Departamento de Informática

PROYECTO FIN DE CARRERA

ALGORITMOS GENÉTICOS PARA LA RESOLUCIÓN DEL PROBLEMA DE BCI

Autor: Ángel Cano Monedero

Tutora: Sandra García Rodríguez

Leganés, 15 de diciembre de 2011

Título: ALGORITMOS GENÉTICOS PARA LA RESOLUCIÓN DEL PROBLEMA DE BCI.
Autor: ÁNGEL CANO MONEDERO
Director:

EL TRIBUNAL

Presidente:_____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 15 de Diciembre de 2011 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Son muchísimas las personas a las que debo agradecer el haber llegado hasta aquí y por ello me gustaría reservarles aunque sea unas pocas líneas de este proyecto para reconocer el apoyo que he recibido en estos años de carrera en los que, sin duda, de todas esas personas, las que han jugado un papel muy especial son mi familia que ha estado a mi lado en todo momento.

Además de mi familia, quiero agradecer a todos mis amigos sobre todo por su apoyo durante esta etapa de mi vida y por haberme ayudado a evadirme y divertirme en los ratos que tenía libres. No hubiera terminado nada sin vosotros.

A todos los que me han ido acompañado en mi vida laboral pasada y presente, en cada uno de los lugares en los que he trabajado, siempre me apoyasteis y me enseñasteis cosas más allá de las que se aprenden en la universidad.

A los que comenzaron siendo mis compañeros de clase y terminaron siendo amigos, por todas las horas estudiando en la biblioteca, por todas las prácticas que nos ha tocado hacer juntos, por todos esos momentos que vivimos día a día durante toda esta etapa de nuestras vidas. Sin vosotros habría sido todo mucho más complicado.

A ese enorme grupo de gente con los que me he divertido en fiestas, viajes, cumpleaños y participado año tras año en multitud de competiciones de deportes en la universidad, donde lo importante era divertirnos fuera cual fuera el resultado. En más de una ocasión me demostrasteis que también estábamos para ayudarnos en lo que hiciera falta. Ahora continuaremos haciéndolo fuera de la universidad. Seguimos siendo un equipo.

Y por supuesto, a Kath por ayudarme con los textos en inglés y a mi tutora Sandra por su paciencia, ayuda y entrega en todo momento.

A todos vosotros muchas gracias.

Resumen

Este proyecto se encuentra situado en el campo de la inteligencia artificial, dentro de la rama de aprendizaje automático. El objetivo ha sido desarrollar un programa que sea capaz de clasificar correctamente los patrones de datos obtenidos a partir de un sistema BCI (Brain Computer Interface). Por medio de este sistema, se obtienen 96 señales eléctricas agrupadas en 8 canales que son producidas por el cerebro del usuario en el momento en el que éste intenta realizar una de las tres acciones requeridas. Estas señales serán preprocesadas y almacenadas en ficheros por medios externos a nuestro programa.

Nuestro sistema está constituido por un algoritmo genético simple en cuya función fitness utiliza un clasificador formado por perceptrones simples. La tarea principal de la clasificación de patrones es realizada por el clasificador de perceptrones simples los cuales utilizan las 96 señales eléctricas y sus operaciones con el fin de introducir datos no lineales a los perceptrones simples. El algoritmo genético se ocupará de seleccionar el conjunto de atributos más válidos. La experimentación demuestra que esta aproximación es capaz de clasificar con el mínimo error posible los patrones de datos obtenidos del BCI desechando en cada ronda del algoritmo los datos que no son útiles o que introducen ruido.

Palabras clave: Inteligencia artificial, aprendizaje automático, red neuronal artificial, perceptrón simple, algoritmos genéticos, computación evolutiva, clasificación de datos.

Abstract

This project lies within the artificial intelligence field, in the branch of machine learning. The aim has been to develop a programme that is able to correctly classify the data patterns obtained from a BCI (Brain Computer Interface) system. By means of this system, 96 electric signals are obtained, grouped into 8 channels that are produced by the user's brain in the moment he or she tries to carry out one of the three required actions. These signals will be pre-processed and stored in files through external applications to this programme.

Our system is designed using a simple genetic algorithm whose fitness function uses a pattern classifier formed by simple perceptrons. The main task of the pattern classification is carried out by simple perceptron classifiers, which use the 96 electric signals and their operations in order to introduce non-linear data to the simple perceptrons. The genetic algorithm will select the collection of most valid attributes. Experimentation has proved that this approximation is able to classify the data patterns obtained from the BCI with the minimum error possible, rejecting in each round of the algorithm the data which is not useful or which introduces some noise.

Key words: Artificial intelligence, machine learning, neural networks, perceptrons, genetic algorithms, evolutionary computation, data classification.

Índice general

Índice de ecuaciones

Índice de figuras

Índice de tablas

Índice de gráficas

Capítulo 1

Introducción y objetivos

1.1 Introducción

En un intento de facilitar la vida a miles de personas, los sistemas BCI o interfaz cerebro-computadora están siendo investigados y su uso se está extendiendo al posibilitar que gente con discapacidades físicas puedan realizar operaciones que no podían realizar. Estos sistemas consisten en captar las señales eléctricas producidas por el cerebro para posteriormente tratarlas y conseguir realizar la comunicación con el ordenador.

Para la realización de este proyecto, hemos utilizado unos ficheros que contienen datos que fueron obtenidos con esa técnica. Estos ficheros contienen miles de registros de señales eléctricas referentes a tres patrones diferentes de datos: desplazar el ratón a la izquierda, desplazar a la derecha y clic. Cada registro poseerá 96 señales eléctricas obtenidas para un instante de tiempo y corresponderá a un sólo patrón de datos de los tres existentes. Por ello, deberemos realizar un programa capaz de distinguir estas tres clases diferentes de patrones existentes en los ficheros. En definitiva, tendremos que clasificar estos registros determinando a qué tipo de patrón corresponden y usando para ello las 96 señales eléctricas que lo forman más los productos obtenidos de la multiplicación, división, resta y suma de sus datos. Esto hace que tengamos una cantidad grande de datos por cada registro a clasificar y que además, no todos estos datos nos sean útiles. Algunas de estas señales perjudican a la clasificación introduciendo ruido ya que pueden ser impulsos del cerebro que no afectan al pensamiento. Deberemos encontrar la forma de seleccionar los datos con el objetivo de quedarnos sólo con los que realmente nos sirvan en la clasificación.

Para la clasificación de datos no siempre es fácil implementar soluciones que sean capaces de clasificar un conjunto de patrones de datos. Debemos dejar a un lado los paradigmas convencionales de programación y aplicar otros paradigmas como el aprendizaje. Nosotros hemos decidido

emplear el uso de redes neuronales (y más concretamente el perceptrón simple) las cuales son eficaces para estas tareas. Estas redes necesitarán un período de aprendizaje en el que usarán un conjunto de patrones de entrenamiento (parte de los registros de datos anteriores). Una vez la red esté entrenada, ya estará lista para poder distinguir entre los diferentes patrones de datos con el mínimo error posible logrando cumplir así gran parte de nuestro objetivo principal.

Pero esta tarea de clasificación no será realmente efectiva si no implementamos un método que consiga seleccionar las señales u operaciones que utilizaremos para ella. De esta tarea se ocupará un algoritmo genético que, combinando sus diferentes operadores y características, seleccione los mejores atributos o señales eléctricas. Además esta reducción de señales también redundará en una disminución del tiempo computacional de nuestra solución.

En resumen, la solución que trataremos a lo largo del proyecto deberá ser capaz de clasificar estos registros de datos obtenidos por el BCI con el mínimo error posible. Para esto deberemos encontrar la forma de realizar una selección de las 96 señales eléctricas existentes para cada registro con el fin de quedarnos sólo con las que verdaderamente nos son útiles, desechando aquellas señales que no aporten información útil para nuestra clasificación. Por ello, nuestra solución se compondrá principalmente de dos partes: la red neuronal que usaremos para la clasificación de los datos y el algoritmo genético que se encargará de la selección de los atributos.

1.2 Objetivos

El objetivo principal de este proyecto es desarrollar un método para resolver el conocido problema de BCI, que consiste en la clasificación de patrones de datos en tres clases diferentes. Además de este objetivo, tenemos otros objetivos complementarios como el encontrar los atributos que sean relevantes y ayuden a la clasificación (ignorando en la medida de lo posible aquellos que solo metan “ruido” al clasificador y por lo tanto no nos sean útiles) o introducir combinaciones entre estos atributos (mediante operaciones matemáticas). Una vez conseguidos estos objetivos fundamentales, podremos intentar añadir mejoras (opcionales) al sistema como el intentar disminuir ligeramente el número de entradas del clasificador consiguiendo reducciones en el tiempo de ejecución.

En definitiva, todos los objetivos que pretendemos conseguir con la realización de este proyecto y su orden son los siguientes:

- Obtener los conocimientos necesarios para conocer el alcance del problema y las diferentes materias que pueden llegar a dar solución a este.

- Desarrollar un algoritmo genético que sea capaz de seleccionar los atributos más relevantes para la clasificación descartando aquellos que no son útiles.

- Encontrar una red neuronal que nos permita la clasificación de los datos de forma correcta con el mínimo error posible.

- Realizar un conjunto de pruebas de test para hacer un estudio que nos sirva para determinar cuáles son los mejores parámetros o características del algoritmo genético, con el fin de tratar de obtener el máximo rendimiento posible a nuestra solución.

- Por último y como añadido, veremos si tenemos la posibilidad de implementar operadores del algoritmo genético que consigan reducir al máximo el número de entradas del clasificador sin afectar a los resultados de éste. La finalidad será conseguir reducir el tiempo computacional (al reducir las entradas del clasificador el tiempo de ejecución también disminuye) obteniendo resultados similares en la clasificación.

1.3 Fases del desarrollo

Las fases de desarrollo de este proyecto se pueden dividir de forma sencilla en:

- **Análisis**: Es la fase donde se estudió el problema al que nos enfrentábamos, vimos las diferentes formas posibles de solucionarlo, nos documentamos sobre las materias y temas que tratamos, determinamos que requisitos debía cumplir y se realizo un esquema primitivo de nuestra solución.

- **Diseño de la solución**: Buscamos el lenguaje de programación adecuado para la realización del proyecto y el compilador que se iba a usar. También se estudió que tipo de recursos iban a ser necesarios para la realización de todo el proyecto. Finalizamos esta etapa con un esquema pormenorizado de todo el proyecto a implementar.

- **Desarrollo de la solución**: Aquí se realizó la codificación del problema y algunas pruebas complementarias para asegurarme del correcto funcionamiento del algoritmo implementado.

- **Experimentación y evaluación de la solución**: En esta fase se llevo a cabo un conjunto de pruebas con el fin de ver el funcionamiento de nuestro algoritmo genético y tratar de hallar cuáles eran los parámetros correctos tanto para nuestra red neuronal como para nuestro algoritmo genético. Gracias a ellas pudimos determinar el comportamiento de la solución implementada.

- **Desarrollo de la documentación**: En esta etapa del proyecto se preparó el presente documento con el fin de explicar todo lo realizado y enseñar las conclusiones a las que he llegado.

1.4 Medios empleados

La principal herramienta empleada para la realización de este proyecto ha sido mi ordenador personal. Se trata de un ordenador portátil con un procesador Pentium i3 2.23 GHz y 4GB de RAM. El sistema operativo que he usado principalmente para la realización de todo el proyecto ha sido Ubuntu (en las diferentes versiones que han ido surgiendo a lo largo de la realización del proyecto) aunque en alguna ocasión se ha usado Windows 7 para algunos aspectos de la documentación.

En cuanto a las herramientas software utilizadas en la parte del proyecto relativa a la redacción de este documento, fue necesario emplear un procesador de texto (usamos tanto Word 2007 como Openoffice.org Writer) además de programas de edición de imágenes como GIMP, programas para la edición de diagramas y gráficos como DIA y hojas de cálculo de Microsoft Excel y Openoffice.org Calc. En el caso de la realización de las gráficas utilizamos Gnuplot y su front-end PlotDrop y en cuanto al presupuesto del proyecto usamos el programa Microsoft Proyect 2010.

En lo que se refiere a la implementación de la solución, hemos usado el editor de textos gedit, el compilador gcc para poder compilar el código y los ficheros de datos utilizados para el entrenamiento de la red y para el test. Estos ficheros de datos que contienen registros obtenidos por BCI pre-procesados y que contienen tres clases diferentes de datos no fueron obtenidos por nosotros durante la realización del proyecto sino que utilizamos unos ya existentes. Todo el código fuente de la aplicación ha sido realizado en el lenguaje de programación C.

El lenguaje de programación C fue desarrollado a principios de los años 70 en los laboratorios Bell, y hoy en día es uno de los más populares. Las principales características que hacen que este lenguaje sea tan popular y usado son [1]:

- Presenta características de bajo nivel con lo que trabaja con la misma clase de objetos que los computadores lo que hace que sus programas sean más eficientes.

- Está muy asociado con el sistema operativo UNIX.

- Es muy adecuado para la programación de sistemas ya que C permite tener un control casi absoluto del computador.

- Sus programas pueden ejecutarse casi sin cambios en multitud de máquinas.

- Permite la programación estructurada y el diseño modular facilitando la compresión y el mantenimiento de sus programas.

- Tiene sentencias de control sencillas y funciones lo que le hace ser muy flexible. Además contiene muchas funcionalidades añadidas por medio de bibliotecas, como el manejo de archivos, funciones matemáticas, etc.

- Acceso de memoria a bajo nivel por medio del uso de punteros.

- Es adecuado para cualquier tipo de aplicación ya que su diseño fue hecho sin limitaciones. Se ajusta al diseño bien jerárquico.

Todas estas características que acabamos de enumerar fueron en gran parte las que hicieron que nos decantáramos por este lenguaje frente a otros muchos disponibles.

Y en lo que respecta a la parte de experimentación, además de usar el ordenador portátil descrito anteriormente, en alguna ocasión también fue necesario emplear el servidor de la universidad para aquellas pruebas que requerían más tiempo de ejecución.

Todas estas herramientas, tanto software como hardware, descritas anteriormente han sido las empleadas para la realización del proyecto. Lo que hizo que eligiésemos las aplicaciones nombradas anteriormente frente a otras fueron aspectos como estar familiarizado con ellas, el tener mayor experiencia en su uso, mayor funcionalidad, etc.

1.5 Estructura de la memoria

Tal y como aparece en el índice anterior, este proyecto fin de carrera se divide en 6 capítulos más las referencias y anexos. A modo de resumen y para facilitar la compresión del documento, a continuación describiremos el contenido de cada una de estas partes.

- El presente capítulo **Introducción y objetivos** trata sobre la motivación que nos ha llevado a realizar este proyecto, los objetivos que esperamos conseguir con él y la estructura de este documento.

- En el capítulo 2, **Estado del arte** resumiremos los distintos conceptos teóricos necesarios para comprender los diferentes campos que abarcan tanto nuestro problema como nuestra solución. Con él intentaremos dar una visión general sobre algoritmos genéticos, redes de neuronas y todo lo necesario para comprender la solución propuesta en los siguientes capítulos.

- En el capítulo 3, **Diseño e implementación de la solución** se describirá nuestra solución implementada así como las diferentes razones que nos ha llevado a ella. Describiremos todo tipo de parámetros y opciones posibles sobre el algoritmo genético y el clasificador.

- El capítulo 4, **Experimentación** recogerá las pruebas realizadas primeramente con el fin de resolver nuestras dudas en la elección del clasificador correcto y a posteriori para ver el funcionamiento del algoritmo genético variando las distintas opciones disponibles.

- El capítulo 5, **Presupuesto** resume los costes que han sido necesarios para la realización del proyecto.

- El capítulo 6, **Conclusiones y líneas futuras** estará compuesto por las conclusiones a las que hemos llegado tras la realización de las pertinentes pruebas y además también contendrá las posibles mejoras o campos de investigación futuros que creemos que podrían llegar a hacer mejorar los resultados de nuestro trabajo.

