos.
6. Funciones de activación (capas ocultas): Muestra las funciones de activación usadas en cada una de las capas ocultas de la red neuronal. Así pues cada función que aparece a la función de activación relativa a su posición en el arreglo mostrado.
7. Función de activación (capa de salida): Corresponde a la función de activación usada en la capa de salida en la configuración respectiva.

8. Gráfica errores: Comportamiento de la salida de la red simulada con los datos de prueba.
9. Guardar red: Una vez obtenidas las configuraciones que mejor se adapten al problema, se tiene la posibilidad de guardarlas para trabajar directamente sobre ellas. Esto es posible ya que cada una de las soluciones mostradas corresponde a la red neuronal entrenada que produce los valores que se indican. Estas redes se guardan en un archivo con extensión .mat propio de Matlab, las cuales se pueden posteriormente cargar el comando load.

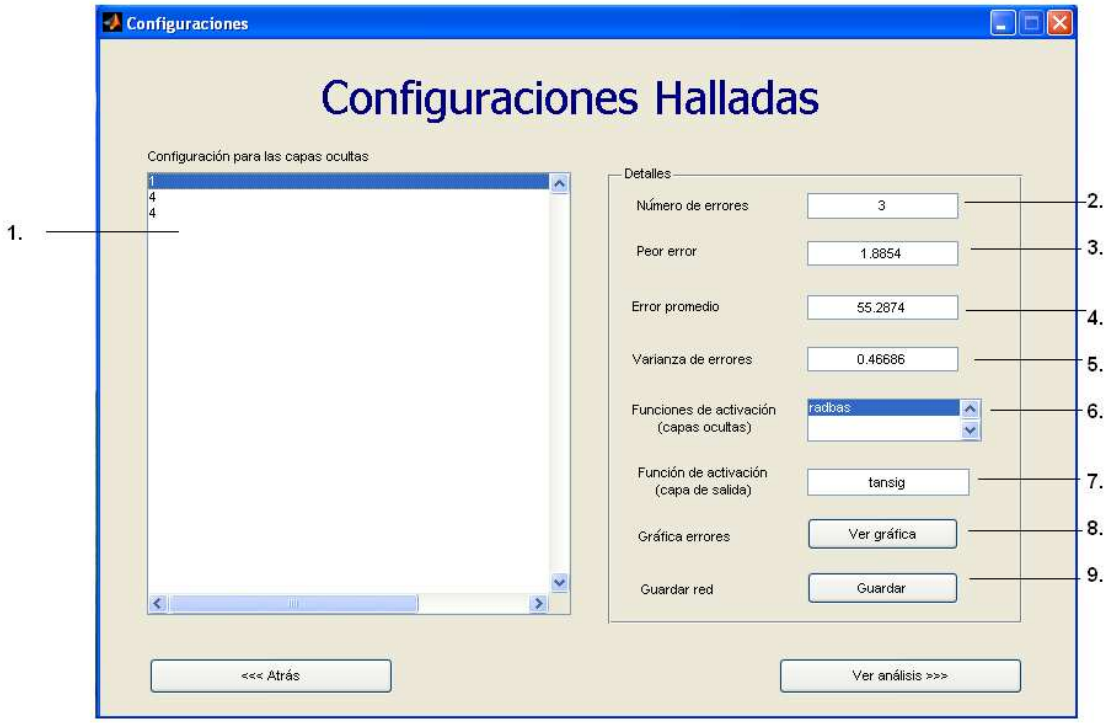

Figura 49. Ventana de configuraciones.

Figura 50. Guardar red seleccionada.

10.6 Análisis del proceso

Los detalles de cada configuración si bien son importantes no proporcionan información sobre el comportamiento general de la búsqueda. Es en este punto donde se muestra un análisis global, con el cual el usuario tiene la posibilidad de visualizar el comportamiento del problema al variar las condiciones de búsqueda tales como el número de capas usado o el número de neuronas.

El menú de análisis del problema esta divido en dos secciones, el primero ubicado al lado izquierdo donde se seleccionan las opciones que se desean visualizar. Y el segundo al lado derecho está ubicada el área de graficación donde son mostradas todas las imágenes correspondientes a la opción seleccionada.

