UNIVERSIDAD DE ORIENTE
NÚCLEO DE ANZOÁTEGUI
ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS
DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS

"DESARROLLO DE UNA APLICACIÓN QUE PERMITA EVOLUCIONAR REDES NEURONALES ARTIFICIALES PARA RESOLVER PROBLEMAS DE CLASIFICACIÓN"

REALIZADO POR:

Cordero Ch., Luis Antonio

Medina L., Enoes Carolina

Trabajo de grado presentado como requisito parcial para optar al Título de
INGENIERO EN COMPUTACIÓN

Barcelona, Marzo de 2009

UNIVERSIDAD DE ORIENTE
NÚCLEO DE ANZOÁTEGUI
ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS
DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS

"DESARROLLO DE UNA APLICACIÓN QUE PERMITA EVOLUCIONAR REDES NEURONALES ARTIFICIALES PARA RESOLVER PROBLEMAS DE CLASIFICACIÓN"

ASESOR:

_____

Lic. José Luis Bastardo. Msc.
Asesor Académico

Barcelona, Marzo de 2009

UNIVERSIDAD DE ORIENTE

NÚCLEO DE ANZOÁTEGUI

ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS

DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS

"DESARROLLO DE UNA APLICACIÓN QUE PERMITA EVOLUCIONAR REDES NEURONALES ARTIFICIALES PARA RESOLVER PROBLEMAS DE CLASIFICACIÓN"

EXCELENTE

JURADO CALIFICADOR:

_____

Lic. José Luis Bastardo. Msc.
Asesor Académico

_____ _____

Profesor Claudio Cortínez
Jurado Principal

Profesor Luis Rojas
Jurado Principal

Barcelona, Marzo de 2009

RESOLUCIÓN

ARTÍCULO Nº 44 Del Reglamento de Trabajo de Grado

"Los trabajos de grado son de exclusiva propiedad de la Universidad y sólo podrán ser utilizados para otros fines con el conocimiento del Consejo de Núcleo respectivo, quién lo participará al Consejo Universitario".

RESUMEN

En el presente trabajo se desarrolló una Aplicación que permitiera evolucionar Redes Neuronales Artificiales para resolver problemas de clasificación. El software desarrollado le brinda al usuario una solución expresada como una Red Neuronal Artificial que puede ser obtenida de dos formas: Algoritmo Genético y Algoritmo Backpropagation; el usuario puede elegir que método usar. Esta aplicación se realizó utilizando tecnologías de software libre, rigiéndose por el decreto 3390, el cual dispone que "La Administración Pública Nacional deberá emplear prioritariamente Software Libre desarrollado con Estándares Abiertos, en todos sus Sistemas, Proyectos y Servicios Informáticos". El desarrollo del proyecto estuvo dirigido por la metodología OMT (*Object Modeling Technique*), la cual utiliza el análisis y el diseño orientado a objetos por medio del Lenguaje de Modelado UML. Para construir el software se utilizó JAVA como lenguaje de programación.

AGRADECIMIENTOS

A Dios por haberme dado la confianza y paciencia para llegar a esta fase de mi carrera. A mis padres Enoes López y Jesús Medina por su apoyo y amor incondicional, para que yo pudiera sacar mi carrera, a mi hermana Mercedes, mis hermanos Jesús Francisco, Alejando y Julio, mi cuñada Naife por darme apoyo en todo momento. A mis sobrinos Arístides, Naife, Arianna y Aurora por comprender que el tiempo que no pude compartir con ellos era por culminar esta meta.

A Luis Cordero por tenerme paciencia, tolerarme y ayudarme durante la mitad de mi carrera y también por ayudarme a cumplir esta meta.

Al profesor José Luis Bastardo por ser mi asesor y guía durante el desarrollo de este proyecto. A la profesora Liliana Varrone por prestarme su apoyo tanto como amiga como profesora durante la realización de este proyecto, aportando una pequeña pero importante información para poder llevar a cabo este proyecto.

A mi amiga Maribel Mayattis por ofrecerme su amistad desde inicios de mi carrera y prestarme su ayuda y apoyo incondicionales cuando lo necesite. A Luis Martínez, por su paciencia, apoyo y amistad brindados en todo momento. A Eliannys Lugo quien me brindo su amistad desde el colegio hasta la universidad y siempre estuvo ahí en todo momento.

A mis amigas Joselin González, Romaira Brito, mis amigos Roger Garban, José Hernández, Gazi El Halabi, Osmar Lunar, Tarik Saab, Jóvito Aranguren, Ronel Zapata, Luis Farías, Israel Pereira, Andrés Segura, David Suniaga, Franco Flores quienes estuvieron conmigo en todo momento durante mi carrera.

Enoes Carolina Medina López.

AGRADECIMIENTOS

A Dios, la Virgen María y a todos los seres de luz que estuvieron conmigo a largo de mi carrera.

A mis padres José Cordero, Nereyda Chang, por su apoyo incondicional en todo momento de mi vida y carrera.

A mi hermano José Cordero Chang, por su ayuda y por ser mi norte y mi ejemplo a seguir en la vida.

A mi cuñada Wendy La Rosa, por ser como otra hermana más y su amistad incondicional en todo momento desde que entró a nuestra familia.

A mi hermana Neyaluisa Cordero Chang, y mis sobrinos Gabriel Cordero y Gianfranco Landi, por los momentos gratos que me han dado.

A mi asesor José Luis Bastardo, el cual, nos guió en mis últimos pasos de la carrera.

A Enoes Carolina Medina López, ya que sin ella no hubiese podido culminar este proyecto que iniciamos juntos.

A Luis Martínez Mendoza, por ser un amigo, compañero y un hermano más desde mis primeros pasos en la universidad.

A Ronel José Zapata, por su amistad, compañerismo y su gran ayuda aportada a este proyecto.

A Roger Luis Garban, por ser un buen amigo y compañero incondicional en todo momento.

A la profesora Liliana Varrone, por su ayuda en el inicio de este proyecto, ya que sus consejos fueron de gran ayuda.

A José Hernández, por ser un compañero y amigo cuyos conocimientos de la vida me han ayudado mucho.

A Luis Farías, ya que sin su ayuda en las últimas materias no hubiese podido culminar esa etapa de la carrera.

A Israel Pereira, por su apoyo que fue de gran ayuda en los últimos momentos de la carrera.

A mis amigos Lenin López, José Sandoval, Richard Fernández, por su paciencia hacia mí en las últimas materias.

A mis amigas Eliannys Lugo, Aivett Bilbao, Ángeles Manzo, Magaly Manzo, por ser excelentes personas conmigo desde el inicio de nuestra amistad.

Al profesor Claudio Cortínez, por su asesoría dada con los diagramas realizados en este trabajo.

A Maribel Mayattis, por ayudarme en las materias más difíciles de la carrera y por su amistad incondicional

A Franco Flores, por ser una persona de gran paciencia conmigo durante toda la carrera.

A Jóvito Aranguren, por su amistad y su apoyo dado en momentos críticos de mi carrera.

A mis compañeros y amigos Tarik Saab, Osmar Lunar, Gazi El Halabi e Irene Andrade, por esos grandes momentos que compartimos durante la carrera.

A mis primos Alberto Ruíz Chang y Carlos Ruiz Chang, por su ejemplo y ayuda dada en todo momento.

A mis familiares Adriana Ruíz, Rubén Ruíz, Carlos Cordero, Amarelis Cordero, Elida Cordero, Delfín Francisco Madriz, Mariela Hernández, Yolicia Aguilera, Liberia Navas, por sus consejos, ayuda y todos los momentos gratos compartidos.

A todos mis familiares y compañeros que de una u otra forma me han brindado su apoyo incondicional a lo largo de mi vida.

Luis Antonio Cordero Chang

LISTA DE FIGURAS

LISTA DE TABLAS

CAPITULO 1: PLANTEAMIENTO DEL PROBLEMA

1.1 INTRODUCCIÓN

La Inteligencia Artificial en los últimos años ha tenido un gran auge, principalmente en el área de Redes Neuronales Artificiales, donde los ingenieros y programadores se esfuerzan por crear sistemas inteligentes similares en funcionamiento a las redes neuronales biológicas.

Los sistemas de Redes Neuronales Artificiales aún cuando están basados en los sistemas neurales biológicos del cerebro, resultan incompletos ya que los sistemas neurales biológicos son infinitamente complejos.

Las Redes Neuronales Artificiales son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los animales.

Las Redes Neuronales al experimentar cambios entre sus conexiones pueden aprender, estos cambios son conocidos como proceso de entrenamiento, el cual se guía por una regla de aprendizaje, donde la experiencia es la que impera en las modificaciones que se realicen.

Las Redes Neuronales son aplicadas con éxito en problemas donde la imprecisión no puede ser ignorada. Entre los problemas que se benefician de esta característica están los de reconocimiento, los de clasificación y los de predicción. **[25]**

En un problema de clasificación la finalidad es dividir un conjunto de objetos en grupos o clases de forma que los perfiles de los objetos dentro de una misma clase sean similares entre sí. Durante la fase de entrenamiento se debe determinar una relación que permita clasificar correctamente cada uno de los *n* vectores de un conjunto de entrenamiento T.

Cada vector X Є T pertenece a una de las *m* clases $C_1,C_2,\ldots,C_m$; la relación sintetizada mediante el entrenamiento de la Red Neuronal debe permitir la clasificación de vectores no pertenecientes a T. De esta manera la relación establecida permite la clasificación de nuevos vectores. La capacidad de clasificar vectores nunca antes vista es conocida como capacidad de generalización.

En los últimos años, la comunidad científica internacional ha mostrado un creciente interés en una nueva técnica de búsqueda basada en la teoría de la evolución y que se conoce como Algoritmo Genético. Estos son una variedad de los llamados algoritmos evolutivos y por lo tanto, una clase dentro de la rama de la Inteligencia Artificial que se conoce con el nombre de Computación Evolutiva.

El concepto de Algoritmo Genético describe un conjunto de sistemas para resolver problemas que usan un modelo computacional similar a los procesos evolutivos de la naturaleza, es decir, un modelo basado en el principio darwiniano de reproducción y supervivencia de los individuos que mejor se adapten al entorno donde viven.

Los Algoritmos Genéticos generan un conjunto de hipótesis mediante la mutación y recombinación de parte del conjunto de hipótesis conocido. En cada paso el conjunto de hipótesis conocido como "población actual" se renueva remplazando una proporción de esta población por los sucesores más adecuados mediante el uso de una función de evaluación. **[10]**

1.2 EL PROBLEMA

Se tiene un archivo de texto con coordenadas x e y de puntos de dos clases diferentes las cuales se deben dividir por medio de Redes Neuronales Artificiales que tienen que aprender y evolucionar hasta lograr una división de estas clases. Esta Red Neuronal se formará por los datos leídos en el archivo de entrada, los pesos iniciales son creados de manera aleatoria, esta red está formada por tres capas, la primera es la capa de entrada la cual tiene dos neuromas, una capa oculta la cual posee n cantidad de neuronas y por último la capa de salida con una sóla neurona.

El aprendizaje de la Red Neuronal se realiza por medio de patrones de entrada donde se modifican los pesos de las conexiones entre neuronas en función de la diferencia de los valores deseados y los obtenidos, es decir, en función del error cometido por la red. Los pesos se modifican por medio de los métodos de cruce y mutación del Algoritmo Genético, luego de encontrar una arquitectura a esta red se le aplicara Backpropagation para ajustar los pesos.

Con el fin de verificar la capacidad de generalización de la red obtenida, el conjunto de entrenamiento T es construido al tomar puntos de una plantilla bidimensional que contiene un patrón donde dos colores (clases) se reparten el área total de la plantilla.

Luego de encontrar una Red Neuronal el usuario podrá comprobar que esta funcione por medio de la generalización, donde el usuario debe cargar la imagen inicial o plantilla la cual representa al problema, el usuario puede elegir si desea ver la generalización del Algoritmo Genético solo o con Backpropagation.

Durante el proceso de generalización la Red Neuronal clasifica todos los puntos de la plantilla. Una buena capacidad de generalización corresponde a una buena reproducción de la plantilla original.

El problema planteado también puede ser resuelto por medio de Backpropagation sin Algoritmo Genético, en este caso el usuario tendrá que introducir la cantidad de neuronas de la capa oculta de la red, los pesos iniciales son aleatorios. Esta solución también puede ser comprobada por medio de la generalización.

1.3 EL PROPÓSITO

El propósito de este proyecto es desarrollar una aplicación capaz de encontrar una solución al problema planteado. Se desea evolucionar la Red Neuronal hasta dar con la solución que sintetice la relación que permite la clasificación de vectores en el mismo espacio de los vectores en T.

Ya que los Algoritmos Genéticos son excelentes en la búsqueda en un espacio de estados, utilizarlos para la búsqueda de los pesos de una Red Neuronal es una aplicación ideal.

Se puede pedir a un Algoritmo Genético que busque y encuentre cómo establecer la arquitectura de la Red y/o los valores de los parámetros de la misma. De esta forma, el proceso de elección del detalle de una Red Neuronal para resolver un problema se hace menos heurístico.

Se han aplicado Algoritmos Genéticos a las Redes Neuronales de diferentes maneras, entrenándolas ajustando la conexión y los pesos, también para explorar la

arquitectura para encontrar el mejor número de unidades ocultas para un problema particular e incluso para seleccionar que modelo de Red Neuronal debe usarse.

1.4 IMPORTANCIA

Académicamente, éste proyecto servirá como base a otros proyectos de desarrollo de software, orientados a la utilización de herramientas de Redes Neuronales Artificiales y Algoritmos Genéticos, siguiendo un proceso sistematizado y organizado para su análisis y desarrollo.

1.5 ALCANCE

Se abordará la evolución de Redes Neuronales para resolver problemas de clasificación en un espacio bidimensional con dos clases.

En el desarrollo de este estudio, se especificarán los requisitos del sistema y sus riesgos; se realizarán los diagramas pertinentes, al igual que la arquitectura del sistema propuesto, se diseñará y construirá una interfaz accesible, interactiva, amigable y de fácil manejo para los usuarios.

1.6 ORIGINALIDAD

Este proyecto es el primero en el Departamento de Computación y Sistemas en utilizar Algoritmos Genéticos para la evolución de Redes Neuronales Artificiales para resolver un problema en específico. Representa un punto de partida para una línea de investigación dentro del Departamento de Computación y Sistemas.

1.7 OBJETIVOS

1.7.1 Objetivo General

Construir una aplicación que permita la evolución de Redes Neuronales Artificiales para la resolución problemas de clasificación.

1.7.2 Objetivos Específicos

- Establecer la representación de las Redes Neuronales.
- Diseñar el Algoritmo Genético para la evolución de las Redes Neuronales Artificiales.
- Codificar el Algoritmo Genético en un lenguaje de programación adecuado.
- Ajustar el programa para que se establezcan los parámetros que garanticen un buen rendimiento del Algoritmo Genético.
- Probar el programa para que se adquieran estadísticas de rendimiento.

CAPITULO 2: MARCO TEÓRICO

2.1 ANTECEDENTES

Los proyectos usados como antecedentes están basados en Redes Neuronales Artificiales y Algoritmos Genéticos.

- Para el año 1994, la estudiante Melisendra Trejo Padrino de la Universidad de Oriente Núcleo Anzoátegui, desarrolló el proyecto titulado **"Diseño e Implantación de un Simulador de una Red Neuronal Backpropagation"**, para optar al título de Ingeniero en Computación. Este proyecto se realizó con el propósito de simular e implementar el algoritmo Backpropagation, puede simular redes de hasta cinco niveles y utilizar una de tres funciones de activación (threshold, sigmoide y tangente hiperbólica). Fue desarrollado en el Lenguaje de Programación C. **[24]**

  Para el año 1997, los estudiantes Sarbjit Singh Kaur y Franklin Ysaccis Betancourt de la Universidad de Oriente Núcleo Anzoátegui, desarrollaron el proyecto titulado **"Simulador de Redes Neurales Competitivas con Aprendizaje no Supervisado del Tipo SOM y ART1"** para optar al título de Ingeniero en Computación. Este proyecto se realizó con el propósito de modelar Redes Neuronales competitivas de aprendizaje no supervisado del tipo SOM (Self Organizing Map) y ART1 (Adaptative Resonance Theory), con la finalidad de evaluar su aplicación en la solución de problemas de categorización o agrupamiento. El aspecto más resaltante de este simulador es un Asistente para Redes Neuronales; este asistente guía al usuario a través del

diseño de la red, haciendo sugerencias durante el proceso y encapsulando muchos detalles. El simulador fue diseñado para operar en ambiente MS-Windows con código de 32 bits y fue desarrollado bajo una arquitectura Orientada a Objetos y codificado en el Lenguaje de Programación C++. **[23]**

- Para el año 2001, los estudiantes Rosa Lérida Lunar Fernández y Antonio Da Silva Moreira de la Universidad de Oriente Núcleo Anzoátegui, desarrollaron el proyecto titulado **"Diseño de un Sistema Basado en Redes Neuronales para el Control de un Vehículo"** para optar al título de Ingeniero en Computación. Este proyecto se realizó con el propósito de diseñar un sistema que permita el control de un vehículo basado en Redes Neuronales. El sistema consta de dos subsistemas: uno en bajo nivel encargado de controlar la circuitería o hardware del vehículo y otro en alto nivel que lleva el control basado en el algoritmo Backpropagation de Redes Neuronales. Este proyecto utilizó para el subsistema de control del hardware el Lenguaje Ensamblador y para el subsistema de control neuronal los diagramas de Lenguaje Unificado para el Modelado de objeto (UML), y se define con una Programación Orientada a Objetos con el Lenguaje de Programación Borland C++.**[11]**

- Para el año 2003, los estudiantes Víctor Manuel Díaz y Luis Augusto Herrera Guevara de la Universidad de Oriente Núcleo Anzoátegui, desarrollaron el proyecto titulado **"Desarrollo de un Algoritmo Genético para Solucionar Problemas de Programación Lineal"**, para optar al título de Ingeniero en Computación. Este proyecto se realizó con el propósito de solucionar problemas de Programación Lineal implementando Algoritmos Genéticos, a través de la herramienta el usuario ingresa la función lineal con sus variables y el tipo de función, también puede configurar el Algoritmo Genético, estableciendo la cantidad de individuos de la población, el número máximo de

generaciones por la que va a pasar el Algoritmo, el tamaño de la élite, la población de cruce, tipo de cruce, mutación, tipo de selección y tipo de reemplazo. El análisis y diseño de la aplicación fue realizado utilizando lenguaje gráfico UML (Unified Modeling Language), y la herramienta de cuarta generación Borland C++ Builder. **[6]**

2.2 REDES NEURONALES

2.2.1 Historia

Las primeras explicaciones teóricas sobre el cerebro y el pensamiento fueron dadas por algunos filósofos griegos, como Platón y Aristóteles, quienes fueron apoyados después por Descartes y filósofos empiristas. **[4]**

Alan Turing, en 1936, fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación, pero quienes primero concibieron algunos fundamentos de la computación neuronal fueron Warren McCulloch y Walter Pitts, Estos dos investigadores propusieron un modelo matemático de neurona. En este modelo cada neurona estaba dotada de un conjunto de entradas y salidas. Cada entrada está afectada por un peso. La activación de la neurona se calcula mediante la suma de los productos de cada entrada y la salida es una función de esta activación**. [4]**

Después otras teorías iniciales fueron expuestas por Donald Hebb, quien en 1949 definió dos conceptos muy importantes y fundamentales que han pesado en el campo de las redes neuronales: **[4]**

- El aprendizaje se localiza en las sinapsis o conexiones entre las neuronas.

• La información se representan en el cerebro mediante un conjunto de neuronas activas o inactivas.

Pero solo hasta 1957 Frank Rosenblatt comenzó el desarrollo del Perceptrón, esta es la red neuronal más antigua; utilizándose hoy en día para aplicación como identificador de patrones. Este modelo era capaz de generalizar, es decir, después de haber aprendido una serie de patrones podía reconocer otros similares, aunque no se le hubiesen presentado en el entrenamiento. **[4]**

Más adelante en 1959 Widrow desarrolló la Teoría sobre la Adaptación Neuronal y el Adaline (Adaptative Linear Neuron) y el Madaline (Multiple Adaline). Es la primera aplicación de las redes a problemas reales: filtros adaptativos para eliminar ecos en las líneas telefónicas. **[4]**

Stephen Grossberg en 1967 realizó una red: **Avalancha**, que consistía en elementos discretos con actividad que varía en el tiempo que satisface ecuaciones diferenciales continuas, para resolver actividades como reconocimiento continuo de habla y aprendizaje de los brazos de un robot. **[4]**

Hasta 1982 el crecimiento se frenó pero surgieron luego investigaciones sobre redes como la de Marvin Minsky y Seymour Papert, después James Anderson desarrolló el Asociador Lineal, en Japón Kunihiko Fukushimika y Teuvo Kohonen que se centraron en Redes Neuronales para el reconocimiento de patrones; en USA John Hopfield también realizó importantes investigaciones. **[4]**

2.2.2 Definición

Las Redes Neuronales Artificiales son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los animales. Se trata de un sistema de interconexión de neuronas artificiales en una red que colabora para producir un estímulo de salida como se puede apreciar en la figura 2.1**. [7]**

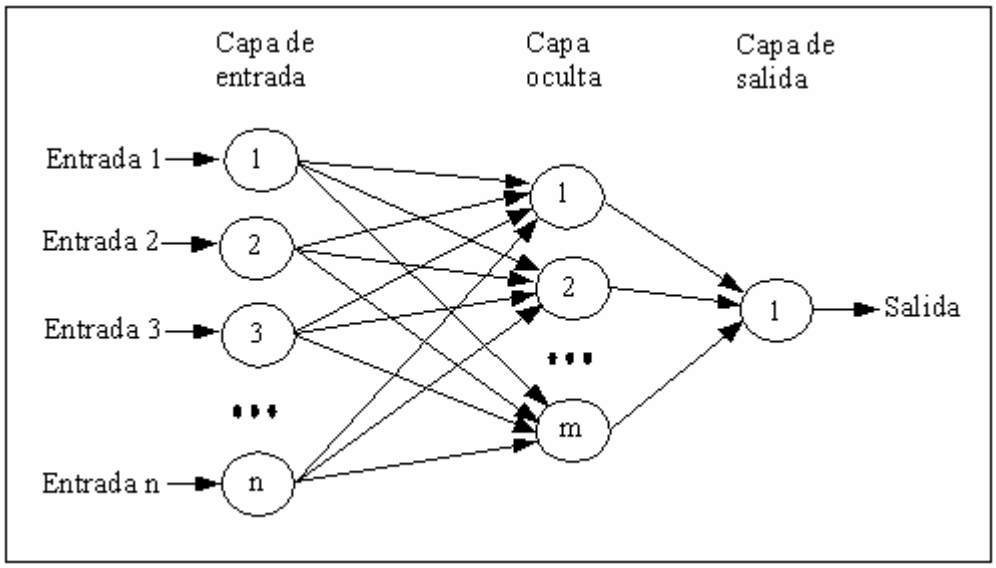


Figura 2.1. Red Neuronal Artificial perceptrón multicapa con n neuronas de entrada, m neuronas en su capa oculta y una neurona de salida**. [7]**

Las Redes Neuronales Artificiales son modelos que intentan reproducir el comportamiento del cerebro. Como tal modelo, realiza una simplificación, averiguando cuáles son los elementos relevantes del sistema, porque la cantidad de información de que se dispone es excesiva o porque es redundante. Una elección adecuada de sus características, más una estructura conveniente, es el procedimiento convencional utilizado para construir redes capaces de realizar determinada tarea**. [4]**

2.2.3 Unidad de proceso: La Neurona Artificial.

La neurona artificial es un elemento que posee un estado interno, llamado nivel de activación, y recibe señales que le permiten cambiar de estado. Este nivel de activación depende de las entradas recibidas y de los valores sinápticos, pero no de valores anteriores de estado de activación. **[7]**

Las neuronas poseen una función que les permite cambiar de nivel de activación a partir de señales que reciben provenientes del exterior o de las neuronas a las cuales están conectadas; a dicha función se le denomina función de transición de estado o función de activación. **[7]**

2.2.4 Funcionamiento

Una de las misiones en una Red Neuronal consiste en simular las propiedades observadas en los sistemas neuronales biológicos a través de modelos matemáticos recreados mediante mecanismos artificiales (como un circuito integrado, un ordenador o un conjunto de válvulas). El objetivo es conseguir que las máquinas den respuestas similares a las que es capaz de dar el cerebro, estas respuestas se caracterizan por su generalización y su robustez. **[28]**

Una Red Neuronal se compone de unidades llamadas neuronas. Cada neurona recibe una serie de entradas a través de interconexiones y emite una salida. Esta salida viene dada por tres funciones:

1. **Una función de propagación** (también conocida como función de excitación), que por lo general consiste en el sumatorio de cada entrada multiplicada por el peso de su interconexión (valor neto). Si el peso es

positivo, la conexión se denomina excitatoria; si es negativo, se denomina inhibitoria. **[28]**

2. **Una función de activación**, Una neurona biológica puede estar activa (excitada) o inactiva (no excitada); es decir, que tiene un estado de activación. Las neuronas artificiales también tienen diferentes estados de activación; algunas de ellas solamente dos, al igual que las biológicas, pero otras pueden tomar cualquier valor dentro de un conjunto determinado. **[2]**

   La función activación calcula el estado de actividad de una neurona; transformando la entrada global (menos el umbral) en un valor (estado) de activación, cuyo rango normalmente va de (0 a 1) o de (−1 a 1). Esto es así, porque una neurona puede estar totalmente inactiva (0 o −1) o activa (1). **[2]**

3. **Una función de transferencia**, que se aplica al valor devuelto por la función de activación. Se utiliza para acotar la salida de la neurona y generalmente viene dada por la interpretación que queramos darle a dichas salidas. Algunas de las más utilizadas son la función sigmoide (para obtener valores en el intervalo [0,1]) y la tangente hiperbólica (para obtener valores en el intervalo [-1,1]). **[28]**

2.2.5 Conexiones entre Neuronas

Existen diferentes formas en las cuales las neuronas de una Red Neuronal pueden estar conectadas, las cuales son:

- **Unión Todos con Todos:** Consiste en unir cada neurona de una capa con todas las neuronas de la otra capa, como se puede observar en la figura 2.2.

Este tipo de conexionado es el más usado en las redes neuronales, se usa en todo tipo de uniones desde el Perceptrón Multicapa a las redes de Hopfield o BAM. **[2]**

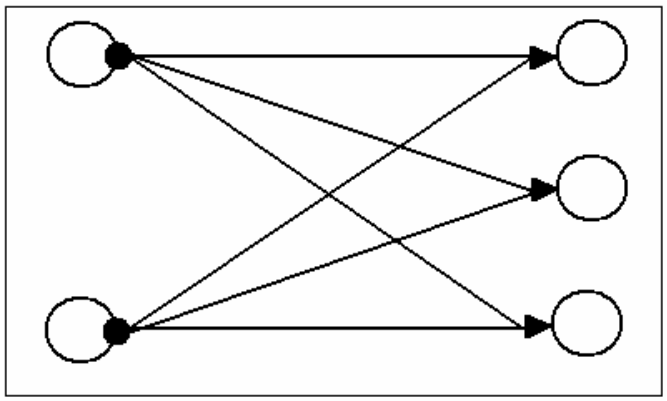


Figura 2.2 Ejemplo de la Unión Todos con Todos **[2]**

- **Unión Lineal:** Consiste en unir cada neurona con otra neurona de la otra capa, un ejemplo de esto se puede observar en la figura 2.3. Este tipo de unión se usa menos que el anterior y suele usarse para unir la capa de entrada con la capa de procesamiento, si la capa de entrada se usa como sensor. También se usa en algunas redes de aprendizaje competitivo. **[2]**

Figura 2.3 Ejemplo de la Unión Lineal **[2]**

- **Predeterminado:** Este tipo de conexión aparece en redes que tienen la propiedad de agregar o eliminar neuronas de sus capas y de eliminar también conexiones, un ejemplo sería el de la figura 2.4. **[2]**

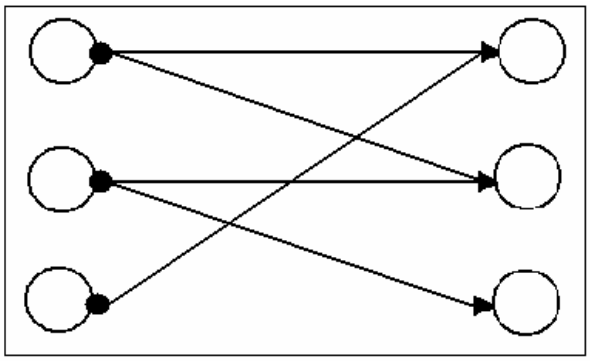


Figura 2.4 Ejemplo de Predeterminado **[2]**

Si establecemos un orden en las capas podemos establecer conexiones hacia delante, hacia atrás o conexiones laterales. Esto sirve para clasificar las redes en feedforward o hacia delante, no tienen ninguna conexión hacia atrás, y feedback o hacia atrás que si permiten este tipo de conexiones.

Las conexiones laterales son conexiones entre neuronas de la misma capa, este tipo de conexión son muy comunes en las redes mono capa. Si la red admite que las neuronas tengan conexiones a sí mismas se dice que la red es auto recurrente. **[2]**

Las conexiones que unen a las neuronas que forman una Rede Neuronal Artificial tiene asociado un peso, que es el que hace que la red adquiera conocimiento. Se considera que el efecto de cada señal es aditivo, de tal forma que la entrada neta que recibe una neurona es la suma del producto de cada señal individual por el valor de la sinapsis que conecta ambas neuronas y es lo que se conoce como red de propagación.

Se utiliza una matriz W con todos los pesos, si w_{ji} es positivo indica que la relación entre las neuronas es excitatoria, es decir, siempre que la neurona i esté activada, la neurona j recibirá una señal que tenderá a activarla. Si w_{ji} es negativo, la sinapsis será inhibidora. En este caso si i está activada, enviará una señal que desactivará a j. Finalmente si w_{ji} es 0 se supone que no hay conexión entre las neuronas. **[4]**

2.2.6 Estructura de la Red Neuronal Artificial

Existen tres tipos de unidades en cualquier sistema: entradas, salidas y ocultas, como se observa en la figura 2.1, las unidades de entrada reciben señales desde el entorno; las de salida envían la señal fuera de la red, y las unidades ocultas son aquellas cuyas entradas y salidas se encuentran dentro del sistema. **[4]**

Se conoce como capa o nivel al conjunto de neuronas cuyas entradas provienen de una misma fuente, ya sea del exterior o de otra capa de neuronas, y cuyas salidas se dirigen al mismo destino, también puede ser otra capa de neuronas o la capa de salida. **[4]**

- **Capa de Entrada**: recibe señales desde el entorno, que pueden provenir desde sensores o de otros sectores del sistema.
- **Capa de Salida**: envía la señal fuera del sistema.
- **Capas Ocultas**: son aquellas donde la entrada y la salida están dentro del sistema, es decir no tienen contacto con el exterior.

Cuando ninguna salida de las neuronas es entrada de neuronas del mismo nivel o de niveles precedentes, la red se describe como de propagación hacia adelante.

Cuando las salidas pueden ser conectadas como entradas de neuronas de niveles previos o del mismo nivel, incluyéndose ellas mismas, la red es de propagación hacia atrás. **[4]**

2.2.7 Características de las Redes Neuronales

Existen cuatro aspectos que caracterizan una Red Neuronal: su topología, el mecanismo de aprendizaje, tipo de asociación realizada entre la información de entrada y salida, y la forma de representación de estas informaciones. **[2]**

2.2.7.1 Topología de las Redes Neuronales.

La arquitectura de las redes neuronales consiste en la organización y disposición de las neuronas formando capas más o menos alejadas de la entrada y salida de la red. En este sentido, los parámetros fundamentales de la red son: el número de capas, el número de neuronas por capa, el grado de conectividad y el tipo de conexiones entre neuronas.

- **Redes Monocapa**: Se establecen conexiones laterales, cruzadas o auto recurrentes entre las neuronas que pertenecen a la única capa que constituye la red. Se utilizan en tareas relacionadas con lo que se conoce como auto asociación; por ejemplo, para generar informaciones de entrada que se presentan a la red incompleta o distorsionada.

- **Redes Multicapa**: Son aquellas que disponen de conjuntos de neuronas agrupadas en varios niveles o capas. Una forma de distinguir la capa a la que pertenece la neurona, consiste en fijarse en el origen de las señales que recibe a la entrada y el destino de la señal de salida. Según el tipo de conexión, como

se vió previamente, se distinguen las redes feedforward, y las redes feedforward/feedback. **[2]**

2.2.7.2 Mecanismo de Aprendizaje.

El aprendizaje es el proceso por el cual una red neuronal modifica sus pesos en respuesta a una información de entrada. Los cambios que se producen durante el proceso de aprendizaje se reducen a la destrucción, modificación y creación de conexiones entre las neuronas, la creación de una nueva conexión implica que el peso de la misma pasa a tener un valor distinto de cero, una conexión se destruye cuando su peso pasa a ser cero. Se puede afirmar que el proceso de aprendizaje ha finalizado (la red ha aprendido) cuando los valores de los pesos permanecen estables ($dw_{ij} / dt = 0$). **[4]**

Un criterio para diferenciar las reglas de aprendizaje se basa en considerar si la red puede aprender durante su funcionamiento habitual, o si el aprendizaje supone la desconexión de la red.

Otro criterio suele considerar dos tipos de reglas de aprendizaje: aprendizaje supervisado y aprendizaje no supervisado. La diferencia fundamental entre ambos tipos estriba en la existencia o no de un agente externo (supervisor) que controle el aprendizaje de la red. **[4]**

- **Redes con Aprendizaje Supervisado.** El proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo (supervisor, maestro) que determina la respuesta que debería generar la red a partir de una entrada determinada. El supervisor comprueba la salida de la red y en el caso de que ésta no coincida con la deseada, se procederá a modificar los pesos de las conexiones, con el fin de conseguir que la salida se aproxime a la deseada.

Se consideran tres formas de llevar a cabo este tipo de aprendizaje:

- **Aprendizaje por corrección de error**: Consiste en ajustar los pesos en función de la diferencia entre los valores deseados y los obtenidos en la salida de la red; es decir, en función del error.

- **Aprendizaje por refuerzo**: Se basa en la idea de no indicar durante el entrenamiento exactamente la salida que se desea que proporcione la red ante una determinada entrada. La función del supervisor se reduce a indicar mediante una señal de refuerzo si la salida obtenida en la red se ajusta a la deseada (éxito=+1 o fracaso=-1), y en función de ello se ajustan los pesos basándose en un mecanismo de probabilidades.

- **Aprendizaje estocástico**: Este tipo de aprendizaje consiste básicamente en realizar cambios aleatorios en los valores de los pesos de las conexiones de la red y evaluar su efecto a partir del objetivo deseado y de distribuciones de probabilidad. **[4]**

- **Redes con Aprendizaje No Supervisado**. Estas redes no requieren influencia externa a la Red Neuronal Artificial para ajustar los pesos de las conexiones entre neuronas. La red no recibe ninguna información por parte del entorno que le indique si la salida generada es o no correcta, así que existen varias posibilidades en cuanto a la interpretación de la salida de estas redes.

  En algunos casos, la salida representa el grado de familiaridad o similitud entre la información que se le está presentando en la entrada y las informaciones que se le han mostrado en el pasado. En otro caso podría realizar una codificación de los datos de entrada, generando a la salida una versión codificada de la entrada, con menos bits, pero manteniendo la

información relevante de los datos, o algunas redes con aprendizaje no supervisado lo que realizan es un mapeo de características, obteniéndose en las neuronas de salida una disposición geométrica que representa un mapa topográfico de las características de los datos de entrada, de tal forma que si se presentan a la red informaciones similares, siempre sean afectadas neuronas de salidas próximas entre sí, en la misma zona del mapa.

En general en este tipo de aprendizaje se suelen considerar dos tipos:

- **Aprendizaje Hebbiano**: Consiste básicamente en el ajuste de los pesos de las conexiones de acuerdo con la correlación, así si las dos unidades son activas (positivas), se produce un reforzamiento de la conexión. Por el contrario cuando una es activa y la otra pasiva (negativa), se produce un debilitamiento de la conexión.