Por último incluiremos las **referencias** que hemos utilizado para la realización del proyecto y los **anexos** que creemos necesarios para la completa explicación del proyecto.

Capítulo 2

Estado del arte

2.1 Introducción

En este capítulo intentaremos explicar de forma clara y sencilla los conceptos teóricos del problema a solucionar con el fin de dejar claro todos los aspectos sobre lo que se sustenta la solución obtenida.

Este capítulo se subdivide en diferentes secciones en las que hablaremos de diferentes temas tales como el problema del BCI, conceptos básicos sobre redes neuronales artificiales (perceptrón simple en concreto), explicaremos de forma general el funcionamiento y la teoría de los algoritmos genéticos y conceptos sobre la optimización general.

2.2 BCI (Brain-Computer Interface)

La interfaz cerebro computadora (“Brain-Computer Interface” o BCI en inglés) es un medio de comunicación entre las funciones mentales o cognitivas de una persona o un animal y un medio externo como por ejemplo un ordenador o un hardware especifico. Consiste en captar las señales eléctricas producidas por el cerebro para poder pre-procesarlas, clasificarlas y conseguir comunicarse con el medio externo (ordenador). Estos procesos pueden servir de gran ayuda y mejorar la calidad de vida a personas que sufren discapacidades motoras ayudando a mover una silla de ruedas, prótesis, manejo de un puntero en una pantalla de un ordenador, brazos robotizados, etc.

Estas señales provenientes de la actividad neuronal del individuo pueden ser captadas de forma invasiva como la electrocorticografía (los electrodos son ubicados directamente en la superficie cerebral mediante una

craneotomía) o de forma no invasiva como la electroencefalografía (por medio de electrodos en el cuero cabelludo) [2]. En el caso de que dichas señales sean conseguidas de forma no invasiva, su registro no representa la actividad de una neurona en solitario sino que constituye una suma de la actividad de las neuronas cercanas a la región en la cual estamos llevando a cabo el seguimiento. La electroencefalografía permite pues el registro de la actividad eléctrica de distintas áreas cerebrales a partir de unos electrodos externos. Es necesario aclarar que una vez hayamos conseguido captar estas señales es necesario procesar la información, interpretarla y asociarla a las intenciones voluntarias del sujeto.

Figura 1. Imagen donde se observa el sistema de extracción de datos.

(Esta fotografía ha sido extraída de [3]).

En el caso particular que nos atañe, está comunicación entre el cerebro y el ordenador se realiza con el fin de poder manejar un puntero en una pantalla (movimiento a la izquierda, a la derecha y pulsación de clic) utilizando las señales eléctricas producidas por la actividad cerebral del sujeto que pretende moverlo [3]. Esta información fue conseguida de forma no invasiva por medio de 96 electrodos divididos en 8 canales. Por lo tanto, nuestro problema en sí consistirá en recibir los datos obtenidos por medio de la electroencefalografía del sujeto ya procesados y tratar de descubrir por medio de nuestro algoritmo de cuál de las tres clases anteriormente explicadas se trata.

2.3 Redes neuronales artificiales

2.3.1 Introducción general

Las redes de neuronas artificiales son un paradigma de aprendizaje automático inspirado en la forma en que funciona el sistema nervioso de las personas. Consisten en una simulación de las propiedades del cerebro humano a través de modelos matemáticos y ordenadores u otros mecanismos artificiales (circuitos integrados, robots). El objetivo de estas

redes es conseguir programas de ordenador o máquinas que den respuestas similares a las que es capaz de dar el cerebro humano.

Es un hecho para todos que las personas son capaces de solucionar muchos problemas de forma sencilla que para una computadora sería más complejo de solucionar. Esto es debido a que las computadoras realizan operaciones mecánicas sin ningún problema basándose en la realización de instrucciones sobre un dato y en cambio tienen grandes dificultades en otro tipo de acciones como la clasificación y el reconocimiento de patrones, la generalización de eventos futuros a partir de otros pasados, etc. [4].

Los avances en la investigación biológica han ayudado a profundizar en los conocimientos de los mecanismos del pensamiento natural. El cerebro humano almacena la información en forma de patrones. Durante años los científicos persiguieron la construcción de estas redes con el fin de imitar el cerebro humano y conseguir almacenar información en forma de patrones, utilizarlos y resolver problemas con ellos [4].

Los primeros estudios de cómo trabajar con este tipo de redes fueron hechos por MCCulloch y Pitts mostrando la habilidad de un grupo de células conectadas para realizar ciertas funciones lógicas. Este campo fue creciendo y en 1958 se introdujo la primera red de neuronas artificiales que tenía capacidad de aprendizaje. Esta red es el perceptrón simple. (Explicado más detalladamente en la sección 2.3.6 El perceptrón simple).

Tras muchas investigaciones se comprobaron y demostraron muchas limitaciones a esta red por lo que se dejó de lado la investigación de este campo. En cambio, algunos investigadores fueron viendo que la solución de algunas de estas limitaciones como la de solucionar ciertos problemas no lineales podía conseguirse con la combinación de varios perceptrones simples. En los años 80 las redes de neuronas volvieron a surgir a partir de que en 1986 se propusiera el perceptrón multicapa y el algoritmo de retropropagación para el cálculo de los pesos de esta red multicapa.

En definitiva, las redes neuronales artificiales son estructuras de procesamiento paralelo distribuido, biológicamente inspiradas, en las que unos elementos simples llamados neuronas artificiales se interconectan entre ellas a través de arcos llamados conexiones y con el medio externo a través de entradas. Cada una de estas conexiones o entradas tiene asignado un peso.

Estas neuronas se distribuyen en capas de distintos niveles con conexiones que las unen a neuronas de otras capas o de la suya misma de forma que la salida de una de las neuronas generalmente sirve como entrada a otras neuronas. El objetivo final es que el conjunto de la red neuronal consiga obtener una salida como respuesta a las señales de entrada.

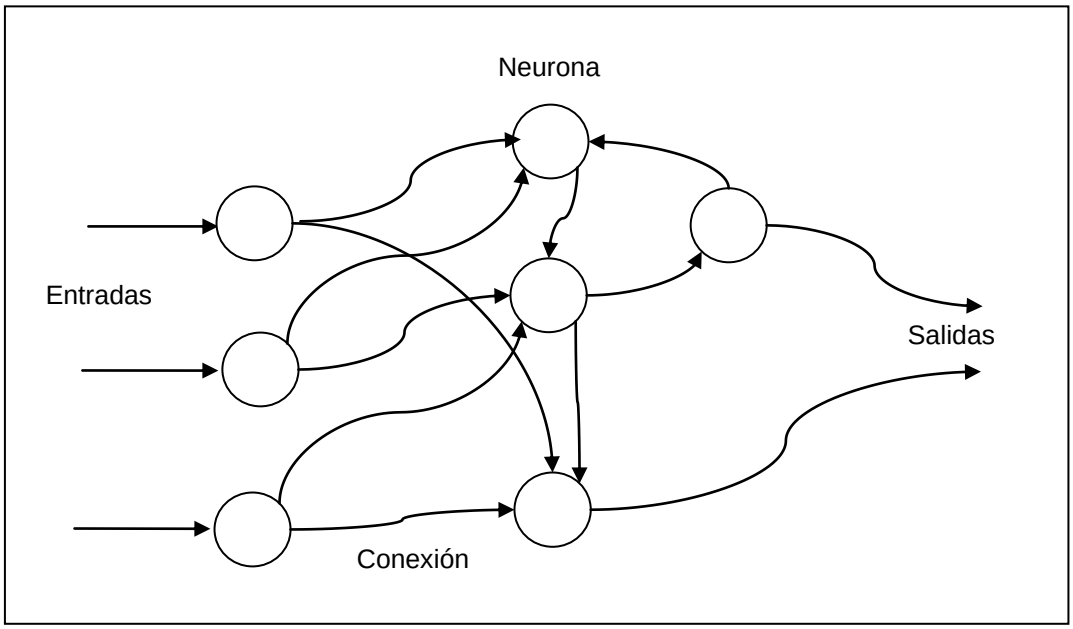


Figura 2. Red de neuronas con sus distintas partes

2.3.2 Neurona

La neurona artificial es la unidad elemental de las redes de neuronas artificiales. Sobre ella se fundamenta la operación de la red neuronal artificial. Cada neurona recibe una serie de entradas a través de sus interconexiones. Estas interconexiones tienen un peso asignado. La información se procesa a partir de operaciones simples y la neurona emite una salida como respuesta de estas señales de entrada.

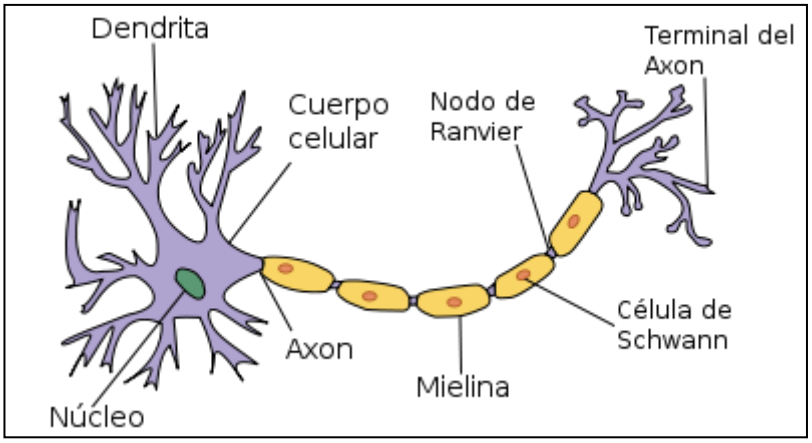


Figura 3. Modelo genérico de una neurona biológica.

(En ella se basa la neurona artificial. Esta imagen no es propia y ha sido extraída de [5]).

Los pesos son los elementos que representan la configuración de la red. Estos serán los que se modificarán o adaptarán en el proceso de entrenamiento consiguiendo que la neurona se adapte al problema.

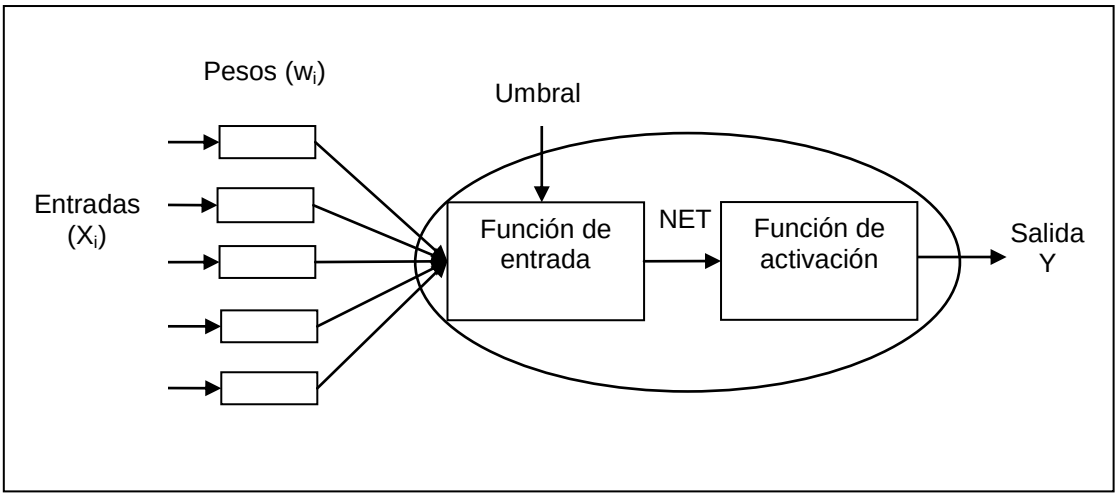


Figura 4. Modelo genérico de una neurona artificial.

Como se puede ver en la Figura 4 esta salida de la neurona viene dada por varias funciones:

- **La función de entrada** que consiste en el sumatorio de cada entrada multiplicada por el peso correspondiente a esa entrada más la suma de una entrada adicional llamada Umbral.

$$NET = \sum_{i=1}^{n} \left((x_i . w_i) + U \right)$$

Ecuación 1. Función de entrada de la neurona.

- **La función de activación** que es la que determina la salida Y de la neurona transformando la función de entrada y adaptándola para que esta se acote a unos determinados valores. Define el nuevo estado de la neurona según su estado de excitación.

$$Y = F(NET)$$

Ecuación 2. Función de activación de la neurona.

Esta función de activación puede ser de diferentes tipos. En la siguiente tabla podemos observar algunos de ellos:

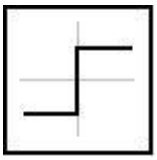
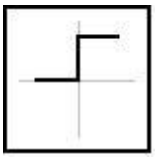
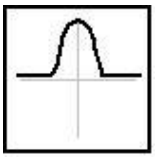
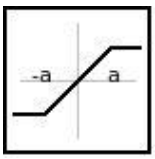
	Función	Rango	Gráfica
Función Identidad	F(x)= x	(-α,α)	
Función sigmoidea	$F(x) = \frac{1}{1+e^{-x}}$	(0,1)	
Función umbral	$F(x)=\begin{cases} 1 & \text{si } x>0 \\ -1 & \text{si } x\leq 0 \end{cases}$	(-1,1)	
Función Umbral (0,1)	$F(x)=\begin{cases} 1 & \text{si } x>0 \\ 0 & \text{si } x\leq 0 \end{cases}$	(0,1)	
Función gaussiana	$F(x) = e^{\frac{-x^2}{2}}$	(0,1)	
Función lineal a tramos	$F(x)=\begin{cases} 1 & \text{si } x>a \\ x & \text{si } a \geq x \geq -a \\ -1 & \text{si } x<-a \end{cases}$	(-1,1)	

Tabla 1. Algunos diferentes tipos de funciones de activación.

(Basada en [4] y [6])

Por lo tanto, la salida de la neurona dependerá de las señales de entrada, de los pesos asociados a cada una de éstas, del umbral y de las funciones de activación y de entrada. De todas estas características los pesos y el umbral serán las que se adapten en el proceso de aprendizaje [7].

2.3.3 Tipos de redes según su topología

Las redes neuronales artificiales se pueden clasificar de varias formas. Una de ellas es su clasificación dependiendo del patrón de conexiones que presentan sus neuronas. Siguiendo este patrón podemos diferenciar dos tipos básicos de redes [4]:

- **Redes de propagación hacia delante:** En estas redes todas las conexiones y la propagación de las señales se realizan hacia delante. De la entrada hacia la salida. No se producen ningún tipo de ciclos o de realimentación. Dentro de este grupo podemos distinguir entre:
    - Redes monocapa: Estas redes solo contienen una capa de neuronas como podemos ver en la siguiente imagen. El perceptrón simple pertenece a este grupo.

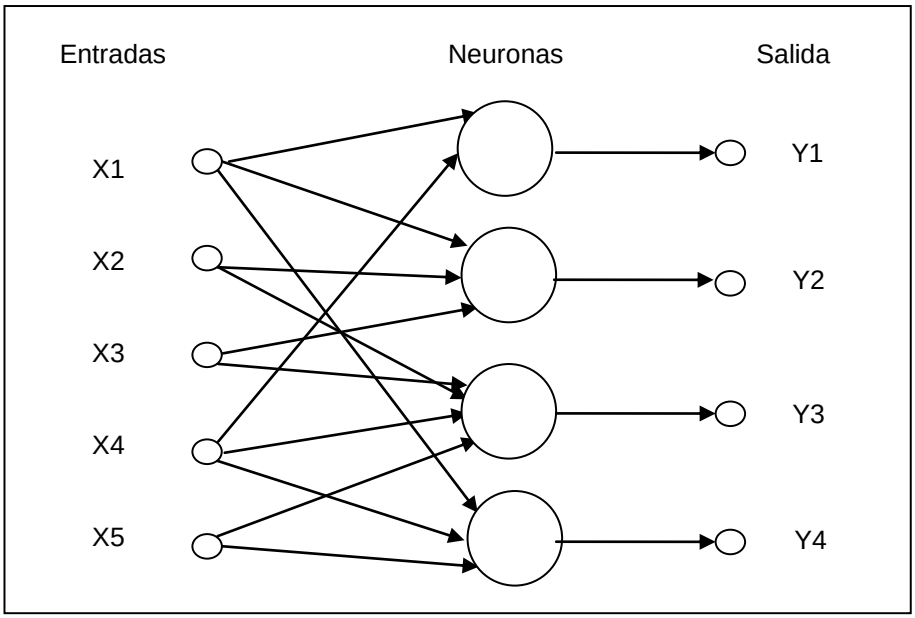


Figura 5. Red neuronal monocapa

    - Redes multicapa: Estas redes contienen varias capas de neuronas. Incluyen una o varias capas intermedias u ocultas. Un ejemplo de este tipo de redes es el perceptrón multicapa. En la Figura 6 podemos ver un ejemplo simple de este tipo de redes.