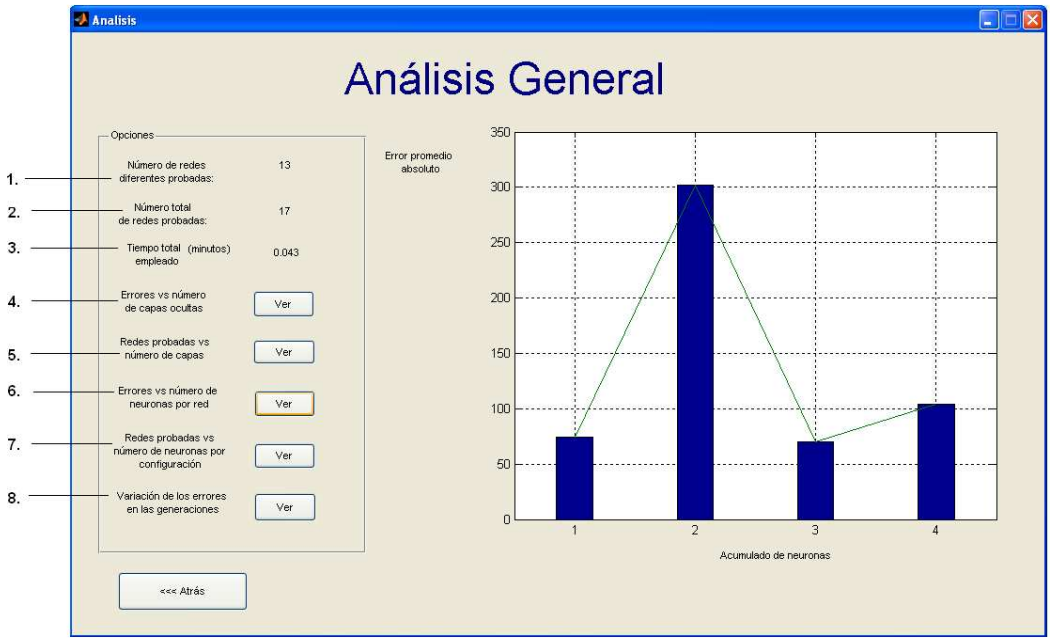

Figura 51. Ventana de análisis general.

1. Número de redes diferentes probadas: Muestra la cantidad de topologías heterogéneas evaluadas.

2. Número de redes probadas: Indica la cantidad de configuraciones que fueron consideradas en el algoritmo, así estas hayan aparecido más de una vez.

3. Tiempo total empleado: Lapso empleado en la búsqueda de las soluciones.

4. Errores vs Número de capas ocultas: Muestra el comportamiento de los errores al variar el número de capas en cada configuración.

5. Redes probadas vs Número de capas: Indica el número de configuraciones evaluadas de acuerdo al número de capas usado. Esto es de especial

utilidad para observar la tendencia del algoritmo de converger al uso de una cantidad de capas para resolver el problema.

6. Errores vs Número de neuronas por red: Muestra el comportamiento del problema al variar el número de neuronas (acumulado) usado en cada configuración.

7. Redes probadas vs Número de neuronas probadas por configuración: Se muestra la cantidad de redes evaluadas de acuerdo al número de neuronas ocultas (acumulado) usado en cada topología.

8. Variación de los errores en las generaciones: Muestra el comportamiento de errores producidos al simular cada configuración, el punto más bajo en esta grafica representa la configuración con menor error promedio en salida, como muestra de proceso se indica con una cruz roja la generación en la que apareció esta solución y como se conserva hasta el final debido al método de reemplazo utilizado en el algoritmo.

De acuerdo a estas gráficas se puede conocer el comportamiento del problema al variar los parámetros más influyentes para lograr una mejor salida al problema, de acuerdo a estos datos el usuario está en capacidad de decidir que configuración usar para resolver su problema o incluso de utilizar una de las configuraciones entrenadas por el algoritmo.