- **Aprendizaje competitivo y cooperativo**: Las neuronas compiten (y cooperan) unas con otras con el fin de llevar a cabo una tarea dada. Con este tipo de aprendizaje se pretende que cuando se presente a la red cierta información de entrada, solo una de las neuronas de salida se active (alcance su valor de respuesta máximo). Por tanto las neuronas compiten por activarse, quedando finalmente una, o una por grupo, como neurona vencedora. **[4]**

2.2.8 Tipos de Redes Neuronales Artificiales

2.2.8.1 Red Hopfield.

La Red Hopfield es recurrente y completamente interconectada. Funciona como una memoria asociativa no lineal, que puede almacenar internamente patrones presentados de forma incompleta o con ruido. De esta forma puede ser usada como una herramienta de optimización; también se han utilizado en aplicaciones de segmentación y restauración de imágenes y optimización combinatoria.

Esto significa que si existe una conexión desde la neurona N_i a la neurona N_j, también existe la conexión desde *Nj* a *Ni* ambas exhibiendo el mismo peso ($w_{ij} = w_{ji}$). Vale aclarar que la conexión de una neurona con sí misma no está permitida.

El conjunto permitido de valores de entrada y salida es {0, 1} (o en alguna oportunidad {−1, 1}); o sea, es un conjunto binario. De esta manera todas las neuronas en una Red Hopfield son binarias, tomando solamente uno de los dos estados posibles: activo (1) o inactivo (−1 o 0).

La clave del aprendizaje Hopfield es que si un patrón que tiene que ser aprendido se conoce, los pesos sobre cada conexión de la red neuronal pueden ser calculados. En esta circunstancia, solamente el estado de las neuronas cambia durante el proceso de aprendizaje. Este cálculo garantiza que cada patrón aprendido corresponda a un mínimo de la función energía.

Es importante entender que para este tipo de redes la definición de aprendizaje es diferente al dado anteriormente, donde aprendizaje significaba simplemente la adaptación de los pesos. **[2]**

2.2.8.2 Perceptrón

La red tipo Perceptrón fue inventada por el sicólogo Frank Rosenblatt en el año 1957. Su intención era ilustrar algunas propiedades fundamentales de los sistemas inteligentes en general, sin entrar en mayores detalles con respecto a condiciones específicas y desconocidas para organismos biológicos concretos.

Rosenblatt creía que la conectividad existente en las redes biológicas tiene un elevado porcentaje de aleatoriedad, por lo que se oponía al análisis de McCulloch y Pitts en el cual se empleaba lógica simbólica para analizar estructuras bastante idealizadas. Rosenblatt opinaba que la herramienta de análisis más apropiada era la

teoría de probabilidades, y esto lo llevó a una teoría de *separabilidad estadística* que utilizaba para caracterizar las propiedades más visibles de estas redes de interconexión ligeramente aleatorias. **[4]**

El primer modelo de Perceptrón fue desarrollado en un ambiente biológico imitando el funcionamiento del ojo humano, el fotoperceptrón como se le llamo era un dispositivo que respondía a señales ópticas, la luz incide en los puntos sensibles de la estructura de la retina, cada punto responde en forma todo-nada a la luz entrante, los impulsos generados por los puntos se transmiten a las unidades de asociación de la capa de asociación; cada unidad de asociación está conectada a un conjunto aleatorio de puntos, denominados conjunto fuente de la unidad de asociación, y las conexiones pueden ser tanto excitatorias como inhibitorias. **[4]**

Las conexiones tienen los valores posibles +1, -1 y 0, cuando aparece un conjunto de estímulos en la retina, una unidad de asociación se activa si la suma de sus entradas sobrepasa algún valor umbral; si la unidad esta activada, produce una salida que se envía a la siguiente capa de unidades.

El Perceptrón era inicialmente un dispositivo de aprendizaje, en su configuración inicial no estaba en capacidad de distinguir patrones de entrada muy complejos, sin embargo mediante un proceso de aprendizaje era capaz de adquirir esta capacidad. **[4]**

La única neurona de salida del Perceptrón realiza la suma ponderada de las entradas, resta el umbral y pasa el resultado a una función de transferencia de tipo escalón. La regla de decisión es responder +1 si el patrón presentado pertenece a la clase A, o –1 si el patrón pertenece a la clase B (figura 2.5), la salida depende de la entrada neta (n = suma de las entradas p_i ponderadas). El Perceptrón es un tipo de red

de aprendizaje supervisado, es decir necesita conocer los valores esperados para cada una de las entradas presentadas.

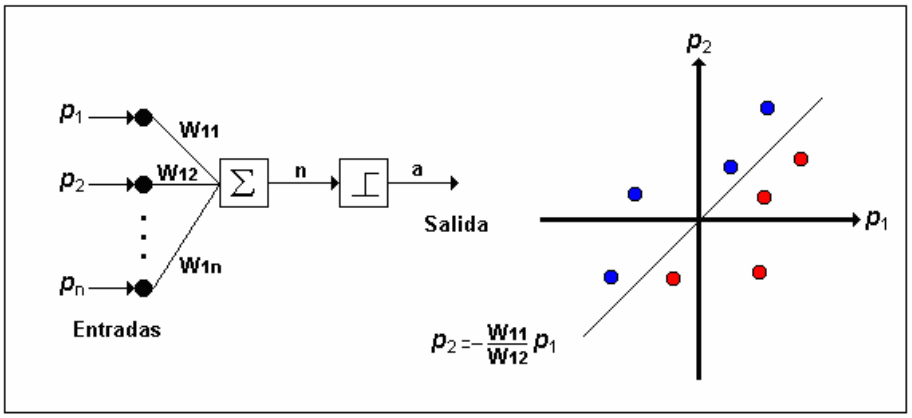


Figura. 2.5 Perceptrón **[1]**

El Perceptrón, al constar de una sola capa de entrada y otra de salida con una única neurona, tiene una capacidad de representación bastante limitada, este modelo sólo es capaz de discriminar patrones muy sencillos, patrones linealmente separables, el caso más conocido es la imposibilidad del Perceptrón de representar la función OR exclusiva. **[4]**

La falta de métodos de entrenamiento para los perceptrones multicapa, hizo que declinara el interés en las Redes Neuronales Artificiales en los años 60 y 70 , en 1986 Rumelhart reformuló el entrenamiento del Perceptrón multicapa Backpropagation. **[5]**

2.2.8.2.1 Backpropagation

La regla de aprendizaje del Perceptrón de Rosenblatt y el algoritmo LMS de Widrow y Hoff fueron diseñados para entrenar redes de una sola capa. Estas redes tienen la

desventaja que solo pueden resolver problemas linealmente separables, fue esto lo que llevo al surgimiento de las redes multicapa para sobrepasar esta dificultad en las redes hasta entonces conocidas.

El primer algoritmo de entrenamiento para redes multicapa fue desarrollado por Paul Werbos en 1974, este se desarrolló en un contexto general, para cualquier tipo de redes, siendo las redes neuronales una aplicación especial, razón por la cual el algoritmo no fue aceptado inicialmente dentro de la comunidad de desarrolladores de Redes Neuronales. **[2]**

Fue solo hasta mediados de los años 80 cuando el algoritmo Backpropagation o algoritmo de propagación inversa fue redescubierto al mismo tiempo por varios investigadores, David Rumelhart, Geoffrey Hinton y Ronal Williams, David Parker y Yann Le Cun. El algoritmo se popularizó cuando fue incluido en el libro "Parallel Distributed Processing Group" por los sicólogos David Rumelhart y James McClelland. La publicación de este libro trajo consigo un auge en las investigaciones con redes neuronales, siendo la Backpropagation una de las redes más ampliamente empleadas, aún en nuestros días. **[2]**

Uno de los grandes avances logrados con la Backpropagation es que esta red aprovecha la naturaleza paralela de las Redes Neuronales para reducir el tiempo requerido por un procesador secuencial para determinar la correspondencia entre unos patrones dados. Además el tiempo de desarrollo de cualquier sistema que se esté tratando de analizar se puede reducir como consecuencia de que la red puede aprender el algoritmo correcto sin que alguien tenga que deducir por anticipado el algoritmo en cuestión.

La mayoría de los sistemas actuales de cómputo se han diseñado para llevar a cabo funciones matemáticas y lógicas a una velocidad que resulta asombrosamente

alta para el ser humano. Sin embargo la destreza matemática no es lo que se necesita para solucionar problemas de reconocimiento de patrones en entornos ruidosos, característica que incluso dentro de un espacio de entrada relativamente pequeño, puede llegar a consumir mucho tiempo. **[2]**

El problema es la naturaleza secuencial del propio computador; el ciclo tomar–ejecutar de la naturaleza Von Neumann solo permite que la máquina realice una operación a la vez. En la mayoría de los casos, el tiempo que necesita la máquina para llevar a cabo cada instrucción es tan breve (típicamente una millonésima de segundo) que el tiempo necesario para un programa, así sea muy grande, es insignificante para los usuarios. Sin embargo, para aquellas aplicaciones que deban explorar un gran espacio de entrada o que intentan correlacionar todas las permutaciones posibles de un conjunto de patrones muy complejo, el tiempo de computación necesario se hace bastante grande. **[2]**

Lo que se necesita es un nuevo sistema de procesamiento que sea capaz de examinar todos los patrones en paralelo. Idealmente ese sistema no tendría que ser programado explícitamente, lo que haría es adaptarse a sí mismo para aprender la relación entre un conjunto de patrones dado como ejemplo y ser capaz de aplicar la misma relación a nuevos patrones de entrada. Este sistema debe estar en capacidad de concentrarse en las características de una entrada arbitraria que se asemeje a otros patrones vistos previamente, sin que ninguna señal de ruido lo afecte. Este sistema fue el gran aporte de la red de propagación inversa, Backpropagation. **[4]**

Backpropagation es un tipo de red de aprendizaje supervisado, que emplea un ciclo propagación – adaptación de dos fases. Una vez que se ha aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas superiores de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas. **[19]**

Las salidas de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo las neuronas de la capa oculta solo reciben una fracción de la señal total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total. **[19]**

Basándose en la señal de error percibida, se actualizan los pesos de conexión de cada neurona, para hacer que la red converja hacia un estado que permita clasificar correctamente todos los patrones de entrenamiento. **[19]**

La importancia de este proceso consiste en que, a medida que se entrena la red, las neuronas de las capas intermedias se organizan a sí mismas de tal modo que las distintas neuronas aprenden a reconocer distintas características del espacio total de entrada. **[19]**

Después del entrenamiento, cuando se les presente un patrón arbitrario de entrada que contenga ruido o que esté incompleto, las neuronas de la capa oculta de la red responderán con una salida activa si la nueva entrada contiene un patrón que se asemeje a aquella característica que las neuronas individuales hayan aprendido a reconocer durante su entrenamiento. Y a la inversa, las unidades de las capas ocultas tienen una tendencia a inhibir su salida si el patrón de entrada no contiene la característica para reconocer, para la cual han sido entrenadas. **[19]**

Varias investigaciones han demostrado que, durante el proceso de entrenamiento, la red Backpropagation tiende a desarrollar relaciones internas entre neuronas con el fin de organizar los datos de entrenamiento en clases. **[19]**

Esta tendencia se puede extrapolar, para llegar a la hipótesis consistente en que todas las unidades de la capa oculta de una Backpropagation son asociadas de alguna manera a características específicas del patrón de entrada como consecuencia del entrenamiento. Lo que sea o no exactamente la asociación puede no resultar evidente para el observador humano, lo importante es que la red ha encontrado una representación interna que le permite generar las salidas deseadas cuando se le dan las entradas, en el proceso de entrenamiento. Esta misma representación interna se puede aplicar a entradas que la red no haya visto antes, y la red clasificará estas entradas según las características que compartan con los ejemplos de entrenamiento. **[19]**

2.2.8.2.1.1 Estructura de la Red Backpropagation

La estructura típica de una red multicapa se observa en la figura 2.7

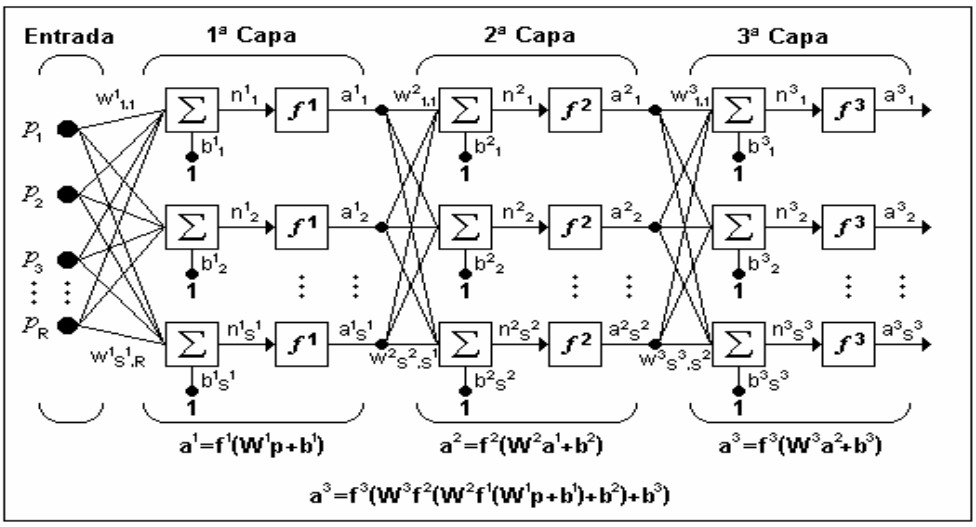


Figura 2.7 Red Perceptrón Multicapa de tres capas **[1]**

Puede notarse que esta red de tres capas equivale a tener tres redes tipo Perceptrón en cascada; la salida de la primera red, es la entrada a la segunda y la salida de la segunda red es la entrada a la tercera. Cada capa puede tener diferente número de neuronas, e incluso distinta función de transferencia.

En la figura 2.7 W^1 representa la matriz de pesos para la primera capa, W^2 los pesos de la segunda y así similarmente para todas las capas que incluya una red. Para identificar la estructura de una red multicapa, se empleará una notación abreviada, donde el número de entradas va seguido del número de neuronas en cada capa: R: S^1: S^2: S^3.

Donde S representa el número de neuronas y el exponente representa la capa a la cual la neurona corresponde.

La notación de la figura 2.7 es bastante clara cuando se desea conocer la estructura detallada de la red, e identificar cada una de las conexiones, pero cuando la red es muy grande, el proceso de conexión se torna muy complejo y es bastante útil utilizar el esquema de la figura 2.8 **[2]**

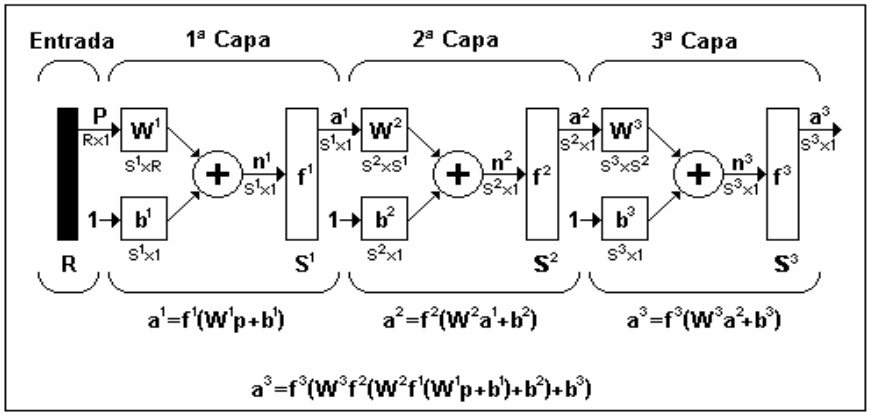


Figura 2.8 Notación compacta de una red de tres capas **[1]**

2.2.8.2.1.2 Funcionamiento

El sistema de entrenamiento mediante Backpropagation consiste en:

- Empezar con unos pesos sinápticos cualquiera (generalmente elegidos al azar).

- Introducir unos datos de entrada (en la capa de entradas) elegidos al azar entre los datos de entrada que se van a usar para el entrenamiento.
- Dejar que la red genere un vector de datos de salida (propagación hacia delante).
- Comparar la salida generada por la red con la salida deseada.
- La diferencia obtenida entre la salida generada y la deseada (denominada *error*) se usa para ajustar los pesos sinápticos de las neuronas de la capa de salidas.
- El error se *propaga hacia atrás* (Backpropagation), hacia la capa de neuronas anterior, y se usa para ajustar los pesos sinápticos en esta capa.
- Se continua propagando el error hacia atrás y ajustando los pesos hasta que se alcance la capa de entradas.

Este proceso se repetirá con los diferentes datos de entrenamiento. **[4]**

2.2.8.2.1.3 Regla de Aprendizaje

El algoritmo Backpropagation para redes multicapa es una generalización del algoritmo LMS, ambos algoritmos realizan su labor de actualización de pesos y ganancias con base en el error medio cuadrático. La Red Backpropagation trabaja bajo aprendizaje supervisado y por tanto necesita un set de entrenamiento que le describa cada salida y su valor de salida esperado de la siguiente forma:

$$T = \{(X_1,Y_1),\ (X_2,Y_2), \ldots ,(X_Q, Y_Q)\ \}\quad (2.2.1)$$

Donde X_Q es una entrada a la red e Y_Q es la correspondiente salida deseada para el patrón q-ésimo. El algoritmo debe ajustar los parámetros de la red para minimizar el error medio cuadrático. **[2]**

El entrenamiento de una Red Neuronal multicapa se realiza mediante un proceso de aprendizaje. Para realizar este proceso se debe inicialmente tener definida la topología de la red, esto es: número de neuronas en la capa de entrada el cual depende del número de componentes del vector de entrada, cantidad de capas ocultas y número de neuronas de cada una de ellas, número de neuronas en la capa de la salida el cual depende del número de componentes del vector de salida o patrones objetivo y funciones de transferencia requeridas en cada capa, con base en la topología escogida se asignan valores iniciales a cada uno de los parámetros que conforma la red.

Es importante recalcar que no existe una técnica para determinar el número de capas ocultas, ni el número de neuronas que debe contener cada una de ellas para un problema especifico, esta elección es determinada por la experiencia del diseñador, el cual debe cumplir con las limitaciones de tipo computacional. **[2]**

En la figura 2.9 se observa la disposición de una red de 3 capas, con q componentes el vector de entrada, m neuronas en la capa oculta y l neuronas en la de salida.

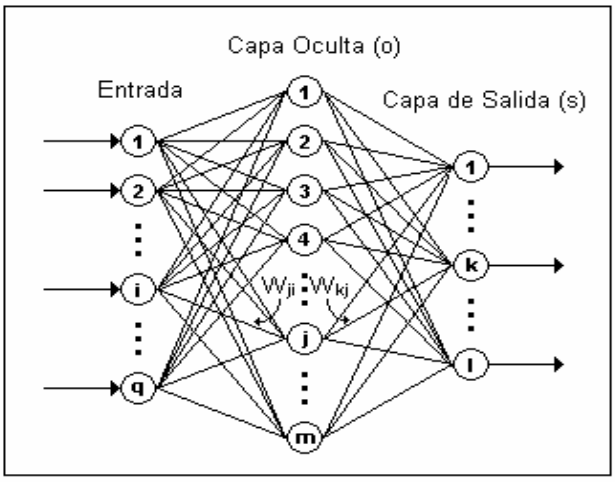


Figura 2.9 Disposición de una red sencilla de 3 capas **[1]**

Cada patrón de entrenamiento se propaga a través de la red y sus parámetros para producir una respuesta en la capa de salida, la cual se compara con los patrones objetivo o salidas deseadas para calcular el error en el aprendizaje, este error marca el camino más adecuado para la actualización de los pesos y ganancias que al final del entrenamiento producirán una respuesta satisfactoria a todos los patrones de entrenamiento, esto se logra minimizando el error medio cuadrático en cada iteración del proceso de aprendizaje. **[4]**

2.2.8.2.1.4 Calculo del Error Cuadrático Promedio

El error es una función que evalúa la diferencia entre las salidas de la red y las salidas deseadas. En la mayor parte de los casos, la función error se define como**: [25]**

$$E = \frac{1}{2} \left(\sum_v Y_i^v - S_i^v\right)^2 \quad (2.2.2)$$

Donde:

Yi = denotan las salidas deseadas de las neuronas de la capa de salida.
v = es el número de patrones o muestras del conjunto de entrenamiento.
Si = denotan las salidas obtenidas en la capa de salida.

Las salidas Yi vienen dadas en el conjunto de entrenamiento, para el cálculo de las salidas Si se utiliza la siguiente ecuación**: [25]**

$$S_i^v = F\left(\sum_j W_{ij} F\left(\sum_k w_{jk} X_k^v\right)\right) \quad (2.2.3)$$

Donde:

F = es la función de transferencia.

Wij = denota los pesos sinápticos de las neuronas de salida;

wjk = denota los pesos sinápticos de las neuronas de la capa oculta;

Xk = denota los componentes de los patrones de entrenamiento.

La función de transferencia se denota por la siguiente ecuación**: [17]**

$$F(x) = \frac{1}{1+e(-x)} \quad (2.2.4)$$

2.2.8.2.1.5 Modificación de los Pesos

La forma explícita para la modificación de los pesos es de gran importancia práctica.

Para los pesos sinápticos W_{ij} de las neuronas de la capa de salida se tienen los siguientes cálculos: **[17]**

$$\delta W_{ij} = -\lambda \frac{\partial E}{\partial w_{ij}} = \lambda \sum_v \Delta_i^v V_j^v \quad (2.2.5)$$

Donde:

E = representa la ecuación de error.

Y Δi se calcula de la siguiente forma:

$$\Delta_i^v = F'(\sum_h W_{ih} V_h^v)(Y_i^v - S_i^v) \quad (2.2.6)$$

Donde:

h= representa el número de neuronas de la capa oculta.

En este caso V juega el papel de patrón de entrada.

En el caso de los pesos sinápticos w_{jk} de las neuronas de la capa oculta, al calcular el gradiente, se ha derivar con respecto a w_{jk} variable implícita en la expresión de la función de error (2.2.2), de tal modo que empleando la regla de la cadena tenemos: **[17]**

$$\delta w_{jk} = -\lambda \frac{\partial E}{\partial w_{jk}} = \lambda \sum_v \delta_j^v X_k^v \quad (2.2.7)$$

Donde tenemos que:

$$\delta_j^v = F'\left(\sum_h w_{jh} X_h^v\right) \sum_i \Delta_i^v W_{ij} \quad (2.2.8)$$

La derivada de la función de transferencia F' para las ecuaciones 2.2.6 y 2.2.8 se denota por:

$$F'(x) = \frac{1}{1+e(-x)}(-e(-x)) = \frac{1}{1+e(-x)} \frac{e(-x)}{1+e(-x)} \quad (2.2.9)$$

Simplificando tenemos:

$$F'(x) = F(x)(1 - F(x)) \quad (2.2.10)$$

Donde F(x) está representada en la ecuación 2.2.4.

2.2.8.3 Adaline

La red Adaline es similar al Perceptrón, excepto en su función de transferencia, la cual es una función continúa sigmoide en lugar de un limitador fuerte como en el caso del Perceptrón.

El algoritmo LMS (del inglés, *Least-Mean-Square algorithm*) se usa para encontrar los coeficientes que permiten obtener el valor esperado mínimo del cuadrado de la señal de error, definida como la diferencia entre la señal deseada y la señal producida a la salida del filtro.

La red Adaline presenta la misma limitación del Perceptrón en cuanto al tipo de problemas que pueden resolver, ambas redes pueden solo resolver problemas linealmente separables, sin embargo el algoritmo LMS es más potente que la regla de aprendizaje del Perceptrón ya que minimiza el error medio cuadrático, la regla sirvió de inspiración para el desarrollo de otros algoritmos, este es el gran aporte de esta red.

El término Adaline es una sigla, sin embargo su significado cambió ligeramente a finales de los años sesenta cuando decayó el estudio de las redes neuronales, inicialmente se llamaba ADAptive LInear NEuron (Neurona Lineal Adaptiva), para pasar después a ser Adaptive LInear Element (Elemento Lineal Adaptivo), este cambio se debió a que la Adaline es un dispositivo que consta de un único elemento de procesamiento, como tal no es técnicamente una red neuronal.

El elemento de procesamiento realiza la suma de los productos de los vectores de entrada y de pesos, y aplica una función de salida para obtener un único valor de salida, el cual debido a su función de transferencia lineal será +1 si la sumatoria es positiva o –1 si la salida de la sumatoria es negativa. En términos generales la salida de la red está dada por: $a= W^T p$. **[2]**

En este caso, la salida es la función identidad al igual que la función de activación; el uso de la función identidad como función de salida y como función de

activación significa que la salida es igual a la activación, que es la misma entrada neta al elemento.

El Adaline es adaptivo en el sentido de que existe un procedimiento bien definido para modificar los pesos con objeto de hacer posible que el dispositivo proporcione el valor de salida correcto para la entrada dada; el significado de correcto para efectos del valor de salida depende de la función de tratamiento de señales que esté siendo llevada a cabo por el dispositivo. **[2]**

El Adaline es Lineal porque la salida es una función lineal sencilla de los valores de la entrada. Es una Neurona tan solo en el sentido (muy limitado) del Patrón de Entrada. También se podría decir que el Adaline es un Elemento Lineal, evitando por completo la definición como Neurona. La estructura general de la red tipo Adaline puede visualizarse en la figura 2.6

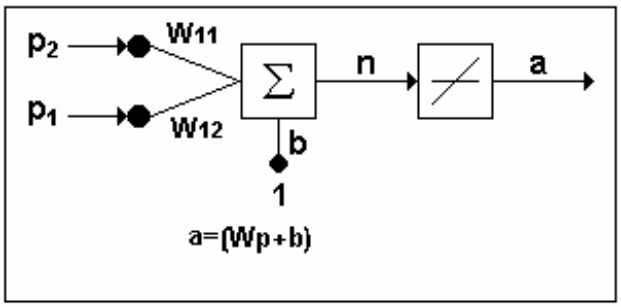


Figura 2.6 Estructura de la Red Adaline **[1]**

2.2.8.4 Red de Kohonen

Existen evidencias que demuestran que en el cerebro hay neuronas que se organizan en muchas zonas, de forma que las informaciones captadas del entorno a través de los órganos sensoriales se representan internamente en forma de mapas bidimensionales.

Aunque en gran medida esta organización neuronal está predeterminada genéticamente, es probable que parte de ella se origine mediante el aprendizaje, esto sugiere que el cerebro podría poseer la capacidad inherente de formar mapas topológicos de las informaciones recibidas del exterior, de hecho esta teoría podría explicar su poder de operar con elementos semánticos: algunas áreas del cerebro simplemente podrían crear y ordenar neuronas especializadas o grupos con características de alto nivel y sus combinaciones, en definitiva se construirían mapas especiales para atributos y características. **[4]**

A partir de estas ideas Tuevo Kohonen presentó en 1982 un sistema con un comportamiento semejante, se trataba de un modelo de red neuronal con capacidad para formar mapas de características de manera similar a como ocurre en el cerebro; el objetivo de Kohonen era demostrar que un estímulo externo (información de entrada) por sí solo, suponiendo una estructura propia y una descripción funcional del comportamiento de la red, era suficiente para forzar la formación de los mapas, como se puede observar en la figura 2.10. **[4]**

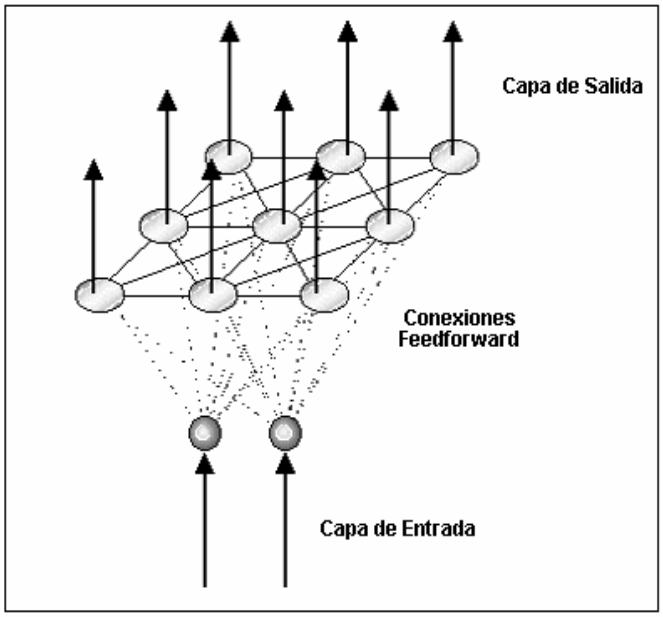


Figura 2.10 Conexiones de una red de Kohonen **[1]**

El aprendizaje en el modelo de Kohonen es de tipo Off-line, por lo que se distingue una etapa de aprendizaje y otra de funcionamiento. En la etapa de aprendizaje se fijan los valores de las conexiones (feedforward) entre la capa de entrada y la salida. Esta red utiliza un aprendizaje no supervisado de tipo competitivo, las neuronas de la capa de salida compiten por activarse y sólo una de ellas permanece activa ante una determinada información de entrada a la red, los pesos de las conexiones se ajustan en función de la neurona que haya resultado vencedora. **[4]**

Durante la etapa de entrenamiento, se presenta a la red un conjunto de informaciones de entrada (vectores de entrenamiento) para que ésta establezca en función de la semejanza entre los datos las diferentes categorías (una por neurona de salida), que servirían durante la fase de funcionamiento para realizar clasificaciones de nuevos datos que se presenten a la red. Los valores finales de los pesos de las conexiones entre cada neurona de la capa de salida con las de entrada se

corresponderán con los valores de los componentes del vector de aprendizaje que consigue activar la neurona correspondiente. **[4]**

Un concepto muy importante en la red de Kohonen es la zona de vecindad, o vecindario alrededor de la neurona vencedora *i**, los pesos de las neuronas que se encuentren en esta zona a la que se le dará el nombre de $\boldsymbol{X}$ (q), serán actualizados junto con el peso de la neurona ganadora, en un ejemplo de aprendizaje cooperativo. Como podemos ver en la figura 2.11 una representación de la zona de vecindad.

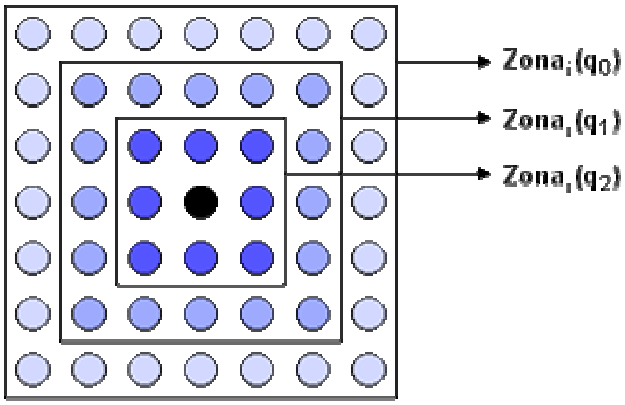


Figura 2.11 Representación de la zona de vecindad en una red de Kohonen **[1]**

En definitiva lo que hace una red de Kohonen es realizar una tarea de clasificación, puesto que la neurona de salida activada ante una entrada representa la clase a la que pertenece dicha información de entrada, además ante otra entrada parecida se activa la misma neurona de salida, u otra cercana a la anterior debido a la semejanza entre las clases, así se garantiza que las neuronas topológicamente próximas sean sensibles a entradas físicamente similares; por esta causa la red es especialmente útil para establecer relaciones desconocidas previamente entre conjuntos de datos. **[4]**

2.2.8.5 Red de Hamming

La red de Hamming ilustrada en la figura 2.12 es uno de los ejemplo más simples de aprendizaje competitivo, a pesar de ello su estructura es un poco compleja ya que emplea el concepto de capas recurrentes en su segunda capa y aunque hoy en día en redes de aprendizaje competitivo se ha simplificado este concepto con el uso de funciones de activación más sencillas, la red de Hamming representa uno de los primeros avances en este tipo de aprendizaje, convirtiéndola en un modelo obligado de referencia dentro de las redes de aprendizaje competitivo.

Las neuronas en la capa de salida de esta red compiten unas con otras para determinar la ganadora, la cual índica el patrón prototipo más representativo en la entrada de la red, la competición es implementada por inhibición lateral (un conjunto de conexiones negativas entre las neuronas en la capa de salida).

Esta red consiste en dos capas; la primera capa realiza la correlación entre el vector de entrada y los vectores prototipo, la segunda capa realiza la competición para determinar cuál de los vectores prototipo está más cercano al vector de entrada. **[2]**

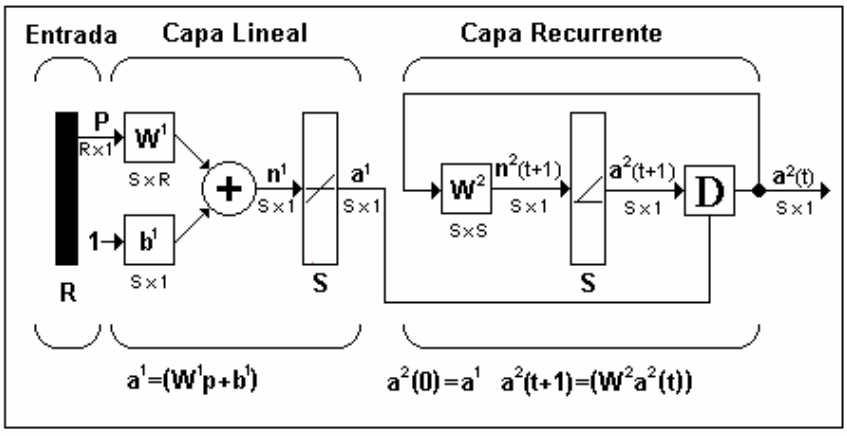


Figura 2.12 Red de Hamming **[1]**

2.2.9 Tipo De Asociación Entre Las Informaciones De Entrada Y Salida

Las Redes Neuronales son sistemas que almacenan cierta información aprendida; esta se registra de forma distribuida en los pesos asociados a las conexiones entre neuronas. Hay que establecer cierta relación o asociación entre la información presentada a la red y la salida ofrecida por esta. Es lo que se conoce como memoria asociativa. **[2]**

Existen dos formas primarias de realizar esta asociación entrada/salida y que generan dos tipos de redes:

- **Redes Heteroasociativas**: La red aprende parejas de datos [(A1, B1), (A2, B2),.... (An, Bn)], de tal forma que cuando se le presente determinada información de entrada Ai responda con la salida correspondiente Bi. Al asociar informaciones de entrada con diferentes informaciones de salida, precisan al menos de 2 capas, una para captar y retener la información de entrada y otra para mantener la salida con la información asociada. Si esto no fuese así se perdería la información inicial al obtenerse la salida asociada; es necesario mantener la información de entrada puesto que puede ser necesario acceder varias veces a ella, por lo que debe permanecer en la capa de entrada. El aprendizaje de este tipo de redes puede ser con supervisión.

- **Redes autoasociativas**: La red aprende ciertas informaciones A1, A2,.., An de forma que cuando se le presenta una información de entrada realizará una autocorrelación, respondiendo con uno de los datos almacenados, el más parecido al de entrada. Este tipo de redes pueden implementarse con una sola capa de neuronas. El tipo de aprendizaje utilizado habitualmente es el no supervisado y suelen utilizarse en tareas de filtrado de información para la

reconstrucción de datos, eliminando distorsiones o ruido, explorar relaciones entre informaciones similares para facilitar la búsqueda por contenido en bases de datos y para resolver problemas de optimización. **[13]**

2.2.10 Representación de la Información de Entrada y Salida

- **Redes Continuas**: En un gran número de redes, tanto los datos de entrada como de salida son de naturaleza analógica (valores reales continuos y normalmente normalizados, por lo que su valor absoluto será menor que la unidad). En este caso las funciones de activación de las neuronas serán también continuas, del tipo lineal o sigmoidal.

- **Redes Discretas**: Por el contrario, otras redes sólo admiten valores discretos [0,1] a la entrada, generando también en la salida respuestas de tipo binario. La función de activación en este caso es del tipo escalón.

- **Redes Híbridas**: La información de entrada es continua pero a la salida ofrecen información binaria. **[4]**

2.3 ALGORITMOS GENÉTICOS

2.3.1 Historia

En los años 70, de la mano de John Holland, surgió una de las líneas más prometedoras de la Inteligencia Artificial, la de los Algoritmos Genéticos. Son llamados así porque se inspiran en la evolución biológica y su base genético-molecular.

Estos algoritmos hacen evolucionar una población de individuos sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución biológica (mutaciones y recombinaciones genéticas), así como también a una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados.**[15]**

2.3.2 Definición

Los Algoritmos Genéticos forman parte de una familia denominada algoritmos evolutivos, que incluye las estrategias de evolución, la programación evolutiva y la programación genética.