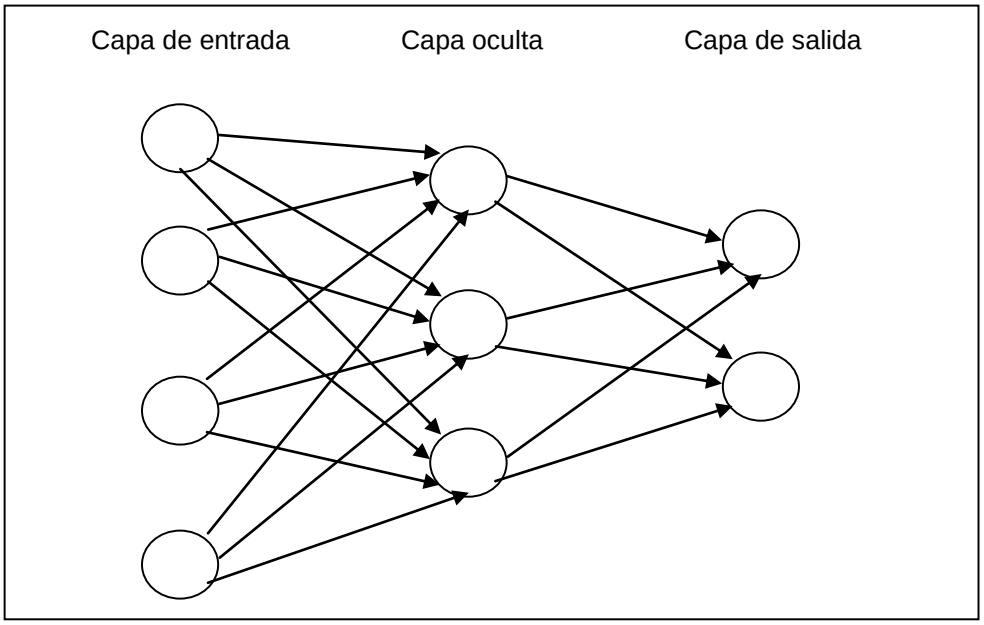


Figura 6. Red neuronal multicapa con una capa intermedia.

- **Redes recurrentes.** Presentan al menos algún ciclo o circuito cerrado dentro de la red neuronal. Con esto se consigue que las señales no solo se transmitan desde la entrada hacia la salida sino que también existirá una realimentación.

  Si este ciclo se produce sólo en una neurona, es decir, que la salida de la neurona es a su vez una de sus entradas, se denomina autorrealimentación.

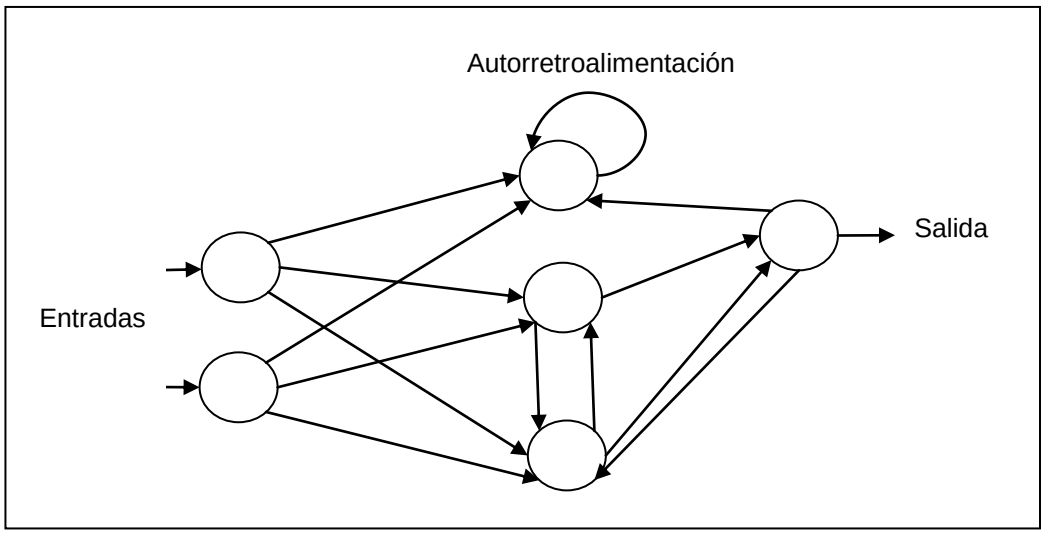


Figura 7. Red recurrente.

2.3.4 Entrenamiento de la red

Una de las principales características de las redes neuronales artificiales es la capacidad de aprender del entorno en el que actúa. Esta capacidad llamada aprendizaje consiste en adaptar los parámetros de la red neuronal como consecuencia de la entrada del conjunto de datos de entrenamiento para conseguir que la red de la respuesta deseada. Los parámetros que se modifican en este proceso son los pesos.

Este aprendizaje puede ser de varios tipos [4]:

- **Aprendizaje supervisado**. Para este aprendizaje se requiere un conjunto de entradas de las que se conoce ya su salida esperada. A este conjunto se le llama patrones de entrenamiento. La red recibe las entradas, las procesa y compara la salida obtenida con la salida esperada. En el caso de que ambas no coincidan, propaga el error por toda la red ajustando los pesos y consiguiendo así modificar el comportamiento de la red. El perceptrón simple que veremos más tarde (en la sección 2.3.6 El perceptrón simple) utiliza este tipo de aprendizaje.

- **Aprendizaje no supervisado.** Con este tipo de aprendizaje no se necesita un conjunto de patrones de entrenamiento. La red recibe los patrones de entrada sin conocer las salidas que se desean asociar a estas. Por lo tanto, la red modifica automáticamente sus parámetros al ir clasificando y agrupando los patrones de entrada según características que va encontrando en los datos.

- **Aprendizaje por grados o por fortalecimiento.** No se dispone de de vectores de entrenamiento, lo que se hace es contar con índice de rendimiento de la red. Se basa en que de forma externa se proporciona una valoración de la red.

2.3.5 Características de las redes de neuronas artificiales y utilidad

Las principales características de las redes de neuronas artificiales son las siguientes:

- Su estructura es paralela. Esto hace que puedan ser implementadas de forma paralela en computadoras u otros dispositivos mejorando su tiempo de ejecución.

- Auto organización. Estas son capaces de crear su propia representación de la información en su interior para poder trabajar.

- Aprendizaje. Tienen la habilidad de aprender a partir de ejemplos adquiriendo las propiedades deseadas a partir de ellos, como ya hemos visto anteriormente.

- Flexibilidad. Puede manejar cambios no importantes en la información de entrada (señales con ruido).

- Tolerancia a fallos. Gracias a su forma de almacenar la información estas pueden funcionar de forma correcta aunque parte de la red esté dañada.

Debido a todo esto, las redes neuronales artificiales son muy útiles para problemas en los que no se dispone de un modelo identificable que pueda ser programado pero se dispone de un conjunto amplio de ejemplos de entrada dada su capacidad para el aprendizaje y para la aproximación a partir de ejemplos.

Gracias a ello, las redes neuronales son muy efectivas y utilizadas para reconocimiento de patrones, compresión y análisis de datos, robótica, aproximación, predicción y clasificación de datos. También pueden usarse para casos en los que no existen modelos matemáticos para su resolución (por ejemplo, el problema del viajante) o usarse conjuntamente con algoritmos genéticos como será nuestro caso.

2.3.6 El perceptrón simple

El perceptrón simple es el modelo más sencillo de red neuronal. Fue desarrollado por Frank Rosenblatt en 1959. Es un modelo de red unidireccional o feedforward ya que no presenta ciclos y la información circula en un único sentido. Está inspirada en el modelo de célula de McCulloch-Pitts. Una de sus principales características es su carácter de neurona entrenable. Es muy utilizado para tareas de clasificación lineal ya que es capaz de discriminar entre dos clases de datos siempre que estos sean linealmente separables [7] y [8] (Ver Figura 8). En ella, en la imagen de la izquierda podemos ver un ejemplo de datos que son separables linealmente por un hiperplano. A la derecha, en cambio los datos no pueden separarse linealmente dado que la superficie de separación no es lineal.

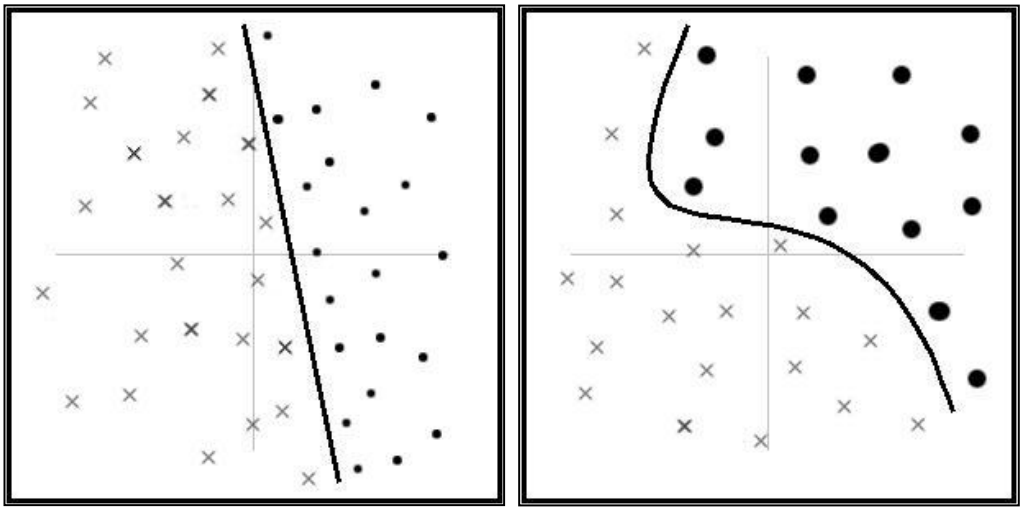


Figura 8. Ejemplo de datos separables y no separables linealmente.

Este modelo tiene un aprendizaje supervisado por lo que requiere de un conjunto de ejemplos o patrones de clases diferentes en la fase de entrenamiento para poder adaptar sus parámetros (pesos y umbral) al caso determinado y poder hallar el hiperplano capaz de separar ambas clases.

Por lo tanto, los perceptrones simples suelen utilizarse para tareas de clasificación o también para representación de funciones booleanas [9].

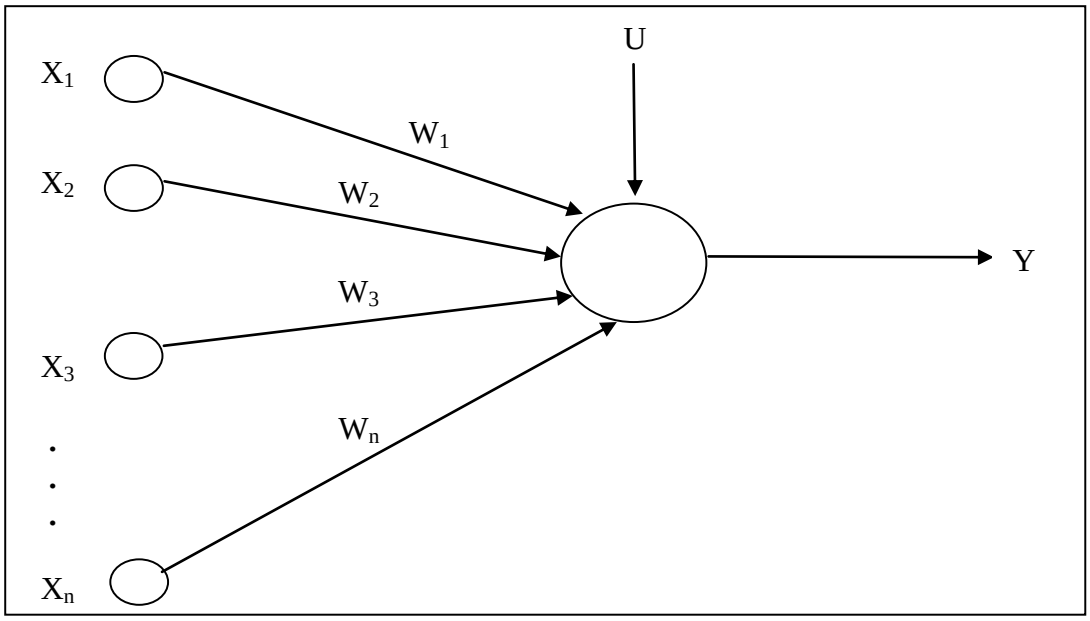


Figura 9. Estructura de un perceptrón simple.

La arquitectura de un perceptrón es muy simple y está formada por un conjunto de células de entrada que tienen asociadas a cada una de ellas un peso, y una o varias células de salida. Cada una de las células de entrada se conecta con todas las células de salida. Estas conexiones son las que

determinan la recta o el hiperplano discriminante entre las clases. Un sencillo ejemplo es el que se puede observar en la Figura 9 donde podemos ver un perceptrón simple con sus entradas ($X_1 \ldots X_n$), los pesos correspondientes a esas entradas ($W_1 \ldots W_n$) y una salida (Y). También existe otra entrada llamada umbral (U) [4], [8], [9] y [10].

Como vemos en la Figura 9, la salida del perceptrón (Y) se halla aplicando la función de activación a la suma ponderada por los pesos de todas las entradas más el umbral.

Por lo tanto, la salida y es:

$$Y = F\left(\sum_{i=1}^{n}(x_i.w_i)+U\right)$$

Ecuación 3. Función de entrada del perceptrón simple.

Donde F(x) es la función de activación, que es:

$$F(x)=\begin{cases}1 \text{ si } x>0\\ 0 \text{ en otro caso}\end{cases}$$

Ecuación 4. Función de activación del perceptrón simple.

A este tipo de función de activación se la conoce como función umbral. (Figura 10) También podría adaptarse para obtener otro tipo de valores como (-1,1).

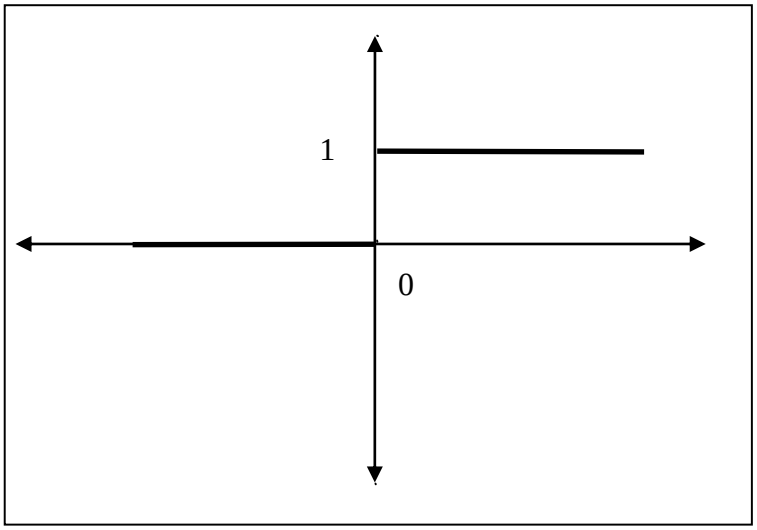


Figura 10. Efecto producido por la función de activación del perceptrón.