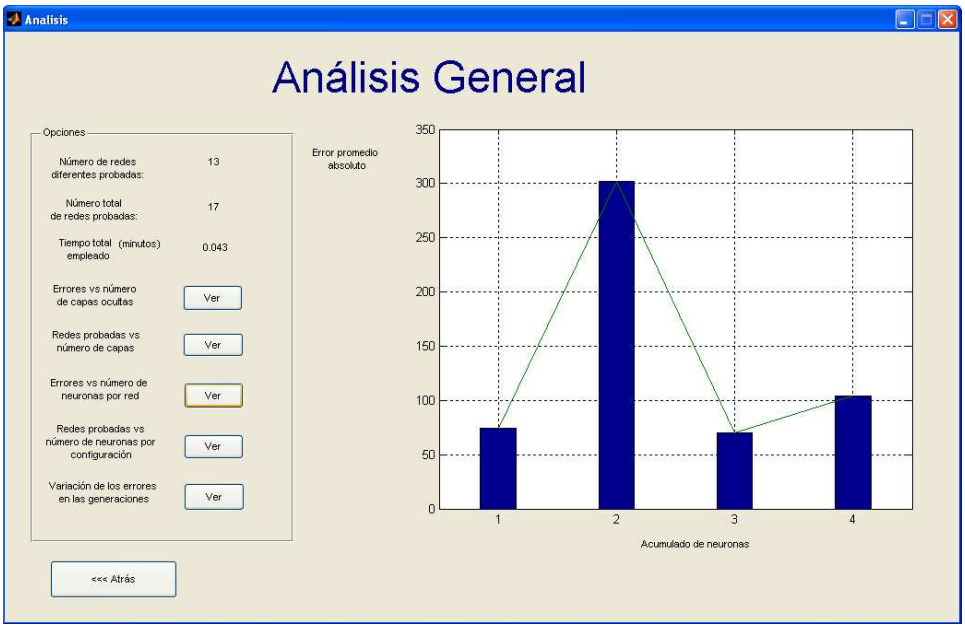

Figura 52. Errores vs numero de neuronas por red

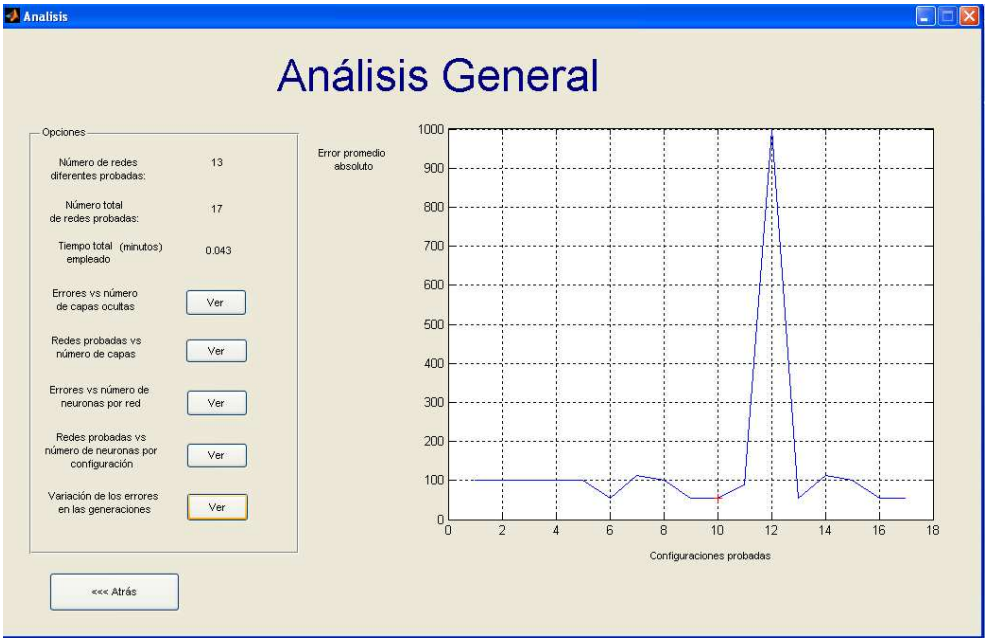

Figura 53. Variación de los errores en las generaciones.

11. ANEXO 2: ARTÍCULO PRESENTADO EN EL CONGRESO INTERNACIONAL DE INGENIERÍA ELECTRÓNICA Y MECATRÓNICA, UNIVERSIDAD DE SAN BUENAVENTURA 2008.

ENTRENAMIENTO DE REDES NEURONALES CON OPTIMIZACIÓN EN LA TOPOLOGÍA Y FUNCIONES DE ACTIVACIÓN CON ALGORITMOS GENÉTICOS.

Edgar A. Méndez Ortiz, Juan Sebastián Mariño Mesa, Msc. Henry Arguello Fuentes
edgar9000@gmail.com, jhusema@gmail.com, henryarguellofuentes@hotmail.com
Grupo de Investigación en ingeniería biomédica (GIIB)
Universidad Industrial de Santander

RESUMEN.
Este trabajo explica el diseño y funcionamiento de un algoritmo híbrido para la configuración de forma no convencional de redes neuronales artificiales, aplicadas en la solución de problemas particulares. En este caso un algoritmo genético es el encargado de realizar el diseño de la arquitectura (capas- neuronas) y las funciones de activación que se deben usar en cada una de estas capas, para lo cual se realiza una codificación directa sobre los diferentes parámetros que van a ser modificados por métodos evolutivos. Se utiliza un algoritmo genético para hallar estos parámetros dada su capacidad como sistema de búsqueda óptima en conjuntos de gran tamaño. El entrenamiento de las configuraciones se realiza por métodos convencionales basados en propagación hacia atrás (*backpropagation*), además se aplican algunas técnicas para lograr una mejor generalización.