Los Algoritmos Genéticos son métodos sistemáticos para la resolución de problemas de búsqueda y optimización que aplican a estos los mismos métodos de la evolución biológica: selección basada en la población, reproducción sexual y mutación.

Un Algoritmo Genético consiste en lo siguiente: hallar de qué valores depende el problema, codificarlos en un cromosoma, y aplicar los métodos de la evolución: selección y reproducción sexual con intercambio de información y alteraciones que generan diversidad. **[18]**

2.3.3 Características

Algunas de las características de los Algoritmos Genéticos son [**18**]:

- Son algoritmos estocásticos. Dos ejecuciones distintas pueden dar dos soluciones distintas.

- Son algoritmos de búsqueda múltiple, por lo tanto dan varias soluciones.

- Son los algoritmos que hacen una barrida amplia al subespacio de posibles soluciones válidas, y con diferencia. De hecho, se considera que, de todos los algoritmos de optimización estocásticos, los Algoritmos Genéticos son de los más exploratorios disponibles.

- La convergencia del algoritmo es poco sensible a la población inicial si esta se escoge de forma aleatoria y es lo suficientemente grande.

- Por su grado de penetración casi nulo, la curva de convergencia asociada al algoritmo presenta una convergencia excepcionalmente rápida al principio, que casi enseguida se bloquea. Esto se debe a que el Algoritmo Genético es excelente descartando subespacios realmente malos. Cada cierto tiempo, la población realiza el proceso evolutivo, y se produce un incremento en la velocidad de convergencia excepcional. La razón de esto es que algunas veces aparece una mutación altamente beneficiosa, o un individuo excepcional, que propaga algún conjunto de cromosomas excepcional al resto de la población.

- La optimización es función de la representación de los datos. Una buena codificación puede hacer la programación y resolución muy sencillas, mientras que una codificación errada obligará a estudiar que los individuos cumplan las restricciones del problema. Además, la velocidad de convergencia va a estar fuertemente influenciada por la representación.

- Es una búsqueda paramétrica robusta. Eso quiere decir que los parámetros del algoritmo escogidos deben ser realmente malos para que no converja.

- Los Algoritmos Genéticos son intrínsecamente paralelos. Esto significa que, independientemente de que se hayan implementado de forma paralela o no, buscan en distintos puntos del espacio de soluciones de forma paralela.

2.3.4 Analogía de un Algoritmo Genético con la Naturaleza

La idea básica de un algoritmo genético es la siguiente: generar un conjunto con algunas de las posibles soluciones. Cada una va a ser llamada individuo, y a dicho conjunto se le denominará población [**20**].

Cada individuo tiene una información asociada a él. En un problema de optimización corresponde a las variables libres, es decir, aquellas a las que el algoritmo tiene que asignar un valor para que una función sea mínima o máxima para esos valores. Esa función es la denominada función de adaptación y determina el grado de adaptación de un individuo. A dicha información se la va a denominar código genético.

Las características de los individuos, sean beneficiosas o no, se van a denominar fenotipos. La información asociada a un individuo se compone de partes indivisibles denominados cromosomas.

Un fenotipo puede estar en más de un cromosoma, en cuyo caso puede ser que el hijo herede un fenotipo que no tenía ni el padre ni la madre, sino una combinación de ambos. Un ejemplo en el humano es el color de la piel o la estructura del cráneo. En caso de que el hijo tenga parte de los genes del padre y parte de los genes de la madre que intervienen en un fenotipo, se va a crear una característica nueva asociada a ese fenotipo.

De todas formas, no es un enfoque muy frecuente, ya que debemos asegurar que el conjunto de los fenomas tendrá ley de composición interna respecto al operador de cruce definido sobre el alfabeto cromosómico. Por otro lado, que un cromosoma codifique más de un fenotipo es más raro todavía. El cromosoma debe tener en dicho caso tantos valores como el producto del número de valores posibles que tenga cada fenotipo del cromosoma.

Es de vital importancia la forma de codificar los fenotipos en los cromosomas y la determinación de qué es fenotipo, es decir, como la información va a ser almacenada en el código genético. Escoger equivocadamente la forma de almacenar la información puede ralentizar la convergencia, es decir, que se tardará más en encontrar la solución ó nunca convergerá en una solución debido a que la población esta errando aleatoriamente por efecto de las mutaciones y de los cruzamientos sin llegar nunca a un punto estable, a este fenómeno se le denomina deriva genética [**20**].

2.3.5 Fitness

También se llama función de evaluación, ésta representa los requisitos a los cuales la población debe adaptarse, asigna un valor singular real a cada fenotipo y forma la base para la selección, cuanto mejor discrimine distintos fenotipos mejor. **[21]**

Es sabido que la función de fitness es una de las componentes principales de los Algoritmos Genéticos. Ésta es la que permite decidir que individuos continúan en la población y cuales expiran. Es decir que si se obtienen funciones de fitness correctas los Algoritmos Genéticos pueden mantener su habilidad para encontrar soluciones adecuadas.

Se trata de una función evaluadora de la calidad de un individuo como solución a nuestro problema. Permite la ordenación de los individuos de la población en cuanto a bondad de los mismos**. [14]**

2.3.6 Codificación

Los Algoritmos Genéticos requieren que las soluciones factibles se codifiquen en cromosomas. Cada cromosoma tiene varios genes, que corresponden a los valores iniciales. Para poder trabajar con estos genes en el ordenador, es necesario codificarlos en una cadena, es decir, una ristra de símbolos (números o letras) que generalmente va a estar compuesta de 0s y 1s.

Hay otras codificaciones posibles, usando alfabetos de diferente cardinalidad; sin embargo, uno de los resultados fundamentales en la teoría de algoritmos genéticos, el teorema de los esquemas el cual proporciona el fundamento teórico de porqué los Algoritmos Genéticos pueden resolver diversos problemas, este teorema afirma que la codificación óptima, es decir, aquella sobre la que los Algoritmos Genéticos funcionan mejor, es aquella que tiene un alfabeto de cardinalidad 2. **[20]**

La mayoría de las veces, una codificación correcta es la clave de una buena resolución del problema. Generalmente, la regla heurística que se utiliza es la llamada regla de los bloques de construcción, es decir, valores relacionados entre sí deben estar cercanos en el cromosoma. Por ejemplo, si queremos codificar los pesos de una Red Neuronal, una buena elección será poner juntos todos los pesos que salgan de la misma neurona de la capa oculta.

Normalmente, la codificación es estática, pero en casos de optimización numérica, el número de bits dedicados a codificar un valor puede variar, o incluso lo que representen los bits dedicados a codificar cada valor. Algunos paquetes de

Algoritmos Genéticos adaptan automáticamente la codificación según van convergiendo los bits menos significativos de una solución. **[20]**

Para comenzar la competición, se generan aleatoriamente una serie de cromosomas. El algoritmo genético procede de la forma siguiente: **[20]**

1. Evaluar la puntuación (fitness) de cada uno de los cromosomas.
2. Permitir a cada uno de los individuos reproducirse, de acuerdo con su puntuación.
3. Emparejar los individuos de la nueva población, haciendo que intercambien material genético, y que alguno de los bits de un gen se vea alterado debido a una mutación espontánea.

Cada uno de los pasos consiste en una actuación sobre las cadenas de bits, es decir, la aplicación de un operador a una cadena binaria. Se les denominan, por razones obvias, operadores genéticos, y hay tres principales: selección, crossover o recombinación y mutación; aparte de otros operadores genéticos no tan comunes. **[20]**

Un Algoritmo Genético tiene también una serie de parámetros que se tienen que fijar para cada ejecución, como los siguientes:

- **Población:** Conjunto de individuos con los que se trabaja en el algoritmo genético. En un algoritmo genético los individuos que constituyen la población van cambiando pero generalmente el tamaño de la misma permanece constante.
- **Tamaño de la población**: debe de ser suficiente para garantizar la diversidad de las soluciones, y además, tiene que crecer más o menos con el número de bits del cromosoma, aunque nadie ha aclarado cómo tiene que hacerlo. Por supuesto, depende también del ordenador en el que se esté ejecutando.

- **Cromosomas**: cada cromosoma es una estructura de datos que representa una de las posibles soluciones del espacio de búsqueda del problema. Los Cromosomas son sometidos a un proceso de evolución que envuelve evaluación, selección, recombinación sexual (cruce) y mutación. Después de varios ciclos de evolución la población deberá contener individuos más aptos.
- **Condición de terminación:** lo más habitual es que la condición de terminación sea la convergencia del algoritmo genético o un número prefijado de generaciones. **[26] [27]**

Los individuos, pueden representarse como un conjunto de genes, los cuales agrupados forman una ristra de valores (cromosoma). Si bien el alfabeto utilizado para representar los individuos no debe necesariamente estar constituido por el 0 y 1, buena parte de la teoría en la que se fundamentan los Algoritmos Genéticos utiliza dicho alfabeto.

En términos biológicos, el conjunto de parámetros representando un cromosoma particular se denomina fenotipo. El fenotipo contiene la información requerida para construir un organismo, el cual se refiere como genotipo. Los mismos términos se utilizan en el campo de los Algoritmos Genéticos**. [10]**

La adaptación al problema de un individuo depende de la evaluación del genotipo. Esta última puede inferirse a partir del fenotipo, es decir puede ser computada a partir del cromosoma, usando la función de evaluación.

La función de adaptación debe ser diseñada para cada problema de manera específica. Dado un cromosoma particular, la función de adaptación le asigna un número real, que se supone refleja el nivel de adaptación al problema del individuo representado por el cromosoma.

Durante la fase reproductiva se seleccionan los individuos de la población para cruzarse y producir descendientes, que constituirán, una vez mutados, la siguiente generación de individuos. La selección de padres se efectúa al azar usando un procedimiento que favorezca a los individuos mejor adaptados, ya que a cada individuo se le asigna una probabilidad de ser seleccionado que es proporcional a su función de adaptación. Este procedimiento se dice que está basado en la ruleta sesgada. **[10]**

Según dicho esquema, los individuos bien adaptados se escogerán probablemente varias veces por generación, mientras que los pobremente adaptados al problema, no se escogerán más que de vez en cuando. **[18]**

Una vez seleccionados dos padres, sus cromosomas se combinan, utilizando habitualmente los operadores de cruce y mutación. Las formas básicas de dichos operadores se describen a continuación. **[18]**

El operador de cruce, toma dos padres seleccionados y corta sus ristras de cromosomas en una posición escogida al azar, para producir dos sub-ristras iniciales y dos sub-ristras adicionales. Después se intercambian las sub-ristras adicionales, produciéndose dos nuevos cromosomas completos (véase la Figura 2.13). Ambos descendientes heredan genes de cada uno de los padres. Este operador se conoce como operador de cruce basado en un punto. **[18]**

Habitualmente el operador de cruce no se aplica a todos los pares de individuos que han sido seleccionados para emparejarse, sino que se aplica de manera aleatoria, normalmente con una probabilidad comprendida entre 0.5 y 1.0. En el caso en que el operador de cruce no se aplique, la descendencia se obtiene simplemente duplicando los padres. **[18]**

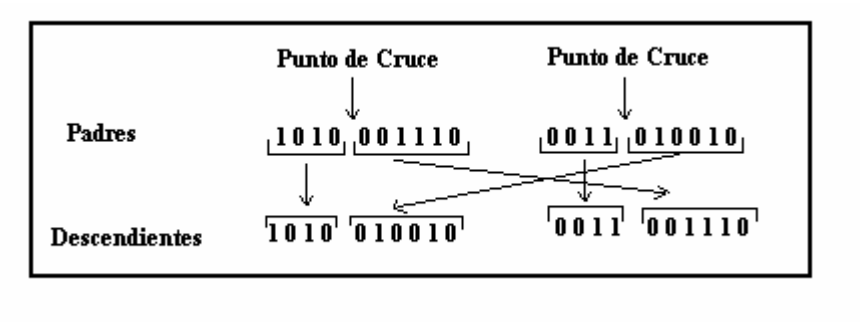


Figura 2.13 Operador de cruce basado en un punto **[18]**

El operador de mutación se aplica a cada hijo de manera individual, y consiste en la alteración aleatoria (normalmente con probabilidad pequeña) de cada gen componente del cromosoma. La Figura 2.14 muestra la mutación del quinto gen del cromosoma. **[18]**

Si bien puede en principio pensarse que el operador de cruce es más importante que el operador de mutación, ya que proporciona una exploración rápida del espacio de búsqueda, este último asegura que ningún punto del espacio de búsqueda tenga probabilidad cero de ser examinado, y es de capital importancia para asegurar la convergencia de los Algoritmos Genéticos. **[10]**

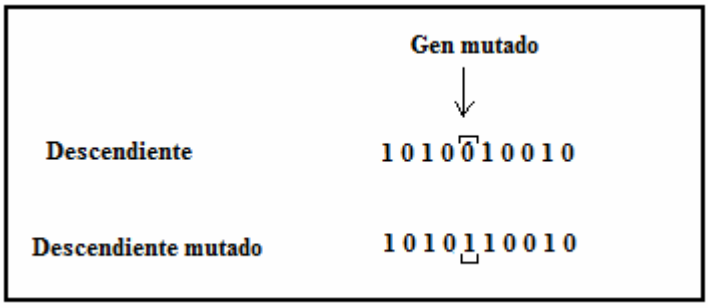


Figura 2.14 Operador de Mutación **[18]**

Si el Algoritmo Genético ha sido correctamente implementado, la población evolucionará a lo largo de las generaciones sucesivas de tal manera que la adaptación media extendida a todos los individuos de la población, así como la adaptación del mejor individuo se irán incrementando hacia el óptimo global. **[18]**

2.3.7 Cruce

El Algoritmo Genético utiliza varios tipos de cruce, uno de ellos es el cruce basado en un punto, en el cual los dos individuos seleccionados para jugar el papel de padres, son recombinados por medio de la selección de un punto de corte, para posteriormente intercambiar las secciones que se encuentran a la derecha de dicho punto. **[18]**

Se han investigado otros operadores de cruce, habitualmente teniendo en cuenta más de un punto de cruce. De Jong (1975) investigó el comportamiento del operador de cruce basado en múltiples puntos, concluyendo que el cruce basado en dos puntos, representaba una mejora mientras que añadir más puntos de cruce no beneficiaba el comportamiento del algoritmo. **[18]**

La ventaja de tener más de un punto de cruce radica en que el espacio de búsqueda puede ser explorado más fácilmente, siendo la principal desventaja el hecho de aumentar la probabilidad de ruptura de buenos esquemas. **[18]**

En el operador de cruce basado en dos puntos, los cromosomas (individuos) pueden contemplarse como un circuito en el cual se efectúa la selección aleatoria de dos puntos, tal y como se indica en la figura 2.15. **[18]**

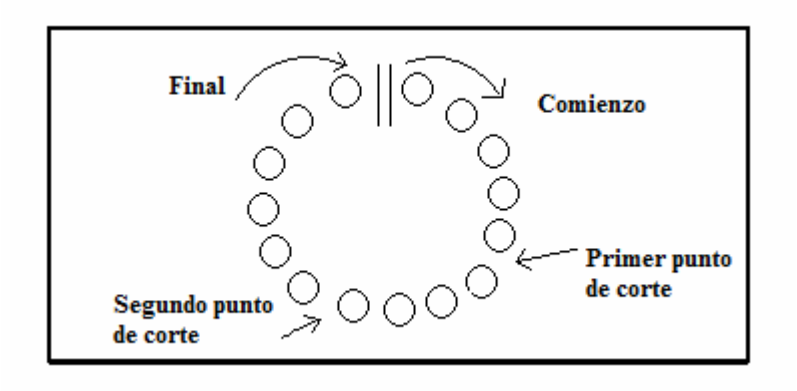


Figura 2.15 Individuo visto como un circuito **[18]**

Desde este punto de vista, el cruce basado en un punto, puede verse como un caso particular del cruce basado en dos puntos, en el cual uno de los puntos de corte se encuentra fijo al comienzo de la ristra que representa al individuo. Véase figura 2.16. **[18]**

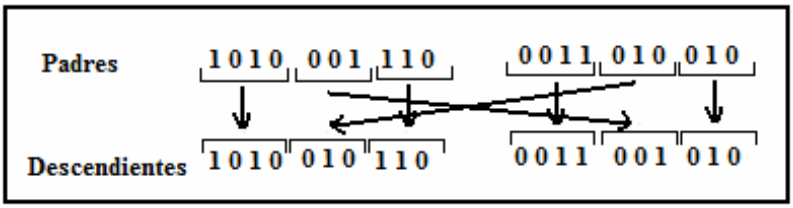


Figura 2.16 Operador de cruce basado en un punto **[18]**

En el denominado operador de cruce uniforme, cada gen en la descendencia se crea copiando el correspondiente gen de uno de los dos padres, escogido de acuerdo a una “máscara de cruce" generada aleatoriamente. Cuando existe un 1 en la “máscara de cruce", el gen es copiado del primer padre, mientras que cuando exista un 0 en la máscara, el gen se copia del segundo padre, tal y como se muestra en la figura 2.17.

Hablando en términos de la teoría de la probabilidad la máscara de cruce está compuesta por una muestra aleatoria de tamaño extraída de una distribución de probabilidad de Bernouilli de parámetro 1/2. **[18]**

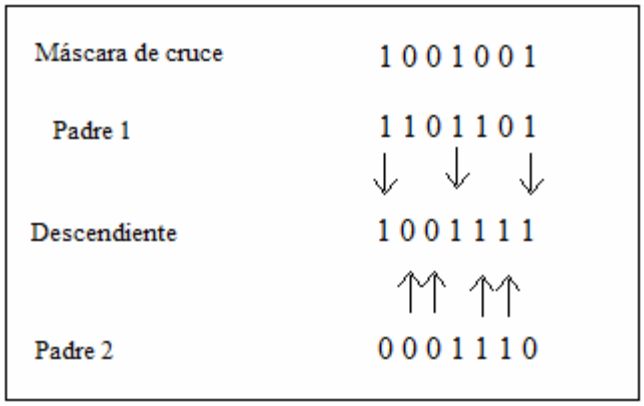


Figura 2.17: Operador de cruce uniforme **[18]**

2.3.8 Mutación

La mutación se considera un operador básico, que proporciona un pequeño elemento de aleatoriedad en la vecindad (entorno) de los individuos de la población. Si bien se admite que el operador de cruce es el responsable de efectuar la búsqueda a lo largo del espacio de posibles soluciones, también parece desprenderse de los experimentos efectuados por varios investigadores que el operador de mutación va ganando en importancia a medida que la población de individuos va convergiendo. Véase figura 2.14.

Schaffer (1989) encuentra que el efecto del cruce en la búsqueda es inferior al que previamente se esperaba. Utiliza la denominada evolución primitiva, en la cual, el proceso evolutivo consta tan sólo de selección y mutación. Encuentran que dicha evolución primitiva supera con creces a una evolución basada exclusivamente en la selección y el cruce. **[18]**

Otra conclusión de su trabajo es que la determinación del valor óptimo de la probabilidad de mutación es mucho más crucial que el relativo a la probabilidad de cruce. **[18]**

La búsqueda del valor óptimo para la probabilidad de mutación, es una cuestión que ha sido motivo de varios trabajos. Así, De Jong (1975) recomienda la utilización de una probabilidad de mutación del bit de l - 1, siendo l la longitud del string. Schaffer utiliza resultados experimentales para estimar la tasa optima proporcional a $1 / \lambda^{0.9318} l^{0.4535}$, donde λ denota el número de individuos en la población.

Si bien en la mayoría de las implementaciones de Algoritmos Genéticos se asume que tanto la probabilidad de cruce como la de mutación permanecen constantes, algunos autores han obtenido mejores resultados experimentales modificando la probabilidad de mutación a medida que aumenta el número de iteraciones. **[18]**

2.3.9 Selección

La función de selección de padres más utilizada es la denominada función de selección proporcional a la función objetivo, en la cual cada individuo tiene una probabilidad de ser seleccionado como padre que es proporcional al valor de su función objetivo. **[18]**

Un esquema de selección, introducido por Brindle (1991), y que empíricamente ha proporcionado buenos resultados, es el denominado muestreo estocástico con reemplazo del resto, en el cual cada individuo es seleccionado un

número de veces que coincide con la parte entera del número esperado de ocurrencias de dicho suceso compitiendo los individuos por los restos.

Baker (1987) introduce un método denominado muestreo universal estocástico, el cual utiliza un único giro de la ruleta siendo los sectores circulares proporcionales a la función objetivo. Los individuos son seleccionados a partir de marcadores (véase Figura 2.18), igualmente espaciados y con comienzo aleatorio. **[18]**

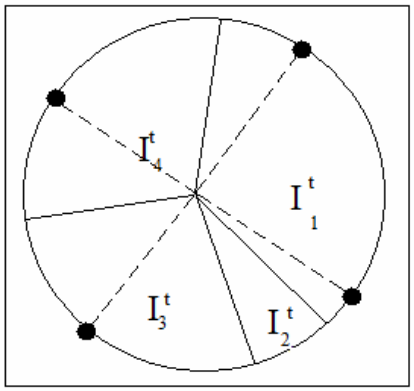


Figura 2.18 Método de selección de padres denominado muestreo universal estocástico. **[18]**

En el modelo de selección elitista se fuerza a que el mejor individuo de la población en el tiempo t, sea seleccionado como padre. **[18]**

La selección por torneo, constituye un procedimiento de selección de padres muy extendido y en el cual la idea consiste en escoger al azar un número de individuos de la población, tamaño del torneo, (con o sin reemplazo), seleccionar el mejor individuo de este grupo, y repetir el proceso hasta que el número de individuos seleccionados coincida con el tamaño de la población. **[18]**

Habitualmente el tamaño del torneo es 2, y en tal caso se ha utilizado una versión probabilística en la cual se permite la selección de individuos sin que necesariamente sean los mejores. **[18]**

Una posible clasificación de procedimientos de selección de padres consistirá en: métodos de selección dinámicos, en los cuales las probabilidades de selección varían de generación a generación, (por ejemplo la selección proporcional a la función objetivo), frente a métodos de selección estáticos, en los cuales dichas probabilidades permanecen constantes (por ejemplo la selección basada en rangos). **[18]**

Si se asegura que todos los individuos tienen asignada una probabilidad de selección distinta de cero el método de selección se denomina preservativo. En caso contrario se acostumbra a denominarlo extintivo. **[21].**

2.3.10 Teoremas de Convergencia

Rudolph (1994) demuestra la no convergencia hacia el óptimo global del Algoritmo Genético, así como la garantía, de convergencia expresada en términos probabilísticos, del Algoritmo Genético que mantiene a la mejor solución en la población. **[18]**

Davis y Principe (1993) extrapolan los fundamentos teóricos del algoritmo simulated annealing a un modelo de algoritmo genético basado en cadenas de Markov. **[18]**

Se efectúa un estudio de las matrices de transición de estados teniendo en cuenta en primer lugar tan sólo la reproducción, a continuación la reproducción y la mutación y finalmente la reproducción, la mutación y el cruce. **[18]**

Suzuki (1993) efectúa un estudio de la convergencia de los Algoritmos Genéticos por medio de cadenas de Markov. **[18]**

Liepins (1992) demuestra la convergencia del Algoritmo Genético hacia poblaciones que contienen al óptimo, en el caso de Algoritmos Genéticos sin operador de mutación, pero en los cuales el reemplazo de individuos es elitista el mejor individuo no se pierde nunca y además se efectúa de tal manera que en cada paso cualquier punto del espacio sea potencialmente alcanzable por medio de la operación de cruce. **[18]**

Chakraborty y Dastidar (1993), presentan un modelo de fiabilidad estocástica de un esquema para el Algoritmo Genético binario con longitud de representación fija, y obtienen una estimación para el número de generaciones necesarias hasta obtener la convergencia. **[18]**

Eiben y col. (1990) modelan la evolución del Algoritmo Genético por medio de una cadena de Markov, obteniendo condiciones suficientes para la convergencia en probabilidad del proceso evolutivo hacia el óptimo global. **[18]**

2.3.11 Diferencias entre los Métodos Tradicionales y los Algoritmos Genéticos

Podemos decir que los Algoritmos Genéticos difieren de los principales métodos tradicionales de optimización en cuatro puntos fundamentales **[16]:**

- Los Algoritmos Genéticos trabajan con codificaciones de los puntos del espacio de búsqueda en lugar de los puntos propiamente dichos.

- Los Algoritmos Genéticos realizan la búsqueda a partir de una población de puntos en lugar de un solo punto.

- Los Algoritmos Genéticos no utilizan derivadas ni otras propiedades de la función objetivo, sino únicamente la propia función objetivo.

- Los Algoritmos Genéticos se rigen mediante reglas de transición probabilísticas, no determinísticas.

2.4 LENGUAJE UNIFICADO DE MODELADO (UML)

2.4.1 Definición

El Lenguaje de Modelado Unificado contiene una notación robusta para el modelado y desarrollo de sistemas orientados a objeto. Proporciona la tecnología necesaria para apoyar la práctica de la ingeniería del software orientada a objetos.

Como resultado de la aplicación de UML se puede producir un arreglo de modelos y documentos de trabajo. Sin embargo, éstos los reducen los ingenieros de software para lograr que el desarrollo sea más ágil y reactivo ante el cambio.

Otros métodos de modelaje como OMT (Object Modeling Technique) o Booch sí definen procesos concretos. En UML los procesos de desarrollo son diferentes según los distintos dominios de trabajo; no puede ser el mismo el proceso para crear una aplicación en tiempo real, que el proceso de desarrollo de una aplicación orientada a gestión, por poner un ejemplo. **[12]**

2.4.2 Elementos de UML

- **Actor**: es una entidad externa (de fuera del sistema) que interacciona con el sistema participando (y normalmente iniciando) en un caso de uso. Los actores pueden ser gente real (por ejemplo, usuarios del sistema), otros ordenadores o eventos externos.

  Los actores no representan a personas *físicas* o a sistemas, sino su *papel*. Esto significa que cuando una persona interacciones con el sistema de diferentes maneras, estará representado por varios actores. **[12]**

- **Clases**: En una clase se agrupan todos los objetos que comparten los mismos atributos, métodos y relaciones. Los atributos son características y propiedades comunes en todos los objetos de la clase. Los métodos son operaciones que deben cumplir las instancias de la clase. Las clases se representan como un rectángulo donde figuran el nombre de la clase, sus atributos y sus métodos. **[12]**

- **Artefacto:** es una información que es utilizada o producida mediante un proceso de desarrollo de software. Pueden ser artefactos un modelo, una descripción o un software. Los artefactos de UML se especifican en forma de diagramas, éstos, junto con la documentación sobre el sistema constituyen los artefactos principales que el modelador puede observar. **[12]**

- **Estado:** Los estados son los ladrillos de los diagramas de estado. Un estado pertenece a exactamente una clase y representa un resumen de los valores y atributos que puede tener la clase. Un estado UML describe el estado interno de un objeto de una clase particular**. [12]**

Tenga en cuenta que no todos los cambios en los atributos de un objeto deben estar representados por estados, sino únicamente aquellos cambios que pueden afectar significativamente a la forma de funcionamiento del objeto. **[15]**

- **Actividad**: es un único paso de un proceso**. [22]**

2.4.3 Diagramas de UML

Un diagrama es una representación gráfica de una colección de elementos del modelo, que habitualmente toma forma de grafo donde los arcos que conectan sus vértices son las relaciones entre los objetos y los vértices se corresponden con los elementos del modelo.

Los distintos puntos de vista de un sistema real que se quieren representar para obtener el modelo se dibuja de forma que se resaltan los detalles necesarios para entender el sistema**. [15]**

- **Diagramas de Casos de Uso**: Un diagrama de casos de uso es un diagrama que muestra un conjunto de casos de uso con sus relaciones y los actores implicados. Es un diagrama que sirve para modelar la vista estática de un programa. La vista estática nos permite visualizar el comportamiento externo del programa; de esta forma se consigue conocer qué es lo que debe hacer el programa independientemente de cómo lo haga y sabremos los elementos que interactúan con el sistema. Los elementos implicados en un diagrama de casos de uso son los casos de uso, las relaciones y los actores. Las relaciones y los casos de uso ya han sido explicados anteriormente y el papel del actor también ha sido comentado pero merece la pena detallarlo más: Un actor es un rol que

interactúa con el sistema. Se define como rol porque un actor puede ser tanto un usuario de la aplicación como otro sistema o dispositivos externos.

Los diagramas de casos de uso se utilizan para ilustrar los requerimientos del sistema al mostrar cómo reacciona una respuesta a eventos que se producen en el mismo. En este tipo de diagrama intervienen algunos conceptos nuevos: un actor es una entidad externa al sistema que se modela y que puede interactuar con él; un ejemplo de actor podría ser un usuario o cualquier otro sistema. Las relaciones entre casos de uso y actores pueden ser las siguientes: **[15]**

- Un actor se comunica con un caso de uso.
- Un caso de uso extiende otro caso de uso.
- Un caso de uso usa otro caso de uso.

- **Diagramas de Secuencia**: Un diagrama de secuencia es un diagrama de interacción UML. Estos diagramas muestran la secuencia de mensajes que se van lanzando los objetos implicados en una determinada operación del programa. Dentro del diagrama los objetos se alinean en el eje X respetando su orden de aparición. En el eje Y se van mostrando los mensajes que se envían, también respetando su orden temporal. Cada objeto tiene una línea de vida donde se sitúa su foco de control. El foco de control es un rectángulo que representa el tiempo durante el que un objeto está activo ejecutando una acción. Con este sencillo esquema podemos visualizar la comunicación y sincronización bajo un estricto orden temporal de los objetos implicados en las distintas funcionalidades de un sistema. **[3]**

- **Diagrama de Clases de Análisis:** Es utilizado por los desarrolladores de software para determinar los requerimientos funcionales, considerando una o

varias clases, o sub-sistemas del sistema a desarrollar. Los casos de uso se describen mediante clases de análisis y sus objetos. El diagrama de clases de análisis se construye examinando los casos de usuarios, cerrando sus reacciones e identificando los roles de los clasificadores. **[15]**

- **Diagrama de Clases de Diseño:** Se emplean para modelar la estructura estática de las clases en el sistema, sus tipos, sus contenidos y las relaciones que se establecen entre ellos. A través de este diagrama se definen las características de cada una de las clases, interfaces, colaboraciones y relaciones de dependencia y generalización**. [22]**

- **Diagramas de actividad**: Son similares a los diagramas de flujo de otras metodologías Orientadas a Objetos. En realidad se corresponden con un caso especial de los diagramas de estado donde los estados son estados de acción (estados con una acción interna y una o más transiciones que suceden al finalizar esta acción, o lo que es lo mismo, un paso en la ejecución de lo que será un procedimiento) y las transiciones vienen provocadas por la finalización de las acciones que tienen lugar en los estados de origen. Siempre van unidos a una clase o a la implementación de un caso de uso o de un método (que tiene el mismo significado que en cualquier otra metodología OO). Los diagramas de actividad se utilizan para mostrar el flujo de operaciones que se desencadenan en un procedimiento interno del sistema**.[16]**

- **Diagramas de Implementación**: Se derivan de los diagramas de proceso y módulos de la metodología de Booch, aunque presentan algunas modificaciones. Los diagramas de implementación muestran los aspectos

físicos del sistema. Incluyen la estructura del código fuente y la implementación, en tiempo de implementación. Existen dos tipos:

- **Diagramas de componentes:** Muestra la dependencia entre los distintos componentes de software, incluyendo componentes de código fuente, binario y ejecutable. Un componente es un fragmento de código software (un fuente, binario o ejecutable) que se utiliza para mostrar dependencias en tiempo de compilación.

- **Diagrama de plataformas o despliegue:** Muestra la configuración de los componentes hardware, los procesos, los elementos de procesamiento en tiempo de ejecución y los objetos que existen en tiempo de ejecución. En este tipo de diagramas intervienen nodos, asociaciones de comunicación, componentes dentro de los nodos y objetos que se encuentran a su vez dentro de los componentes. Un nodo es un objeto físico en tiempo de ejecución, es decir una máquina que se compone habitualmente de, por lo menos, memoria y capacidad de procesamiento, a su vez puede estar formado por otros componentes.**[16]**

- **Modelo de Dominio*:*** es un artefacto de la disciplina de análisis, construido con las reglas de UML durante la fase de concepción, en la tarea construcción del modelo de dominio, presentado como uno o más diagramas de clases y que contiene, no conceptos propios de un sistema de software sino de la propia realidad física.

  Los modelos de dominio pueden utilizarse para capturar y expresar el entendimiento ganado en un área bajo análisis como paso previo al diseño de un sistema, ya sea de software o de otro tipo. Similares a los mapas mentales

utilizados en el aprendizaje, el modelo de dominio es utilizado por el analista como un medio para comprender el sector industrial o de negocios al cual el sistema va a servir. **[22]**

- **Diagrama de Paquetes**: Los diagramas de paquetes se usan para reflejar la organización de paquetes y sus elementos. Cuando se usan para representaciones, los diagramas de paquete de los elementos de clase se usan para proveer una visualización de los espacios de nombres. Los elementos contenidos en un paquete comparten el mismo espacio de nombre, el hecho de compartir espacios de nombres requiere que los elementos contenidos en un espacio de nombre específico tengan nombres únicos. Los paquetes se pueden construir para representar relaciones tanto físicas como lógicas. **[3]**

2.4.4 Metodología OMT

La metodología OMT (*Object Modeling Technique*) fue creada por James Rumbaugh y Michael Blaha en 1991, mientras James dirigía un equipo de investigación de los laboratorios General Electric**. [15]**

OMT es una de las metodologías de análisis y diseño orientadas a objetos, más madura y eficiente que existe en la actualidad. **[15]**
Las fases que conforman a la metodología OMT son:

- **Análisis**. El analista construye un modelo del dominio del problema, mostrando sus propiedades más importantes. El modelo de análisis es una abstracción resumida y precisa de lo que debe de hacer el sistema deseado y no de la forma en que se hará. Los elementos del modelo deben ser conceptos del dominio de aplicación y no conceptos informáticos tales como estructuras

de datos. Un buen modelo debe poder ser entendido y criticado por expertos en el dominio del problema que no tengan conocimientos informáticos. **[12]**

- **Diseño del sistema**. El diseñador del sistema toma decisiones de alto nivel sobre la arquitectura del mismo. Durante esta fase el sistema se organiza en subsistemas basándose tanto en la estructura del análisis como en la arquitectura propuesta. Se selecciona una estrategia para afrontar el problema. **[16]**

- **Diseño de objetos**. El diseñador de objetos construye un modelo de diseño basándose en el modelo de análisis, pero incorporando detalles de implementación. El diseño de objetos se centra en las estructuras de datos y algoritmos que son necesarios para implementar cada clase. OMT describe la forma en que el diseño puede ser implementado en distintos lenguajes (orientados y no orientados a objetos, bases de datos, etc.). **[15]**

- **Implementación**. Las clases de objetos y relaciones desarrolladas durante el análisis de objetos se traducen finalmente a una implementación concreta. Durante la fase de implementación es importante tener en cuenta los principios de la ingeniería del software de forma que la correspondencia con el diseño sea directa y el sistema implementado sea flexible y extensible. **[15]**

La metodología OMT emplea tres clases de modelos para describir el sistema:

- **Modelo de objetos.** Describe la estructura estática de los objetos del sistema (identidad, relaciones con otros objetos, atributos y operaciones). El modelo de objetos proporciona el entorno esencial en el cual se pueden situar el modelo dinámico y el modelo funcional. El objetivo es capturar aquellos

conceptos del mundo real que sean importantes para la aplicación. Se representa mediante diagramas de objetos.

- **Modelo dinámico.** Describe los aspectos de un sistema que tratan de la temporización y secuencia de operaciones (sucesos que marcan los cambios, secuencias de sucesos, estados que definen el contexto para los sucesos) y la organización de sucesos y estados. Captura el control, aquel aspecto de un sistema que describe las secuencias de operaciones que se producen sin tener en cuenta lo que hagan las operaciones, aquello a lo que afecten o la forma en que están implementadas. Se representa gráficamente mediante diagramas de estado.