De este modo la salida de la función de activación del perceptrón es binaria, lo que facilita que esta salida pueda ser traducida fácilmente a una clasificación en dos clases [8].

Por ejemplo:

Y = 0. La entrada pertenece a la clase A.
Y = 1. La entrada pertenece a la clase B.

La ecuación del hiperplano que separara ambas clases sería la siguiente:

$$x_1.w_1 + x_2.w_2 + \ldots + x_n.w_n + U = 0.$$

Ecuación 5. Hiperplano.

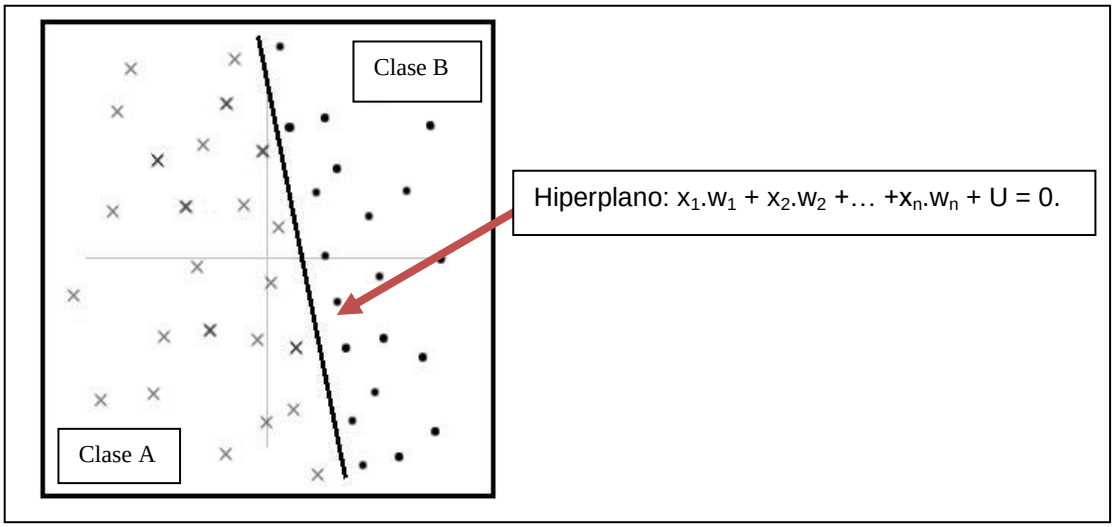


Figura 11. Imagen de un hiperplano discriminante.

2.3.6.1 Algoritmo de aprendizaje del perceptrón simple

La importancia de este tipo de red neuronal artificial es que ésta puede ser entrenada y que sean calculados de forma automática los parámetros para conseguir clasificar los patrones para los que ha sido entrenado. Este algoritmo de de aprendizaje del perceptrón simple es de los que se denominan de corrección de errores. Este lo que hace es ajustar los pesos en proporción a la diferencia producida entre la salida de la red y la salida que se deseaba [9].

Partiendo de unos pesos iniciales (aleatorios o escogidos), se van introduciendo a la red los patrones de entrenamiento. En el caso de que la salida producida por el perceptrón sea la deseada para ese patrón no se

actualizaría nada y se continuaría con el entrenamiento, pero en el caso de que el perceptrón nos devuelva una salida que no sea la correcta, se procede a actualizar los pesos y el umbral de la siguiente forma:

$$w_i\,(t+1) = w_i\,(t) + d(x).x_i \qquad u(t+1) = u\,(t) + d(x)$$

Ecuación 6. Aprendizaje del perceptrón.

Donde:

w son los pesos.
x son las entradas.
d(x) es la salida deseada para esa entrada.

Por lo tanto, este proceso de aprendizaje es iterativo, se parte de una configuración de pesos inicial y la entrada de patrones van produciendo que estos pesos vayan siendo modificados hasta que todos los patrones queden bien clasificados. Gráficamente este efecto se puede observar en la Figura 17 al ver como el hiperplano se va desplazando en cada corrección hasta conseguir separar ambas clases. Esta separación solo es posible si ambas clases son linealmente separables.

Para facilitar la compresión del funcionamiento del algoritmo de aprendizaje a continuación podemos ver un sencillo ejemplo de dos dimensiones donde el perceptrón simple va adaptando sus parámetros hasta conseguir clasificar correctamente el conjunto de puntos dado.

2.3.6.1.1 Ejemplo de funcionamiento

En la Figura 12 podemos ver representados en la gráfica cuatro puntos dados. Estos puntos se dividen en dos clases diferentes. El punto (2,1) y el punto (0,-1) son de la clase A y los puntos (0,1) y el (-2,-1) pertenecen a la clase B.

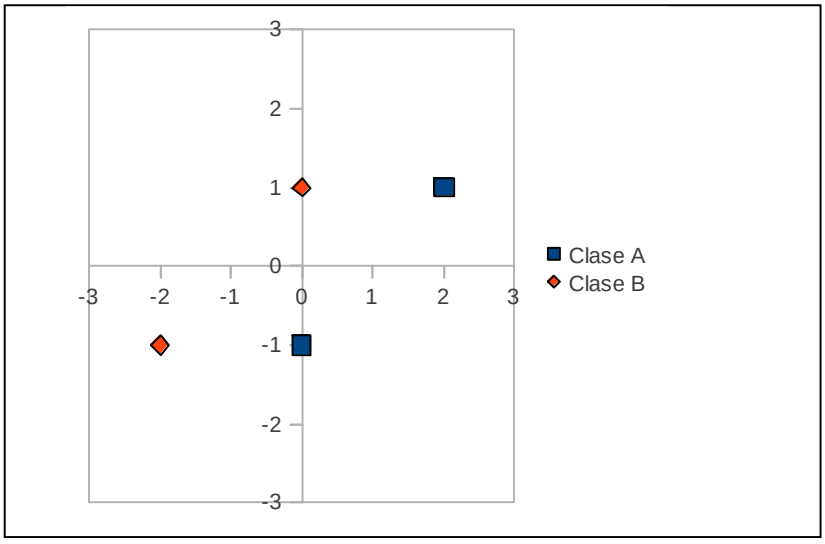


Figura 12. Eje de coordenadas con los puntos descritos en el ejemplo.

O lo que es lo mismo:

Puntos	X1	X2	Y
(0, 1)	0	1	Clase B
(2, 1)	2	1	Clase A
(0,-1)	0	-1	Clase A
(-2,-1)	-2	-1	Clase B

Tabla 2. Puntos a clasificar y clases a las que pertenecen.

Dado que la salida del perceptrón simple será -1 ó 1 (ó 0 y 1 dependiendo del caso) asignaremos un valor a cada una de las clases. Por lo tanto, la clase A será representada por el valor 1 y la clase B por el valor -1. Las salidas deseadas que queremos conseguir con el perceptrón son las siguientes:

Puntos	X1	X2	Y
(0, 1)	0	1	-1
(2, 1)	2	1	1
(0,-1)	0	-1	1
(-2,-1)	-2	-1	-1

Tabla 3. Salida que deseamos que tenga el perceptrón.

Necesitaremos por tanto, un perceptrón simple con dos entradas (X_1 y X_2) y una salida (Y) para poder clasificar los puntos descritos anteriormente.

Partiendo de la configuración inicial de pesos y umbral mostrada en la Figura 13 comenzamos la fase de entrenamiento introduciendo los patrones de entrenamiento anteriores hasta hallar el hiperplano discriminante.

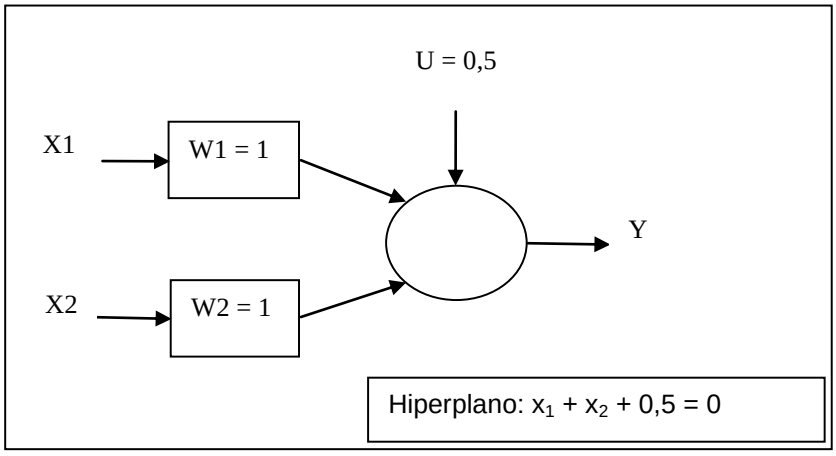


Figura 13. Perceptrón simple con los valores iniciales del ejemplo.

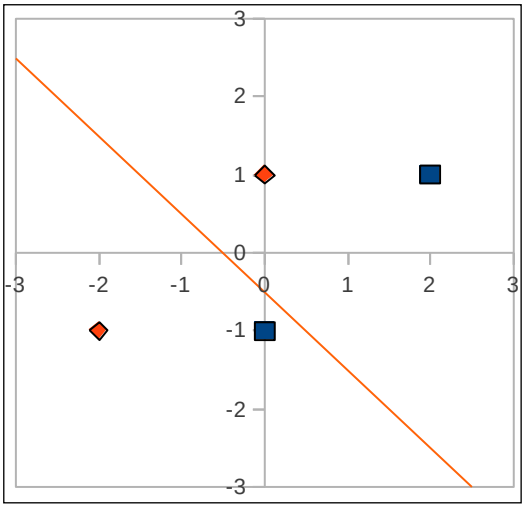


Figura 14. Muestra gráfica de los valores a clasificar y el hiperplano inicial.

- Para el primer patrón (0,1) el resultado de la función de entrada es:

$$x_1.w_1+x_2.w_2+u= 0.1+1.1+0,5= 1,5$$

Si aplicamos la función de activación explicada anteriormente la salida del perceptrón es la siguiente:

$$F(1,5)= 1.$$

La salida que debía haber dado era -1 por lo tanto, el perceptrón ha clasificado mal este patrón y se procede a aplicar el algoritmo de aprendizaje modificando los pesos y el umbral. Los nuevos pesos y umbral son los siguientes:

$$W_1= 1+ (-1).0= 1$$
$$W_2= 1+ (-1).1= 0$$
$$U= 0,5+ (-1)= -0,5$$

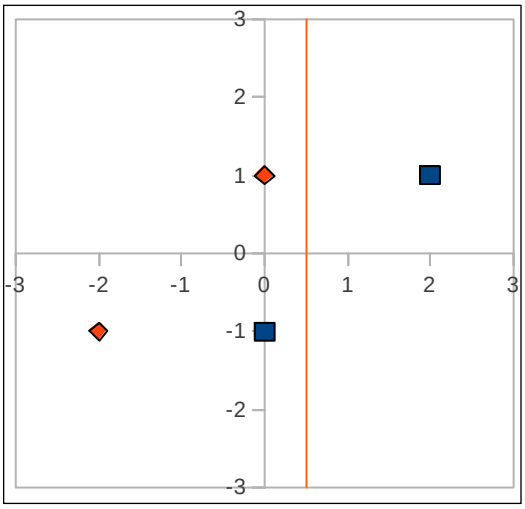


Figura 15. El hiperplano tras el primer aprendizaje.

Por lo tanto el nuevo hiperplano discriminante es:

$$X_1 - 0{,}5 = 0.$$ (Figura 15)

- Introducimos el segundo patrón (2,1):

$$x_1.w_1+x_2.w_2+u = 2.1+1.0+ (-0{,}5)= 1{,}5$$
$$F (1{,}5)= 1.$$

La salida del perceptrón coincide con la salida que se deseaba por lo tanto no se realiza ninguna operación y se pasa al siguiente patrón de entrenamiento.

- El tercer patrón es el siguiente (0,-1) y el resultado:

$$x_1.w_1+x_2.w_2+u = 0.1+ (-1).0+(-0{,}5)= -0{,}5$$
$$F (-0{,}5)= -1.$$

La clasificación es incorrecta. Se modifican los pesos y el umbral:

$$W_1= 1+1.0= 1$$
$$W_2= 0+1.(-1)=-1$$
$$U= (-0{,}5)+ (1)= 0{,}5$$

Por lo tanto el nuevo hiperplano discriminante es:

$$X_1-X_2+0{,}5=0.$$

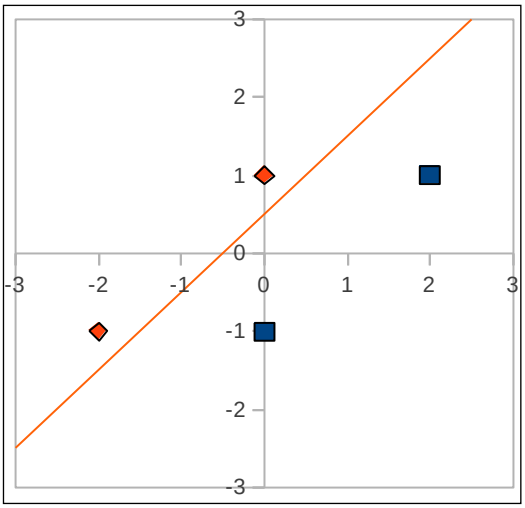


Figura 16. El hiperplano tras el segundo aprendizaje.

- Por último introducimos el patrón que nos falta (-2,-1):

$$x_1.w_1+x_2.w_2+u = (-2).1+(-1).(-1)+0,5= -0,5.$$

$$F(-0,5)= -1.\text{ Clasifica bien el patrón.}$$

Por lo tanto el hiperplano que da solución al problema y separa ambas clases es:

$$X_1 - X_2+ 0,5=0.$$

A continuación vemos las diferentes iteraciones realizadas por el hiperplano en el proceso de aprendizaje hasta conseguir clasificar bien los datos. En naranja la clase A, en blanco la clase B.

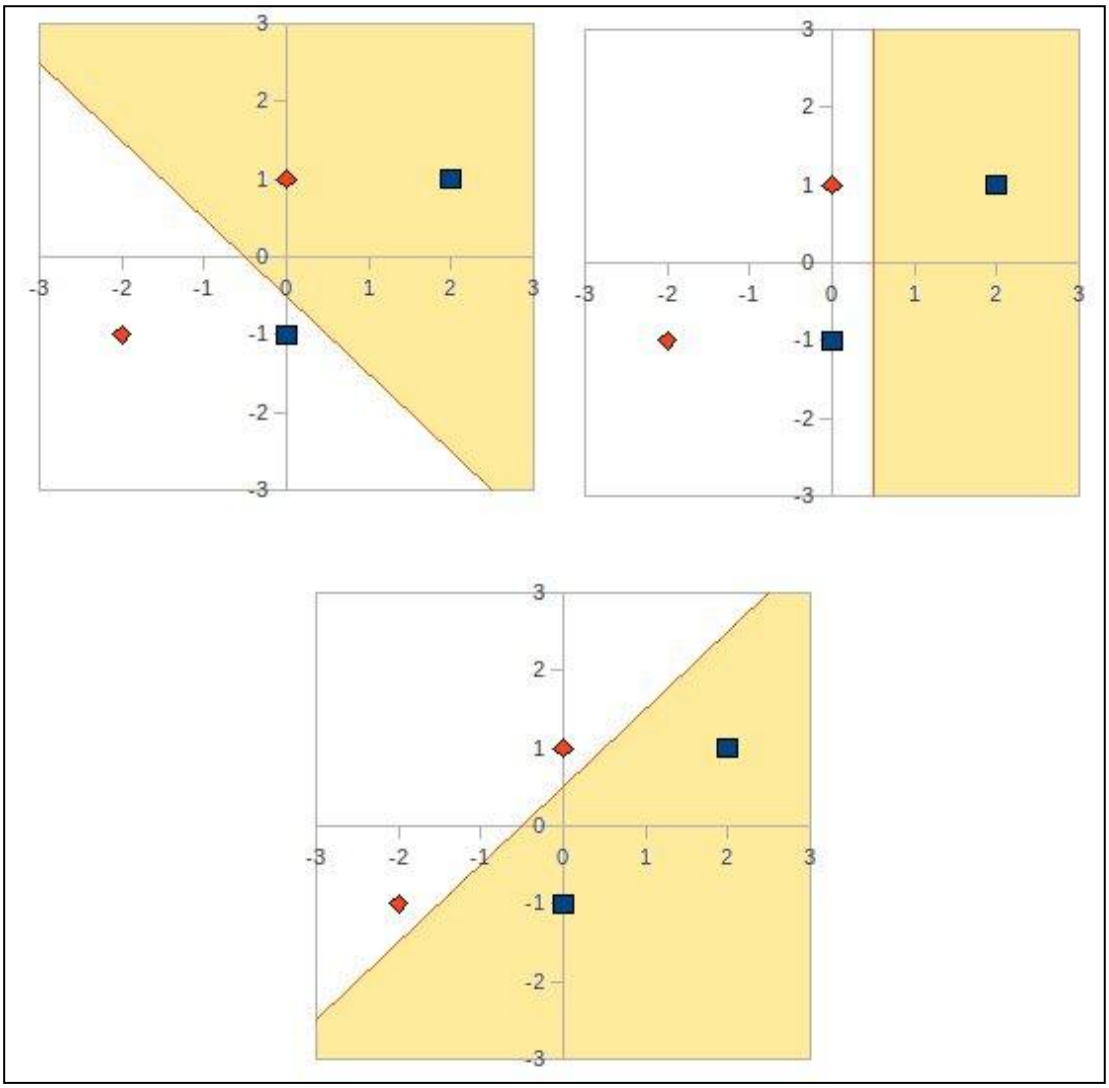


Figura 17. Iteraciones producidas durante el proceso de aprendizaje.