PALABRAS CLAVES: redes neuronales, algoritmos genéticos, computación evolutiva, capas ocultas, funciones de activación

1. INTRODUCCIÓN.

Las redes neuronales artificiales, como parte de la teoría de inteligencia artificial, han mostrado su impacto en diferentes áreas de la ciencia, donde se aplican a problemas de clasificación y predicción de patrones. Aunque esta técnica ha sido utilizada frecuentemente por muchos investigadores, no se tiene aun una teoría solida que permita identificar la estructura de una red neuronal de forma adecuada; por lo tanto la selección de esta estructura se busca por medio de pruebas y error, o en el mejor de los casos la aplicación de algunas reglas para configuración; además de jugar un papel importante la experiencia adquirida por el experto.

Se hace entonces de utilidad desarrollar un software que busque diferentes configuraciones y se ajuste para obtener soluciones sobre las cuales el experto analice y decida la opción que esté de acuerdo con sus necesidades; el problema se encuentra en que existen infinitas estructuras, por lo cual se hace necesario utilizar alguna técnica, diferente al de fuerza bruta, para encontrar de forma rápida esas soluciones.

Los algoritmo genéticos, junto con la programación evolutiva, son métodos de búsqueda en espacios muestrales de gran tamaño, teoría que hace parte de los algoritmos de búsqueda local y problemas de optimización de la inteligencia artificial.

El presente artículo tratará diferentes resultados de un proyecto investigativo, en el cual pretendemos por medio de técnicas de algoritmos genéticos, realizar una búsqueda optimizada sobre posibles soluciones de estructuras en redes neuronales artificiales, examinando valores en la cantidad de capas ocultas y neuronas en estas, junto con cambios de las funciones de activación.

2. PLANTEAMIENTO DEL PROBLEMA

El diseño de la estructura de redes neuronales, es un proceso tedioso, que requiere por parte del usuario un grado de experiencia de diseño/entrenamiento, junto con un proceso de prueba y error dependiente de la complejidad del problema. Este proceso no se limita al simple hecho de determinar el número de capas y neuronas ocultas que debe tener la red, sino que se involucran muchos más parámetros que se pueden modificar en aras de encontrar una red que produzca una mejor generalización.

La evolución de las redes neuronales puede ser clasificada de acuerdo a la meta a alcanzar. Algunos esquemas han propuesto la evolución de los pesos sinápticos, otros afirman que lo más importante es la arquitectura y otros acercamientos han incluido funciones de activación y reglas de aprendizaje. Sin embargo el área más interesante para nuevas investigaciones radica en la combinación de estos enfoques evolutivos[112].

En este caso se utilizaron combinadamente dos de las técnicas anteriormente mencionadas, la evolución simultanea de la arquitectura y las funciones de activación.

3. GENERALIZACIÓN DE LA RED.

La topología de una red neuronal es un aspecto fundamental para lograr una buena solución al problema, la densidad de neuronas y conexiones de una red determinan su habilidad para generalizar. Si la red no tiene suficientes neuronas conectadas por capa, el algoritmo de entrenamiento probablemente no realice una buena adaptación de los pesos haciendo que la red no pueda dar una buena solución. Todo lo contrario sucede si se tiene una red densamente conectada.

En este caso puede ocurrir un sobre-ajuste, en este caso la red en vez de aprender a aproximar la función, comienza a memorizar cada ejemplo de entrenamiento. En estos casos el ruido de los datos de entrenamiento se aprende, limitando la capacidad de generalizar, es decir de producir soluciones correctas para datos que no hayan sido parte del entrenamiento.

Existen también diferentes funciones utilizadas en las redes neuronales artificiales y estudios referentes a las características y aplicaciones en las cuales deben ser aplicadas. En la elección de las funciones de activación (pueden ser escogidas diferentes funciones en una sola red neuronal) deben de tenerse en cuenta la velocidad de entrenamiento y la convergencia en poco ciclos; la función debe ser fácilmente computable, tanto original como su derivada, además debe tener una amplia parte lineal[113].
Para el desarrollo del proyecto se han implementado técnicas para mejorar la generalización de la red como el uso de entrenamiento con detención temprana o la normalización de los datos para el ajuste de los datos al rango de las funciones de activación.

3.1 ENTRENAMIENTO CON DETENCIÓN TEMPRANA (*EARLY STOPPING*)

Lo que se desea es que la red se entrene bien de forma tal que aprenda lo suficiente acerca del pasado para generalizar en el futuro.