- **Modelo funcional.** Describe las transformaciones de valores de datos (funciones, correspondencias, restricciones y dependencias funcionales) que ocurren dentro del sistema. Captura lo que hace el sistema, independientemente de cuándo se haga o de la forma en que se haga. Se representa mediante diagramas de flujo de datos. **[15]**

La OMT, por ejemplo, intenta abstraer la realidad utilizando tres clases de modelos OO: el modelo de objetos, que describe la estructura estática; el modelo dinámico, con el que describe las relaciones temporales entre objetos; y el modelo funcional que describe las relaciones funcionales entre valores. Mediante estas tres fases de construcción de modelos, se consigue una abstracción de la realidad que tiene en sí misma información sobre las principales características de ésta**. [15]**

Los modelos además, al no ser una representación que incluya todos los detalles de los originales, permiten probar más fácilmente los sistemas que modelan y determinar los errores.

Según se indica en la Metodología OMT (Rumbaugh), los modelos permiten una mejor comunicación con el cliente por distintas razones:

- Es posible enseñar al cliente una posible aproximación de lo que será el producto final.

- Proporcionan una primera aproximación al problema que permite visualizar cómo quedará el resultado.

- Reducen la complejidad del original en subconjuntos que son fácilmente tratables por separado. **[15]**

Se consigue un modelo completo de la realidad cuando el modelo captura los aspectos importantes del problema y omite el resto. Los lenguajes de programación que estamos acostumbrados a utilizar no son adecuados para realizar modelos completos de sistemas reales porque necesitan una especificación total con detalles que no son importantes para el algoritmo que están implementando.

En OMT se modela un sistema desde tres puntos de vista diferentes donde cada uno representa una parte del sistema y una unión lo describe de forma completa. En esta técnica de modelado se utilizó una aproximación al proceso de implementación de software habitual donde se utilizan estructuras de datos (modelo de objetos), las operaciones que se realizan con ellos tienen una secuencia en el tiempo (modelo dinámico) y se realiza una transformación sobre sus valores (modelo funcional). **[15]**

En este proyecto la OMT es utilizada para la descripción de todo el proceso de modelado.

2.5 TÉCNICAS DE DESARROLLO DE SISTEMAS DE OBJETOS (TDSO)

Existen diversas formas tradicionales de expresar la resolución programada de problemas a través del computador digital anteriores al uso de la orientación por objetos, entre ellas se tienen el refinamiento paso a paso, el diseño modular y estructurado, los algoritmos estructurados, el método deductivo, entre otros, que pueden ser utilizados independientemente unos de otros, aunque ellos, normalmente se usan en conjunto para analizar, diseñar y documentar sistemas programados. La TDSO está basada en la técnica OMT. **[3]**

En este proyecto la TDSO se utiliza para la definición del universo de clases, y para especificación formal de las clases y los métodos.

CAPÍTULO 3: FASE DE ANÁLISIS

3.1 INTRODUCCIÓN

En este capítulo se procederá a describir por medio de diagramas UML la fase de inicio del sistema a desarrollar el cual recibe el nombre de Sistema Divisor de Clases (S.D.C).

El propósito fundamental de este capítulo es guiar el desarrollo hacia el sistema correcto. Esto se consigue mediante una descripción de los requisitos del sistema, es decir, que las condiciones o capacidades que el sistema debe cumplir, sean suficientemente buenas como para que pueda llegarse a un acuerdo entre los usuarios y los desarrolladores sobre qué debe y qué no debe hacer el sistema.

Existen diferentes puntos de partida para la captura de los requisitos correctos y para la construcción del sistema deseado. Para ello, es necesario que los desarrolladores, posean un firme conocimiento del contexto en el que se emplaza el sistema, pudiendo tomar en consideración para lograrlo. **[16]**

3.2 REQUISITOS

En esta parte del capítulo se presentaran los requisitos necesarios para el buen funcionamiento del sistema SDC.

3.2.1 Requisitos Funcionales

Los requisitos funcionales, son aquellos que muestran la forma en la que el sistema interactúa con él o los usuarios finales, es decir, se toman en cuenta los procesos y métodos que se necesitan para ejecutar la interacción. A continuación se muestran los requisitos funcionales identificados en el sistema.

A continuación se enumeran los requisitos funcionales del sistema SDC:

- El sistema debe ser capaz de cargar por archivo de texto las coordenadas X e Y de los puntos pertenecientes al conjunto de entrenamiento T.

- El sistema debe permitir que el usuario visualice la solución obtenida por la aplicación tanto por medio del Algoritmo Genético como por medio del Algoritmo Backpropagation.

- El sistema debe guardar en un archivo de texto las soluciones obtenidas.

- El sistema debe ser capaz de dar una solución factible al problema planteado por el usuario.

3.2.2 Requisitos No Funcionales

Los requisitos no funcionales especifican propiedades del sistema, como restricciones del entorno o de la implementación, rendimiento, dependencias de la plataforma, facilidad de mantenimiento, extensibilidad, y fiabilidad.

Los requisitos que no son vitales para el funcionamiento del sistema SDC:

- El sistema debe contar con una interfaz gráfica agradable de tal forma que, permita interactuar, de forma fácil, segura y cómoda; entre los usuarios y el sistema.

- El sistema debe ser realizado bajo plataforma de software libre basándose en el decreto presidencial 3.390, con el objeto de permitir que el código fuente pueda ser modificado sin restricciones legales además de reducir el costo de desarrollo.

- La estructura y diseño del sistema deben adaptarse fácilmente a cualquier cambio o mejora en los mismos.

3.2.3 Requisitos de Software

- Sistema Operativo Windows XP service pack 2 o superior o Sistema Operativo Linux.
- Java Runtime Environment (JRE) 1.7.

3.2.4 Requisitos de Hardware

- Pentium 4 3.0 GHz o Superior
- 512 MB de memória RAM o superior.

3.3 RIESGOS DEL SISTEMA

En todo desarrollo de sistema es importante considerar los riesgos; entiéndase por riesgo la variable del proyecto que pone en peligro o impide el éxito del mismo. Los riegos constituyen la probabilidad de que un proyecto sufra sucesos no deseables.

- El Algoritmo Genético no consiga una solución al problema planteado por el usuario.

  La prioridad de este riesgo es crítica ya que de darse esto el sistema no estaría cumpliendo con los requisitos del usuario. La solución sería que luego de cierto tiempo de ejecución del programa este se detenga y de cómo solución la mejor obtenida hasta ese momento.

- El Backpropagation no encuentre una solución al problema con la arquitectura suministrada por el usuario.

  La prioridad es crítica, y se soluciona deteniendo la ejecución de la aplicación y enviando un mensaje de advertencia al usuario donde se le indique la razón por la cual no se encontró solución al problema.

- El problema es muy complejo y la aplicación tarda mucho tiempo en conseguir una solución.

  La prioridad es crítica, la solución sería que luego de un tiempo determinado de ejecución de la aplicación esta se detenga y muestre la mejor solución encontrada hasta ese momento o un mensaje al usuario donde se le indique que no se encontró solución al problema suministrado.

- El archivo de entrada no tiene el formato correcto, la prioridad es secundaria, la solución sería indicarle al usuario que visite la ayuda del sistema donde se le indicaría el formato correcto del archivo.

3.4 DIAGRAMA DE DOMINIO

Un modelo de dominio es una representación visual de las clases conceptuales u objetos del mundo real en un dominio de interés. También se les domina modelos conceptuales, modelo de objetos del dominio y modelos de objetos de análisis. **[20]**

Para la descripción del modelo de dominio debe realizarse diagramas de clases basado en Lenguaje de Modelado UML. En la figura 3.1 se representan las clases más importantes del modelo de dominio.

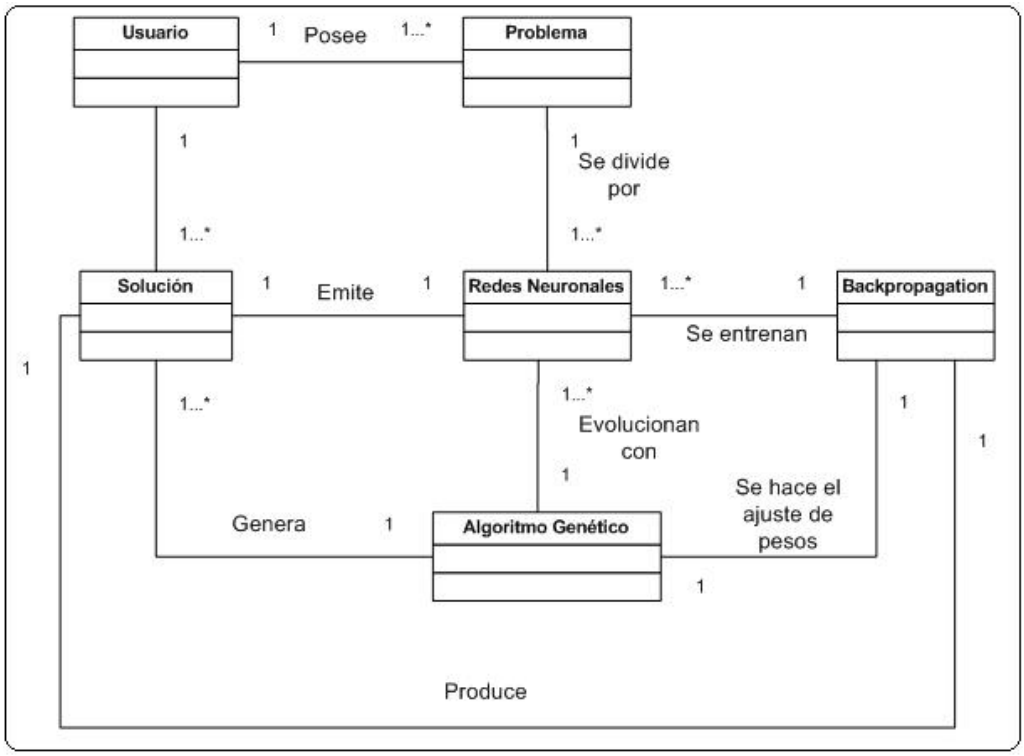


Figura 3.1 Diagrama de Dominio

3.4.1 Glosario de Términos del Diagrama de Dominio

- **Usuario:** es quien introduce le archivo de entrada con los datos del problema que desea resolver.
- **Problema:** este está contenido en el archivo de entrada suministrado por el usuario.
- **Backpropagation:** en esta clase se resuelve el problema por medio de la red Backpropagation, donde el usuario debe establecer, además de las clases, la arquitectura que tendrá la Red Neuronal que debe resolver el problema dado por el usuario.
- **Redes Neuronales:** en esta clase se establece la arquitectura inicial de la Red Neuronal, así como también de manera aleatoria los pesos iniciales de la misma.
- **Algoritmo Genético:** en esta clase se entrena la Red Neuronal establecida inicialmente hasta que logre resolver de manera factible el problema planteado por el usuario.
- **Solución:** aquí el usuario visualiza la solución obtenida y puede almacenarla en un archivo de salida, también si el usuario desea puede ver la generalización de la red obtenida.

3.5 DIAGRAMA DE CASO DE USO

El modelo de casos de uso incluye los casos de usos y actores. Los casos de uso son fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. **[16]**

En la figura 3.2 se muestra el caso de uso del sistema SDC.

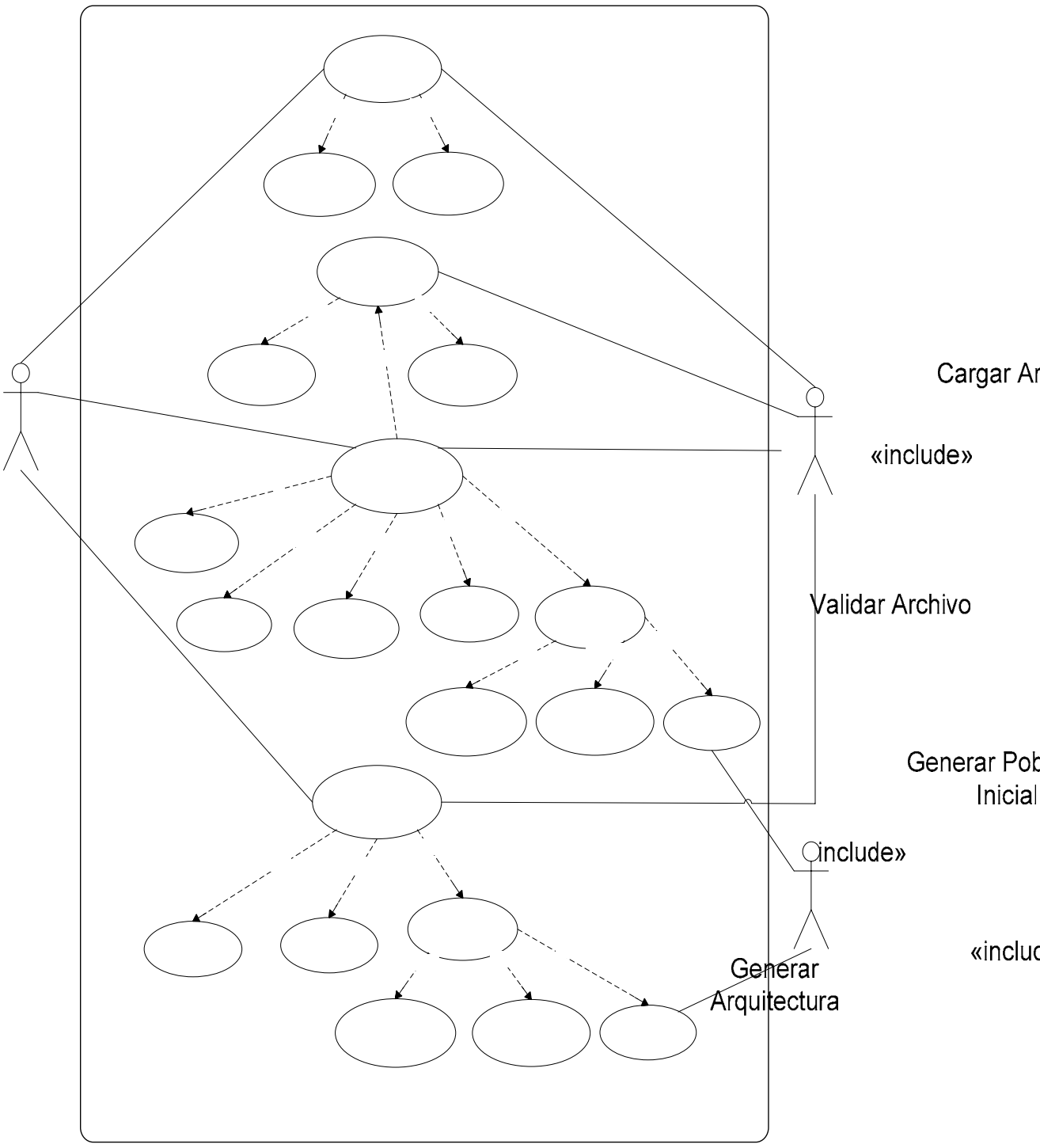


Figura 3.2 Caso de Uso del sistema SDC

3.5.1 Identificación de Actores

Los actores representan a todos aquellos que de una u otra manera interactúan con el sistema. Estos pueden ser los usuarios (personas que hacen uso del sistema) y también pueden estar representados por sistemas o dispositivos externos que se comunican con el sistema. Es importante realizar la identificación de los actores ya que ello permite obtener una visión del entorno externo del sistema. **[16]**

A continuación se describe cada uno de los actores que forman parte del ambiente en que se desempeña el sistema.

- **Usuario:** como su nombre lo indica es quien va a usar el sistema, este se encarga de suministrar al sistema el problema por medio de un archivo de texto y de elegir con que método quiere resolverlo, eligiendo entre una Red Neuronal evolucionada por Algoritmo Genético o por una red entrenada por el algoritmo Backpropagation.

- **Archivo de Entrada:** se trata del archivo de texto que contiene las coordenadas de los puntos que representan a las clases que se desean dividir, estos puntos serán utilizados como la entrada a la Red Neuronal que se establezca para resolver el problema. Este archivo es la representación del conjunto de entrada T.

- **Plantilla**: se trata del archivo imagen que contiene la plantilla original del problema.

3.5.2 Identificación de Casos de Uso

- **Caso de Uso 1 : Cargar Archivo**

  Este caso de uso se encarga de leer el archivo que contiene las coordenadas de los puntos de las clases, este archivo es suministrado por el usuario y consta de 2 casos de usos los cuales son:

  - **Caso de Uso 1.1: Validar Archivo**

    Este caso de uso se encarga de comprobar que el archivo contenga el formato que requiere el sistema, en caso contrario se le emite un mensaje de advertencia al usuario para que haga las modificaciones pertinentes al archivo.

  - **Caso de Uso 1.2: Dibujar Puntos**

    Una vez que se verifico el archivo se procede a dibujar los puntos que representan a las clases, para luego crear la matriz que contiene la información de los individuos de la Red Neuronal.

- **Caso de Uso 2: Generar Población Inicial**

  Este caso de uso, como lo indica su nombre, se encarga de generar la población inicial, la primera arquitectura de la Red Neuronal y los pesos de las conexiones de las neuronas de la Red. Este caso de uso está formado por:

  - **Caso de Uso 2.1: Generar Arquitectura**

    Aquí se procede a crear la primera Red Neuronal que luego será entrenada por el Algoritmo Genético para conseguir una solución al problema. Aquí se establecerán las neuronas de la capa de entrada,

capa oculta y de la capa de salida así como las conexiones entre las neuronas de las distintas capas.

- **Caso de Uso 2.2: Generar Pesos Iniciales**

  En este caso de uso se generan de manera aleatoria los pesos de las conexiones entre las neuronas, estos peso están formados por un valor entre -1 y 1.

- **Caso de Uso 3: Evolucionar Redes Neuronales**

  En este caso de uso se procede a aplicar el Algoritmo Genético para el entrenamiento de la Red Neuronal establecida anteriormente con el fin de obtener una solución para resolver el problema, todo esto se realiza por medio de los siguientes casos de uso:

  - **Caso de Uso 3.1: Buscar Fitness**

    Aquí se les calcula el fitness a todos los individuos de la población inicial, para encontrar aquel con mejor fitness que optimice la solución al problema.

  - **Caso de Uso 3.2: Seleccionar Individuos**

    En este caso de uso se seleccionan los individuos que el Algoritmo Genético va a evolucionar con el fin de optimizar la Red Neuronal.

  - **Caso de uso 3.3: Generar Cruce y Mutación**

    Aquí luego de seleccionar a los individuos se procede a cruzarlos y mutarlos por medio del Algoritmo Genético.

- **Caso de Uso 3.4: Generar Nueva Población**

  En este caso de uso se crea una nueva población con los individuos resultantes de la mutación, luego con esta población se repite el mismo proceso de evolución hasta que se encuentre una solución.

- **Caso de Uso 3.5: Generar Solución**

  Aquí luego de obtener la Red Neuronal que resuelve de manera óptima al problema, se procede a:

  - **Caso de Uso 3.5.1: Generar Archivo de Salida**

    Aquí se guarda en un archivo de texto la solución obtenida.
  - **Caso de Uso 3.5.2: Visualizar Solución**

    Luego de encontrar la solución se le muestra al usuario la Red Neuronal que resuelve el problema y se le muestra la generalización arrojada por la Red.
  - **Caso de Uso 3.5.3: Mostrar Generalización**

    Aquí se procesa y se visualiza la generalización la cual se hace sobre la base de la plantilla que representa al problema planteado por el usuario.

- **Caso de Uso 4: Aplicar Backpropagation**

  En este caso de uso se resuelve el problema por medio de una Red Backpropagation.

  - **Caso de Uso 4.1: Establecer Arquitectura**

    En este caso de uso se le solicita al usuario que indique cual sería la arquitectura de la Red Neuronal que él cree puede resolver el problema.

- **Caso de Uso 4.2: Generar Pesos**

  Aquí luego que el usuario indico la arquitectura, se procede a generar los pesos de manera aleatoria de las conexiones de la Red Neuronal.

- **Caso de Uso 4.3: Generar solución**

  Aquí el sistema le envía al usuario la solución obtenida por la Red Backpropagation y también se encarga de la generalización.

  - **Caso de Uso 4.3.1: Generar Archivo de Salida**

    Aquí se guarda en un archivo de texto la salida obtenida.
  - **Caso de Uso 4.3.2: Visualizar Solución**

    Aquí se le muestra al usuario el resultado obtenido por la Red Backpropagation y se le muestra la generalización arrojada por la Red.
  - **Caso de Uso 4.3.3: Mostrar Generalización**

    Aquí se procesa y se visualiza la generalización la cual se hace en base a la plantilla que representa al problema planteado por el usuario.

3.6 DIAGRAMA DE ACTIVIDADES

Son similares a los diagramas de flujo de otras metodologías Orientadas a Objetos. Con estos se representa el funcionamiento en forma general de las principales operaciones que realizará el sistema, la secuencia en que se ejecutan y cómo el sistema reacciona y se comporta en respuesta a los procesos de realización de los casos de uso. **[27]**

3.6.1 Diagrama de Actividades: Cargar Archivo

Es la actividad que se realiza para cargar el archivo, recibe la solicitud del usuario, y se evalúan los datos del archivo para comprobar que éste posea el formato correcto, de ser así se procede a graficar las coordenadas contenidas en el archivo y luego se genera la matriz de individuos, en caso de ser incorrecto el formato, se le mostrara un mensaje de error al usuario.

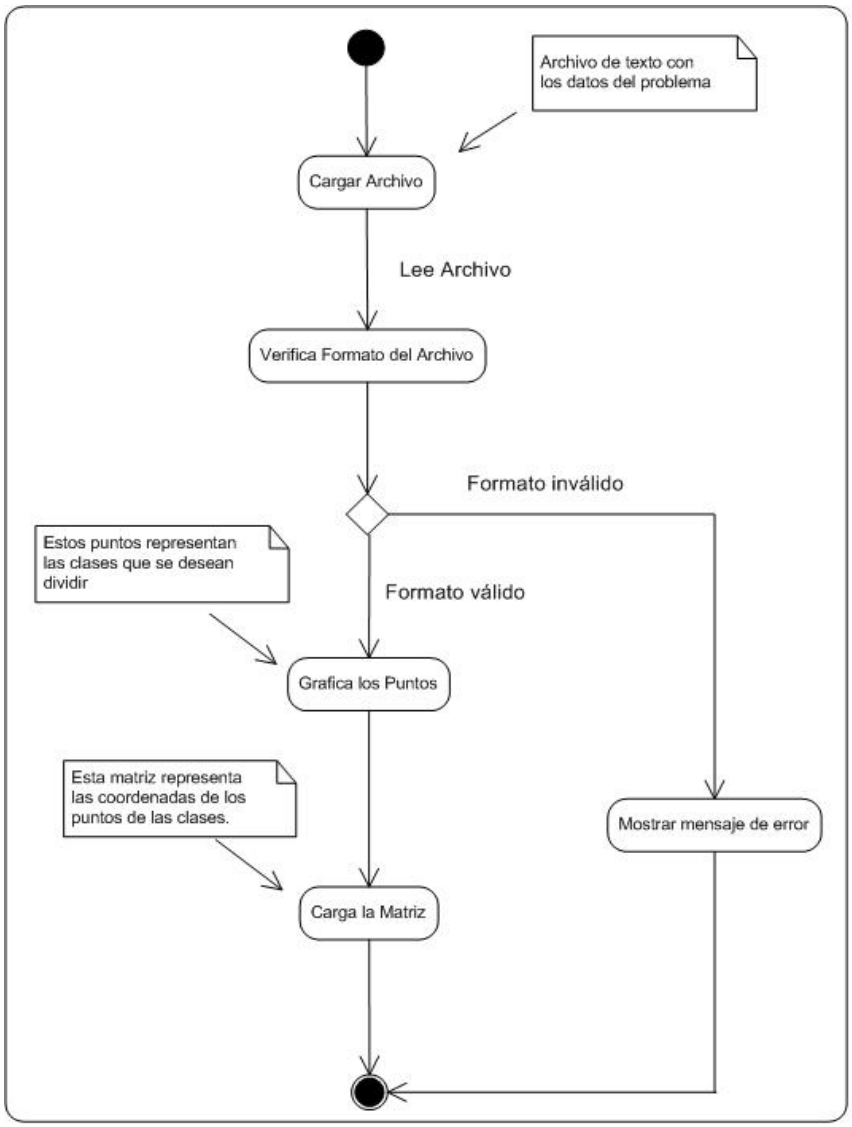


Figura 3.3 Diagrama de Actividades Cargar Archivo

3.6.2 Diagrama de Actividades Generar Población Inicial

En esta actividad se procede a generar la arquitectura inicial que tendrá la Red Neuronal inicial, donde se tendrá una capa de entrada que contiene 2 neuronas, dos capas ocultas con 5 neuronas como máximo, y una capa de salida con una sola salida, y también se generan los pesos iniciales de las conexiones entre las neuronas, por último se genera la población inicial del Algoritmo Genético.

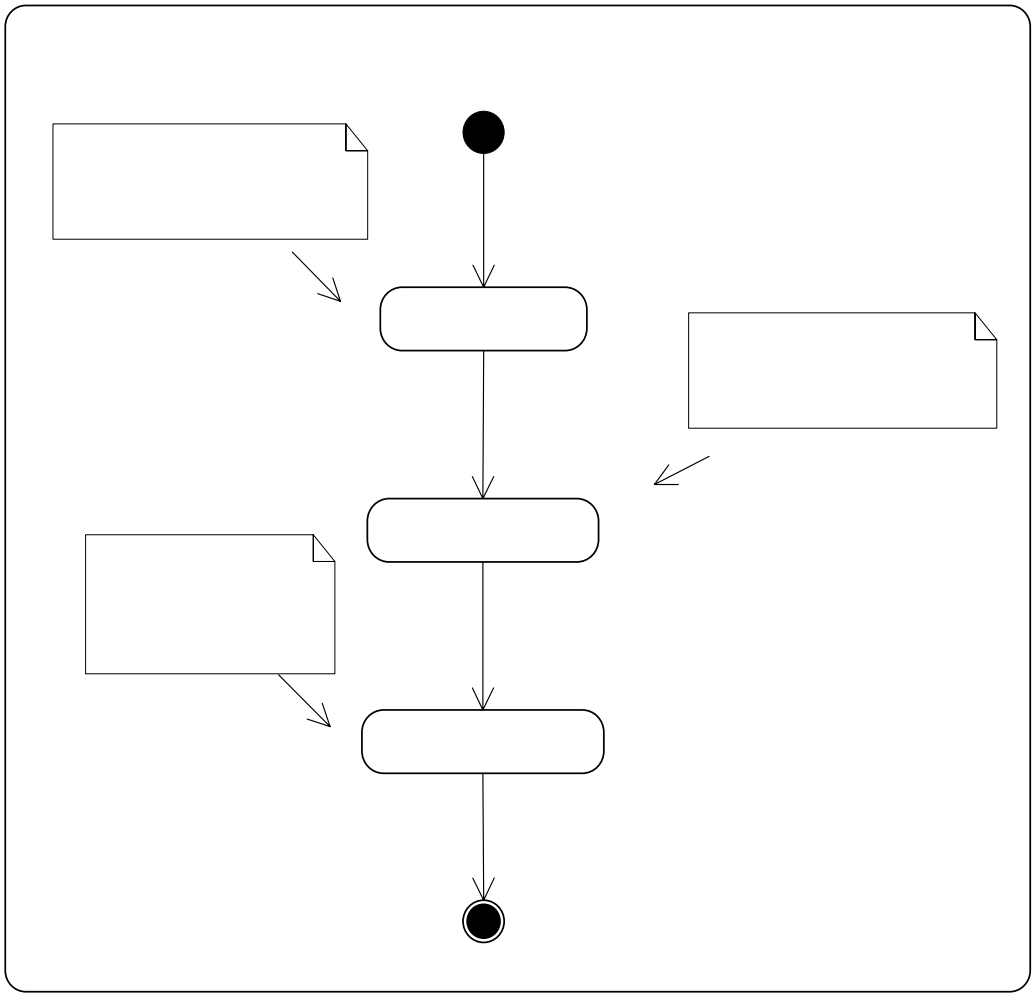


Figura 3.4 Diagrama de Actividades Generar Población Inicial

3.6.3 Diagrama de Actividades Evolucionar Redes Neuronales

A cada individuo de la población inicial se le calcula el fitness para luego ir seleccionando a los individuos a los que se les aplicará el cruce y mutación para generar una nueva población más apta para la resolución del problema, este proceso se realiza hasta que se encuentre la población que resuelva el problema. En la figura 3.5 se muestra el diagrama de Actividades Evolucionar Redes Neuronales.

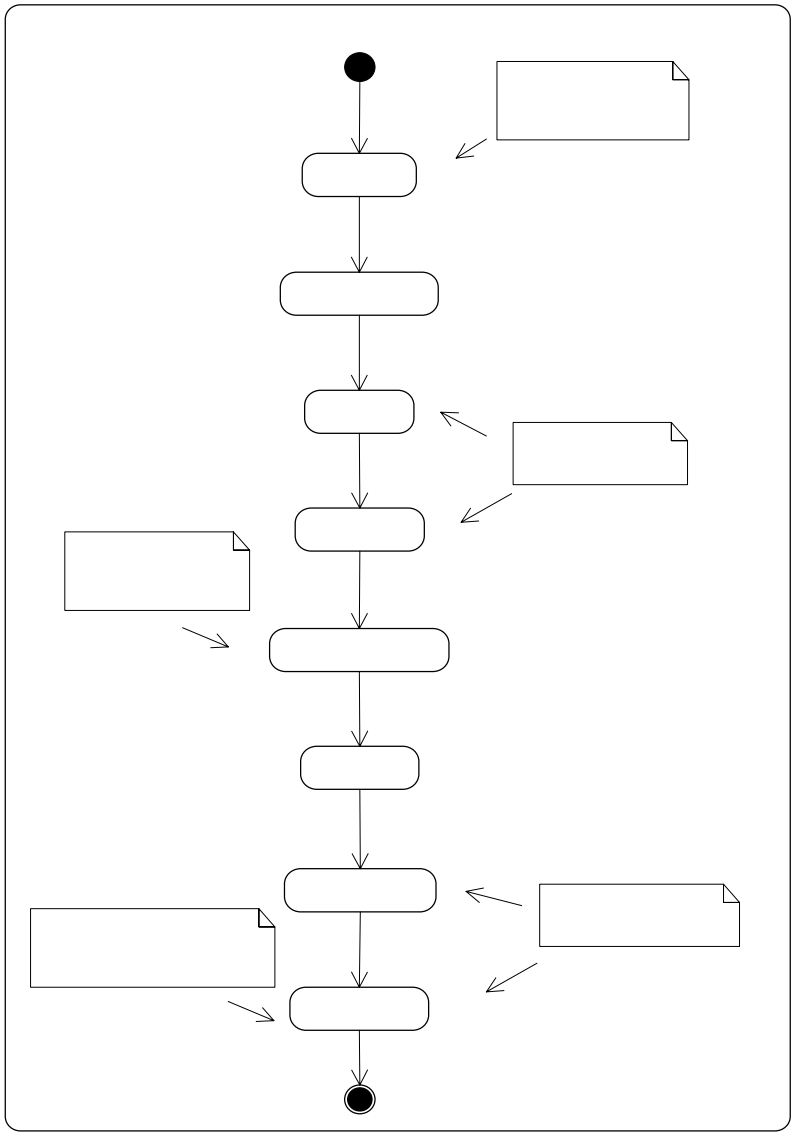


Figura 3.5 Diagrama de Actividades Evolucionar Redes Neuronales

3.6.4 Diagrama de Actividades Aplicar Backpropagation

Esta actividad corresponde a otra forma de resolver el problema, aquí luego de que el usuario cargue el archivo de entrada, se le solicita que ingrese el número de neuronas que desea en la capa oculta de la Red Neuronal, y en base a este número se genera la arquitectura de la red, que tendrá una capa de entrada de dos neuronas, una capa oculta con la cantidad de neuronas elegida por el usuario y una capa de salida con una sola neurona, luego de esto se procede a generar los pesos de las conexiones entre las neuronas, para calcular la solución se aplican todas las ecuaciones que corresponden al aprendizaje Backpropagation para comprobar si esta arquitectura resuelve o no el problema. En la figura 3.6 se muestra el diagrama de Actividades Aplicar Backpropagation.

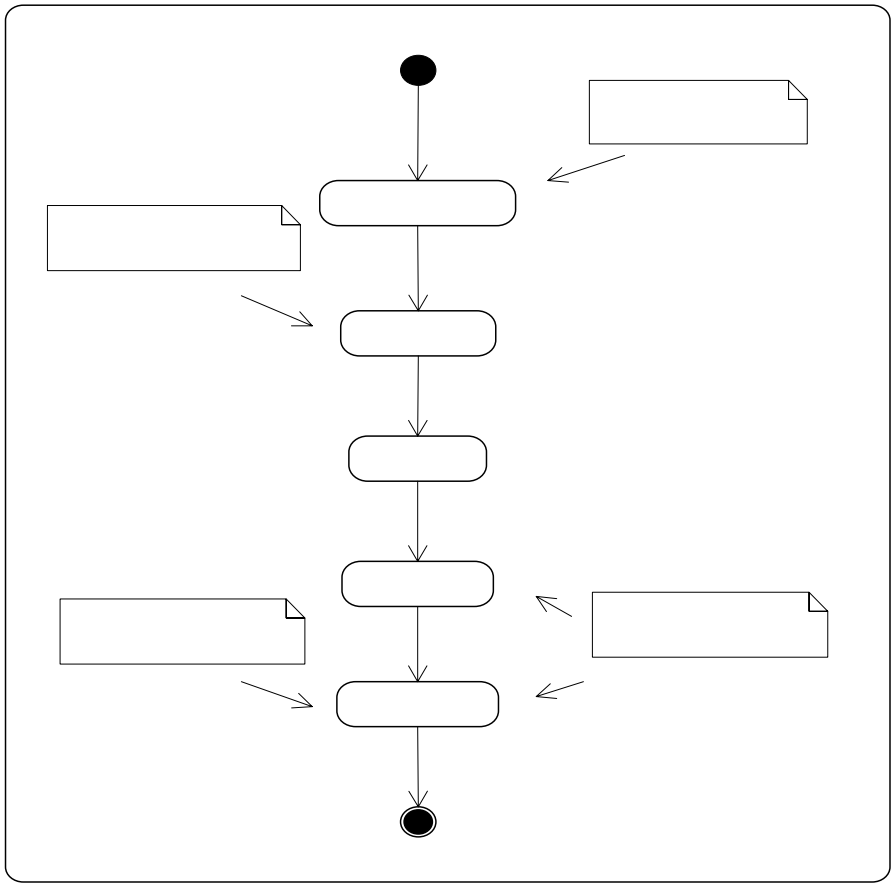


Figura 3.6 Diagrama de Actividades Aplicar Backpropagation

CAPITULO 4: FASE DE DISEÑO

4.1 DIAGRAMA DE CLASE DE DISEÑO

En este diagrama se describe la estructura del sistema SDC, mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases de diseño son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro.