2.3.6.2 Razón de aprendizaje en el perceptrón simple

En algunos casos, puede ser útil añadir un nuevo parámetro (α) a la regla de aprendizaje. Esta nueva variable se la conoce como razón de aprendizaje o ritmo de aprendizaje. El objetivo principal de esta es conseguir que los cambios realizados en los pesos y del umbral sean menos bruscos y por lo tanto que el hiperplano discriminante no oscile demasiado durante el período de adaptación del perceptrón.

Las nuevas fórmulas tras insertar este parámetro quedarían así:

$$w_i\,(t+1) = w_i\,(t) + \alpha.d(x).\,x_i \qquad u(t+1) = u\,(t) + \alpha.d(x)$$

Ecuación 7. Aprendizaje del perceptrón con razón de aprendizaje.

Dónde:

w son los pesos.
x son las entradas.
d(x) es la salida deseada para esa entrada.
α es la razón de aprendizaje. Su valor estará contenido entre 0 y 1.

Por lo tanto, para valores altos (cercanos a 1), habrá variaciones en los pesos bastante amplias y se producirán oscilaciones en el entrenamiento. El resultado que se obtendrá será similar a un proceso de aprendizaje sin este factor. En cambio, a medida que disminuyamos el valor de esta nueva variable veremos que las oscilaciones producidas serán menores. Si el valor es bajo el proceso de aprendizaje será mucho más lento y se necesitará emplear más tiempo y más iteraciones con los patrones de entrenamiento para conseguir la adaptación del perceptrón ya que la corrección de los pesos y del umbral será mínima [9].

Esto se puede ver mejor en la siguiente imagen (Figura 18) compuesta de varias gráficas. En todas ellas vemos en naranja el hiperplano inicial antes de corregirse los parámetros y en azul claro el nuevo hiperplano que resultaría al realizar un aprendizaje normal (sin razón de aprendizaje). Los distintos hiperplanos pintados de azul oscuro y que varían dependiendo de la gráfica son los hiperplanos resultantes aplicando distintos valores de razón de aprendizaje.

Para una razón de aprendizaje α =1 el resultado sería el mismo que un entrenamiento sin razón de aprendizaje ya que si sustituimos en las fórmulas:

$$w_i\,(t+1) = w_i\,(t) + \alpha.d(x).\,x_i \Rightarrow\; w_i\,(t+1) = w_i\,(t) + 1.d(x).\,x_i \Rightarrow\; w_i\,(t+1) = w_i\,(t) + d(x).\,x_i$$

$$u(t+1) = u\,(t) + \alpha.d(x) \Rightarrow\; u(t+1) = u\,(t) + 1.d(x) \Rightarrow\; u(t+1) = u\,(t) + d(x)$$

Ecuación 8. Comprobación con razón de aprendizaje igual a 1.

Las fórmulas resultantes serían iguales a las del aprendizaje normal.

El caso de α = 0 sería inviable ya que no se produciría ninguna variación en los parámetros y por lo tanto el hiperplano seguiría igual en todas las iteraciones no produciéndose ningún tipo de aprendizaje.

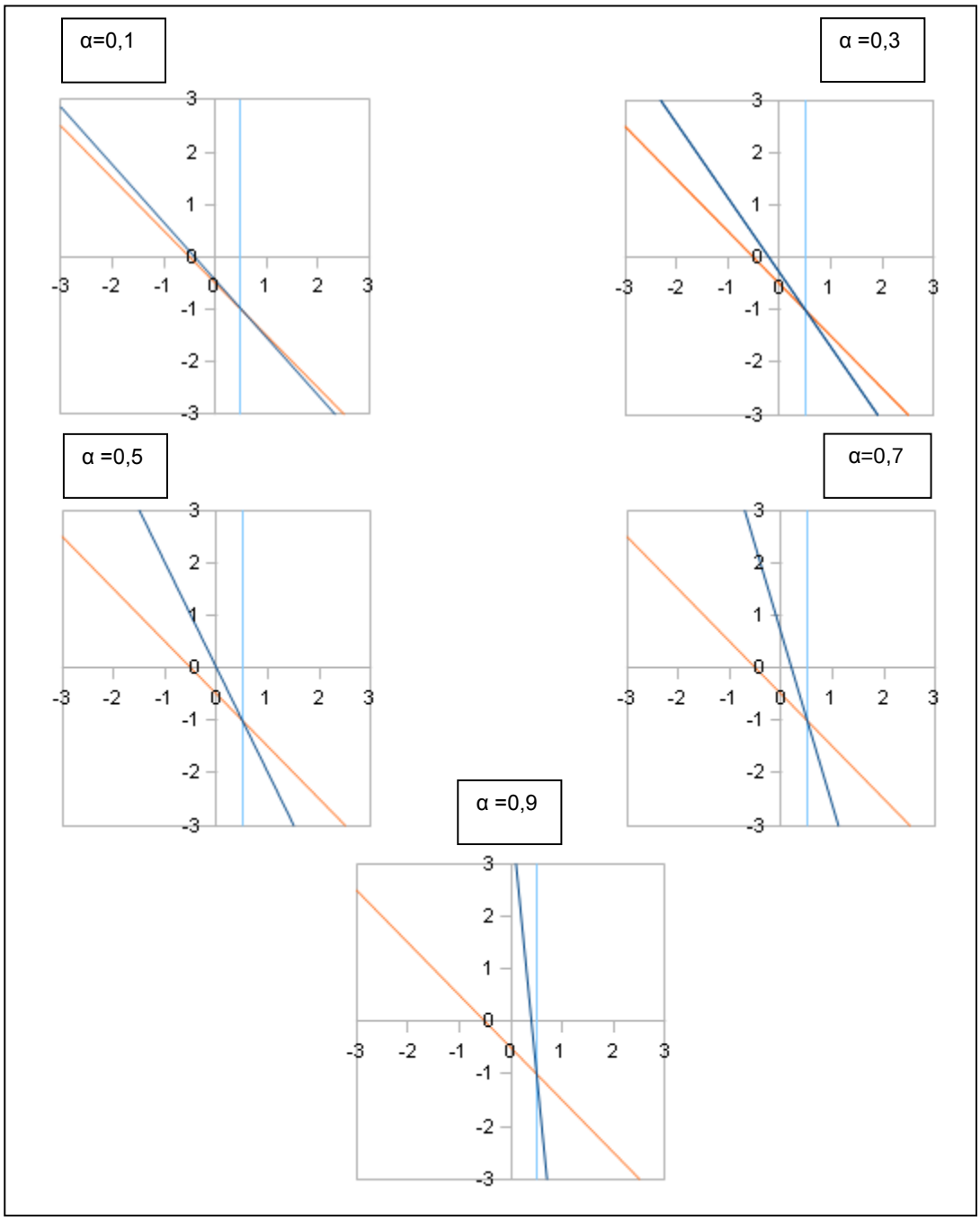


Figura 18. Oscilaciones del hiperplano provocadas por α.

2.3.6.3 Limitaciones del perceptrón simple

Como hemos explicado anteriormente, este perceptrón es muy útil para casos en los que las clases a separar dentro del conjunto de datos sean separables linealmente [9]. En el caso de que esto no sea así, el proceso de entrenamiento irá oscilando intentando buscar una solución al problema sin llegar nunca a una clasificación correcta y las clases no podrán ser representadas por un perceptrón simple. Para estos casos será necesario utilizar una combinación de varios perceptrones simples u otro tipo de red neuronal.

2.4 Algoritmos genéticos

Los algoritmos genéticos fueron desarrollados en los años 70 por John Henry Holland. Estos algoritmos están englobados dentro de lo que se llaman algoritmos evolutivos siendo los algoritmos genéticos los más populares de este campo [7].

En general, estos algoritmos son métodos de búsqueda inspirados en los procesos evolutivos biológicos y en los procesos de selección natural que se dan en el mundo real. Estos algoritmos genéticos se inspiran por tanto en teorías de evolución como la Teoría de la Evolución de Darwin y estudios de Lamarck, Mendel y Wallace sobre procesos naturales de la evolución biológica y su base genético-molecular.

Los algoritmos genéticos parten de una población de individuos inicial y la hacen evolucionar por medios semejantes a los de la evolución biológica (como las mutaciones o combinación de individuos) y por medio de estos procesos y otros procesos de selección semejantes a los naturales consiguen ir seleccionando a los individuos más aptos que van sobreviviendo mientras los menos aptos van desapareciendo, con lo que se consigue que la población vaya evolucionando hasta encontrar la mejor solución al problema.

Independientemente de cuál sea el problema particular que traten, todos los algoritmos genéticos deben tener los siguientes componentes [11]:

- Una representación genética del conjunto de soluciones válidas del problema.

- Un método capaz de crear el conjunto de soluciones que formarán la población inicial.

- Una función de aptitud (o de *fitness*) que evalúe las diferentes soluciones existentes dándole más o menos opciones de subsistir en función de su adaptación. Equivaldría a la selección natural.

- Operadores genéticos que generan nuevas soluciones a partir de los individuos anteriores existentes.

- Otros parámetros necesarios para el funcionamiento del algoritmo genético como puede ser el tamaño de la población, probabilidades de uso de los operadores genéticos, etc.

El método de representar los individuos o soluciones de nuestro problema es por medio de cadenas de dígitos binarias. Estos individuos de la población también son llamados cromosoma. Es muy importante decidir cuál será la mejor representación de la soluciones para el problema ya que, una buena representación nos puede ayudar a aplicar unos operadores genéticos más sencillos, o nos puede ayudar a simplificar y comprender el objetivo del algoritmo. También esta codificación de las soluciones nos afectará a la hora de diseñar la función de aptitud.

En lo que respecta a su funcionamiento, este es sencillo como se puede apreciar en la Figura 19 y se compone básicamente de las siguientes etapas o fases iterativas explicadas brevemente a continuación [7]:

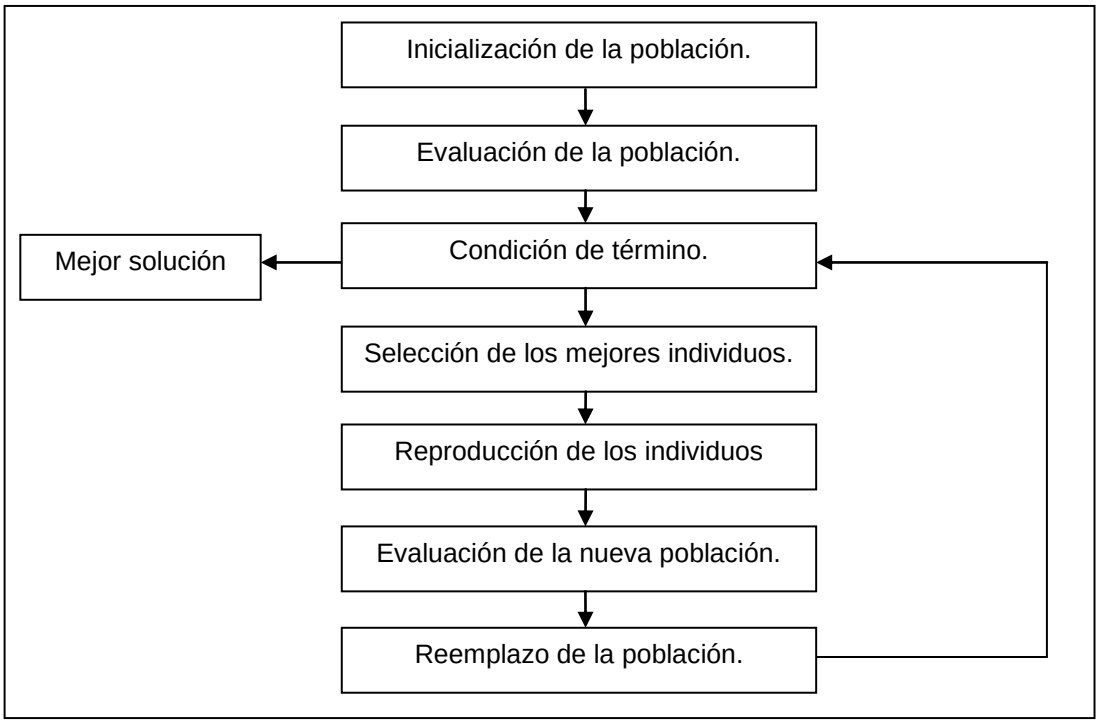


Figura 19. Fases de un algoritmo genético básico.

Inicialización de la población: Como ya hemos explicado, estos algoritmos parten de una población de individuos creada inicialmente de forma aleatoria. Esta población es, en realidad un conjunto de soluciones distintas al problema, con lo que es conveniente que esta población inicial sea lo más diversa posible para así tener representadas el mayor número de soluciones posibles y evitar una convergencia prematura.

Evaluación de la población: Una vez que tenemos la población inicial, el siguiente paso es evaluar esta población según una función de aptitud (en inglés "fitness") o función objetivo para poder determinar como de buena es cada una de las soluciones del problema y asignar a cada individuo de la población un valor de aptitud. Esta función objetivo representa la función que se desea optimizar y dependerá de cada problema concreto. Gran parte del éxito del algoritmo dependerá de la elección de una buena función de aptitud que sea capaz de evaluar a los individuos de una forma correcta. Esta función de fitness puede maximizarse o minimizarse dependiendo del objetivo del problema.

Selección de los mejores individuos: El siguiente paso es seleccionar los mejores individuos de la población para que posteriormente se reproduzcan y creen individuos que contengan las cualidades de los mejores individuos existentes. Hay numerosos algoritmos de selección, como por ejemplo la selección jerárquica, la selección por torneos, selección por ruleta, etc. Todos tienden a elegir los mejores individuos descartando a los peores con pequeños matices. Este proceso es importante ya que se encarga de dirigir al algoritmo hacia regiones del espacio de búsqueda que parecen mejores.

Reproducción de los individuos: Aquí es donde el algoritmo consigue generar nuevas soluciones a partir de las seleccionadas anteriormente. De este modo se conseguirá nuevos individuos mejor adaptados a la función de aptitud. Todo esto se realiza con los siguientes operadores genéticos:

- Cruzamiento: Partiendo de dos individuos padres, se combinan sus cromosomas para crear dos nuevos individuos. Es análogo a la reproducción dada en la naturaleza. Existen diversas técnicas de recombinación o cruzamiento.

- Mutación: Se modifican al azar algunos genes o bits del individuo con el fin de alcanzar soluciones no dadas en la población anterior.