Teniendo como meta a una buena generalización, es muy difícil darse cuenta cuándo es el mejor momento de detener el entrenamiento si solamente estamos mirando a la curva de aprendizaje para el entrenamiento. En particular, como se mencionó anteriormente, es posible que la red termine sobre ajustándose a los datos de entrenamiento si la sesión de entrenamiento no se detiene en el momento correcto.

Aplicando lo aconsejado por el método de validación cruzada (crossvalidation)[114], el conjunto de datos que se escoja debe ser dividido en dos partes, una parte denominada conjunto de entrenamiento y otra llamada conjunto de validación.

Como su nombre lo dice, el conjunto de

---

[112] **D. Curran, C. O'Riordan,** Applying Evolutionary Computation to Designing Neural Networks: A Study of the Satate of the Art, technical report NUIG-IT-111002, National University of Ireland, Galway, 2002

[113] **L. Llano, A. Hoyos, F. Arias, J. Velásquez.** Comparación del desempeño de funciones de activación en redes Feedforward para aproximar funciones de datos con y sin ruido.

[114] **Stone M. (1974)** *Cross-validatory choice and assessment of statistical predictions*. Journal of the Royal Statistical Society, vol. B36, pp. 111-133.

entrenamiento es con el que la red supervisada realiza el aprendizaje, y el conjunto de validación es con el que se simula la red para ver el grado de generalización, entre más bajo sea el error en este punto mejor ha generalizado la red.

3.2 NORMALIZACIÓN.

De acuerdo con el rango de las funciones de activación utilizadas es conveniente escalar o transformar los datos de entrada para ajustarlos a dichos rangos. Comúnmente se utilizan: la función paso, para problemas de clasificación; la función lineal, en distintos tipos de redes frecuentemente en la capa de salida (funciones no lineales en la neurona de salida son comúnmente utilizadas en tareas de clasificación de patrones para restringir los valores de salida a rangos tales como [0,1] o [–1,1])[115].

Para el proyecto se realiza la normalización de los datos de entrada y salida para ajustarlos a las funciones de activación usadas que se obtienen por medio de los operadores genéticos.

4. ESPECIFICACIONES DEL ALGORITMO.

El primer problema que se enfrenta a la hora de utilizar un algoritmo genético, es el de encontrar una representación (genotipo) para simbolizar los parámetros de la red neuronal que se desean evolucionar. Para este caso se utiliza una codificación directa tanto para la arquitectura como para las funciones de activación. Estos cromosomas están representados de forma independiente, pero están directamente relacionados, para que haya convergencia la combinación de las dos codificaciones debe producir una buena solución.

Para la representación de la configuración de usa un gen que puede tomar valores desde uno hasta infinito (limitado por las condiciones definidas por el usuario), estos valores representan el número de neuronas que se encuentran en la capa oculta correspondiente; de igual manera el tamaño del gen de cada individuo está definido por el número máximo de capas tomado para realizar la búsqueda. Así de esta forma el individuo mostrado en la figura representaría una configuración de tres capas, cada una con tres, dos y cinco neuronas respectivamente, lo que un diseñador nombraría como una arquitectura: 3-2-5.

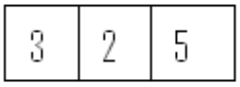

Figura 1. Individuo cualquiera de la población de configuraciones.

Para la representación de las funciones de activación se usa una representación similar, pero cada elemento del cromosoma indica una posición de la función de activación usada, los números del gen varían de acuerdo al número de funciones de activación implementadas en el algoritmo. De esta forma por ejemplo el cromosoma de la figura indicaría una configuración con funciones de activación tangente hiperbólica, lineal y gaussiana. El número de funciones de activación está determinado por la cantidad de capas usadas en el paso previo de generación de las capas ocultas, pues cada capa debe tener su función correspondiente y además agregar otra para la capa de salida.

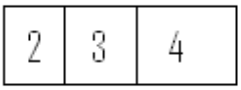

Figura 2. Individuo cualquiera de la población de funciones de activación.

En este punto esta se está listo para realizar todos los procedimientos de configuración evolutiva y entrenamiento de la red neuronal.

4.1 DISEÑO DEL ALGORITMO.

Después de tener la representación de la red se realiza el proceso de selección por medio de torneo. Esto consiste en que para la elección de los padres de la que se van a cruzar se eligen tres individuos al azar de la población, se toma aquel tenga el menor error como elemento a cruzarse. El cruce implementado es uniforme de un punto, esto consiste en elegir una posición al azar del cromosoma (determinado por el tamaño del gen de menor tamaño), y realizar un corte; después por ese punto se realiza un intercambio en la información de cada uno de los individuos seleccionados.