En este diagrama se especifican el nombre de la clase tal como se encuentra en la aplicación, el nombre y tipo de los atributos, donde se indicará con signo más (+) si es público y con un signo menos (-) si es privado, luego se nombran los métodos de la clase, indicando también si son públicos o privados. Así como también se debe indicar la relación entre las clases. **[12]**

En la figura 4.1 se muestra un ejemplo del formato y en la figura 4.2 se muestra el diagrama de caso de diseño del sistema SDC

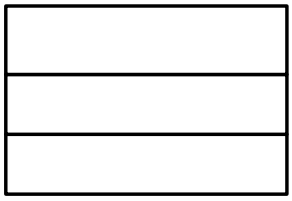


Figura 4.1 Estructura del Diagrama clase de diseño **[12]**

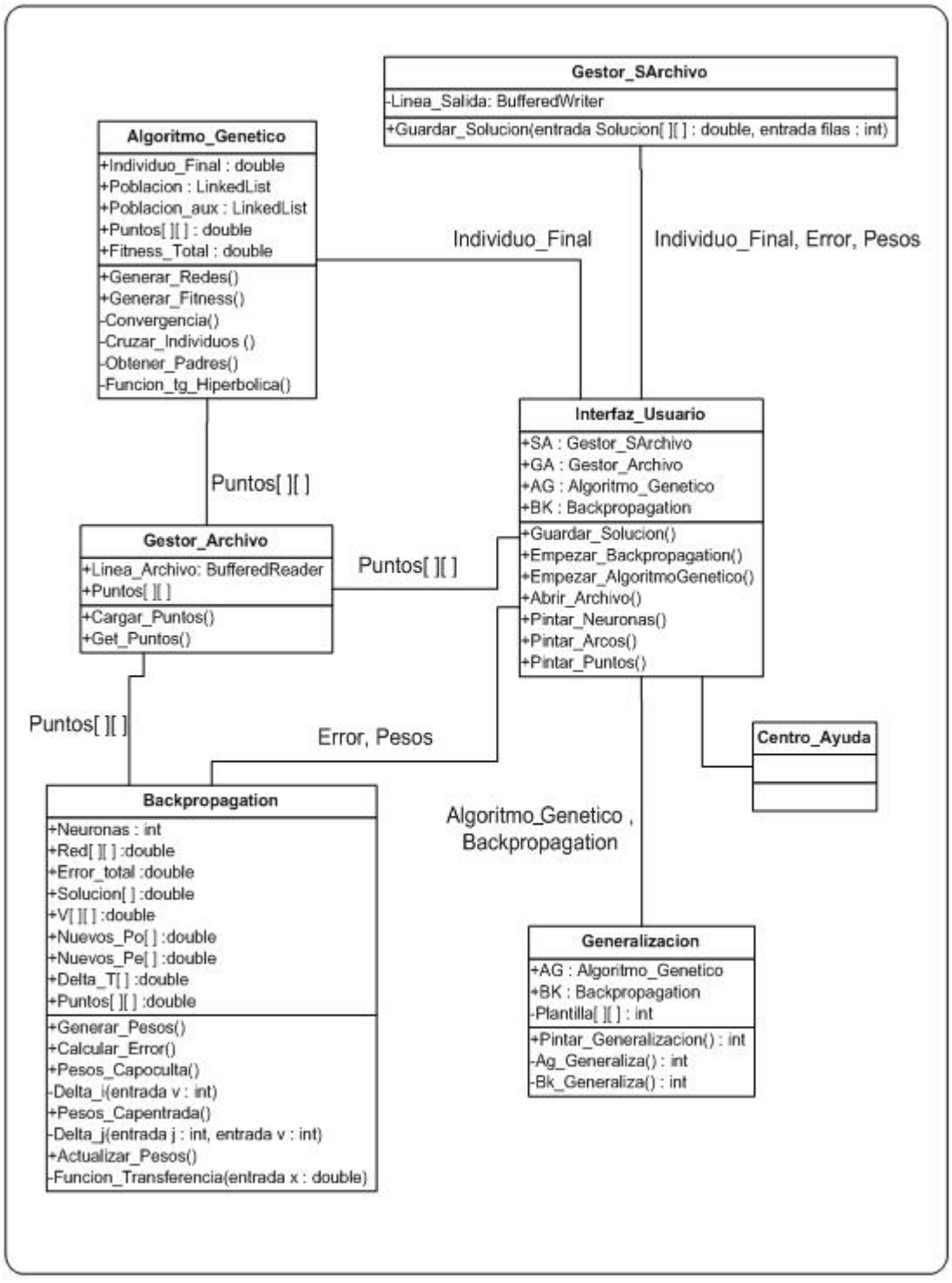


Figura 4.2 Diagrama de Clase de Diseño del sistema SDC

4.2 DIAGRAMA DE SECUENCIA

Éste es un diagrama de interacción UML. Estos diagramas muestran la secuencia de mensajes que se van lanzando los objetos implicados en una determinada operación del programa.

Dentro del diagrama los objetos se alinean en el eje X respetando su orden de aparición. En el eje Y se van mostrando los mensajes que se envían, también respetando su orden temporal. Cada objeto tiene una línea de vida donde se sitúa su foco de control.

Con este sencillo esquema podemos visualizar la comunicación y sincronización bajo un estricto orden temporal de los objetos implicados en las distintas funcionalidades de un sistema.

A continuación se muestran los diagramas de secuencia correspondientes a los casos de uso fundamentales para el sistema SDC. **[16]**

4.2.1 Diagrama de Secuencia para el Caso de Uso Cargar Archivo

En el diagrama de secuencia del caso de uso Cargar Archivo, el proceso de comunicación de los objetos que interactúan, se establece a través de los mensajes que se envían entre ellos. Dicho proceso se describe de la siguiente manera: (**Véase *figura 4.3.***).

1. El actor Usuario realiza un llamado al objeto **IU Principal** por medio del mensaje *Solicitud Cargar Archivo.*
2. El actor Archivo de Entrada le envía al objeto **Gestor Archivo** los datos del archivo de entrada.

3. El objeto **IU Principal** activa al objeto **Gestor Archivo** por medio del mensaje *Leer Archivo*.
4. El **Gestor Archivo** se hace un llamado a sí mismo con el mensaje *Validar Archivo.*
5. El **Gestor Archivo** activa a **IU Principal** por medio del mensaje *datos archivo entrada* para enviarle los datos leídos en el archivo.
6. El objeto **IU Principal** se envía a sí mismo el mensaje *Dibujar Puntos.*
7. El objeto **IU Principal** le envía el mensaje *Mostar Puntos* al Usuario.

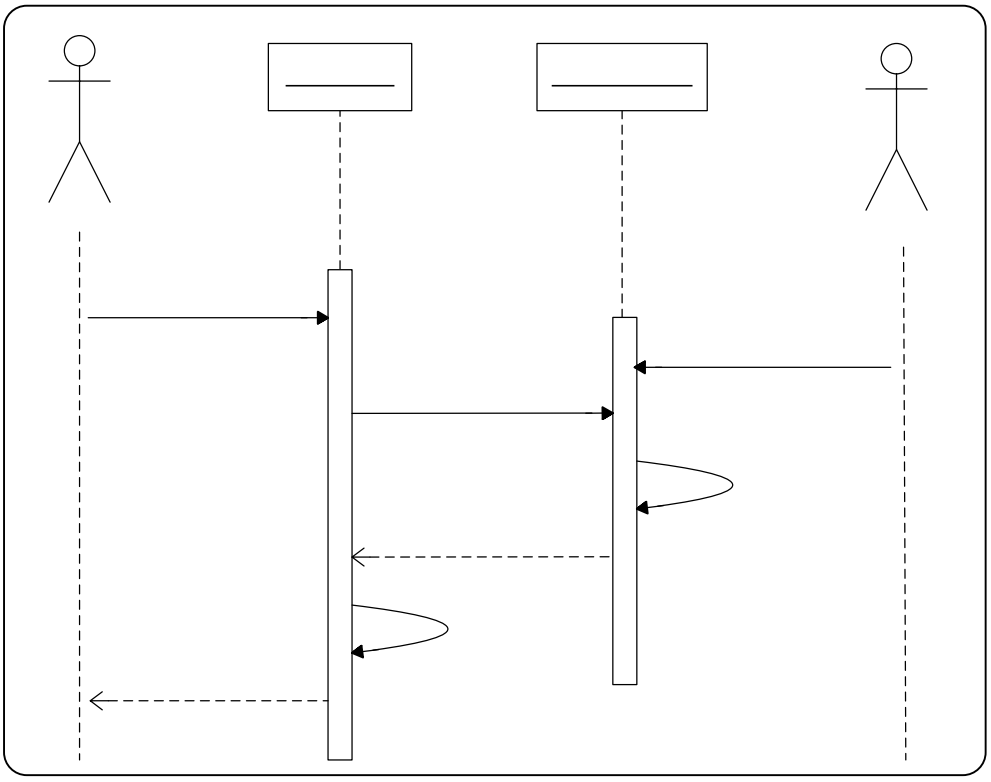


Figura 4.3 Diagrama de Secuencia para el Caso de Uso Cargar Archivo

IU Principal

Usuario

4.2.2 Diagrama de Secuencia para el Caso de Uso Generar Población Inicial

En el diagrama de secuencia del caso de uso Generar Población Inicial, el proceso de comunicación de los objetos que interactúan, se establece a través de los mensajes que se envían entre ellos. Dicho proceso se describe de la siguiente manera: (**Véase *figura 4.4.***).

1. El actor Usuario activa al objeto **IU Principal** con el mensaje *Activar Algoritmo Genético.*
2. El objeto **IU Principal** activa al **Gestor Red Neuronal** con el mensaje *Generar Población Inicial*.
3. El actor Archivo Entrada le envía al **Gestor Red Neuronal** los puntos de las clases por medio del mensaje *Puntos.*
4. El **Gestor Red Neuronal** se envía a sí mismo el mensaje *Generar Arquitectura.*
5. El **Gestor Red Neuronal** se envía a sí mismo el mensaje *Generar Pesos Iniciales.*

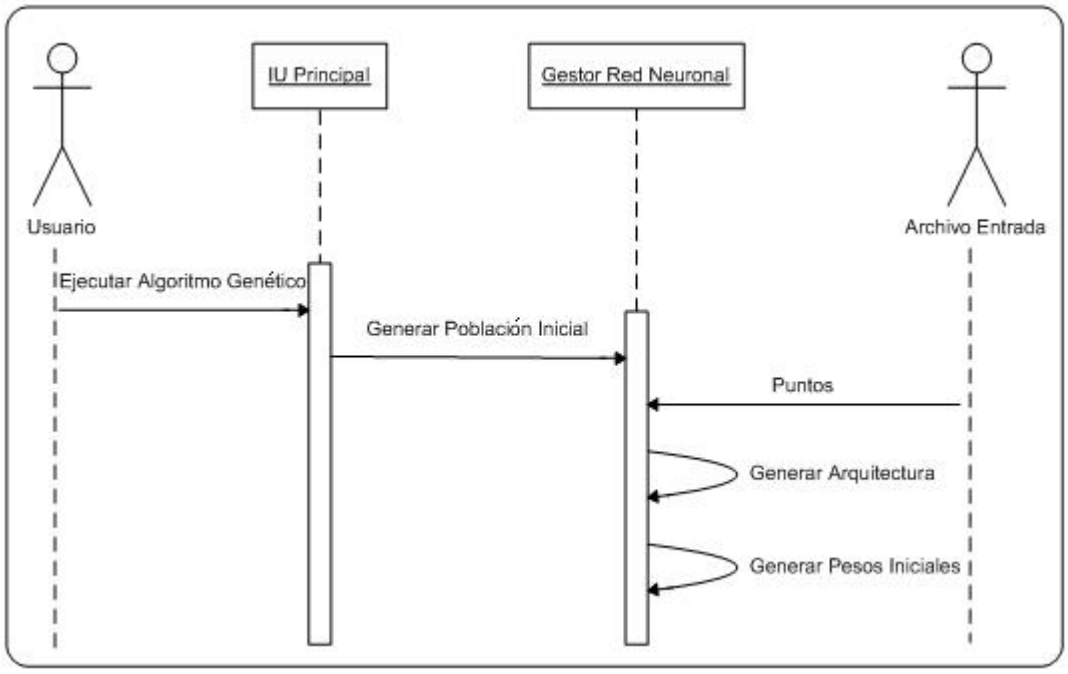


Figura 4.4 Diagrama de Secuencia para el Caso de Uso Generar Población Inicial

4.2.3 Diagrama de Secuencia para el Caso de Uso Evolucionar Redes Neuronales

En el diagrama de secuencia del caso de uso Evolucionar Redes Neuronales, el proceso de comunicación de los objetos que interactúan, se establece a través de los mensajes que se envían entre ellos. Dicho proceso se describe de la siguiente manera: (**Véase *figura 4.5.***).

1. El objeto **Gestor Red Neuronal** activa al **Gestor Algoritmo Genético** con el mensaje *Arquitectura Inicial*.
2. El objeto **Gestor Red Neuronal** le envía al **Gestor Algoritmo Genético** el mensaje *Pesos Iniciales*.
3. El actor **Archivo de Entrada** le envía el mensaje *Puntos* al **Gestor Algoritmo Genético.**

4. El **Gestor Algoritmo Genético** se envía a sí mismo el mensaje *Buscar Fitness*.
5. El **Gestor Algoritmo Genético** se envía a sí mismo el mensaje *Seleccionar Individuos.*
6. El **Gestor Algoritmo Genético** se envía a sí mismo el mensaje *Generar Cruce y Mutación.*
7. El **Gestor Algoritmo Genético** se envía a sí mismo el mensaje *Generar Solución.*
8. El **Gestor Algoritmo Genético** le envía al **Gestor Solución** el mensaje *Solución.*
9. El **Gestor Solución** le envía al gestor **IU Principal** el mensaje *Solución.*
10. El **IU Principal** le envía al actor **Usuario** el mensaje *Visualiza Solución*.
11. El actor **Usuario** le envía al gestor **IU Principal** el mensaje *Mostrar Generalización.*
12. El actor **Plantilla** le envía al gestor **Gestor Solución** el mensaje *Imagen Inicial.*
13. El **IU Principal** le envía al **Gestor Solución** el mensaje *Solicitar Generalización.*
14. El **Gestor Solución** se envía a sí mismo el mensaje *Procesa Generalización.*
15. El **Gestor Solución** le envía a **IU Principal** el mensaje *Generalización.*
16. El **IU Principal** le envía al actor **Usuario** el mensaje *Visualiza Generalización.*

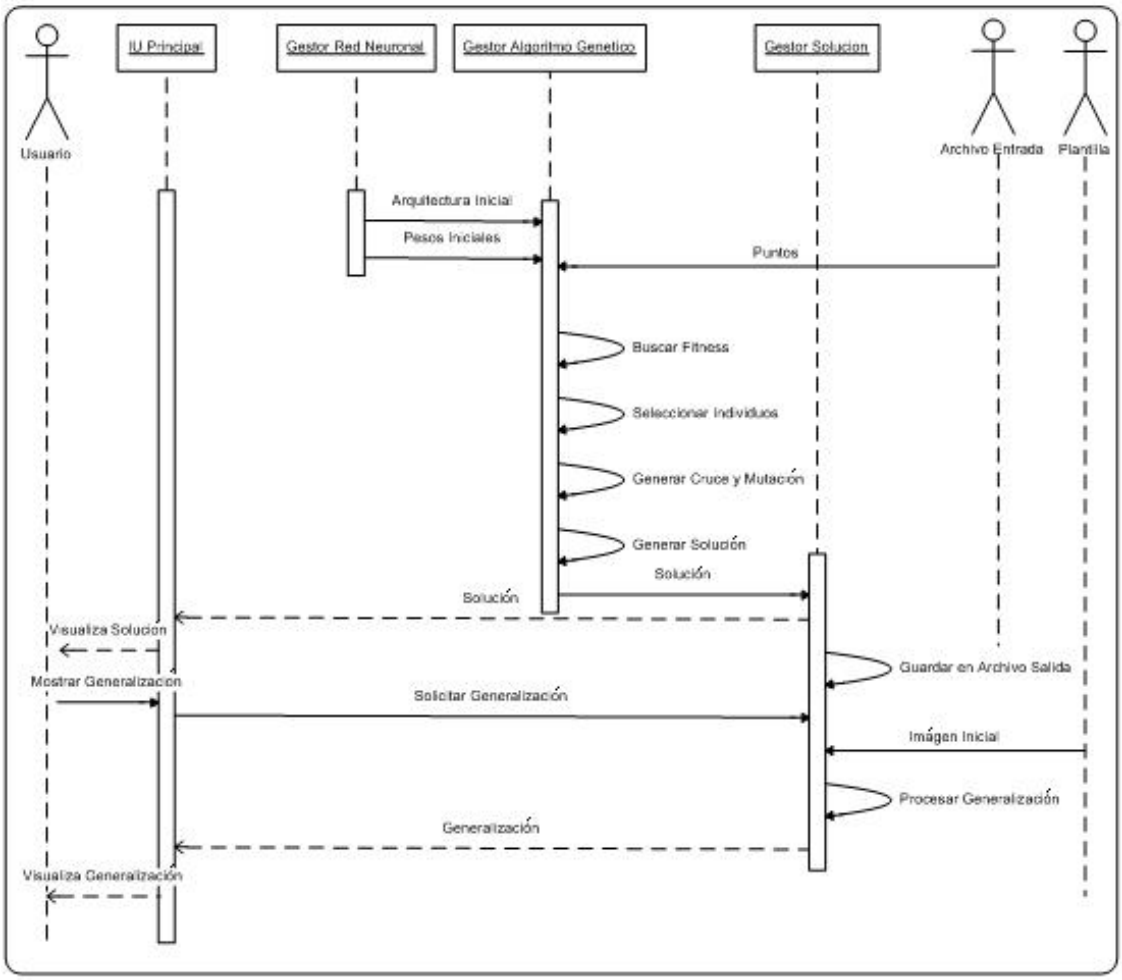


Figura 4.5 Diagrama de Secuencia para el Caso de Uso Evolucionar Red Neuronal

4.2.4 Diagrama de Secuencia para el caso de uso Aplicar Backpropagation

En el diagrama de secuencia del caso de uso Evolucionar Redes Neuronales, el proceso de comunicación de los objetos que interactúan, se establece a través de los mensajes que se envían entre ellos. Dicho proceso se describe de la siguiente manera: (**Véase *figura 4.6.***).

1. El actor **Usuario** le envía a **IU Principal** el mensaje *Aplicar Backpropagation*.
2. El **IU Principal** le envía al Usuario el mensaje *Solicitud Neuronas.*
3. El actor **Usuario** le envía a **IU Principal** el mensaje *Cantidad de Neuronas.*
4. El **IU Principal** le envía a **Backpropagation** el mensaje *Neuronas*.
5. El actor **Archivo Entrada** le envía a **Backpropagation** el mensaje *Puntos.*
6. El **Backpropagation** se envía a sí mismo el mensaje *Generar Pesos*.
7. El **Backpropagation** se envía a sí mismo el mensaje *Generar Solución.*
8. El **Backpropagation** le envía a **Gestor Solución** el mensaje *Solución*.
9. El **Gestor Solución** se envía a sí mismo el mensaje *Guardar Solución.*
10. El **Gestor Solución** le envía a **IU Principal** el mensaje *Solución.*
11. El **IU Principal** le envía a **Usuario** el mensaje *Visualiza Solución.*
12. El actor **Usuario** le envía a **IU Principal** el mensaje *Solicitar Generalización.*
13. El **IU Principal** le envía al **Gestor Solución** el mensaje *Procesar Generalización.*
14. El actor **Plantilla** le envía al **Gestor Solución** el mensaje *Imagen Inicial.*
15. El **Gestor Solución** se envía a sí mismo el mensaje *Procesa Generalización.*
16. El **Gestor Solución** le envía a **IU Principal** el mensaje *Generalización.*
17. El **IU Principal** le envía al actor **Usuario** el mensaje *Visualiza Generalización.*

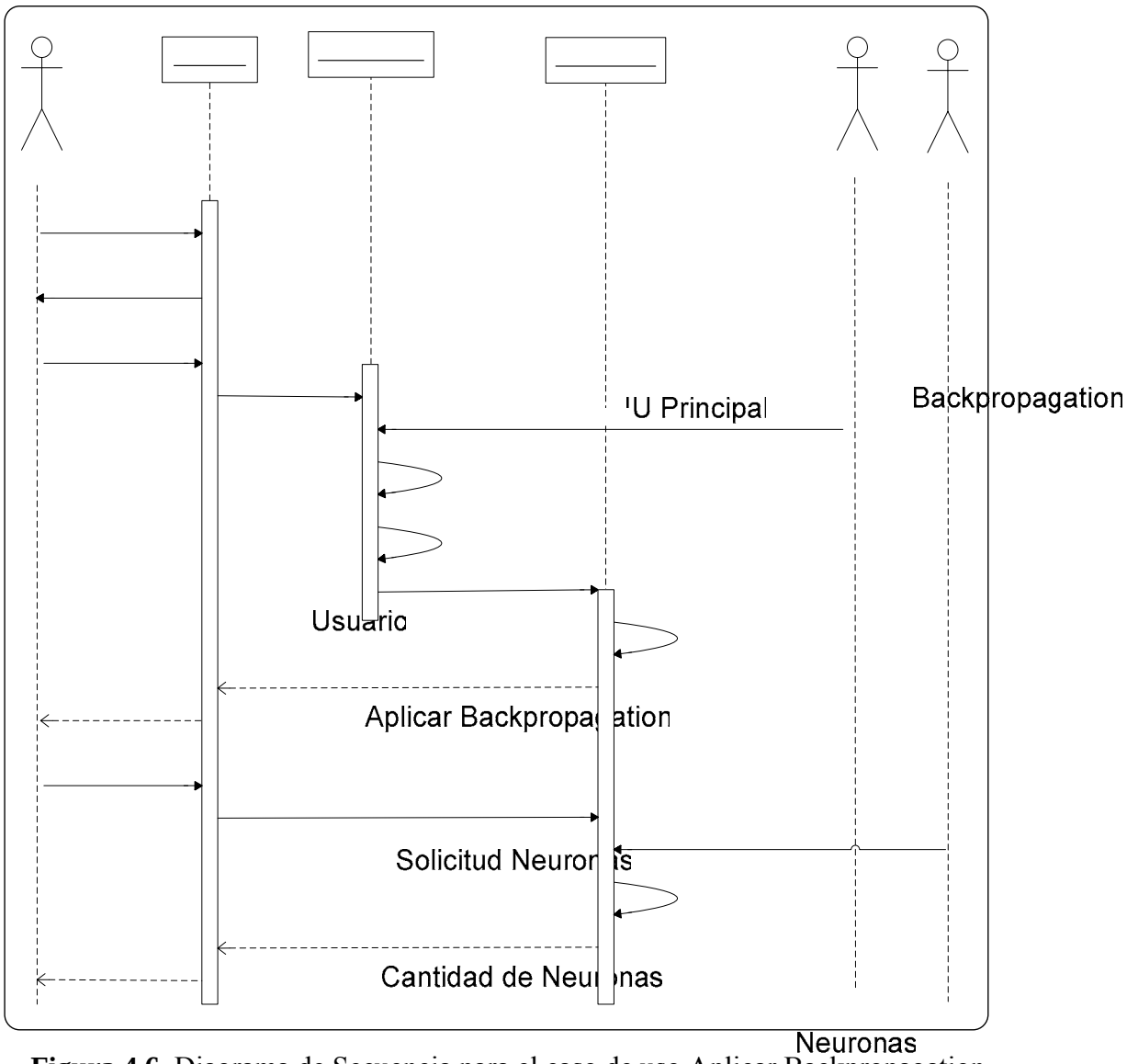


Figura 4.6 Diagrama de Secuencia para el caso de uso Aplicar Backpropagation

4.3 DIAGRAMA DE PAQUETES

Los diagramas de paquetes se usan para reflejar la organización de paquetes y sus elementos. Los elementos contenidos en un paquete comparten el mismo espacio de nombre, el hecho de compartir espacios de nombres requiere que los elementos contenidos en un espacio de nombre específico tengan nombres únicos. Los paquetes se pueden construir para representar relaciones tanto físicas como lógicas.

En la figura 4.7 y 4.8 se puede observar el paquete principal de la aplicación el cual es Redes_Neuronales, donde se encuentran contenidos todos los archivo .java de la aplicación, y está relacionado con todos los casos de uso del sistema. **[16]**

Figura 4.7 Diagrama de Paquete Redes_Neuronales

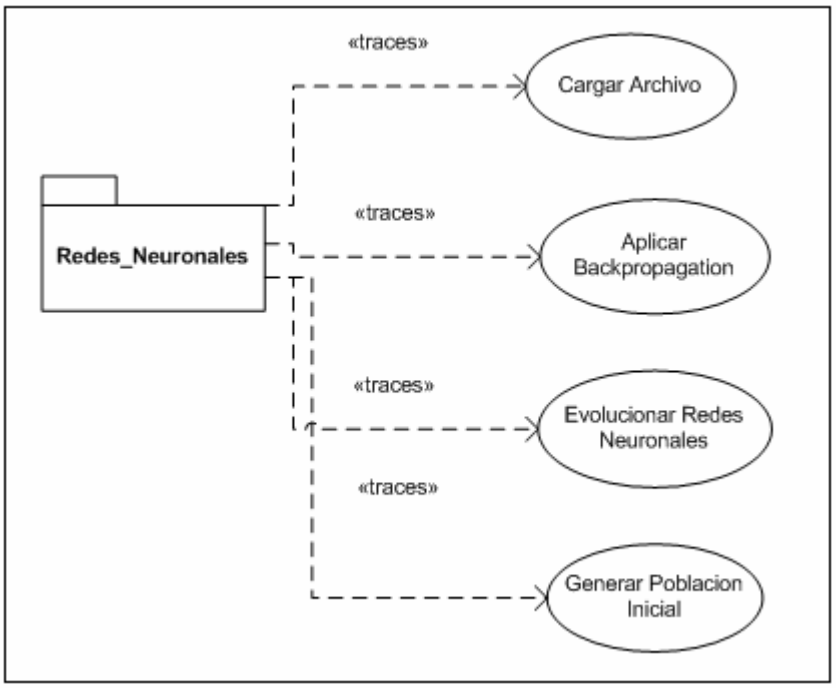


Figura 4.8 Diagrama de Paquete Redes_Neuronales y los Casos de Uso a los que está relacionado

En la figura 4.9 se muestra el paquete Imágenes, donde se encuentran todas las imágenes utilizadas en la aplicación.

Figura 4.9 Diagrama de Paquete Imágenes

En la figura 4.10 se muestra la relación entre los paquetes del sistema.

Figura 4.10 Relación ente los Paquetes

4.4 DIAGRAMA DE CAPAS

En este diagrama se muestra la distribución en cuatro capas de los paquetes que permiten el funcionamiento del sistema SDC, la cual se distribuye de la siguiente manera:

Imagenes

- **Capa Específica de la Aplicación**: aquí se encuentran los paquetes internos del sistema como son el de Imágenes y el Redes_Neuronales.
- **Capa General de la Aplicación**: aquí se muestra el paquete de la aplicación como tal.
- **Capa Intermedia**: en esta capa se encuentra el JRE.
- **Capa de Software:** aquí se encuentra el sistema operativo donde puede funcionar la aplicación. El cual puede ser Windows o Linux.

En la figura 4.11 se muestra el diagrama de capas del sistema SDC.

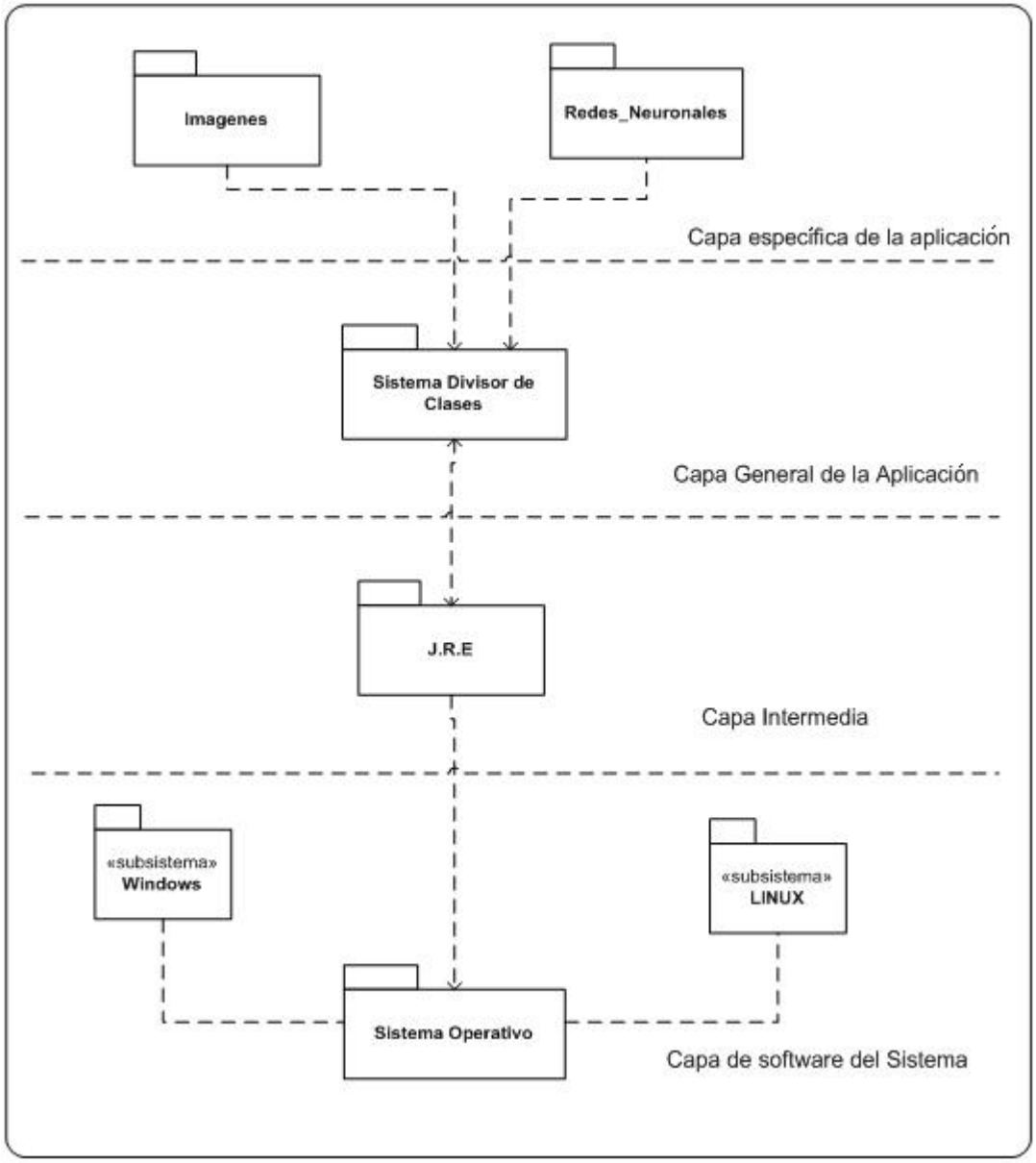


Figura 4.11 Diagrama de Capas del sistema SDC

4.5 TÉCNICA DE DESARROLLO DE SISTEMAS DE OBJETOS (TDSO)

A continuación en la tabla 1 se mostrará la especificación formal del universo de las clases del sistema SDC:

4.5.1 Descripción del Universo de Clases

Tabla 1 Definición del Universo de Clases del sistema SDC

10/10/2008		Versión 1.0
Universo de Clases del Sistema Divisor de Clases (SDC) { Colección de clases que son requisitos para implantar el SDC }		
1	Algoritmo_Genetico (double Puntos[ ][ ])	Clase donde se procede a evolucionar a las Redes Neuronales, por medio del cruce y la mutación.
2	Backpropagation (double Puntos[ ] [ ] , int neurona)	En esta clase se aplica la Red Backpropagation al problema dado por el usuario, y esta red será del tamaño elegido por el usuario.
3	Interfaz_Usuario : [JFrame]	Clase donde está definida toda la interfaz con la que interactúa el usuario.
4	Gestor_Archivos(string Nombre_Archivo)	Esta clase se encarga de cargar el archivo de entrada, y verificar que tenga el formato correcto.
5	Gestor_Sarchivo(string Nombre_Archivo)	Esta clase se encarga de crear y guardar el archivo de salida el cual contiene la solución obtenida ya sea por Algoritmo Genético o por Backpropagation.

4.5.2 Especificación Formal de las Clases

A continuación en las tablas 2 a la 6 se mostrará cada una de las clases del sistema SDC, donde se nombran y explican brevemente cada una de las variables y métodos correspondientes a cada clase.

Tabla 2 Especificación Formal de la Clase Algoritmo_Genetico

10/10/2008		Versión 1.0
1 clase Algoritmo_Genetico (double Puntos[ ] [ ]) {Realiza la evolución de las Redes Neuronales}		
	Especificación de Atributos: **Individuo_Final**: doble **Poblacion:** LinkedList **Poblacion_aux:**LinkedList **Puntos[ ] [ ]:** doble **Fitness_Total**: doble **NEURONAS:** entero **Especificación Sintáctica:**	**Individuo_Final:** es la red que converge. **Poblacion:** representa la población de la red. **Poblacion_aux:** es la población donde se almacenan los nuevos individuos. **Puntos[ ] [ ]:** matriz global que contiene los puntos de las clases extraídos del archivo de entrada **Fitness_Total:** la sumatoria del fitness de todos los individuos de la población. **NEURONAS:** cantidad de neuronas presentes en la capa intermedia. **Generar_Redes( ):** genera la arquitectura y pesos de las Redes. **Generar_Fitness( ):** se calcula el fitness de cada individuo. **Convergencia( ):** verifica si existe un individuo que de una solución. **Cruzar_Individuos( ):** se procede a realizar el cruce y la mutación de los individuos. **Obtener_Padres( ):** se seleccionan los padres de los individuos nuevos. **Funcion_tg_Hiperbolica( ):** se aplica a la salida de las capas ocultas y de salida.
1	Generar_Redes( ):	
2	Generar_Fitness( ):	
3	Convergencia( ):	
4	Cruzar_Individuos( ):	
5	Obtener_Padres( ):	
6	Funcion_tg_Hiperbolica( ):	

Tabla 3 Especificación de la clase Backpropagation

10/10/2008		Versión 1.0
2 clase Backpropagation (double Puntos[ ] [ ] , int neurona) { En esta clase se aplica la Red Backpropagation}		
	Especificación de Atributos: **Neuronas**: Entero **Red[ ][ ]:** doble **Error_Total**: doble **Solucion[ ]:** doble **Nuevos_Po[ ]:** doble **Nuevos_Pe[ ]:** doble **Delta_T[ ]:** doble **Puntos[ ][ ]:**doble **Especificación Sintáctica:**	**Neuronas:** representa la cantidad de neuronas en la capa oculta, es dado por el usuario. **Red[ ][ ]:** matriz que contiene los pesos y la arquitectura de la Red Neuronal Backpropagation. **Error_Total:** almacena el valor total del error de la Red. **Solucion[ ]:** contiene la solución obtenida por el Backpropagation. **Nuevos_Po[ ]:**contiene los nuevos pesos de la capa oculta. **Nuevos_Pe[ ]:** contiene los nuevos pesos de la capa de entrada. **Delta_T[ ]:** valor del delta total de los pesos de la capa oculta. **Puntos [ ][ ]:** matriz global. **Generar_Pesos( ):** crea los pesos de la red Backpropagation. **Calcular_Error( ):** calcula el error cometido por la red al momento de resolver el problema. **Pesos_Capoculta( ):** calcula los pesos de la capa oculta. **Delta_i( ):** calcula los delta para los pesos de la capa oculta. **Pesos_Capentrada( ):** calcula los pesos de la capa de entrada. **Delta_ j( ):** calcula los delta para los pesos de la capa de entrada. **Actualizar_Pesos( ):** actualiza los pesos con los pesos nuevos. **Funcion_Transferencia(doble x):** se le aplica a los delta de los pesos de las capas de la red.
1 2 3 4 5 6 7 8	Generar_Pesos( ) Calcular_Error( ) Pesos_Capoculta( ) Delta_i(entero v) Pesos_Capentrada( ) Delta_j( ) Actualizar_Pesos( ) Funcion_Transferencia(doble x)	

Tabla 4 Especificación de la clase Interfaz_Usuario

10/10/2008		Versión 1.0
	3 clase Interfaz_Usuario( ) {Esta clase es la encargada de la Interfaz con lo que interactúa el usuario}	
	Especificación de Atributos: **SA**: Gestor_Sarchivo **GA**:Gestor_Archivos **AG:**Algoritmo_Genetico **BK**:Backpropagation **Especificación Sintáctica:**	**SA**: es un objeto de la clase Gestor_Sarchivo. **GA**: es un objeto de la clase Gestor_Archivo. **AG**: es un objeto de la clase Algoritmo_Genetico. **BK**: es un objeto de la clase Backpropagation.
1	Guardar_Solucion( )	**Guardar_Solucion( ):** guarda la solución obtenida en una archivo de texto.
2	Empezar_Backpropagation( )	**Empezar_Backpropagation( ):** hace el llamado a la clase Backpropagation enviándole los puntos de las clases y la cantidad de neuronas presentes en la capa oculta, este valor es suministrado por el usuario.
3	Empezar_Algoritmo_Genetico( )	**Empezar_Algoritmo_Genetico( ):** hace el llamado a la clase Algoritmo_Genetico enviándole los puntos de las clases.
4	Abrir_Archivo( )	**Abrir_Archivo( ):** se abre el archivo de entrada y se verifica su formato.
5	Pintar_Neuronas( )	**Pintar_Neuronas( ):** se pintan las neuronas de la red resultante.
6	Pintar_Arcos( )	**Pintar_Arcos( ):** pinta los arcos de conexión de las neuronas de la red resultante
7	Pintar_Punto( )	**Pintar_Punto( ):** pinta los puntos de las clases.