- Inversión: Se da la vuelta a varios genes del cromosoma que se encuentran consecutivos creando un individuo diferente al anterior.

Reemplazo de la población: Tras haber aplicado los operadores genéticos en la fase anterior, se realiza una selección de los mejores individuos creados para que formen parte de la nueva población reemplazando a sus padres peor adaptados. Existen varias estrategias de reemplazo, pero en general se dividen en: Reemplazo generacional (todos los individuos creados sustituyen a sus padres), y reemplazo estacionario (el reemplazo es parcial, ya que solo se sustituyen n padres por n hijos).

Condición de término: En general el algoritmo genético puede finalizar su ejecución por varias causas. La ideal sería que parase cuando se haya alcanzado la solución óptima pero dado que ésta solución óptima no siempre se conoce, se utilizan otro tipo de condiciones de parada como permitir un número determinado de iteraciones (generaciones), finalizar cuando se hayan producido cambios en la población, etc.

Los algoritmos genéticos se suelen utilizar mayormente para solucionar problemas de optimización como por ejemplo:

- Realizar una búsqueda en el espacio de datos para obtener el mejor elemento que solucione el problema.
- Obtener el procedimiento que maximice una tarea de control.
- Solucionar problemas de clasificación encontrando el procedimiento que mejor clasifique los datos.
- Encontrar el procedimiento que tenga menos fallos de predicción.

Estos algoritmos han sido usados durante años para resolver problemas complejos en multitud de áreas [7].

2.5 Concepto de optimización

Uno de los problemas más comunes a los que nos enfrentamos son los problemas de optimización. Su propósito es elegir entre un conjunto de elementos dados el mejor dependiendo de la función objetivo. Un ejemplo sencillo puede ser la búsqueda del mínimo / máximo de una función dada, la maximización de los beneficios, la minimización de los costos, minimizar distancias, etc. Estos ejemplos son de los más sencillos ya que solo había un objetivo que optimizar pero no siempre será así.

En muchos otros casos los problemas de optimización no son mono-objetivo, sino multi-objetivo. Esto significa que tienen dos o más objetivos o funciones objetivo que satisfacer. La principal dificultad de este tipo de problemas es que en la mayoría de los casos estos objetivos entran en conflicto provocando que al intentar optimizar uno de ellos otro salga perjudicado o puede darse el caso que una solución que es óptima para uno de ellos no lo sea para otro objetivo. Por lo tanto, será necesario llegar a un punto intermedio o situación de compromiso en la que se consiga que todos los objetivos sean satisfechos de forma aceptable.

Un ejemplo trivial de este tipo de problemas de optimización puede ser la realización de una obra. En esta obra tendremos dos objetivos, la disminución de los costes de personal y la disminución del tiempo de realización. Si queremos disminuir el tiempo de realización inevitablemente los costes aumentarán al ser necesaria la contratación de más personal. Al contrario, si disminuimos los costes de personal el tiempo de ejecución de la obra aumentará ya que habrá menos gente dedicada a la obra. Por lo tanto, es necesario llegar a una situación intermedia en la que ambos objetivos se cumplan de forma aceptable y ninguno de ellos salga muy perjudicado.

2.5.1 Formas de solucionar el problema

Los diferentes métodos de solucionar este tipo de problemas en los que existe algún tipo de conflicto entre los diferentes objetivos a optimizar son los siguientes:

- **Métodos basados en la eficiencia de Pareto:** El concepto de eficiencia descrito por Wilfredo Pareto determina la situación en la que conseguimos que se cumpla que no es posible mejorar un objetivo sin perjudicar a ningún otro. Por lo tanto, todas aquellas soluciones encontradas no sólo serán óptimas sino que no existirá ninguna otra solución que mejore a estas en varios objetivos a la vez sin perjudicar a ningún otro. Al conjunto de estas soluciones se las denominará frente de Pareto. Esta es la noción más aceptada sobre el concepto de “óptimo” en problemas multi-objetivo.

  Como vemos en la imagen siguiente, (cuyo objetivo es minimizar tanto f1 como f2), el punto p3 es una solución óptima para nuestro problema pero no es una solución pareto-optima ya que la solución p1 tiene el mismo valor de f1 y además mejora el valor de f2. Las soluciones pareto-optimas serán aquellas que estén en el frente de Pareto (línea oscura).

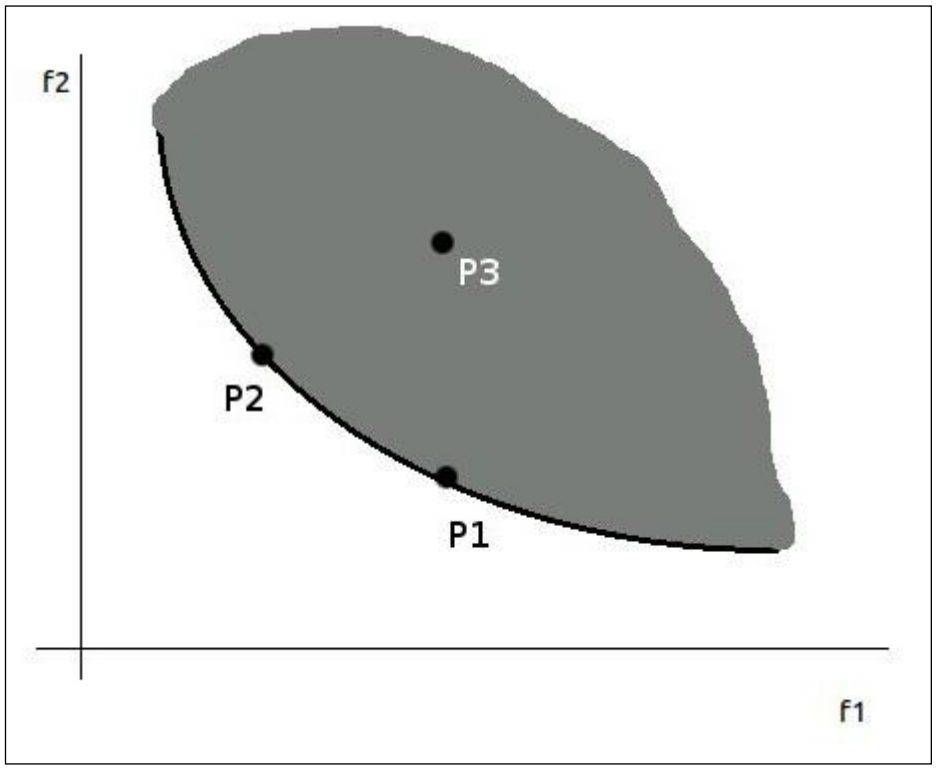


Figura 20. Óptimo de Pareto.

No entraremos más en profundidad en este tema, ya que estos métodos derivan en los algoritmos genéticos multi-objetivo que nosotros no usamos para la resolución de nuestro problema. Estos algoritmos genéticos multi-objetivo usan este concepto de frente de Pareto para solucionar los problemas multi-objetivo, de este modo hallan un frente de soluciones pareto-óptimas para el problema. En nuestro caso, no trataremos de hallar este frente sino que buscaremos una zona concreta del espacio de búsqueda que cumpla nuestro objetivo. Si se quiere conocer más información sobre este tipo de algoritmos genéticos multi-objetivo, se han realizado varios trabajos y estudios para casos de clasificación de datos similares al que nosotros nos enfrentamos (Ver [12],[13] y[14]).

- **Métodos de asignación de prioridades:** Durante el proceso de optimización se asignan un orden de prioridades entre los distintos objetivos. De este modo optimizaremos siguiendo este orden de importancia consiguiendo así que los objetivos más importantes se cumplan en mayor medida.

- **Métodos de combinación de objetivos:** Con este tipo de métodos conseguimos combinar varios de nuestros objetivos disminuyendo el número de ellos o consiguiendo que nuestro problema se resuma en un problema de un solo objetivo. El método de la **suma ponderada (o**

técnica de escalarización) [15] y [16] que consiste en asignar valores a cada objetivo dependiendo de su importancia estaría dentro de este grupo de métodos. Con él resumiríamos el problema multi-objetivo en un problema de optimización simple en el que habría que optimizar una función, en el caso de maximizar los objetivos sería, como la siguiente:

$$\max \sum_{i=1}^{n} w_i \cdot f_i$$

Ecuación 9. Maximización usando el método de suma ponderada.

Dónde:

W_i es el peso asignado al objetivo i.
F_i es la función objetivo i.

Con este método de escalarización, en muchos casos no obtendremos todo el conjunto de soluciones optimales para nuestro problema, es decir que es probable que no quede determinada la línea de eficiencia o frente de Pareto [15] y [16] como si pasaba en los métodos basados en la eficiencia de Pareto.

Utilizaremos este método de suma ponderada para evaluar la población del algoritmo genético como veremos más adelante, ya que creemos que esta puede ser una de las mejores formas de realizar esta evaluación (siempre sin entrar en otros métodos basados en Pareto).

Capítulo 3

Diseño e implementación de la solución

3.1 Introducción

En este capítulo ahondaremos más en los diferentes conocimientos explicados anteriormente con el fin de explicar aspectos más técnicos de nuestra solución para este caso concreto y mostrar las diferentes formas de afrontar el problema. En él veremos varias redes neuronales artificiales usadas como clasificadores válidos para nuestra solución con el fin de intentar ver cuál es la que mejor se adapta a nuestro problema, además también mostraremos todos los detalles de nuestro algoritmo genético como pueden ser los distintos operadores que vamos a usar en él, las diferentes formas de representar las soluciones posibles, etc.

En definitiva, mostraremos y explicaremos las herramientas que hemos usado para tratar de dar solución a nuestro problema de clasificación de los datos obtenidos con el BCI y todas las diferentes características de estas con el fin de comprender aún más el problema a resolver y la forma de abordarlo.

El orden interno de este capítulo será:

- Esquema y explicación general de nuestra solución.
- Diseño y explicación de las características de nuestro algoritmo genético.
- Explicación de los clasificadores diseñados.

3.2 Esquema general de nuestra solución

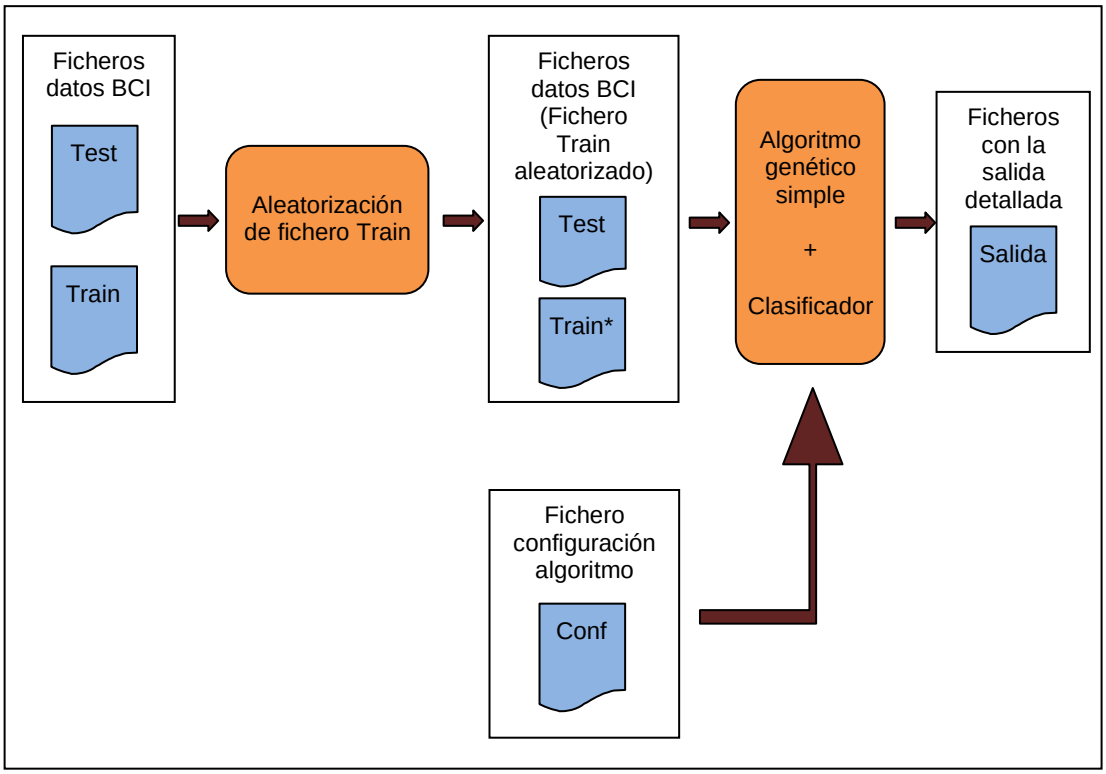


Figura 21. Esquema general de la solución.

En el esquema anterior podemos ver el esquema general de nuestra solución. Como se puede observar, nuestro sistema contará con tres entradas (dos ficheros con los datos pre-procesados obtenidos por el BCI y otro más con la configuración del algoritmo genético). El fichero de Train con los datos obtenidos del BCI necesitará un tratamiento previo antes de ser utilizado por el algoritmo genético, este tratamiento se trata de “aleatorizar” o descolocar sus registros con el fin de evitarnos problemas durante el entrenamiento de la red. Tras realizar esta operación ya tendremos los dos ficheros con los registros que serán la entrada de nuestro algoritmo genético. Una vez conseguido esto, el algoritmo genético leerá los parámetros de configuración que se encuentran en el tercer fichero de entrada y se ejecutará siguiendo todos esos parámetros que explicaremos posteriormente. Tras la finalización de su ejecución obtendremos varios ficheros con toda la salida del algoritmo detallada. Además de estos ficheros el algoritmo mostrará también los resultados por pantalla durante toda su ejecución.

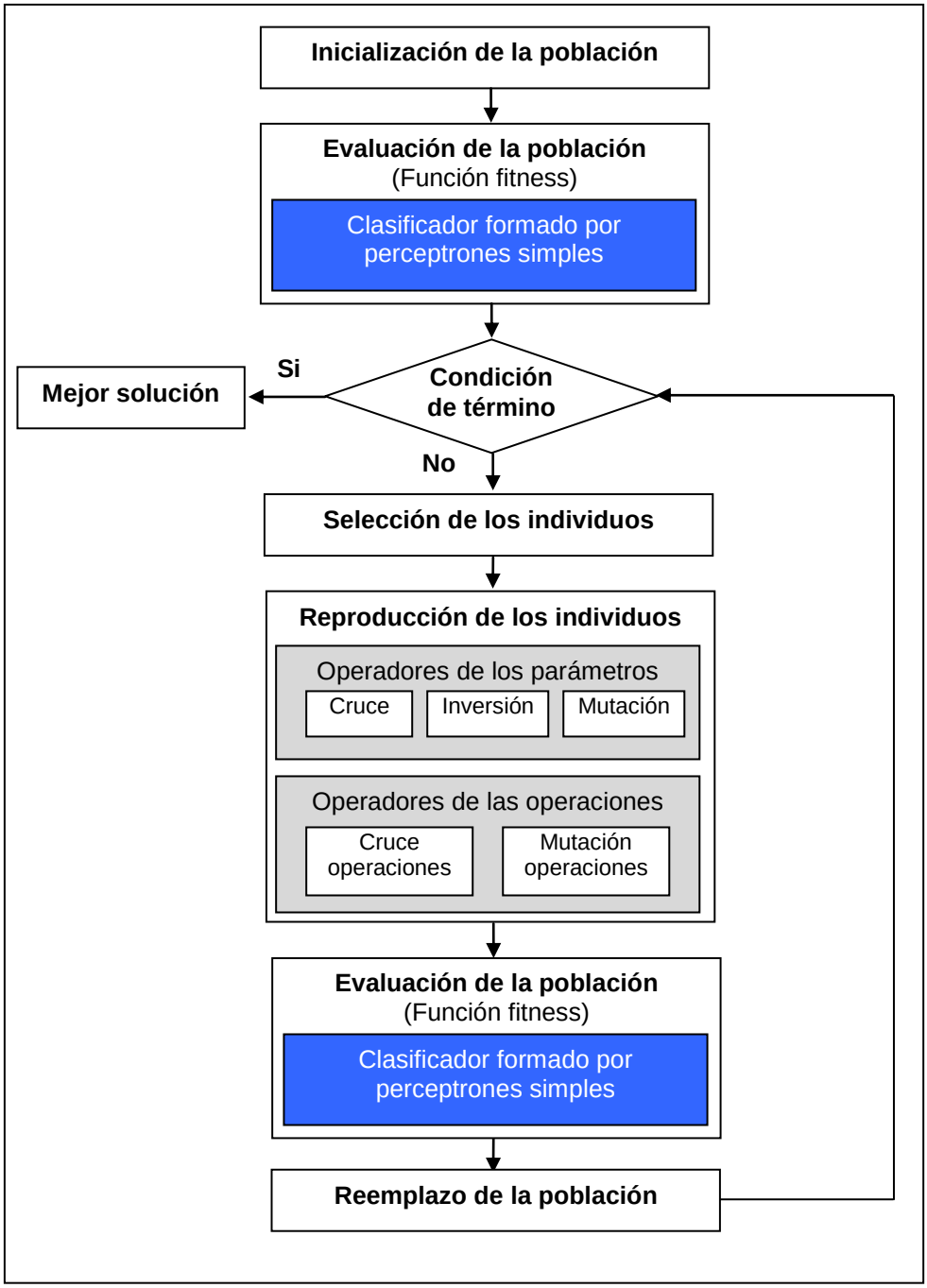


Figura 22. Esquema detallado del algoritmo genético + clasificadores.