[115] **Hong-Choon**, and Wong Y. A Comparative Study on the Multi-layered Perceptrons with One and Two Hidden Layers. TS5B-3, M2USIC, 2004.

Un factor importante para garantizar la diversidad de la población es el operador de mutación, donde con una probabilidad determinada realiza la variación de un gen de los individuos obtenidos del cruce (hijos), seleccionados aleatoriamente.

Estando listos los pasos del ciclo genético se procede a realizar el entrenamiento con los algoritmos disponibles, el número de entrenamientos seleccionado debe ser considerable para garantizar que cada topología logre explotar todo su potencial antes de ser dada como evaluada.

Después de esto se deben seleccionar los individuos que van a formar parte de la población después del entrenamiento, este procedimiento se realiza de nuevo por torneo lo que garantiza una presión selectiva (elitismo); donde los mejores hijos reemplazarán a los individuos de la población inicial con peor desempeño.

Este se mide por la simulación de la red neuronal con un conjunto de datos definido y comparado con la salida esperada para ese conjunto; esta salida produce un error que es el considerado para guiar la evolución (error absoluto o relativo según se seleccione por el usuario). El procedimiento se repite durante el número de generaciones seleccionado.

5. METODOLOGÍA.

Para el desarrollo de las pruebas, se decidió tomar un modelo del cual se tuviera suficiente información como para determinar su comportamiento, y que a su vez no representara una solución demasiado sencilla en la búsqueda de la configuración. El problema considerado como función objetivo corresponde a la aproximación de la función:

$$F(x,y)= \sin(x)*\sin(y)/x*y \qquad (1)$$

Este problema nos brinda suficiente información para introducir al configurador, sin embargo no se debe realizar el entrenamiento con grandes volúmenes de datos, pues no se estaría poniendo a prueba la generalización de la red si se le suministran gran cantidad de ejemplos.

El conjunto de entrenamiento fue introducido a la red mediante cincuenta (50) ejemplos de la red, suficientes para mostrar un comportamiento del problema, pero no demasiados para que la red llegue memorizar todos los casos.

Una técnica utilizada para buscar una mejor generalización fue el entrenamiento con detención temprana (early stopping), para lo cual se definieron dos conjuntos para validación y prueba del comportamiento de la red, de veinte (20) y treinta (30) muestras respectivamente. Después se mostró el comportamiento de algunas de las configuraciones obtenidas, las cuales forman el conjunto solución, de donde el usuario debe decidir la que mejor se adapte a su problema.

Los parámetros introducidos para delimitar el espacio de búsqueda fueron:

Población inicial: 20 individuos.
Probabilidad de cruce: 80 %
Probabilidad de mutación: 10 %
Número máximo de capas: 4
Número máximo de neuronas por capa: 30
Numero de entrenamientos a cada configuración: 40
Algoritmo de entrenamiento: Levenberg-Maquardt
Número de generaciones del algoritmo genético: 30
Tolerancia al error (error absoluto): 0.1

Un aspecto clave para lograr buenos resultados en la búsqueda, es el considerar un número de entrenamientos adecuado para garantizar que el potencial de cada red sea explotado. Esto debido a la capacidad del algoritmo de memorizar las configuraciones que ya han sido evaluadas con el fin de no realizar entrenamientos sobre configuraciones que han aparecido más de una vez, buscando la reducción del tiempo de búsqueda (la mayoría del tiempo del algoritmo, es consumido en el entrenamiento de las configuraciones).

5.1 ANÁLISIS DE LAS CONFIGURACIONES.

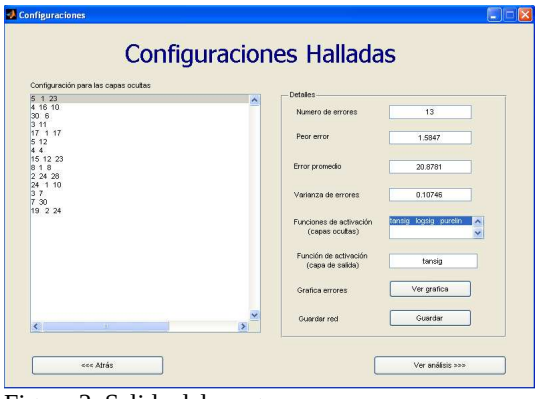

Figura 3. Salida del programa.