Tabla 5 Especificación de la clase Gestor_Archivos

10/10/2008		Versión 1.0
	4 clase Gestor_Archivos(string Nombre_Archivo) {Esta clase se encarga de leer y validar el archivo de entrada}	
	Especificación de Atributos: **LineaArchivo:**BufferedReader **Puntos[ ] [ ]:** doble **Especificación Sintáctica:**	**LineaArchivo:** es la línea que es leída en un momento dado en el archivo de entrada. **Puntos[ ] [ ]:** matriz que contiene los puntos de las clases contenidos en el archivo de entrada.
1	Cargar_Puntos( )	**Cargar_Puntos( ):** carga en la matriz Puntos las coordenadas de los puntos de las clases.
2	Get_Puntos( )	**Get_Puntos( ):** devuelve la matriz Puntos.

Tabla 6 Especificación de la clase Gestor_Sarchivo

10/10/2008		Versión 1.0
	5 clase Gestor_Sarchivo(string Nombre_Archivo) {Esta clase se encarga de crear el archivo que contiene la solución}	
	Especificación de Atributos: **Linea_Salida:**BufferedWriter **Especificación Sintáctica:** Guardar_Solucion(doble Solucion[ ] [ ], entero filas)	**Linea_Salida:** contiene las líneas del archivo de salida **Guardar_Solucion(doble Solucion[ ] [ ], entero filas):** guarda la solución obtenida en un archivo de texto.

4.5.3 Especificación Formal de Métodos

A continuación en las tablas 7 a la 17 se mostrarán los métodos de las clases del sistema SDC, representados en pseudocódigo y una breve explicación de las variables utilizadas en ellos.

4.5.3.1 Métodos Clase Algoritmo_Genetico

Tabla 7 Especificación de Método Generar_Fitness( )

10/10/2008		Versión 1.0
	1, 2 (Procedimiento, Público) **Generar_Fitness( )** { se encarga de calcular el fitness de los individuos de la población }	
{**pre:** void}		{**pos:** valor del fitness de los individuos}
1	Para Posicion=0 hasta poblacion.size	**Posicion:**entero contador **p:** entero contador **i:** entero contador **Entrada_Oculta1[]:**arreglo del tipo doble que contiene los valores de entrada a cada neurona de la capa oculta **Red:** matriz de tipo doble que contiene los pesos y la arquitectura de un individuo dado **X1:** variable de tipo doble que contiene la coordenada x de un punto del conjunto de entrenamiento **X2:** variable del tipo doble que contiene la coordenada y de un punto del conjunto de entrenamiento **Salida:** variable de tipo doble que almacena la salida de la red **error:** error de la Red Neuronal para resolver el problema **Fitness_Total:** fitness total de todos los individuos de la red.
2	Red=poblacion.remove(Posicion)	
3	Para p=0 hasta Puntos.length hacer	
4	X1=Puntos[p][0]	
5	X2=Puntos[p][1]	
6	Para i=0 hasta NEURONAS hacer	
7	Entrada_Oculta1[i]=(X1*Red[0][i]) + (X2*Red[1][i]) + (1*Red[3][i])	
8	fpara	
9	Para i=0 hasta NEURONAS hacer	
10	Salida=Salida + (Entrada_Oculta1[i] * Red[2][i])	
11	fpara	
12	Salida=Salida + (1 * Red[4][0])	
13	Salida=Funcion_tg_Hiperbolica(Salida)	
14	Si Salida es menor que 0 entonces	
15	Salida=-1	
16	Sino	
17	Salida=1	
18	fsino	
19	fsi	
20	Si Puntos[p][2] es distinto de Salida entonces	
21	error=error + 1	
22	fsi	
23	fpara	
24	Fitness_Total=Fitness_Total + error	
25	Red[4][1]=error	
26	poblacion.add(Posicion,Red)	
27	error=0	
28	fpara	

Tabla 8 Especificación Método Cruzar_Individuos( )

10/10/2008		Versión 1.0
	1, 4 (Procedimiento, Público) **Cruzar_Individuos( )** { se encarga de realizar el cruce y la mutación de los individuos}	
{**pre:** void}		{**pos:** nuevos individuos mutados}
1	Mientras Poblacion_aux.size() es menor que 100	**Padre** : matriz del tipo doble que guarda el valor del padre del individuo **Madre** : matriz del tipo doble que almacena el otro padre del individuo **cont**: entero contador **i**: entero contador **j:** entero contador **Hijo:** arreglo del tipo doble que contiene al nuevo individuo generado **Mutar**: entero que indica si se hace la mutación o no del individuo **Fila:** entero contador **Columna**: entero contador **Poblacion_aux:** lista dinámica que contiene la nueva población generada **mutar:** variable de tipo entero que actúa como tasa de mutación va incrementándose a través del tiempo **r:** objeto que retorna un número aleatorio
2	Padre = Obtener_Padres( )	
3	Madre = Obtener_Padres( )	
4	cont = 0	
5	Para i=0 hasta 5 hacer	
6	Para j=0 hasta NEURONAS hacer	
7	Hijo[cont]=Padre[i][j]	
8	Hija[cont]=Madre[i][j]	
9	cont=cont+1	
10	fpara	
11	fpara	
12	Para i=aux hasta aux*2 hacer	
13	aux_hijo=Hijo[i]	
14	Hijo[i]=Hija[i]	
15	Hija[i]=aux_hijo	
16	fpara	
17	cont=0	
18	Para i=0 hasta 5 hacer	
19	Para j=0 hasta NEURONAS hacer	
20	Padre[i][j]=Hijo[cont]	
21	Madre[i][j]=Hija[cont]	
22	cont=cont+1	
23	fpara	
24	fpara	
25	Mutar=r.nextInt(mutar)	
26	Si Mutar=0 entonces	
27	Fila=valor aleatorio con probabilidad de 1:4	
28	Columna=valor aleatorio con probabilidad de 1:NEURONAS	

Tabla 9 Continuación Especificación Método Cruzar_Individuos( )

10/10/2008		Versión 1.0
	1, 4 (Procedimiento, Público) **Cruzar_Individuos( )** { se encarga de realizar el cruce y la mutación de los individuos}	
{**pre:** void}		{**pos:** nuevos individuos mutados}
29	Si Padre[Fila][Columna] es distinto de 0 entonces	
30	Padre[Fila][Columna]= Padre[Fila][Columna] *-1	
31	Sino	
32	Padre[Fila][Columna]=valor aleatorio del tipo doble	
33	fsino	
34	fsi	
35	fsi	
36	Mutar=r.nextInt(mutar)	
37	Si Mutar=0 entonces	
38	Fila=valor aleatorio con probabilidad de 1:4	
39	Columna=valor aleatorio con probabilidad de 1:NEURONAS	
40	Si Madre[Fila][Columna] diferente 0 entonces	
41	Madre[Fila][Columna]= Madre[Fila][Columna] *-1	
42	Sino	
43	Madre[Fila][Columna]=valor aleatorio del tipo doble	
44	fsino	
45	fsi	
46	fsi	
47	Poblacion_aux.add(Padre)	
48	Poblacion_aux.add(Madre)	
49	fmientras	
50	Poblacion.clear()	
51	Poblacion=Poblacion_aux	
52	Poblacion_aux.clear()	

Tabla 10 Especificación Método Obtener_Padres ( )

10/10/2008		Versión 1.0
	1, 5 (Procedimiento, Privado) **Obtener_Padres( )** { se encarga de buscar los padres de los nuevos individuos } {**pre:** void} {**pos:** devuelve los padres de los individuos a ser evolucionados}	
1	Para l = 0 hasta tamaño de la población hacer	**l:** entero contador **fitness_Ind[ ]:** arreglo del tipo doble que almacena el valor del fitness de cada individuo **población:** matriz del tipo doble que almacena la matriz de individuos **n:** entero del tipo doble que contiene un número aleatorio entre 0 y el tamaño del fitness total de la población **i:** entero contador **bandera:** valor del tipo booleano
2	Si l = 0 entonces	
3	fitness_Ind[l] = mat_aux[4][1]	
4	poblacion = mat_aux[l]	
5	fSi	
6	fitness_Ind[l]= mat_aux[4][1] + fitness_Ind[l-1]	
7	poblacion = mat_aux[l]	
8	fPara	
9	n = r.nextInt(Fitness_Total)	
10	Para i = 0 hasta tamaño de la población hacer	
11	Si n = fitness_Ind[i] o fitness_Ind[i] es mayor que n entonces	
12	Bandera = verdadero	
13	fSi	
14	fPara	
15	Si bandera = verdadero entonces	
16	Retornar mat_aux	

4.5.3.2 Métodos Clase Backpropagation

Tabla 11 Especificación Método Calcular_Error( )

10/10/2008		Versión 1.0
	2, 2 (Procedimiento, Público) **Calcular_Error( )** {Calcula el error de la Red Neuronal}	
{**pre:** void}		{**pos:** el error total de la red }
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	Para i=0 hasta Cantidad de Puntos hacer Para j=0 hasta Cantidad de Neuronas hacer Para k=0 hasta 2 hacer Resultado=Resultado + (Red[k][j] * Puntos[i][k]) fpara Resultado=Funcion_Transferencia(Resultado) V[j][i]=Resultado Resultado=Resultado * Red[2][j] ResultadoA=ResultadoA+Resultado fpara ResultadoA=Funcion_Transferencia(ResultadoA) Solucion[i]=ResultadoA Error=Error + (Puntos[i][2] – ResultadoA)^2 fpara Error=Error/2 Error_Total=Error	**i:**entero contador **j:** entero contador **k:** entero contador **Resultado:** valor del tipo doble que almacena el resultado arrojado por la función de transferencia **V[ ][ ]:** matriz del tipo doble que almacena los valores de la función de transferencia para todos los puntos **Red[ ][ ]:** matriz del tipo doble que almacena la arquitectura de la red **ResultadoA:** nuevo valor al que se le calcula la función de transferencia **Solucion[]:** arreglo del tipo doble que almacena los valores de la función de transferencia **Error:** error inicial de la red **Error_Total:** error total de la red

Tabla 12 Especificación Método Pesos_Capoculta( )

10/10/2008		Versión 1.0
2,3 (Procedimiento, Público) **Pesos_Capoculta( )** {Este método calcula los nuevos pesos de la capa oculta de la Red Backpropagation} {**pre:** void }		{**pos:** los nuevos valores de los pesos de la capa oculta}
1 2 3 4 5 6 7 8	Para j=0 hasta Neuronas hacer Para i=0 hasta Cantidad de Puntos hacer delta=Delta_i(i) Resultado=Resultado = (delta * V[j][i]) fpara Resultado=Resultado * 0.01 Nuevos_Po[j]=Resultado + Red[2][j] fpara	**j:** entero contador **i:** entero contador **delta:** valor que almacena el delta del peso i de la capa oculta **Resultado:** almacena el valor del peso anterior multiplicado por la constante de aprendizaje que es 0.01 **Nuevos_Po[ ]:** arreglo que almacena los nuevos valores de los pesos de la capa oculta

Tabla 13 Especificación Método Delta_i ( )

10/10/2008		Versión 1.0
2,4 (Procedimiento, Público) **Delta_i(int v)** {Calcula el delta de los pesos de la capa oculta} {**pre:** entero v }		{**pos:** valor del detlta}
1 2 3 4 5 6 7 8	Para i=0 hasta Neuronas hacer Delta=Delta + (Red[2][i] * V[i][v]) fpara Delta=Funcion_Transferencia(Delta) Delta=Delta * (1 – Delta) Delta=Delta * (Puntos[v][2]-Solucion[v]) Delta_T[v]=Delta Devolver Delta	**i:** entero contador **Delta:** valor que almacena el delta de un peso i de la capa de entrada **Delta_T[ ]:**arreglo que almacena todos los delta de la capa de entrada

Tabla 14 Especificación Método Pesos_Capentrada( )

10/10/2008		Versión 1.0
	2,5(Procedimiento, Público) **Pesos_Capentrada( )** {Calcula los nuevos pesos de la capa de entrada}	
{**pre:** void}		{**pos:** los nuevos valores de los pesos}
1 2 3 4 5 6 7 8 9 10 11	Para j=0 hasta Neuronas hacer Para k=0 hasta 2 hacer Para v=0 hasta Cantidad de Puntos hacer delta=Delta_j(j,v) delta=delta * Puntos[v][k] Resultado=Resultado + delta fpara Resultado=Resultado * 0.01 Nuevo_Pe[k][j]=Resultado + Red[k][j] fpara fpara	**j:** entero contador **k:** entro contador **v:** entero contador **delta:** valor que almacena el resultado del delta de la capa de entrada **Resultado:** valor que almacena el resultado del valor del peso por la constante de aprendizaje **Nuevo_Pe[ ] [ ]:** matriz que almacena los valores de los nuevos pesos de la capa de entrada.

Tabla 15 Especificación Método Delta_j ( )

10/10/2008		Versión 1.0
	2,6 (Procedimiento, Público) **Delta_j(entero j, entero v)** {Calcula el delta para los pesos de la capa de entrada}	
{**pre:** los valores de v y j}		{**pos**: el valor delta de los pesos}
1 2 3 4 5 6	Para i=0 hasta 2 hacer Delta=Delta + (Red[i][j] * Puntos[v][i]) fpara Delta=Funcion_Transferencia(Delta) Delta=Delta*(Delta_[v]*Nuevos_Po[j] Devolver Delta	**i:** entero contador **Delta:** valor que almacena los delta de los nuevos pesos de la capa de entrada

Tabla 16 Especificación Método Actualizar_Pesos( )

10/10/2008		Versión 1.0
2,7 (Procedimiento, Público) **Actualizar_Pesos( )** { se encarga de actualizar la matriz de pesos iniciales con los nuevos calculados} {**pre:** void}		{**pos:** matriz de pesos actualizada}

1
2
3
4
5
6
7
8
	Para i = 0 hasta 2 hacer
	  Para j = 0 hasta Neuronas hacer
	  Red[i][j]= Nuevos_Pe[i][j]
	  fPara
	fPara
	Para i = 0 hasta Neuronas hacer
	  Red[2][i] = Nuevos_Po[i]
	fPara
		i: entero contador
		j: entero contador
		Red[ ][ ]: matriz del tipo doble que contiene los nuevos pesos de la Red Neuronal

Tabla 17 Especificación Método Funcion_Trasferncia( )

10/10/2008		Versión 1.0
2,8 (Procedimiento, Público) **Funcion_Transferencia(doble x)** {se calcula la función de transferencia de las salidas de las capas} {**pre:** recibe el valor del delta}		{**pos:** devuelve el valor obtenido por la función}
1 2 3 4	x = x * -1 doble e = exp (x) doble f = 1 / 1 + e devuelve f	**x:** es el valor del delta del peso ya sea de la capa oculta o de la capa de entrada **e:** contiene el exponencial del delta **f:** almacena el valor resultante de la función de transferencia

CAPITULO 5: IMPLEMENTACIÓN Y PRUEBAS

5.1. DIAGRAMA DE COMPONENTE

Un diagrama de componentes representa como un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, librerías compartidas, módulos, ejecutables, o paquetes. Los diagramas de Componentes prevalecen en el campo de la arquitectura de software pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema.

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones, también muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Pueden ser simples archivos, paquetes, bibliotecas cargadas dinámicamente, etc. Figura 5.1 es la representación gráfica de un componente. **[16]**

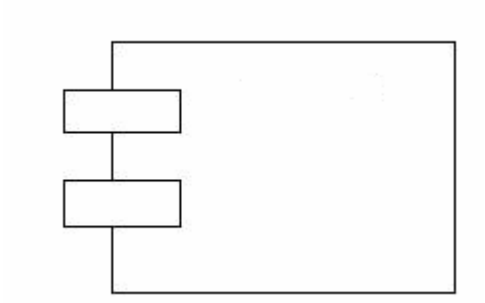


Figura 5.1 Representación Gráfica del Componente **[16]**

5.1.1 Diagramas de Componentes del Sistema Divisor de Clases (SDC)

A continuación se muestran los distintos diagramas de componente del sistema SDC.

5.1.1.1 Diagrama de Componentes del Paquete Redes_Neuronales

En la figura 5.2 se muestra el Diagrama de Componentes del paquete Redes_Neuronales.

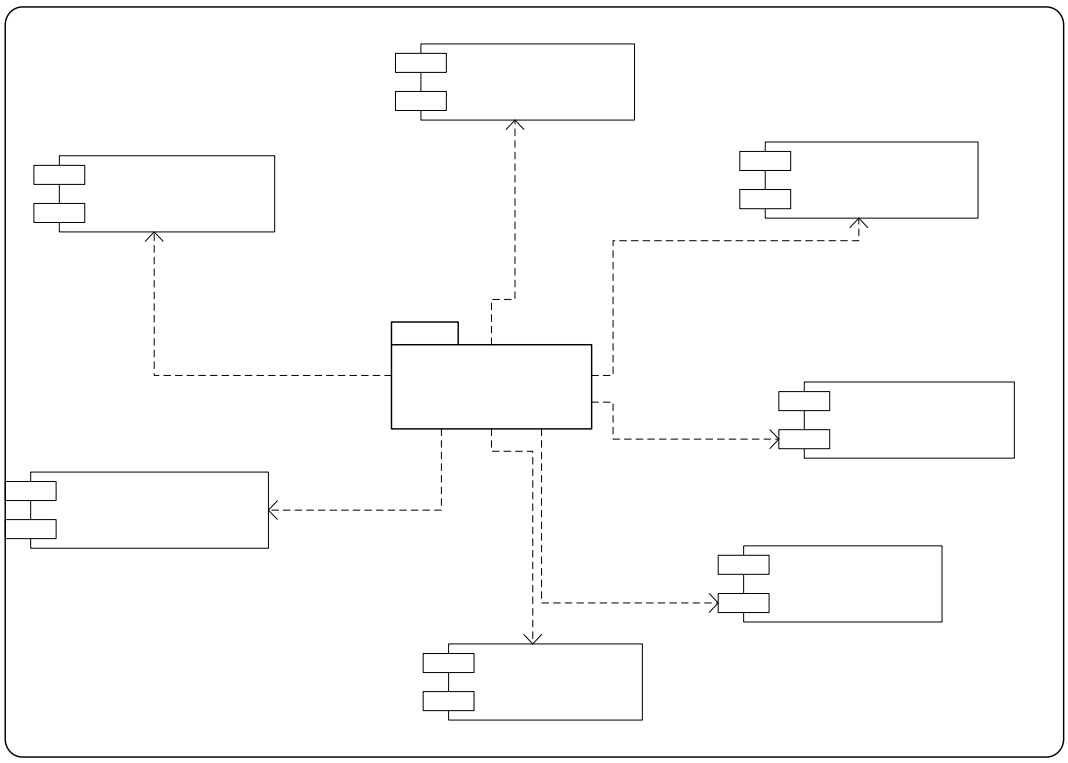


Figura 5.2 Diagrama de componente del paquete Redes_Neuronales.

Gestor_

Backpropagation.java

5.1.1.2 Relación del Componente Interfaz_Usuario

En la figura 5.3 se muestra como está relacionado el Componente Interfaz_Usuario con el resto de los Componentes del paquete Redes_Neuronales.

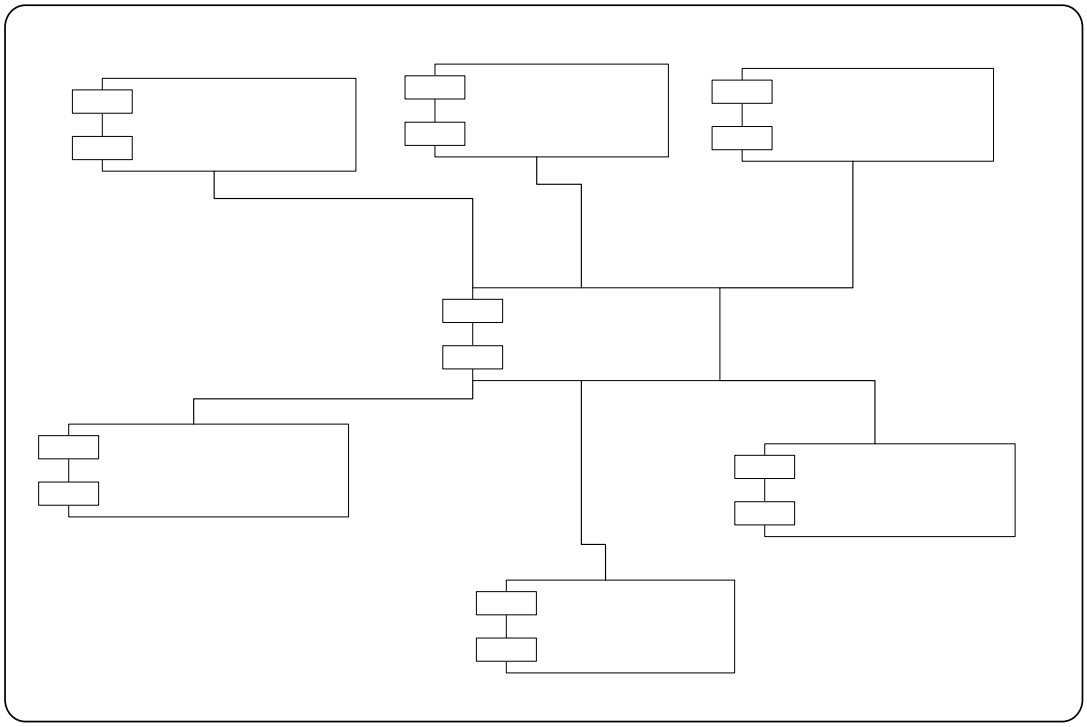


Figura 5.3 Relación entre el Componente Interfaz_Usuario y los Componentes del paquete Redes_Neuronales.

Backpropagation.java

5.1.1.3 Relación del Componente Algoritmo_Genetico

En la figura 5.4 se muestra la relación entre el componente Algoritmo_Genetico y los otros Componentes del Paquete Redes_Neuronales

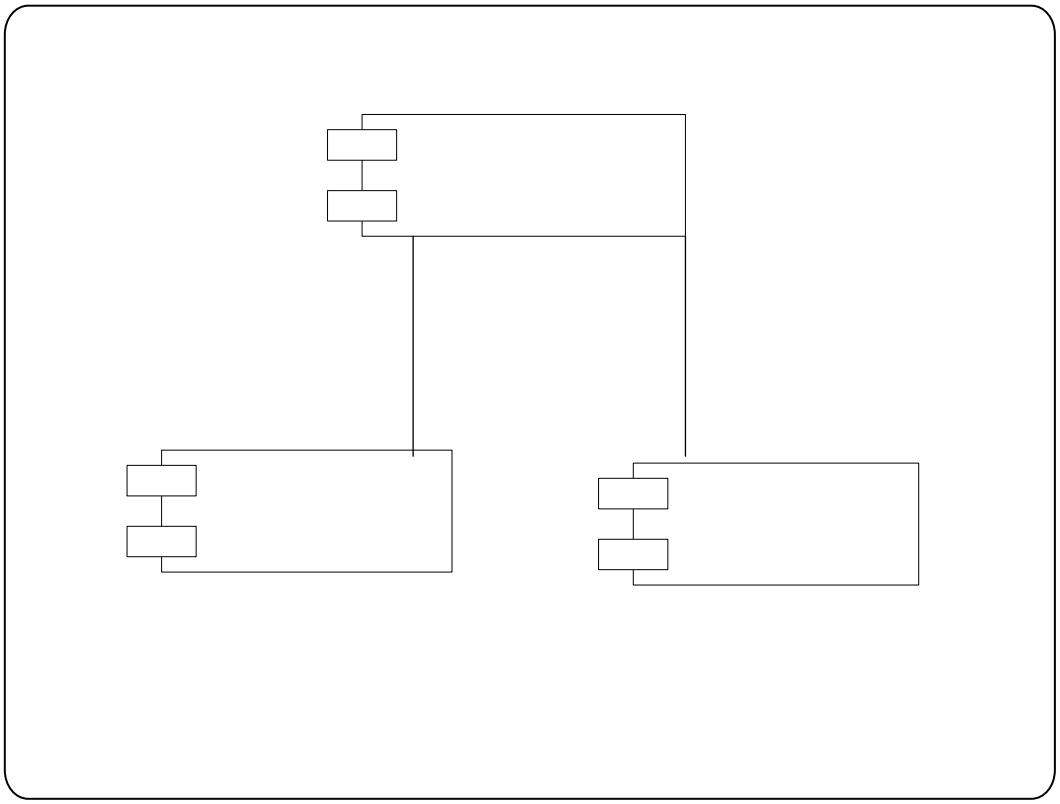


Figura 5.4 Relación entre el Componente Algoritmo_Genetico y los Componentes Gestor_Archivos e Interfaz_Usario

Algoritm

Código en JAVA del Componente Algoritmo Genético

```
public void Generar_Redes()
{
    Barra.setValue(1);
    r.setSeed(new Date().getTime());
    poblacion.clear();
    poblacion_aux.clear();
    int aux=0;
    double mat_aux[][];
    double matriz[][];
    //Se añaden individuos a la poblacion
    for(int i=0;i<700;i++)
    {
        matriz=new double[5][NEURONAS];
        poblacion.add(matriz);
    }
    for(int l=0;l<poblacion.size();l++)
    {
      mat_aux=poblacion.remove(l);
     for(int i=0;i<3;i++)
     {
         for(int j=0;j<NEURONAS;j++)
         {
             mat_aux[i][j]=r.nextDouble();
             aux=r.nextInt(2);
             if(aux==0)
                mat_aux[i][j]=mat_aux[i][j]*-1;
         }

     }
        /*************************************
         ***SE GENERA EL UMBRAL PARA CADA CAPA
         *************************************/
    for(int j=0;j<NEURONAS;j++)
    {
         mat_aux[3][j]=r.nextDouble();
         aux=r.nextInt(2);
         if(aux!=0)
         {
                 mat_aux[3][j]=mat_aux[3][j]*-1;
         }

    }
```

```
    mat_aux[4][0]=r.nextDouble();
    aux=r.nextInt(2);
    if(aux!=0)
    {
      mat_aux[4][0]=mat_aux[4][0]*-1;
    }
        poblacion.add(1,mat_aux); //Se introduce al individuo
    }


}
public void Cruzar_Individuos()
{
    riesgo++;
    int mutar;
    mutar=riesgo;
    r.setSeed(new Date().getTime());
    double Padre[][];
    double Madre[][];
    int aux=NEURONAS*5;
    int aux2=aux/3;
    double Hijo[]=new double[aux];
    double Hija[]=new double[aux];
    double aux_hijo;
    int cont;
    int Mutar;
    int Fila;
    int Columna;
    while(poblacion_aux.size()<100)
    {
            Padre=Obtener_Padres();
            Madre=Obtener_Padres();
            cont=0;
            for(int i=0;i<5;i++)
               for(int j=0;j<NEURONAS;j++)
               {
                 Hijo[cont]=Padre[i][j];
                 Hija[cont]=Madre[i][j];
                 cont++;
               }

            for(int i=aux2;i<aux2*2;i++)
            {
               aux_hijo=Hijo[i];
               Hijo[i]=Hija[i];
               Hija[i]=aux_hijo;
            }
```

```
            cont=0;
            for(int i=0;i<5;i++)
            {
                for(int j=0;j<NEURONAS;j++)
                {
                    Padre[i][j]=Hijo[cont];
                    Madre[i][j]=Hija[cont];
                    cont++;
                }
            }
            Mutar=r.nextInt(mutar);
            if(Mutar==0)
            {
                Fila=r.nextInt(4);
                Columna=r.nextInt(NEURONAS);
                if(Padre[Fila][Columna]!=0)
                {
                    Padre[Fila][Columna]=Padre[Fila][Columna]*-1;
                }
                else
                {
                    Padre[Fila][Columna]=r.nextDouble();
                }
            }
            Mutar=r.nextInt(mutar);
            if(Mutar==0)
            {
                Fila=r.nextInt(4);
                Columna=r.nextInt(NEURONAS);
                if(Madre[Fila][Columna]!=0)
                {
                    Madre[Fila][Columna]=Madre[Fila][Columna]*-1;
                }
                else
                {
                    Madre[Fila][Columna]=r.nextDouble();
                }

            }
            poblacion_aux.add(Padre);
            poblacion_aux.add(Madre);
        }
        poblacion.clear();
        poblacion=(LinkedList<double[][]>) poblacion_aux.clone();
        poblacion aux.clear();
```

```
private double[][] Obtener_Padres()
{
    boolean bandera=false;
    double mat_aux[][] = null;
    double fitness_Ind[]=new double[poblacion.size()];
    double n;
    for(int l=0;l<poblacion.size();l++)
    {
        mat_aux=poblacion.get(l);
        if(l==0)
        {
            fitness_Ind[l]=mat_aux[4][1];
            continue;
        }

        fitness_Ind[l]=mat_aux[4][1]+fitness_Ind[l-1];
    }


    n=r.nextInt((int)fitness_total);
    for(int i=0;i<poblacion.size();i++)
    {
         if(n==fitness_Ind[i]||fitness_Ind[i]>n)
         {
             mat_aux=poblacion.get(i);
             bandera =true;
             break;
         }
    }
    if(bandera==true)
    {
      return mat_aux;
    }
    else
    {
        mat_aux=poblacion.getFirst();
        return mat_aux;
    }

}
private double Funcion_tg_hiperbolica(double x)
{
    x=x*-1;
    double e = Math.exp(x);
    double f=(1-e)/(1+e);
    return f;
}
```

```
public boolean Convergencia()
{

    boolean convergir=false;
    double mat_aux[][];
    for(int l=0;l<poblacion.size();l++)
    {
         mat_aux=poblacion.remove(l);
         if(mat_aux[4][1]<Puntos.length*0.05)
         {
             convergir=true;
              poblacion.add(l,mat_aux);
             Individuo_Final=poblacion.remove(l);
             break;
         }
         poblacion.add(l,mat_aux);

    }

    return convergir;
}
```

5.1.1.4 Relación del Componente Backpropagation

En la figura 5.5 se muestra la relación del Componente Backpropagation y los otros Componentes del Paquete Redes_Neuronales

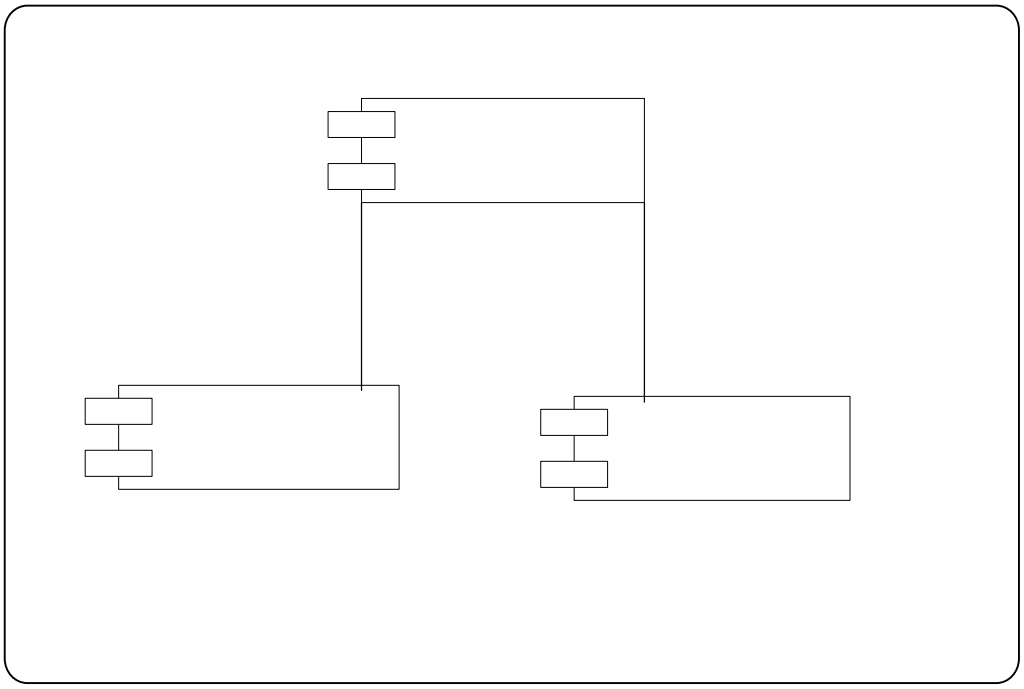


Figura 5.5 Relación entre el Componente Backpropagation y los Componentes Gestor_Archivos e Interfaz_Usuario

Backp

Código JAVA del Componente Backpropagation

```
public void Calcular_Error()
{
    double Error=0;
    double Resultado=0;
    double ResultadoA=0;
    //Calculo de los Si
    for(int i=0;i<Puntos.length;i++)
    {
        for(int j=0;j<Neuronas;j++)
        {
            for(int k=0;k<2;k++)
            {
                Resultado=Resultado + (Red[k][j]*Puntos[i][k]);

            }
            Resultado= Funcion_Transferencia(Resultado);
            V[j][i]=Resultado;
            Resultado= Resultado*Red[2][j];
            ResultadoA=ResultadoA + Resultado;
        }
        ResultadoA=Funcion_Transferencia(ResultadoA);
        Solucion[i]=ResultadoA;
        Error=Error + (Math.pow((Puntos[i][2]-ResultadoA), 2));
    }
    Error=Error/2;
    Error_Total=Error;
}
public void Pesos_Capoculta()
{
    double delta;
    double Resultado=0;
    for(int j=0;j<Neuronas;j++)
    {
        for(int i=0;i<Puntos.length;i++)
        {
            delta=Delta_i(i);
            Resultado=Resultado + (delta*V[j][i]);

        }                    //Constante de aprendizaje=0.01
        Resultado=Resultado*0.01;
        Nuevos_Po[j]=Resultado + Red[2][j];
    }

}
```

```
public double Delta_i(int v)
{
    double Delta=0;
    for(int i=0;i<Neuronas;i++)
    {
        Delta=Delta + (Red[2][i]*V[i][v]);
    }
    Delta=Funcion_Transferencia(Delta);
    Delta=Delta*(1-Delta);//Derivada de la funcion de transferencia
    Delta=Delta*(Puntos[v][2]-Solucion[v]);
    Delta_T[v]=Delta;
    return Delta;
}
public void Pesos_Capentrada()
{
    double delta=0;
    double Resultado=0;
    for(int j=0;j<Neuronas;j++)
    {
        for(int k=0;k<2;k++)
        {
            for(int v=0;v<Puntos.length;v++)
            {
                delta=Delta_j(j,v);
                delta=delta*Puntos[v][k];
                Resultado=Resultado + delta;
            }
            Resultado=Resultado*0.01;
            Nuevos_Pe[k][j]=Resultado + Red[k][j];
        }
    }
}
public double Delta_j(int j,int v)
{
    double Delta=0;
    for(int i=0;i<2;i++)
    {
        Delta=Delta + (Red[i][j]*Puntos[v][i]);
    }
    Delta=Funcion_Transferencia(Delta);
    Delta=Delta*(1-Delta);//Derivada de la funcion de transferencia
    Delta=Delta*(Delta_T[v]*Nuevos_Po[j]);
    return Delta;
}
```

```
public void Actualizar_Pesos()
{
    for(int i=0;i<2;i++)
    {
        for(int j=0;j<Neuronas;j++)
        {
            Red[i][j]=Nuevos_Pe[i][j];
        }
    }
    for(int i=0;i<Neuronas;i++)
    {
        Red[2][i]=Nuevos_Po[i];
    }
}

private double Funcion_Transferencia(double x)
{
    x=x*-1;
    double e = Math.exp(x);
    double f=1/(1+e);
    return f;
}
```

5.1.2 Diagramas de Componentes del Paquete Imágenes

En la figura 5.6 se muestra al paquete Imágenes y los Componentes por los que está formado

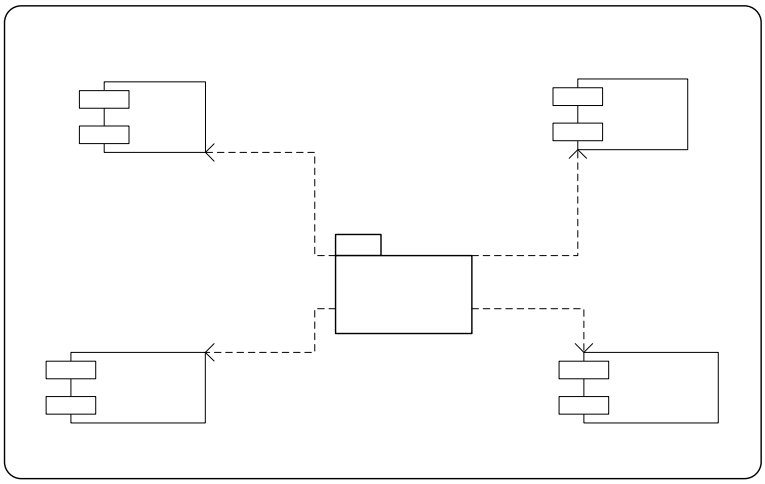


Figura 5.6 Diagrama Componentes Paquete Imágenes

5.2 DIAGRAMA DE INTERFAZ

Los Diagramas de la Interfaz del Usuario se usan para visualmente imitar la interfaz del usuario del sistema usando formas, controles y etiquetas. **[15]**