Por lo tanto, en resumen, nuestra solución para el problema es un algoritmo genético (ver la sección de algoritmos genéticos del capítulo anterior). Además, para la correcta evaluación de los individuos, hemos implementado un clasificador formado por perceptrones simples (partes azul oscuro del esquema detallado Figura 22). Este clasificador formará parte de la función de fitness del algoritmo genético.

Con ello, crearemos una población inicial al azar y la haremos evolucionar generación tras generación a través de los distintos operadores confiando que ello nos lleve a soluciones mejores que nos hagan reducir el número de entradas a la red y a su vez obtengamos errores de esta lo más mínimos posibles.

En las siguientes secciones de este capítulo trataremos de describir de forma detallada cada una de las partes que la componen. Comenzaremos hablando del algoritmo genético y de las características que nos llevaron a escoger este tipo de algoritmo y a continuación describiremos los clasificadores realizados con perceptrones simples y el porqué de la elección de este tipo de red neuronal.

3.3 ¿Por qué nos un algoritmo genético?

Como hemos explicado ya antes, este tipo de algoritmos parten de un conjunto de soluciones que hacen evolucionar generación tras generación con el fin de conseguir las que mejor se adaptan. Si volvemos a nuestro problema, teníamos como uno de nuestros objetivos el tratar de hallar la forma de poder seleccionar las entradas a la red para así quedarnos sólo con las que nos sean útiles desechando las que perjudican a la clasificación. La dificultad es que a simple vista no sabemos cuáles de estas entradas son las que podemos desechar sin perjudicar al resultado.

La idea por lo tanto será usar un algoritmo genético para seleccionar los atributos que sean útiles para la clasificación del patrón. Como hemos comentado antes, sabemos que hay varios atributos que solo introducen ruido en la clasificación, son datos que se han tomado pero a lo mejor esos impulsos eléctricos del cerebro no afectan a ese tipo de pensamiento y por lo tanto solo perjudican al clasificador en lugar de ayudarlo. Es por eso por lo que usamos el algoritmo genético, para intentar localizar y desechar esos atributos que sólo perjudican a la clasificación del patrón. De este modo será el propio algoritmo quién vaya seleccionando las entradas de las red y a la vez comprobando los resultados producidos por ésta, así se conseguirá una reducción de entradas sin aumentar el error producido por la red ya que, en caso de que este error aumente, esa solución es probable que no prospere y no consiga perdurar sin sufrir alguna modificación que haga que mejore su resultado.

3.4 Descripción del algoritmo genético utilizado

A continuación pasamos a describir de forma más concreta los métodos y procesos utilizados en el algoritmo genético implementado, así como aspectos más técnicos que han sido necesarios para adaptar este algoritmo a la resolución de nuestro caso concreto. Por ello, en este apartado explicaremos las distintas formas de representar los individuos o soluciones de nuestro problema, su inicialización, los aspectos relativos a la función de evaluación de los individuos, los métodos de selección disponibles, los distintos métodos implementados para la creación de nuevos individuos, los métodos de reemplazo, la condición de término del algoritmo genético así como otros aspectos o características a tener en cuenta.

3.4.1 Representación de los individuos

Como ya se comentó anteriormente, la población de un algoritmo genético está compuesta por las diferentes soluciones al problema dado. Por lo tanto, cada individuo de la población representará una solución concreta al problema. La forma de codificar estas diferentes soluciones o individuos de la población para nuestro caso es sencilla. En nuestro problema cada registro está formado por 96 datos o atributos recogidos por el casco BCI (Ver 2.2 BCI (Brain-Computer Interface)) que a su vez se agrupan en 8 canales distintos dependiendo de la zona donde se haya recogido ese dato como ya vimos anteriormente.

Además de estos datos que recibimos del fichero, también introduciremos otros que serán combinaciones de datos originales (suma, resta, multiplicación, división) con el objetivo de mejorar el resultado obtenido. Nuestro objetivo es conseguir es que nuestro algoritmo sea capaz de separar las distintas clases de datos sin introducir todas las señales, ya que como explicamos en secciones anteriores (1.1 Introducción, 1.2 Objetivos) debemos descartar los atributos (señales) de entrada que no sean representativos para la clasificación, ya que de esta manera se disminuyen el número de atributos de entrada de la red y, como consecuencia, el tiempo computacional, además de mejorar el resultado. Por lo tanto, para poder representar las posibles soluciones a nuestro problema, necesitaremos genes que serán codificados como bits (valores: 0,1) los cuales se dividirán, principalmente, en dos grupos. Un grupo de bits será utilizado para seleccionar los datos con los que evaluaremos al individuo y otro grupo de bits que serán los encargados de codificar las diferentes operaciones entre ellos.

3.4.1.1 Codificación de operaciones

Para realizar una operación es necesario tener dos operandos y un operador (Operando1 (operador) Operando2). Por ejemplo 3+5, 15/3, etc. Con lo cual, para codificar cada uno de los operandos es necesario 7 bits ya que al tener 96 atributos necesitamos codificar 96 posiciones (0… 95), como vemos a continuación.

6 bits= 2^6 valores= codifica 64 posiciones (0…63). No son suficientes.
7 bits= 2^7 valores= codifica 128 posiciones (0…127).

De este modo, si tenemos la siguiente cadena 0000000 significará que el operando de la operación será el atributo 0 de la trama de datos recogidos por el BCI, si tenemos 1000000 será el atributo 64, 1011111 será el atributo 95, nuestro último valor, etc. (Siempre comenzando a contar por el atributo 0 en adelante).

Para la codificación de los operadores, dado que estos son cuatro (suma, resta, multiplicación y división) necesitaremos 2 bits de la siguiente forma:

Suma= 00
Resta= 01
Multiplicación= 10
División= 11

Por lo tanto, necesitaremos 16 bits para cada una de las operaciones que queramos incluir. (7bits operando1 + 2 bits operador + 7 bits operando2) como podemos ver en la siguiente imagen:

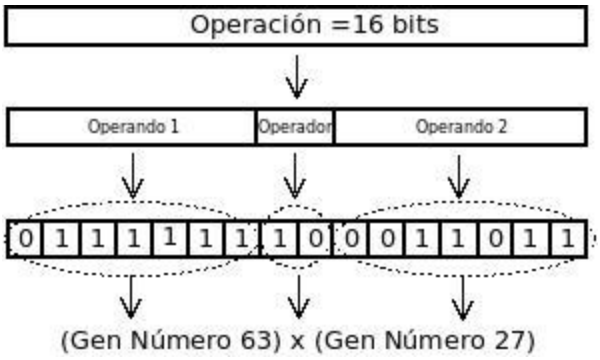


Figura 23. Muestra una operación codificada.
(Atributo nº 63 multiplicado por el atributo nº 27)

En el caso del grupo de bits encargado de indicar cuáles son los datos con los que evaluaremos al individuo, dada la procedencia de nuestros datos, podemos representar los individuos de dos posibles formas:

3.4.1.2 Representación simple por atributos

Esta es la forma más simple de representación de los individuos. Simplemente utilizaremos un bit o gen para codificar cada uno de los 96 atributos del fichero. En el caso de que el valor del bit sea 0, esto indicará que ese atributo no será utilizado para evaluar al individuo, o lo que es lo mismo, ese individuo no contendrá ese atributo. Por lo tanto, el individuo solo estará compuesto por los atributos que tengan valor 1 y serán estos junto a las operaciones la entrada de nuestra función fitness.

Es decir, si el bit número 5 es un 1 significará que se realizará la evaluación de ese individuo usando el atributo número 5 recogido del casco BCI. De este modo, el individuo estará formado por 96 bits que representan los diferentes atributos que forman parte de él y 16 bits más por cada una de las operaciones que use. La trama tendrá la siguiente forma:

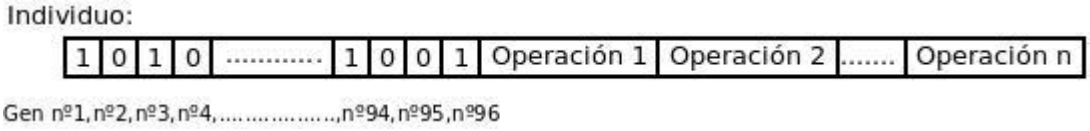


Figura 24. Representación de un individuo por genes.

3.4.1.3 Representación de los individuos utilizando los canales

Otra forma de representar los individuos es utilizando la división de los atributos en canales. Como ya hemos explicado antes, los 96 datos recogidos por el casco por instancia, se dividen en 8 canales distintos dependiendo de la zona en la que se hayan tomado. Se ha escogido esta representación pues se quiere comprobar si, escogiendo los datos por canales y dado que estos pertenecen a una determinada zona del cerebro, el resultado podría ser mejor que escoger varios datos de diferentes zonas.

Si seguimos este criterio entonces podríamos representar los individuos con 8 bits o genes que indiquen si un canal determinado forma parte de un individuo o no y 16 bits más por cada una de las operaciones como explicamos antes. En el caso de que un canal sea seleccionado, la totalidad de los atributos de ese canal formarán parte del individuo. La trama quedará de la siguiente forma:

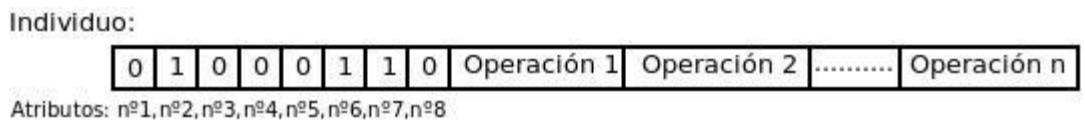


Figura 25. Representación de un individuo por canales.

Como vemos en la imagen, el gen número 2 es un 1 por lo tanto los 12 atributos que forman parte de este canal número 2 (atributos del 12 al 23 si empezamos a contar desde 0) formará parte del individuo y serán introducidos en la función fitness junto a las operaciones y el resto de los canales activados. Si este gen fuera 0, no entraría ningún atributo perteneciente a ese canal.

De este modo, el individuo estará formado siempre por 8 genes que representan los diferentes canales que forman parte de él y 16 bits más por cada una de las operaciones que use.

Es necesario comentar que estas dos formas descritas anteriormente de representar los individuos no se pueden utilizar a la vez sino que deberán usarse por separado ya que son incompatibles. En apartados posteriores, se realizarán pruebas con el fin de comprobar los diferentes resultados obtenidos y ver cuál de ellas es mejor para nuestro problema o si por el contrario la forma de representación elegida no hace variar el resultado obtenido.

3.4.2 Inicialización de los individuos

Independientemente del tipo de representación utilizada (representación simple o representación por canales) a la hora de inicializar los individuos de la población está se hará de forma aleatoria siguiendo las probabilidades que le hallamos indicado. En el caso de las operaciones (operandos y el operador) podremos elegir la probabilidad que le damos a cada uno de los cuatro operadores de ser elegido, incluso llegando a desechar alguno de ellos o haciendo que todas las operaciones sean de un mismo tipo, (todas sumas, todas restas…, etc.). También puede dejarse que el algoritmo seleccione las operaciones completamente al azar seleccionado la misma probabilidad a todos los tipos.

En el caso de los atributos que formarán parte del individuo (existirá una probabilidad de formar parte de este). Esta probabilidad de formar parte del individuo o tanto por ciento de atributos podremos variarlo a la hora de ejecutar el algoritmo pero nosotros utilizaremos en la mayoría de los casos un 50%. Comentar además que con esta inicialización aleatoria en la que los atributos tienen un 50% de pertenecer al individuo, todos los individuos de la población partirán con un porcentaje de atributos sobre el 50% más las operaciones que se halla determinado, con lo que conseguimos reducir el espacio de búsqueda eliminando a individuos con un gran porcentaje de atributos usados y también conseguimos reducir de manera importante el número de entradas de la red.

Con todo esto, conseguimos que en la población inicial con la que partimos en nuestro problema exista diversidad entre las diferentes soluciones y nos evitamos problemas como el de la convergencia prematura que podría surgir en casos en los que la población fuera muy similar.

3.4.3 Función de aptitud o función de fitness

Esta función de aptitud o función de fitness de lo que se ocupa es de valorar los distintos individuos de la población con el objetivo de determinar como de buena es la solución que representa cada uno de los individuos para la resolución del problema. Como ya hemos comentado con anterioridad, el objetivo de nuestro problema consiste en minimizar el número de errores que se producen al intentar clasificar los datos obtenidos por el BCI por el clasificador, a su vez, también en conseguir eliminar las entradas a la red que perjudiquen esta clasificación, (los aspectos relativos al clasificador los explicaremos la sección 3.5). Por lo tanto nuestra función de aptitud deberá valorar correctamente a los individuos de la población según cómo hayan conseguido cumplir los objetivos.

Si no se consigue una buena valoración de los individuos el algoritmo puede fallar en la búsqueda de la mejor solución y perderse por las regiones del espacio de búsqueda. Pero, ¿cómo valorar correctamente a cada uno de los individuos? Tenemos claro que el porcentaje de error de la red será un valor predominante en esta valoración pero ¿será el único? Aquí nos surgieron dudas sobre la mejor forma de valorar a los individuos. Planteamos una función de aptitud para cada una de las diferentes formas que se nos ocurrieron y realizamos sencillas pruebas con cada una de ellas con el fin de ver con los resultados cuál era la que mejor valoraba a los individuos para nuestro problema determinado.

A continuación explicaremos cada una de ellas y comentaré los pros y los contras de cada una. Ni que decir tiene que todas las funciones explicadas son totalmente válidas para nuestro problema, lo que les diferencia es la forma de valorar los diferentes objetivos y la forma de primar unos objetivos sobre otros. Algunas funciones explicadas aquí tendrán como principal objetivo la disminución del error producido por la red y dejarán en un segundo plano la reducción de los parámetros introducidos a ésta ya que estos son reducidos ya considerablemente al crear la población, otras intentarán buscar un equilibrio e ir reduciendo el número de parámetros a la vez que conseguimos disminuir el error, etc.

El principal problema al elegir la función de aptitud es por lo tanto, ver el funcionamiento de estas y ver cuál se adapta más a nuestras necesidades.

3.4.3.1 Función de aptitud básica.

$$Función\ de\ Fitness\ individuo_i = Porcentaje\ de\ error\ del\ individuo_i$$

Ecuación 10. Fórmula de la función de fitness básica.