En la anterior imagen se muestran las configuraciones obtenidas, ordenadas de acuerdo al número de errores cometidos en la salida de la red, al ser evaluada con los datos de prueba, tomando como base la tolerancia al error escogida.
Se muestran algunos parámetros que pueden ayudar a la elección de una u otra configuración; tales como el error promedio cometido, la varianza de los errores en la salida, entre otros. Además se muestra la información de las funciones de activación usadas en cada una de las capas de la red obtenida, halladas igualmente mediante evolución.

A continuación se muestra el comportamiento de algunas de las configuraciones obtenidas.

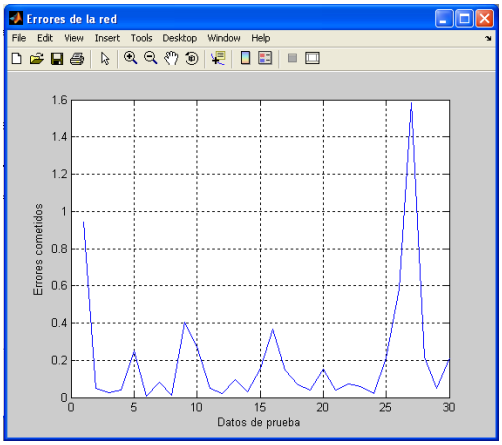

Figura 4. Errores configuración [5 1 23]

Funciones de activación capas ocultas: tansig, logsig, purelin
Función de activación capa de salida: tansig
Error promedio: 20.8781
Varianza de los errores: 0.10746

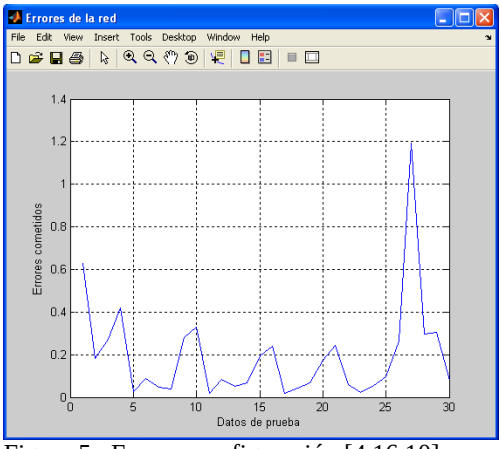

Figura 5. Errores configuración [4 16 10]

Funciones de activación capas ocultas: radbas, purelin, tansig
Función de activación capa de salida: tansig
Error promedio: 19.6253
Varianza de los errores: 0.055994

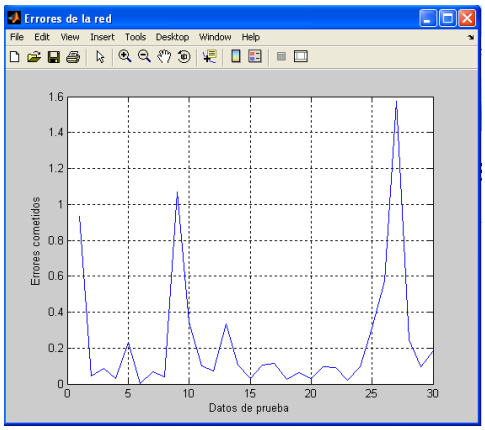

Figura 6. Errores configuración [30 6]

Funciones de activación capas ocultas: radbas, tansig
Función de activación capa de salida: tansig
Error promedio: 23.7177
Varianza de los errores: 0.12798

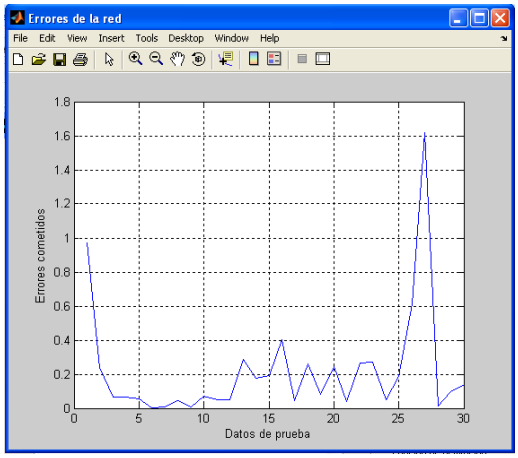

Figura 7. Errores configuración [3 11]

Funciones de activación capas ocultas: tansig, tansig
Función de activación capa de salida: tansig
Error promedio: 22.0794
Varianza de los errores: 0.11124

La variación general del problema también es mostrada para dar una idea del comportamiento de la búsqueda.