UP.png

5.2.1 Diagramas de Interfaz del Sistema Divisor de Clases

5.2.1.1 Diagrama Interfaz Principal

En la figura 5.7 se puede observar la Interfaz Principal de la aplicación.

esferav.png

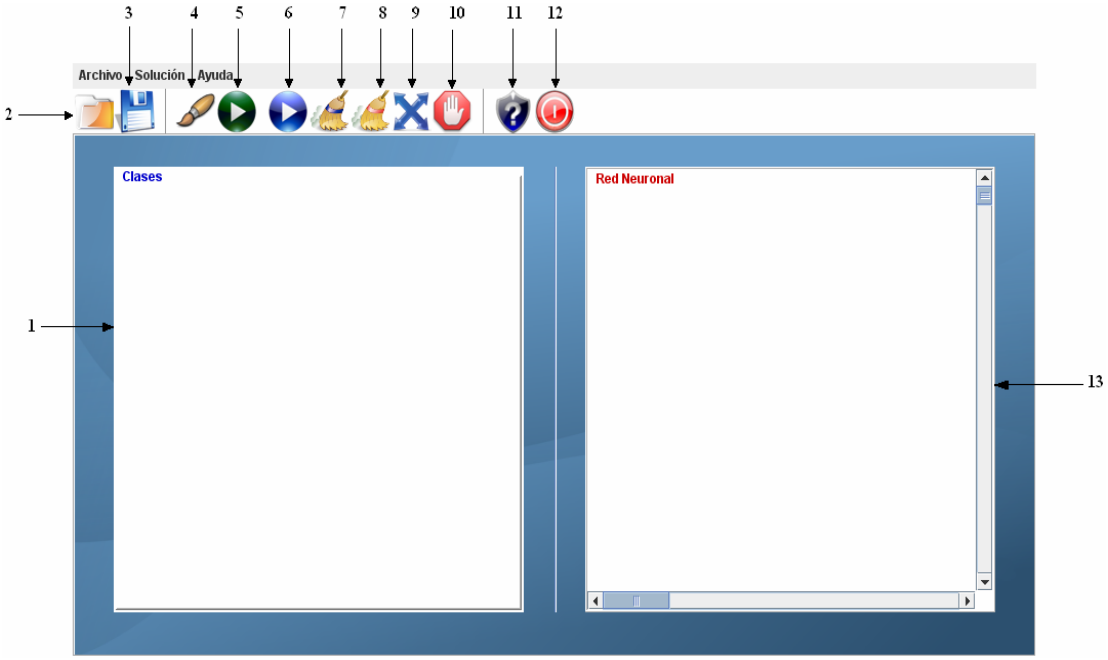


Figura 5.7 Diagrama Interfaz Principal

Leyenda Diagrama Interfaz Principal

1. Panel donde se dibujan los puntos de las clases que se desean dividir
2. Botón de acceso directo para cargar el archivo de entrada
3. Botón de acceso directo para guardar la solución obtenida
4. Botón de acceso directo para pintar los puntos de las clases
5. Botón de acceso directo para iniciar el Algoritmo Genético
6. Botón de acceso directo para iniciar el Algoritmo Backpropagation
7. Botón de acceso directo para borrar el panel de las clases
8. Botón de acceso directo para borrar el panel donde se muestra la Red Neuronal que resuelve el problema
9. Botón de acceso directo para iniciar la generalización
10. Botón de acceso directo para detener la ejecución de la aplicación
11. Botón de acceso directo para activar el centro de ayuda

12. Botón de acceso directo para cerrar la aplicación
13. Panel donde se muestra la Red Neuronal que divide las clases.

5.2.1.2 Interfaz Menú Archivo

En el menú archivo están comprendidas las acciones para abrir el archivo de entrada, guardar archivo de salida y salir de la aplicación, donde se muestra el icono del botón de acceso directo, y las teclas de atajo para activarla. (Ver Figura 5.8)

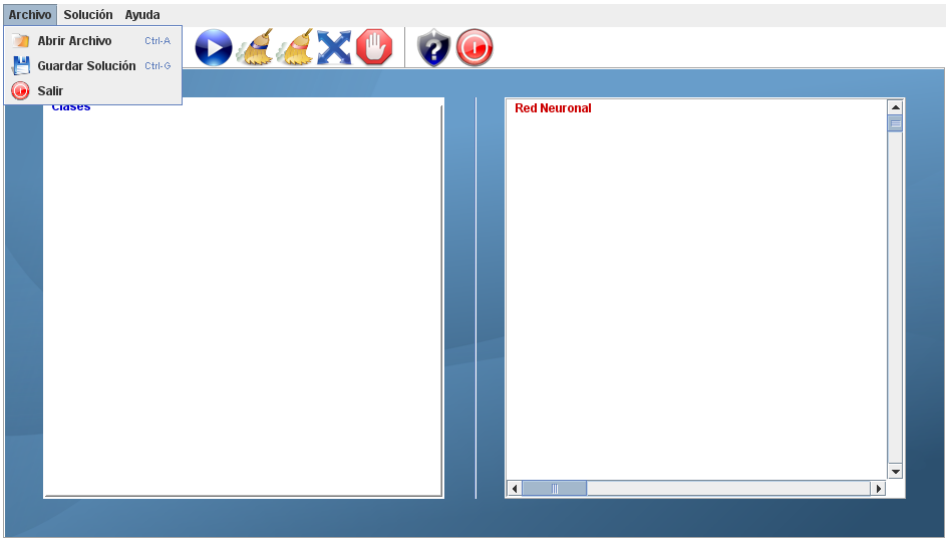


Figura 5.8 Diagrama Interfaz Menú Archivo

5.2.1.2 Interfaz Menú Solución

En la figura 5.9 se muestra la Interfaz del menú Solución, el cual contiene las acciones de pintar puntos, iniciar Algoritmo Genético, iniciar Backpropagation, borrar los puntos de las clases, borrar la Red Neuronal o solución y detener el proceso de ejecución de la aplicación, incluye el icono de acceso directo y las teclas de atajo para cada acción.

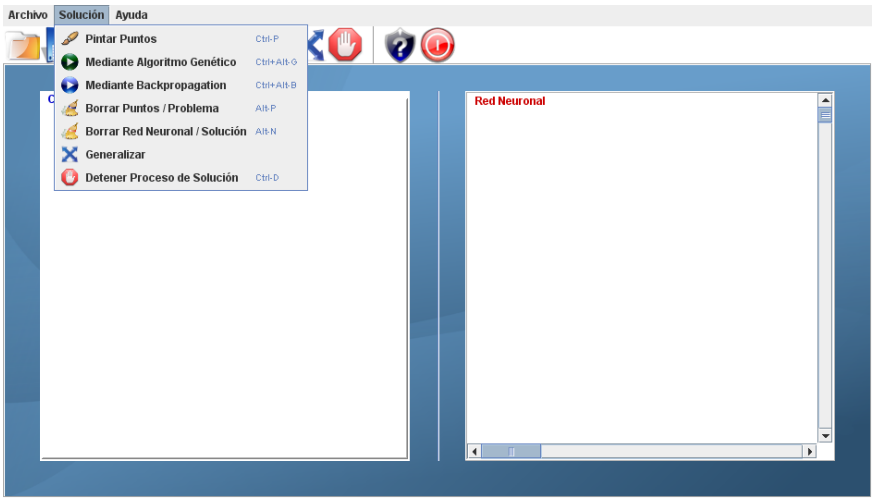


Figura 5.9 Diagrama Interfaz Menú Solución

5.2.1.3 Interfaz Menú Ayuda

En la figura 5.10 se muestra el contenido del menú Ayuda, el cual solo contiene la acción de mostrar la ayuda de la aplicación, incluye el icono de acceso directo y teclas de atajo.

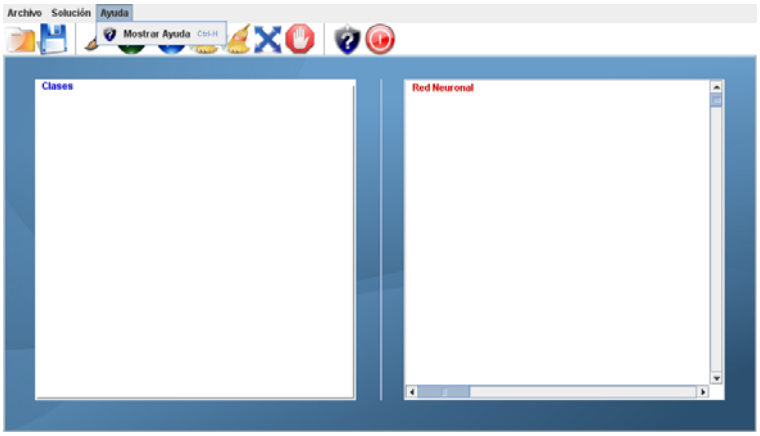


Figura 5.10 Diagrama Interfaz Menú Ayuda

5.3 PRUEBAS

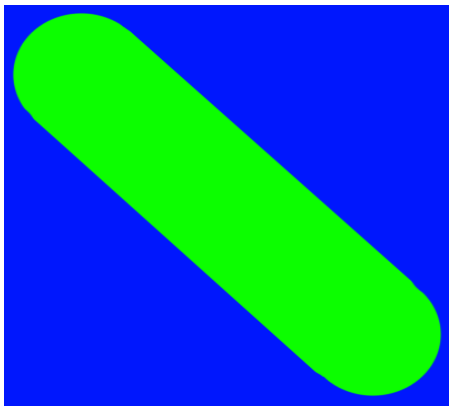
Las pruebas de software tienen como objetivo encontrar defectos en el software, donde la prueba tiene éxito si consigue un defecto y fracasa si no lo hace, también verificar si cumple las especificaciones de diseño y los requisitos de análisis.

A continuación se muestran 3 casos de pruebas diferentes, donde se mostrará la plantilla inicial, esta es una imagen que está formada por las dos clases que se desean separar.

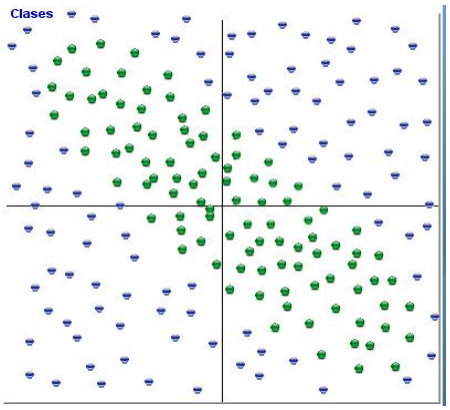
También se mostrará el conjunto de entrenamiento, este está formado un conjunto de puntos tomados aleatoriamente de la plantilla, es decir, contiene información de ambas clases a dividir.

Por último se mostraran las generalizaciones realizadas por el Algoritmo Genético solo, por el Algoritmo Genético con Backpropagation, y por el Backpropagation.

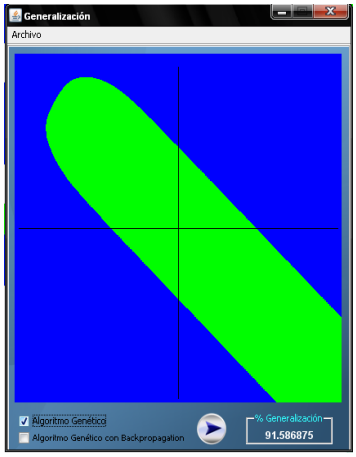
5.3.1 Caso de prueba 1



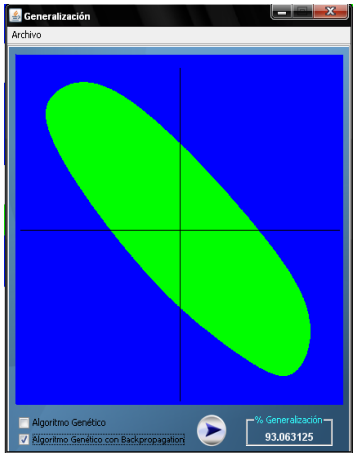
5.11.a)



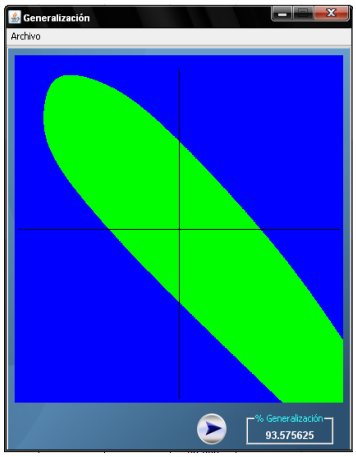
5.11.b)



5.11.c)



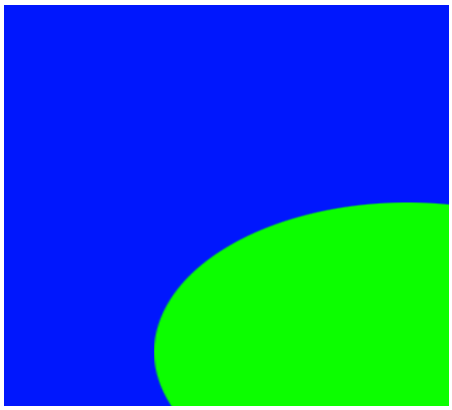
5.11.d)



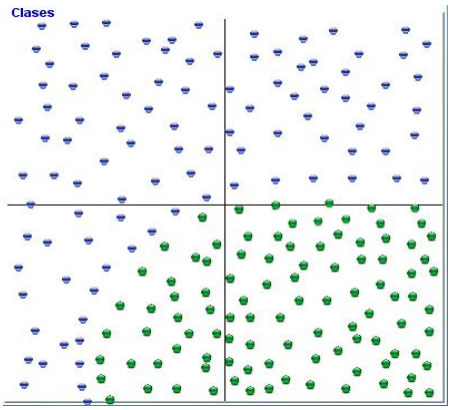
5.11. e)

Figura 5.11. Caso de prueba 1 a) Plantilla imagen inicial, b) conjunto de entrenamiento, c) generalización obtenida solo por Algoritmo Genético, d) generalización obtenida por el Algoritmo Genético y el Backpropagation, e) generalización obtenida por el Backpropagation

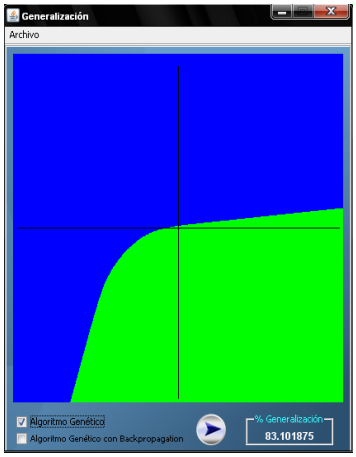
5.3.2 Caso de prueba 2



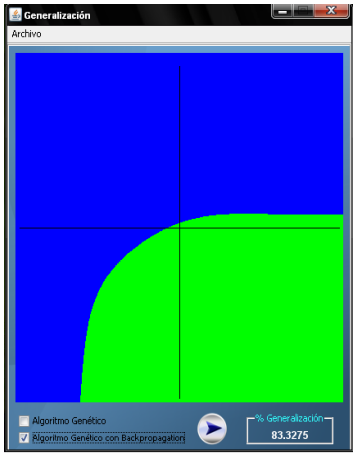
5.12.a)



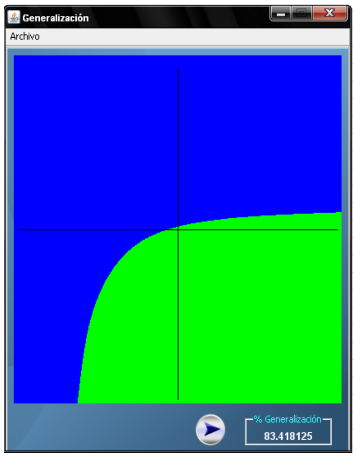
5.12.b)



5.12.c)



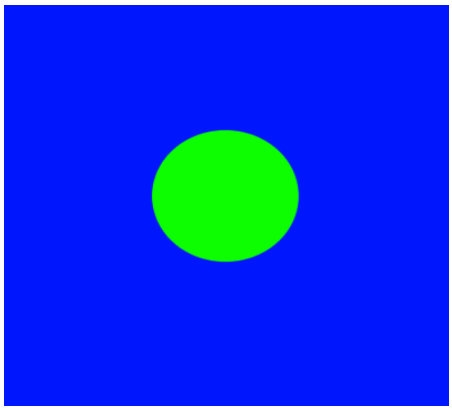
5.12.d)



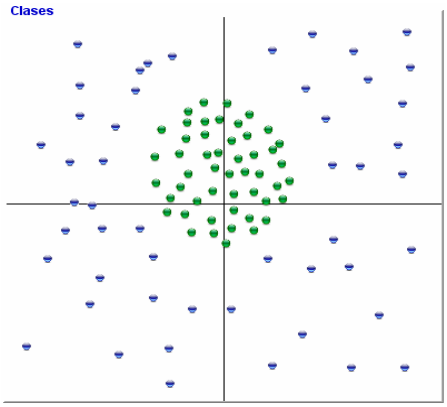
5.12.e)

Figura 5.12. Caso de prueba 2 a) Plantilla imagen inicial, b) conjunto de entrenamiento, c) generalización obtenida solo por Algoritmo Genético, d) generalización obtenida por el Algoritmo Genético y el Backpropagation, e) generalización obtenida por el Backpropagation

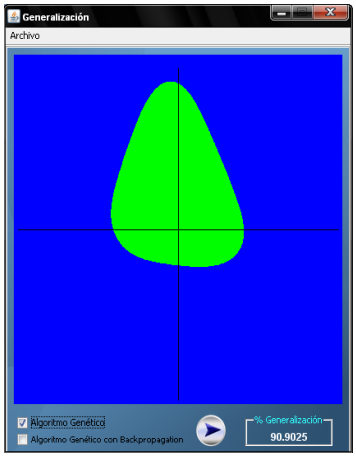
5.3.3 Caso de prueba 3



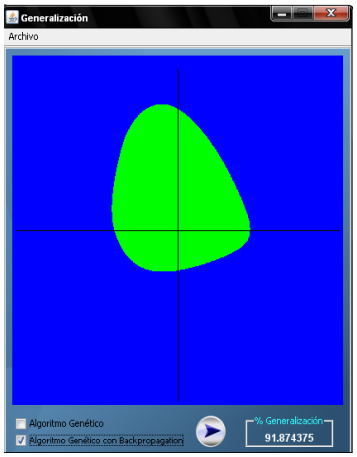
5.13.a)



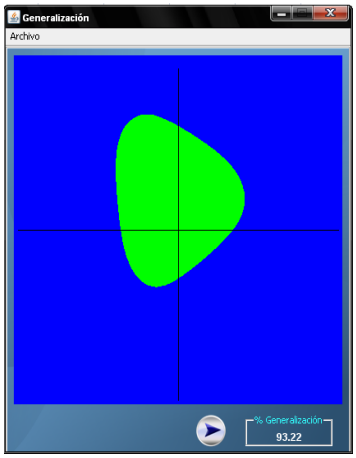
5.13.b)



5.13.c)



5.13.d)



5.13.e)

Figura 5.13. Caso de prueba 3 a) Plantilla imagen inicial, b) conjunto de entrenamiento, c) generalización obtenida solo por Algoritmo Genético, d) generalización obtenida por el Algoritmo Genético y el Backpropagation, e) generalización obtenida por el Backpropagation.

5.3.4 Tabla de Resultados

Esta tabla contiene todos los datos calculados para cada uno de los 30 casos de prueba, los cuales tenían diferentes niveles de dificultad.

A continuación se muestra la tabla de resultados la cual contiene el número de neuronas que utilizó el Algoritmo Genético para resolver el problema el cual representa la cantidad mínima de neuronas necesarias para dar una solución, el número de neuronas que utilizó el Backpropagation para resolver el problema, este valor es introducido por el usuario, también contiene el tiempo de ejecución para el Algoritmo Genético y para el Backpropagation, contiene el porcentaje de generalización que obtuvo la red del Algoritmo Genético sin el ajuste final del Backpropagation, el porcentaje de generalización del Algoritmo Genético más el ajuste del Backpropagation y por último el porcentaje de generalización del Backpropagation.

Valores promedio de la tabla 18:

- Promedio de Neuronas utilizadas por el Algoritmo Genético: 25.2
- Promedio de Neuronas utilizadas por el Backpropagation: 23.5
- Promedio del Tiempo de Ejecución del Algoritmo Genético: 69975.33 ms
- Promedio del Tiempo de Ejecución del Backpropagation: 3876.3 ms
- Promedio Generalización del Algoritmo Genético: 71.387733 %
- Promedio Generalización del Algoritmo Genético y Backpropagation: 82.0287333 %
- Promedio Generalización del Backpropagation: 82.173567 %

Tabla 18 Tabla de resultados obtenidos por los 30 casos de prueba realizados

Problema	Neuronas A.G.	Neuronas B.P.	Tiempo Ejec. A.G.	Tiempo Ejec B.P.	% Gener. A.G.	% Gener. A.G. + B.P.	% Gener. B.P.
1	15	11	1569	2015	80.403	91.874	93.22
2	16	19	2378	2657	75.75	91.948	90.821
3	31	22	76297	2109	72.443	73.591	73.253
4	45	29	295359	4594	79.426	97.861	98.454
5	18	24	9312	6093	75.713	90.903	89.438
6	14	12	1123	2578	75.259	86.296	86.095
7	15	15	1436	3343	68.605	75.89	75.919
8	17	19	2531	4172	48.365	52.531	52.481
9	56	30	636797	3563	91.586	93.063	93.575
10	26	24	49062	4203	75.601	79.398	79.727
11	16	25	7953	3563	69.806	87.427	87.021
12	18	22	22469	2503	73.283	90.827	90.884
13	27	35	100688	7235	58.603	68.595	64.216
14	28	20	20547	4031	68.881	70.345	71.628
15	20	25	20297	3250	62.925	87.191	87.203
16	79	55	497094	8359	70.886	73.679	80.476
17	17	22	8063	3953	75.878	90.333	89.723
18	21	42	25063	6906	74.27	80.129	78.288
19	17	23	8359	4812	75.783	90.725	91.012
20	17	15	8563	3906	67.543	82.921	83.174
21	33	25	83452	3078	68.683	87.322	87.696
22	41	32	20794	5484	73.518	92.841	92.745
23	15	18	1359	3756	77.245	81.592	81.473
24	16	22	2493	2704	73.616	89.729	89.643
25	17	18	2689	2813	69.114	89.467	89.211
26	35	23	79567	2750	48.928	51.626	51.636
27	31	25	75987	3797	83.101	83.327	83.418
28	19	15	13359	2859	69.371	86.302	86.228
29	18	22	12437	2266	68.541	73.88	77.442
30	18	16	12172	2937	68.506	69.249	69.107

En la tabla anterior se muestra a modo de comparación entre Algoritmo Genético y el Algoritmo de Backpropagation al momento de resolver cada uno de los casos de prueba.

Entre los aspectos tomados en la tabla se encuentran: tiempo de ejecución (expresado en milisegundos), cantidad de neuronas utilizadas en la solución, y porcentaje de generalización. Datos que arrojaron para resolver cada uno de los casos de prueba y que sirvieron de base para analizar el comportamiento de ambos algoritmos.

Se puede observar que el algoritmo Backpropagation utiliza menor cantidad de neuronas que el Algoritmo Genético para resolver la mayor parte de los casos de prueba, esta cantidad en promedio es de 1,7 neuronas menos, lo que hace al Backpropagation más rápido al momento de ejecutarlo.

También se puede apreciar la capacidad de generalización de ambos algoritmos, donde se demuestra que el Backpropagation es el mar acertado al momento de arrojar una solución factible al problema planteado.

Otro aspecto importante reflejado en la tabla es que el Algoritmo Genético al ser combinado con el algoritmo Backpropagation arroja una capacidad de generalización mayor, esto no es más que aplicarle un mejoramiento local al resultado obtenido por el Algoritmo Genético para que la solución sea más acertada, esta combinación se conoce como Algoritmo Memético.

CONCLUSIONES

Sobre la base del análisis de 30 casos de prueba con diferentes niveles de dificultad obtuvimos las siguientes conclusiones:

- En promedio la Red Neuronal obtenida por medio del Backpropagation utiliza 1.7 neuronas menos que la obtenida a través del Algoritmo Genético.
- La solución obtenida por el Algoritmo Genético sin el ajuste final de los pesos por medio del Backpropagation tiene un porcentaje de generalización del 71.387% en promedio. Este resultado muestra el potencial del Algoritmo Genético pero también revela la necesidad de realizar ajustes que conduzcan a mejores resultados.
- La solución obtenida por el Algoritmo Genético más el ajuste final de los pesos por medio del Backpropagation tiene un porcentaje de generalización del 82.028% en promedio. Esto muestra la conveniencia de combinar Algoritmos Genéticos con mejoramiento local, idea conocida como Algoritmo Memético.
- La solución obtenida solo a través del Backpropagation tiene un porcentaje de generalización del 82.173% en promedio.
- Los resultados obtenidos por el Algoritmo Genético son muy prometedores y marcan el inicio de futuras investigaciones.
- Mediante el uso de la técnica OMT (Object Modeling Technique), como marco de trabajo o metodología implementada, junto con los diagramas del UML usados como herramienta de modelado, fueron determinantes y esenciales para la correcta realización del diseño, documentación y codificación de todo el sistema.

RECOMENDACIONES

- Se recomienda la exploración de técnica de evolución que permitan un mayor grado de libertad en las conexiones de las Redes Neuronales.

- Se recomienda la exploración de variantes que combina el Algoritmo Genético con búsquedas locales, tal como la empleada aquí al combinar el Algoritmo Genético con el Backpropagation.

- Se propone explorar nuevas formas de asignación de los valores de los parámetros tales como el tamaño dinámico de la población y la tasa dinámica de mutación.

- Se propone utilizar la Regla Delta Generalizada para el Algoritmo Backpropagation.

BIBLIOGRAFÍA

1. - Acosta, M., Salazar,H. **"Redes Neuronales Artificiales"** Universidad Tecnológica de Pereira (2000)
http://ohm.utp.edu.co/neuronales/Capitulo2/General2.htm

2.- Ballesteros, A. **"La Neurona Biológica**" Universidad de Málaga, España.

3.- Besembel, I., y Rivero, D. **"Técnica de Desarrollo de Sistemas de Objetos (TDSO)"** Universidad de Los Andes

4.- Bollella, A. **"Redes Neuronales"** (2000)
http://www.monografias.com/trabajos12/redneuro/redneuro.shtml#histor

5.- Cardona, O. **"Redes Neuronales y Teoria de los Conjuntos Difusos"** (2001)
http://www.tdcat.cesca.es/TESIS_UPC/AVAILABLE/TDX-0416102-075520/26ApendiceD.PDF

6.- Díaz, V. y Herrera, L. **"Desarrollo de un Algoritmo Genético para Solucionar Problemas de Programación Lineal"** (2003)

7.- M.C. Flores, L. **"Red Neuronal Artificial"** (2001)
http://www.itnogales.edu.mx/formatos/Red%20neuronal%20artificial.pdf

8.- González, L. **"Historia de las Redes Neuronales"** (2001)
http://ingenieria.udea.edu.co/investigacion/mecatronica/mectronics/redes.htm

9.- Kuri, A. "**Redes Neuronales y Algoritmos Genéticos"** (2000)
http://cursos.itam.mx/akuri/PUBLICA.CNS/2000/RNSyAGS.PDF

10.- Leon, M. "**Algoritmos evolutivos, genéticos y programación genética"** (2002) http://www.gth.die.upm.es/~macias/doc/pubs/aircenter99/www.aircenter.net/enevolut.html

11- Lunar, R. y Da Silva, A. **"Diseño de un Sistema Basado en Redes Neuronales para el Control de un Vehículo"** (2001)

12.- Larman, C. **"UML y Patrones"** segunda Edición (2006)

13.- Merelo, J. **"Diseño de Redes Neuronales Artificiales Mediante Algoritmos Evolutivos."** (2001)

14. - Michalewicz, Z. **"Genetic Algorithms + Data Structures = Evolution Programs"** 3rd edition, Springer, Berlin (1996).

15.- Medina, J. "**Metodología y Herramientas UML para el Modelado y Análisis de Sistemas de Tiempo Real Orientados a Objetos"** Tesis Doctoral, Santander, junio de 2005

16.- Moreno, G. "**Ingeniería de Software UML"** (2005)

17.- Moreno, J. **"Redes Neuronales Cibernéticas"** Postgrado de Computación Emergente.

18.- Muñoz, M **"ALGORITMOS GENÉTICOS"**
Primera versión: Noviembre, 1997. Última modificación: Abril, 2005.

19.- Piedra, N. y López, J. **"Estudio de la Aplicación de Redes Neuronales Artificiales en la Ingeniería de Tráfico de Internet"** (2006)
http://nopiedra.files.wordpress.com/2007/10/aplicacion-de-redes-neuronales-artificiales-para-la-ingenieria-de-trafico-de-internet.pdf

20.- Orcero, D. "**Los Algoritmos Genéticos."** (2002)
http://www.orcero.org/irbis/disertacion/node1.htm

21.- Orozco, S. "**ALGORITMOS GENÉTICOS"** Guatemala, Febrero de 2007

22.- Rodríguez, B. **"UML"** (2004)

23.- Sarbjit, K. y Ysaccis, F. **"Simulador de Redes Neuronales Competitivas con Aprendizaje no Supervisado del Tipo SOM y ART1"** (1997)

24.- Trejo, M. **"Diseño e Implementación de un Simulador de Redes Neuronales Backpropagation"** (1994)

25.- Viñuela, P y Galván, I. **"Redes Neuronales Artificiales Un Enfoque Práctico",** Editorial Pearson Prentice Hall, Madrid (2004)

26.- Universidad Autónoma de Madrid **"Algoritmo Genético"**
http://www.uam.es/personal_pdi/psicologia/adarraga/studs/Xerxes/Pag4.html

27.- Wikipedia la enciclopedia libre **"Algoritmo Genético"** (2008)
http://es.wikipedia.org/wiki/Aut%C3%B3mata_celular

28.- Wikipedia la enciclopedia libre **"Redes Neuronales"** (2008)
http://es.wikipedia.org/wiki/Red_neuronal_artificial

ANEXO A: MANUAL DE USUARIO

1. INTRODUCCIÓN

Bienvenidos al **SDC**, Sistema Divisor de Clases. Éste representa la herramienta ideal para brindarles a los usuarios que deseen encontrar una Red Neuronal que sea capaz de resolver problemas de clasificación en un espacio bidimensional.

Para ello el SDC le brindará al usuario dos vías para encontrar la solución, una es por medio de Algoritmo Genético y la otra es por medio de la Red Backpropagation, además el usuario podrá observar la capacidad de generalización de la solución obtenida.

2. INICIO

Para ingresar al SDC, es necesario que el computador cuente con los siguientes requisitos para garantizar el correcto funcionamiento del mismo:

- Sistema Operativo Windows XP service pack 2 o superior o Sistema Operativo Linux
- Java Runtime Environment (JRE) 1.7.

3. INTERFAZ PRINCIPAL

A continuación se explican los menús y accesos directos para la navegación entre las distintas funciones de la aplicación.

3.1 MENÚ ARCHIVO

Se encuentra en la parte superior izquierda de la interfaz y este cuenta con tres opciones las cuales son: Abrir Archivo, Guardar Solución y Salir véase la figura A.1.

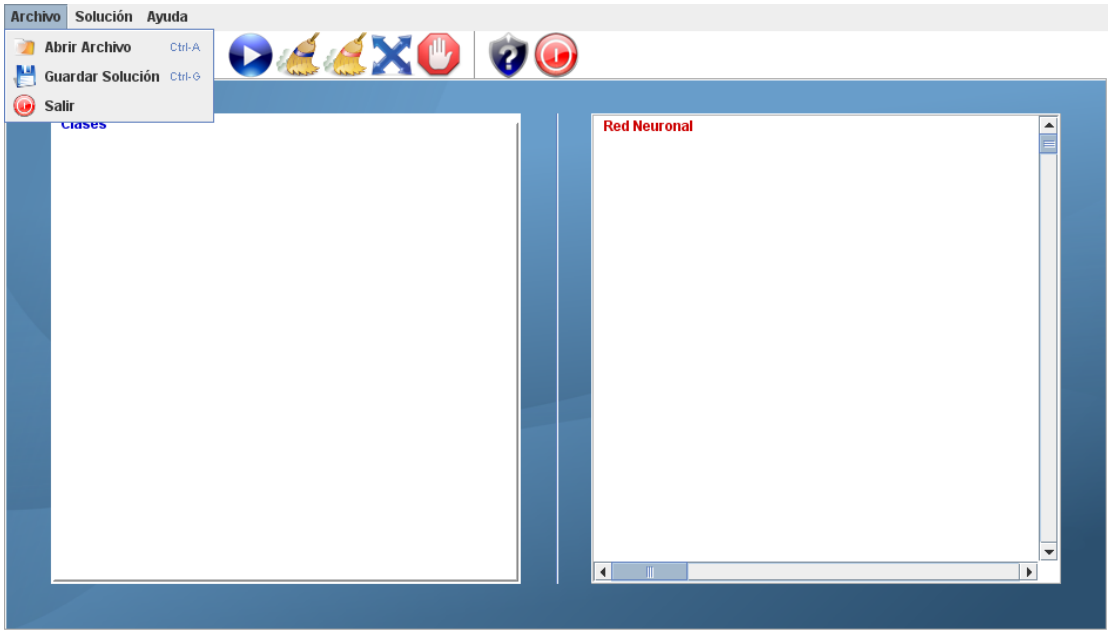


Figura A.1 Menú Archivo

3.1.1 OPCIÓN ABRIR ARCHIVO

A través de esta opción podrá cargar el archivo donde está contenido el conjunto de entrenamiento para las Redes Neuronales. Este contendrá las coordenadas x e y de los puntos de los cuales se desea la división. Está situado de primero en el menú Archivo véase la figura A.2.

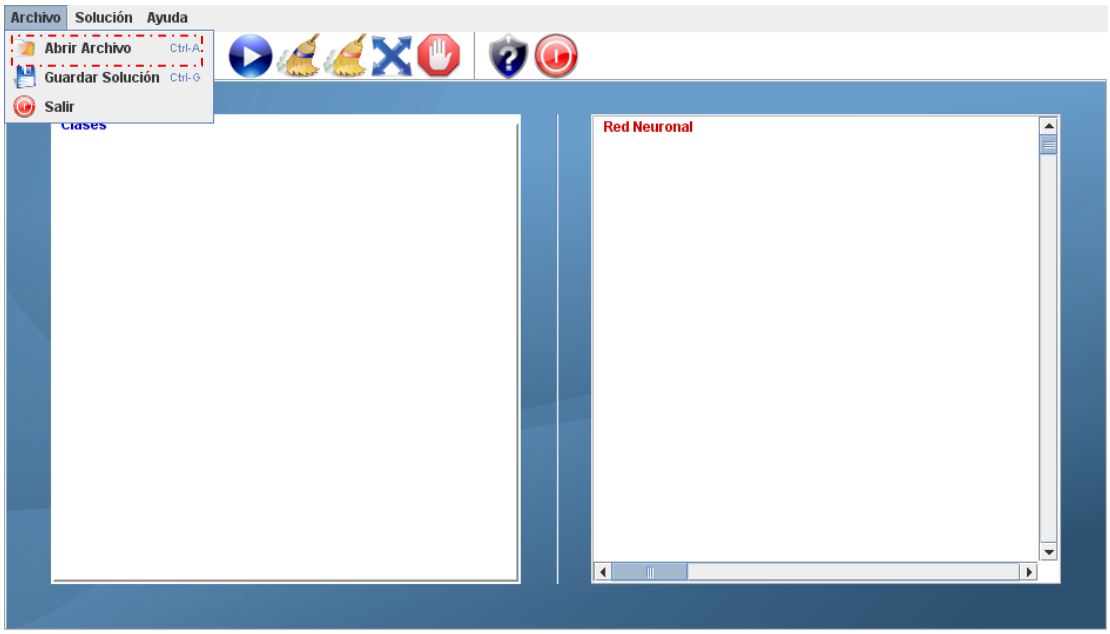


Figura A.2 Opción Abrir Archivo

El formato del archivo de texto debe ser el siguiente (ver figura A.3):

- La primera línea será la cantidad de puntos presentes en el conjunto de entrenamiento.
- Luego cada línea del archivo tendrá la coordenada x del punto, luego la coordenada y por último la clase a la que pertenece, separados por un espacio en blanco.
- El final del archivo se marcará con una línea que contenga la palabra EOF.
- La escala tomada para las coordenadas de los puntos serán desde cero (0) hasta un máximo de uno (1) y hasta un mínimo de menos uno (-1).
- Las clases se identifican con uno (1) para una y menos uno (-1) para la otra.

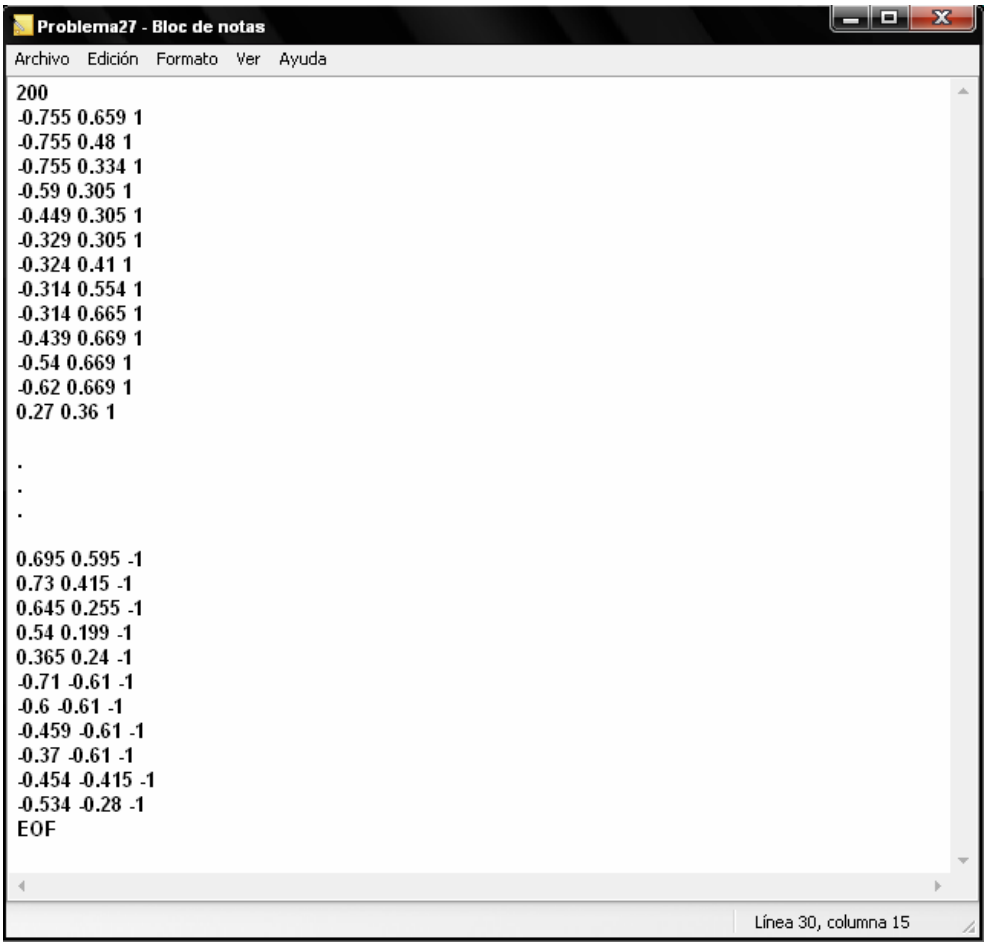


Figura A.3 Formato del archivo de entrada

3.1.2 OPCIÓN GUARDAR ARCHIVO

Luego de obtener la solución, podrá mediante esta opción (véase la figura A.4), guardar en un archivo de texto los pesos sinápticos pertenecientes a la Red Neuronal.

Al momento de colocar el nombre del archivo en la ventana emergente que aparecerá, (véase la figura A.5), puede colocar o no la extensión .txt.

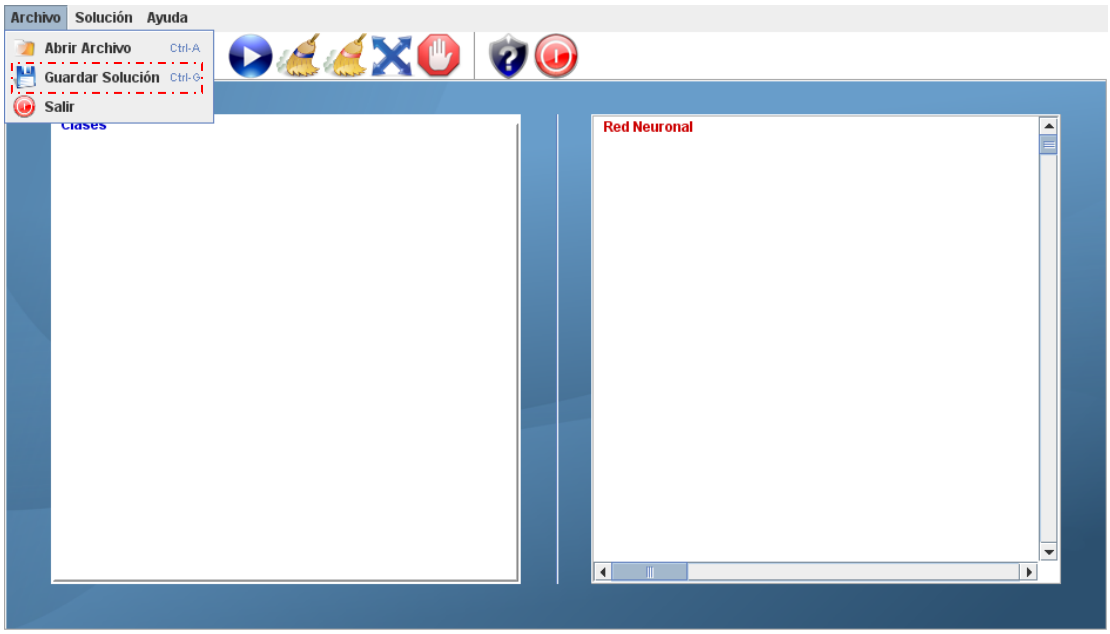


Figura A.4 Opción Guardar Solución

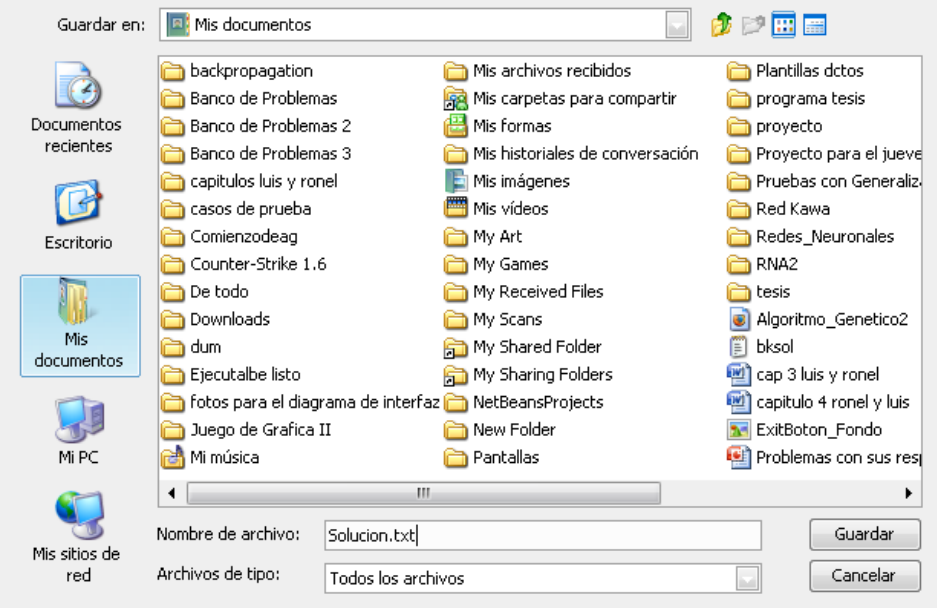


Figura A.5 Ventana Emergente

3.1.3 OPCIÓN SALIR

Esta opción le permite al usuario salir de la aplicación cuando lo desee.(Véase figura A.6)

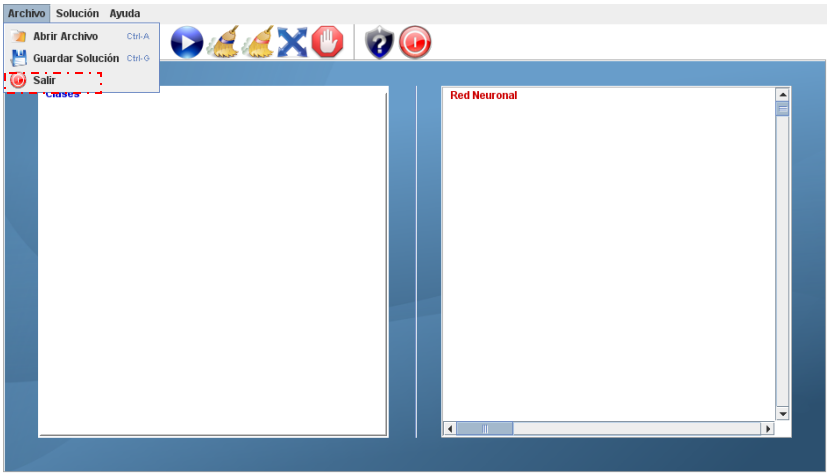


Figura A.6 Opción Salir

3.2 MENÚ SOLUCIÓN

Se encuentra en la parte superior al lado del menú archivo, este cuenta con las opciones: Pintar Puntos, Mediante Algoritmo Genético, Mediante Backpropagation, Borrar Puntos / Problema, Borrar Red Neuronal / Solución, Generalización, Detener Proceso de Solución, véase la figura A.7.

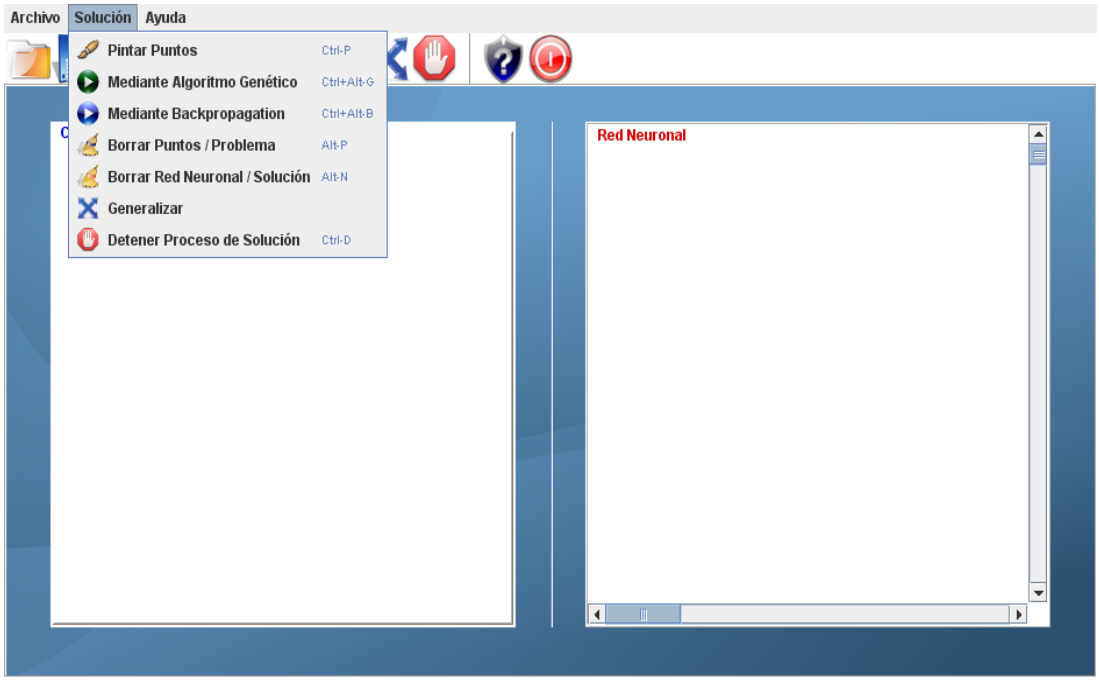


Figura A.7 Menú Solución

3.2.1 OPCIÓN PINTAR PUNTOS.

Esta opción le permite al Usuario dibujar los puntos contenidos en el archivo de entrada en el panel ubicado a la izquierda de la aplicación.

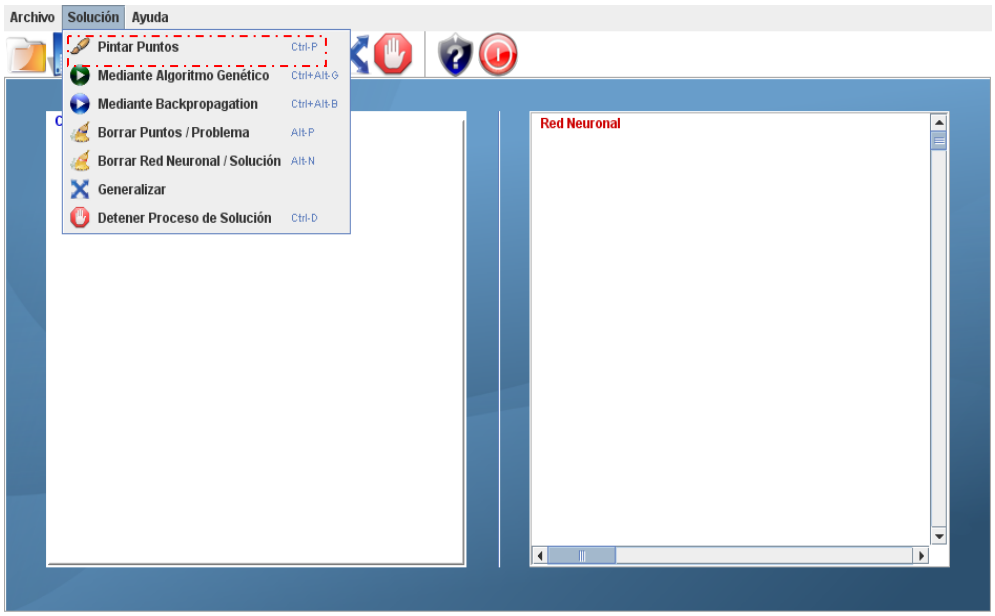


Figura A.8 Opción Pintar Puntos.

3.2.2 OPCIÓN MEDIANTE ALGORITMO GENÉTICO

Al elegir esta opción se inicia el proceso de evolución y cruce del Algoritmo Genético el cual es representado por una Red Neuronal en el panel derecho de la aplicación. (Véase figura A.9)

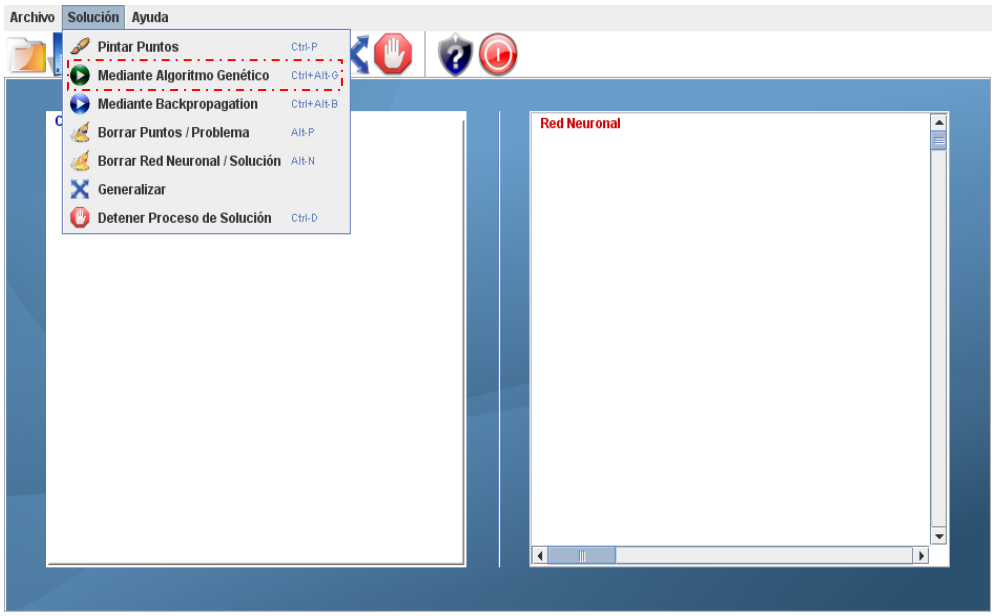


Figura A.9 Opción Mediante Algoritmo Genético

3.2.3 OPCIÓN MEDIANTE BACKPROPAGATION

Al seleccionar esta opción aparece una ventana emergente donde se le solicita al usuario la cantidad de neuronas que desea en la capa oculta de la red, luego se inicia el proceso de Backpropagation (véase figura A.10)

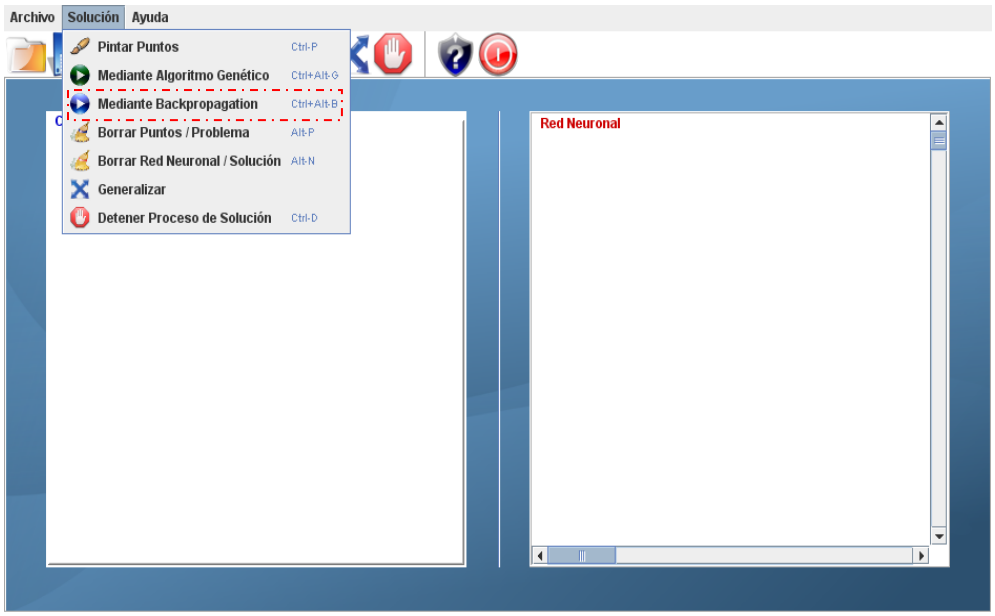


Figura A.10 Opción Mediante Backpropagation

3.2.4 OPCIÓN BORRAR PUNTOS / PROBLEMA

Esta opción le permite al usuario borrar los puntos dibujados en panel izquierdo de la aplicación. (véase la figura A.11)

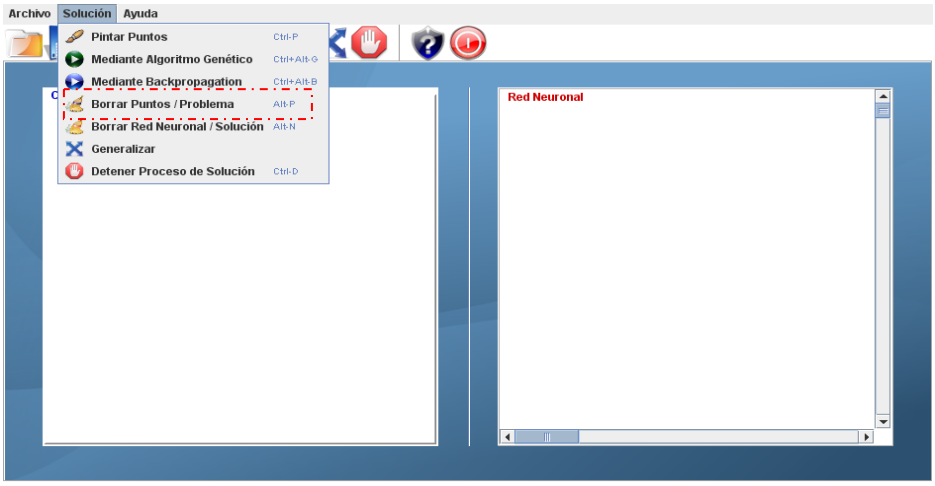


Figura A.11 Opción Borrar Puntos / Problema

3.2.5 OPCIÓN BORRAR RED NEURONAL / SOLUCIÓN

Esta opción le permite al usuario borrar la solución reflejada en al panel derecho de la aplicación, ya sea por el Algoritmo Genético o por el Backpropagation (véase figura A.12).

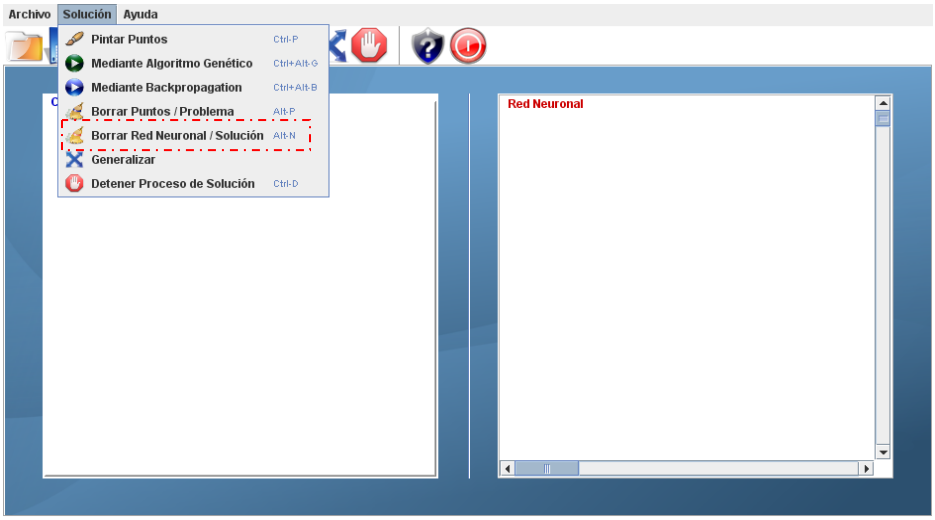


Figura A.12 Opción Borrar Red Neuronal / Solución

3.2.6 OPCIÓN GENERALIZACIÓN

Luego de obtener una solución, ya sea por medio del Algoritmo Genético o por Backpropagation, el usuario podrá seleccionar esta opción, (véase figura A.13).

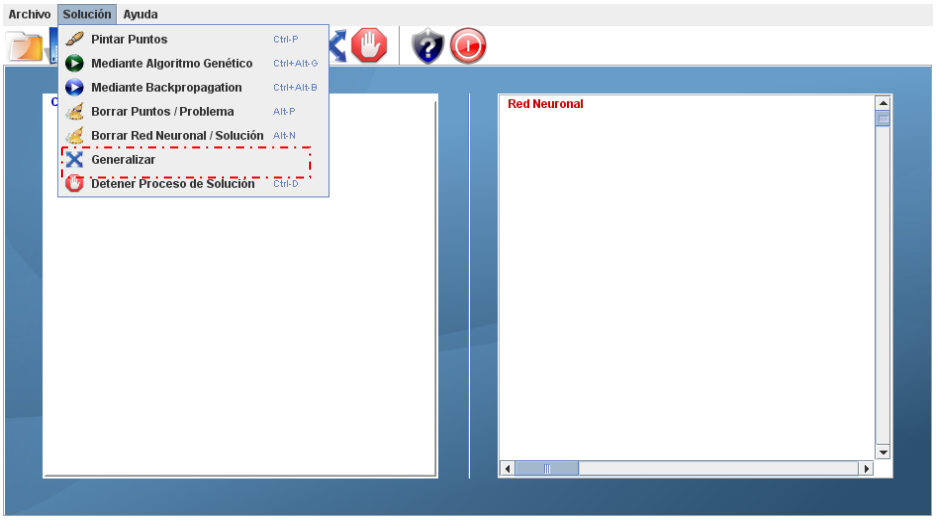


Figura A.13 Opción Generalización

Aparecerá una ventana emergente donde el primer paso es cargar la imagen matriz o plantilla de donde fueron extraídos los puntos de entrenamiento. Para iniciarla presionar el botón ubicado en la parte inferior de la ventana emergente (Véase figura A.14).

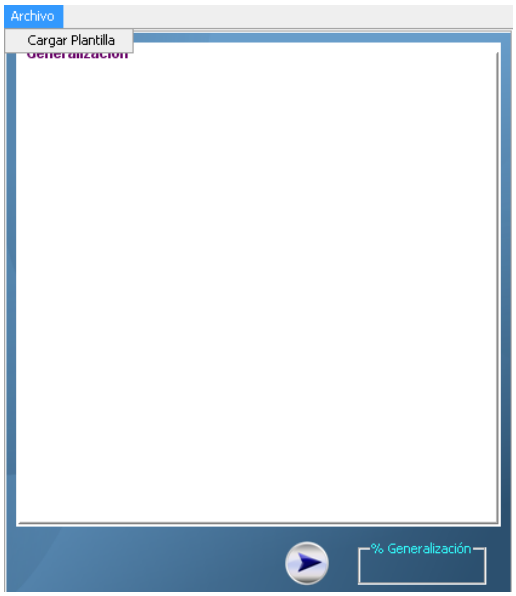


Figura A.14 Ventana Emergente Generalización

3.2.7 OPCIÓN DETENER PROCESO DE SOLUCIÓN

En esta opción el usuario podrá detener la ejecución del Algoritmo Genético o del Backpropagation (véase figura A.15)

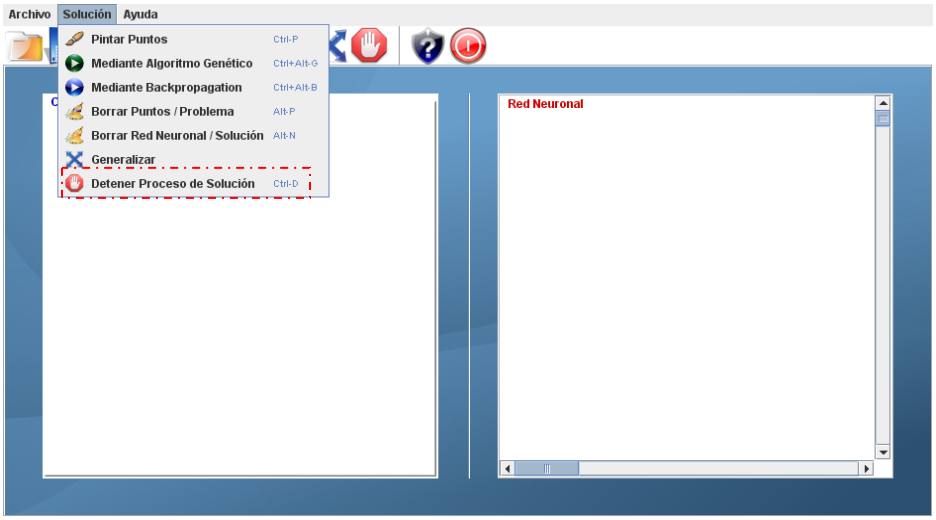
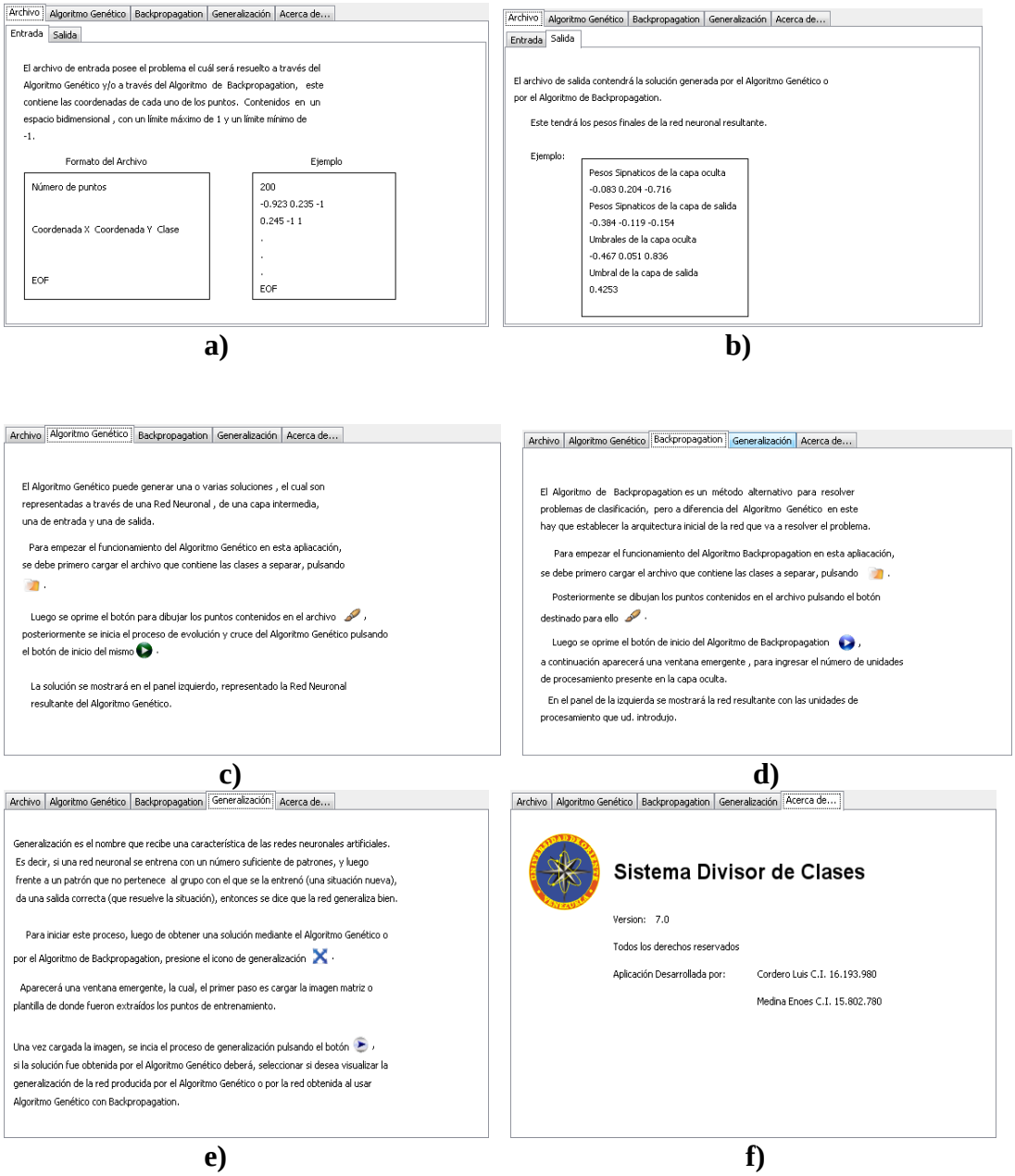


Figura A.15 Opción Detener Proceso De Solución

3.3 MENÚ AYUDA

Este menú le facilitara toda la información necesaria para el buen manejo de la aplicación, este está dividido en 6 secciones las cuales son: Archivo, Algoritmo Genético, Backpropagation, Generalización, Acerca de. (Véase figura A.16)



a) **b)** **c)** **d)** **e)** **f)**

Figura A.16 a) Ayuda para el Archivo de Entrada, **b)** Ayuda para Archivo de Salida, **c)** Ayuda para Algoritmo Genético, **d)** Ayuda para Backpropagation, **e)** Ayuda para Generalización, **f)** Acerca de.

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

TÍTULO	**DESARROLLO DE UNA APLICACIÓN QUE PERMITA EVOLUCIONAR REDES NEURONALES ARTIFICIALES PARA RESOLVER PROBLEMAS DE CLASIFICACIÓN**
SUBTÍTULO	

AUTOR (ES):

APELLIDOS Y NOMBRES	**CÓDIGO CULAC / E MAIL**
CORDERO CH., LUIS ANTONIO	**CVLAC: 16.193.980** **E MAIL: chinochang22@hotmail.com**
MEDINA L., ENOES CAROLINA	**CVLAC: 15.802.780** **E MAIL: enoescarolina@hotmail.com**
	CVLAC: **E MAIL:**
	CVLAC: **E MAIL:**

PALÁBRAS O FRASES CLAVES:

DESARROLLO APLICACIÓN REDES NEURONALES ARTIFICIALES ALGORITMO GENÉTICO BACKPROPAGATION PROBLEMAS DE CLASIFICACIÓN

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

ÀREA	SUBÀREA
INGENIERÍA Y CIENCIAS APLICADAS	**INGENIERÍA EN COMPUTACIÓN**

RESUMEN (ABSTRACT):

En el presente trabajo se desarrolló una Aplicación que permitiera evolucionar Redes Neuronales Artificiales para resolver problemas de clasificación. El software desarrollado le brinda al usuario una solución expresada como una Red Neuronal Artificial que puede ser obtenida de dos formas: Algoritmo Genético y Algoritmo Backpropagation; el usuario puede elegir que método usar. Esta aplicación se realizó utilizando tecnologías de software libre, rigiéndose por el decreto 3390, el cual dispone que "La Administración Pública Nacional deberá emplear prioritariamente Software Libre desarrollado con Estándares Abiertos, en todos sus Sistemas, Proyectos y Servicios Informáticos". El desarrollo del proyecto estuvo dirigido por la metodología OMT (*Object Modeling Technique*), la cual utiliza el análisis y el diseño orientado a objetos por medio del Lenguaje de Modelado UML. Para construir el software se utilizó JAVA como lenguaje de programación

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

CONTRIBUIDORES:

APELLIDOS Y NOMBRES	ROL / CÓDIGO CVLAC / E_MAIL				
BASTARDO, JOSÉ L.	**ROL**	**CA**	**AS** X	**TU**	**JU**
	CVLAC:	**6.890.832**			
	E_MAIL	**josebastardo@gmail.com**			
	E_MAIL				
CORTÍNEZ N., CLAUDIO A.	**ROL**	**CA**	**AS**	**TU**	**JU X**
	CVLAC:	**10.299.576**			
	E_MAIL	**cl_cortinez@hotmail.com**			
	E_MAIL				
ROJAS, LUIS F.	**ROL**	**CA**	**AS**	**TU**	**JU** X
	CVLAC:	**10.945.922**			
	E_MAIL	**lrojas@anz.udo.edu.ve**			
	E_MAIL				
	ROL	**CA**	**AS**	**TU**	**JU**
	CVLAC:				
	E_MAIL				
	E_MAIL				

FECHA DE DISCUSIÓN Y APROBACIÓN:

2009	04	14
AÑO	**MES**	**DÍA**

LENGUAJE. SPA

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

ARCHIVO (S):

NOMBRE DE ARCHIVO	TIPO MIME
TESIS. Desarrollo De Una Aplicacion.doc	application/msword

CARACTERES EN LOS NOMBRES DE LOS ARCHIVOS: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z. a b c d e f g h i j k l m n o p q r s t u v w x y z. 0 1 2 3 4 5 6 7 8 9.

ALCANCE

ESPACIAL: _____Redes Neuronales Artificiales___ (**OPCIONAL**)

TEMPORAL: _____6 meses_____ (**OPCIONAL**)

TÍTULO O GRADO ASOCIADO CON EL TRABAJO:

Ingeniero en Computación

NIVEL ASOCIADO CON EL TRABAJO:

Pregrado

ÁREA DE ESTUDIO:

Departamento de Computación y Sistemas

INSTITUCIÓN:

Universidad de Oriente, Núcleo de Anzoátegui

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

DERECHOS

De acuerdo al artículo 44 del reglamento de trabajos de grado

Los trabajos de grado son de exclusiva propiedad de la Universidad de Oriente y sólo podrán ser utilizados para otros fines con el conocimiento del Consejo de Núcleo respectivo, quién lo participará al Consejo Universitario

Cordero Ch., Luis Antonio.	Medina L., Enoes Carolina
AUTOR	**AUTOR**

Bastardo, José Luís	Cortinez, Claudio	Rojas, Luis
TUTOR	**JURADO**	**JURADO**

POR LA SUBCOMISION DE TESIS