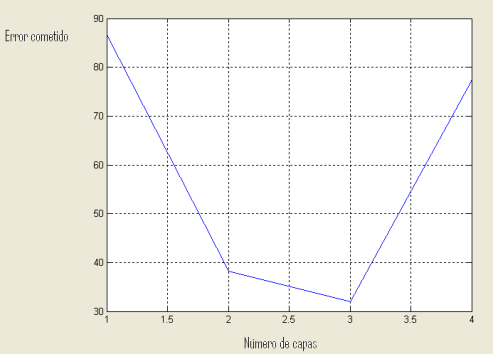

Figura 8. Error promedio (absoluto) contra número de capas ocultas

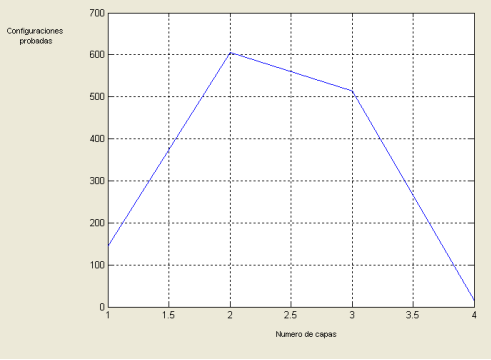

Figura 9. Configuraciones probadas contra número de capas ocultas.

Mediante las graficas de variación de la arquitectura de la red se puede visualizar el comportamiento que tiene el problema al agregar o remover una capa oculta o de igual manera al variar el número de neuronas por capa. Para esto cada una de los configuraciones es probada para determinar que configuración está usando para determinar un error promedio, de igual manera se muestran cuantas redes se han considerado con estas características.

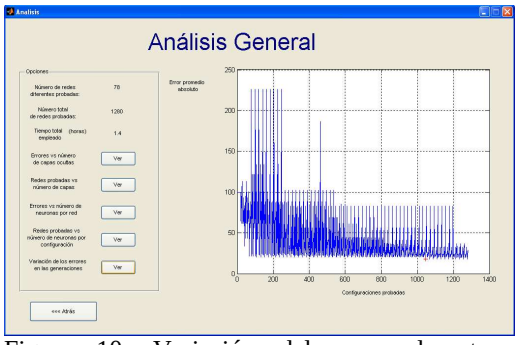

Figura 10. Variación del error durante el algoritmo.

En una visión más general de la salida se muestra la variación de los errores a través del curso del algoritmo genético. Como datos adicionales se muestra que el número total de redes consideradas fue de 1280 de las cuales 78 eran configuraciones diferentes, el tiempo empleado para el cálculo de los resultados fue de 1.4 horas.

Dada la cantidad de datos usados en el entrenamiento la red falla en algunas de las salidas, sin embargo los resultados obtenidos son aceptables de acuerdo a las condiciones del problema.

6. Conclusiones.

Mediante el análisis de los resultados mostrados se puede tener una base de conocimiento sobre la arquitectura de red que se podría usar para lograr una buena solución al problema, así mismo de las funciones de activación encontradas para las capas ocultas.

Si bien existen muchas más posibilidades para considerar en el espacio de búsqueda, la decisión de tomar más neuronas o capas ocultas depende del usuario; pero se debe tener claro que repercutirá directamente en el tiempo que se requerirá para obtener una solución.

Se debe tener en cuenta que si bien para espacios de búsqueda grandes el algoritmo requiere de gran cantidad de tiempo para mostrar una respuesta, este no se puede comparar con los que se requiere para hallar una configuración por métodos tradicionales; ya que al ser estos influidos en gran parte por el azar son siempre variables, e incluso pueden no llegar a una buena solución.

7. Bibliografía.

[1] Darwin, C. On the Origin of Species by Means of Natural Selection. John Murray, London, 1859.

[2] John Henry Holland. Adaptation in Natural and Artificial Systems: 2nd edition. MIT Press. 1992

[3] D. Curran, C. O'Riordan, Applying Evolutionary Computation to Designing Neural Networks: A Study of the Satate of the Art, technical report NUIG-IT-111002, National University of Ireland, Galway, 2002

[4] L. Llano, A. Hoyos, F. Arias, J. Velásquez. Comparación del desempeño de funciones de activación en redes Feedforward para aproximar funciones de datos con y sin ruido.

[5] Stone M. Cross-validatory choice and assessment of statistical predictions. Journal of the Royal Statistical Society, vol. B36, pp. 111-133, 1974

[6] Hong-Choon, and Wong Y. A Comparative Study on the Multi-layered Perceptrons with One and Two Hidden Layers. TS5B-3, M2USIC, 2004.