Esta es la función más sencilla de todas. Simplemente la función de fitness es el porcentaje de error producido al ser evaluado el individuo por la red neuronal. Este será el valor a intentar reducir. A menor porcentaje de error, mejor adaptado estará el individuo a nuestro problema.

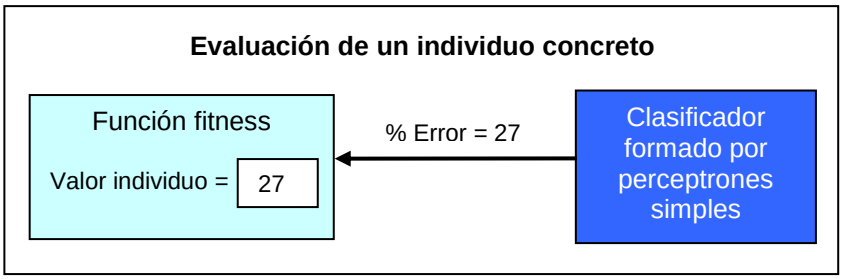


Figura 26. Esquema evaluación de un individuo con función básica.

Como vemos aquí, esta función no tiene en cuenta el número de parámetros introducidos a la red a la hora de valorar los individuos por lo tanto no habrá forma de primar a los individuos con menor número de parámetros para favorecer su reproducción o su permanencia en la población. Esto se hace así porque si recordamos la forma que teníamos de crear los individuos (3.4.2 Inicialización de los individuos) nosotros creábamos a éstos con sólo un 50% de parámetros, lo que hace que lleguemos a la conclusión que con esta reducción inicial el objetivo de minimizar los atributos para quitar las entradas no válidas ha sido ya cumplido y no es necesario controlarlo o ejercer más presión sobre él. Lo único necesario por lo tanto será que el algoritmo busque por si sólo las soluciones más adaptadas y que por tanto, no contengan entradas que no nos sirvan.

Cuando realicemos algunas pruebas veremos si esta función nos es útil para nuestro fin. En el siguiente ejemplo vemos gráficamente el cálculo del fitness de algunos individuos:

	% Error red	% Parámetros	*Fitness = %Error red
Individuo 1	30	53	30
Individuo 2	30	46	30
Individuo 3	28	48	28
Individuo 4	25	65	25

Tabla 4. Cálculo del fitness con la función básica.

Por lo tanto con esta función fitness nos centraremos en el objetivo principal de reducir el error de la red, primando a los individuos con menor error y dejando que el objetivo de reducir el número de entradas a la red se realice en otra parte del algoritmo como es la inicialización de los individuos.

3.4.3.2 Función de aptitud mejorada (suma ponderada de objetivos)

$$Función\ de\ fitness\ individuo_i = (W_1 * Error_i) + (W_0 * Unos_i)$$

Ecuación 11. Fórmula de la función de fitness mejorada.

Dónde:

***n**= Número de individuos de la población.*

***Unos** $_i$ = Porcentaje de parámetros que forman parte del individuo $_i$.*

***Error** $_i$ = Porcentaje de error del individuo $_i$ al ser evaluado por el clasificador.*

***W$_i$**= Pesos encargados de ponderar los diferentes objetivos.*

(Como apunte añadir que si al peso W_1 le damos un valor de 1 y al peso W_0 le damos el valor de 0, la función solamente valorará el porcentaje de error dejando a un lado el porcentaje de parámetros de la red, con lo que obtendremos que para ese ajuste particular de pesos, esta función será igual a la función de fitness básica explicada anteriormente).

Esta función es una versión mejorada con el fin de controlar la reducción de atributos o entradas de la red. En ella tratamos de ponderar el valor de cada uno de los dos objetivos más importantes con pesos. El porcentaje de error será multiplicado por un peso (entre 0 y 1) y el porcentaje de parámetros que contiene por otro peso diferente. La suma de ambos pesos obligatoriamente deberá dar 1. Esta función es más flexible que la anterior, aunque también tiene el problema de que será necesario calcular o hallar el mejor valor para cada uno de los pesos. Además con ella podremos valorar de alguna manera el número de entradas a la red que tiene el individuo valorando positivamente a los individuos que menor porcentaje de entradas tengan.

	% Error red	% Parámetros	*Fitness
Individuo 1	30	53	(W_1.30)+(W_0.53)=32,3
Individuo 2	30	46	(W_1.30)+(W_0.46)=31,6
Individuo 3	28	48	(W_1.28)+(W_0.48)=30
Individuo 4	25	65	(W_1.25)+(W_0.65)=29

*Hemos usado para el cálculo de este ejemplo los siguientes valores: W1= 0,9 W2=0,1.

Tabla 5. Cálculo del fitness con la función mejorada.

Siempre que los pesos sean diferentes a 0 ó 1, con esta nueva función de fitness conseguimos unir los dos objetivos existentes en nuestro problema en una nueva función fitness con un solo objetivo. De este modo, el objetivo de nuestro algoritmo será conseguir individuos con la menor valoración posible, (minimizar nuestro fitness) y al intentar disminuir al máximo los valores obtenidos estaremos consiguiendo que se cumplan ambos objetivos.

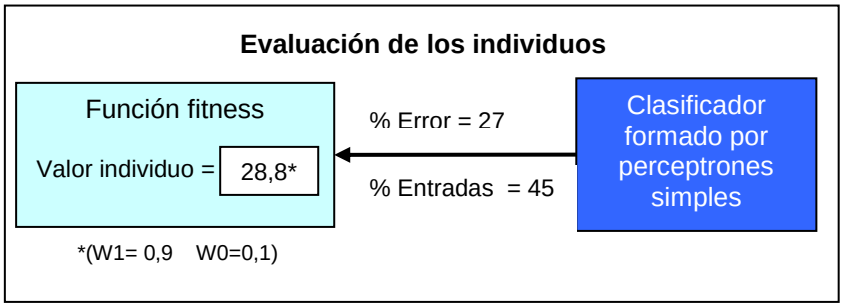


Figura 27. Esquema evaluación de un individuo con función mejorada.

Otro aspecto importante es que, antes de la ejecución de nuestro problema podremos ajustar la importancia que le damos a cada uno de nuestros objetivos en cada ocasión simplemente modificando los valores de los pesos. En nuestro caso, lo que pretendemos con esta función es ejercer una ligera presión sobre el número de entradas a la red para conseguir eliminar sólo aquellas que no afecten a la clasificación o que influyan negativamente en ella y no una reducción drástica, por ello utilizaremos un valor del peso de las entradas muy bajo. A continuación veremos cómo calculamos el valor de estos pesos para conseguir el mayor rendimiento posible a esta función.

3.4.3.2.1 Ajuste de los pesos para nuestro problema

Dado que nosotros comenzamos con una población formada por un 50% de parámetros más las operaciones el peso del porcentaje de entradas deberá de ser menor al del porcentaje de error ya que partimos ya con una reducción importante y no será necesario conseguir una reducción de entradas mucho más drástica. Además no nos servirá de mucho tener individuos formados por una cantidad muy baja de entradas si tienen un porcentaje de error muy alto. En cambio, unos individuos con un porcentaje de error muy bajo pero que tengan un porcentaje de entradas ligeramente superior si nos pueden ser útiles.

Tras la realización de un pequeño estudio (expuesto en el Anexo 1 Cálculo de los pesos para la función de fitness 2) hemos determinado que la mejor combinación de pesos en nuestro caso es la siguiente:

W_1 (Peso del error de la red) =0.95
W_0 (Peso de las entradas de la red) =0.05

3.4.4 Métodos de selección

Los procesos de selección sirven para elegir los individuos mejor adaptados existentes en la población para que estos sean evolucionados en las siguientes etapas del algoritmo genético. Para realizar este proceso de selección será necesaria una función de aptitud que mida la adaptación de los diferentes individuos. Esta función de aptitud es la función de fitness explicada en la sección anterior.

La importancia de este operador reside en que es el encargado de ir dirigiendo al algoritmo hacia regiones del espacio de búsqueda que parezcan mejores o más prometedoras. El éxito del algoritmo, en gran parte depende de este operador.

Existen multitud de métodos para realizar esta operación y cada uno de ellos ejerce una fuerza o presión sobre la población diferente y dependiendo de las necesidades del problema serán más útiles unos que otros. En nuestro caso concreto, los diferentes procesos de selección implementados son los explicados a continuación:

3.4.4.1 Selección jerárquica

Con este método conseguimos mantener la variabilidad genética con lo que nos evitamos problemas como el de la convergencia prematura. Lo primero que realiza este método es ordenar a los individuos según la valoración que le haya dado la función de fitness. Una vez hecho esto, a continuación calculamos la probabilidad de ser seleccionado utilizando la siguiente fórmula:

$$Probabilidad = A * Ranking$$

Ecuación 12. Selección jerárquica

Dónde:

- Ranking es la posición que ocupa cada individuo ordenando sus fitness del peor adaptado al mejor adaptado.
- $A = \frac{1}{\sum_{i=1}^{n} i}$ (n = Número de individuos de la población)

De este modo no se depende tanto del fitness para ser seleccionado. Una vez que hemos calculado todas las probabilidades de ser seleccionado para todos los individuos, se procede a introducirlos en la nueva población siguiendo las probabilidades asignadas.

3.4.4.2 Selección por el método de la ruleta

Con el método de la ruleta, el objetivo es que los individuos que están mejor adaptados según nuestra función de aptitud tengan mayor número de opciones de ser seleccionados que los que están peor adaptados, pero permitiendo también que los peor adaptados tengan opciones. Salvo en casos extremos, (un grupo pequeño de individuos con un fitness muy superior al resto), con este método permitiremos mantener una diversidad genética en nuestra población.

El funcionamiento es el siguiente: primero se calcula el sumatorio de los fitness de todos los individuos y a continuación, y una vez que se conoce el fitness total o sumatorio de los fitness, calculamos el fitness relativo de cada uno de los individuos. Las fórmulas utilizadas para ambos casos son las siguientes:

$$\text{Fitness total} = \sum_{i=1}^{n} \text{Fitness individuo}_i \qquad \text{Fitness relativo}_i = \frac{\text{Fitness individuo}_i}{\text{Fitness total}}$$

Ecuación 13. Fórmulas para selección por el método de la ruleta.

El fitness relativo nos indica la probabilidad de cada uno de ellos de ser seleccionados dando un mayor número de opciones a los individuos mejor adaptados. Una vez que tenemos todos los fitness relativos de los individuos los colocamos uno detrás del otro formando un vector. A continuación generamos un número aleatorio y el individuo seleccionado es el que pasa a formar parte de la población intermedia. Generaremos tantos números aleatorios como individuos sean necesarios. En el siguiente ejemplo podemos ver más detenidamente un proceso de selección por ruleta:

Tenemos 5 individuos cuyos fitness son:

Individuos	Fitness
Individuo 1	83
Individuo 2	64
Individuo 3	71
Individuo 4	41
Individuo 5	48

Tabla 6. Muestra el fitness de cada individuo.

Si calculamos el fitness total con la fórmula anterior nos dará:

Fitness total = 307

Y los distintos fitness relativos son:

Individuos	Fitness relativo
Ind 1	0,27
Ind 2	0,21
Ind 3	0,23
Ind 4	0,13
Ind 5	0,16

Tabla 7. Muestra el fitness relativo de cada individuo.

Colocamos los fitness relativos de todos los individuos con el fin de crear un vector de 0 a 1 como el que mostramos a continuación.

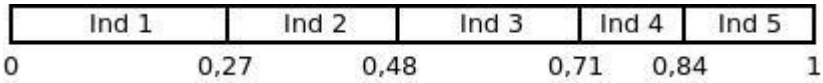


Figura 28. Vector que indica la probabilidad de los diferentes individuos.

De este modo, comenzamos a generar números aleatorios y el individuo seleccionado es que pasa a la población intermedia para que en etapas posteriores pueda ser cruzado, mutado, etc.

Por ejemplo, aleatoriamente se generan los siguientes números: 0.10, 0.24, 0.59, 0.73, 0.86. Siguiendo el vector anterior los individuos seleccionados son:

Figura 29. Vector con los individuos que han sido seleccionados.

Individuos	Fitness	Fitness relativo	Veces que ha sido seleccionado
Ind 1	83	0,27	2
Ind 2	64	0,21	0
Ind 3	71	0,23	1
Ind 4	41	0,13	1
Ind 5	48	0,16	1

Tabla 8. Muestra las veces que han sido seleccionados cada individuo.

3.4.4.3 Selección por torneos

Este método de selección va realizando comparaciones (torneos) entre pequeños grupos de individuos de la población, seleccionando el que mayor valor de aptitud (fitness) tenga de ellos. Estos torneos serán repetidos hasta que la nueva población o población intermedia sea completada.

El tamaño de estos torneos es importante ya que cuanto mayor sea, la competencia será mayor y por lo tanto los individuos menos adaptados no tendrán prácticamente opciones de sobrevivir, pudiendo llegar a dar problemas de convergencia prematura. En definitiva, la elección del tamaño del torneo nos permitirá ajustar la presión que queramos realizar sobre la población y por ello se debe tener especial cuidado en la selección del tamaño de los torneos con el fin de conseguir una buena selección para nuestro caso.

Podemos realizar este tipo de selección de varias formas diferentes, para nuestro algoritmo implementamos las siguientes:

3.4.4.3.1 Selección por torneos deterministas

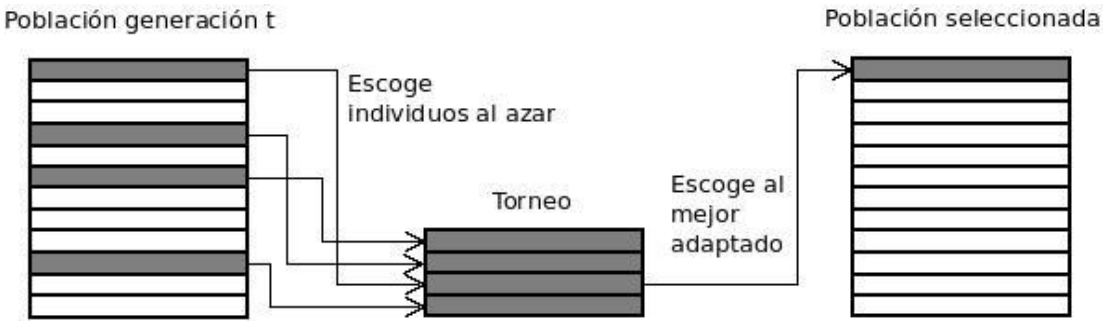


Figura 30. Muestra la realización de un torneo concreto.

Se trata del método de selección por torneos más sencillo. Determinamos el tamaño del torneo a partir del tamaño de la población con el fin de que esta selección sea buena. A continuación se escogen varios individuos al azar hasta completar todas las posiciones del torneo y el individuo con mejor fitness es escogido para acceder a la población intermedia. Se realizan todos los torneos necesarios hasta completar la población intermedia.

Con este método de torneos deterministas, un individuo puede ser seleccionado en varias ocasiones ya que la selección de individuos que forman parte del torneo se hace de forma totalmente aleatoria.

3.4.4.3.2 Selección por torneos probabilística

Con este método el tamaño de los torneos será siempre de dos. Por lo tanto los torneos serán enfrentamientos por parejas entre los distintos individuos. Inicialmente, desordenamos todos los individuos de la población. Después, vamos cogiendo a los individuos por parejas y dependiendo de un número aleatorio (entre 0 y 1) se decidirá cuál de los dos pasa a formar parte de la población intermedia [17]. Si este número es menor de 0.5, escogeremos al individuo menos adaptado de la pareja y este pasará a formar parte de la nueva población. En caso de que este número sea mayor o igual 0.5, escogeremos al individuo mejor adaptado de los dos. De este modo no se seleccionan a todos los individuos mejor adaptados sino que dependerá del azar.

Al realizarse los torneos por parejas, para completar la población intermedia será necesario que una vez completado una primera ronda de enfrentamientos los individuos vuelvan a ser desordenados y se vuelva a repetir el proceso de enfrentamientos, de este modo cada individuo tendrá siempre la posibilidad de ser seleccionado dos veces, una, o ninguna [17].

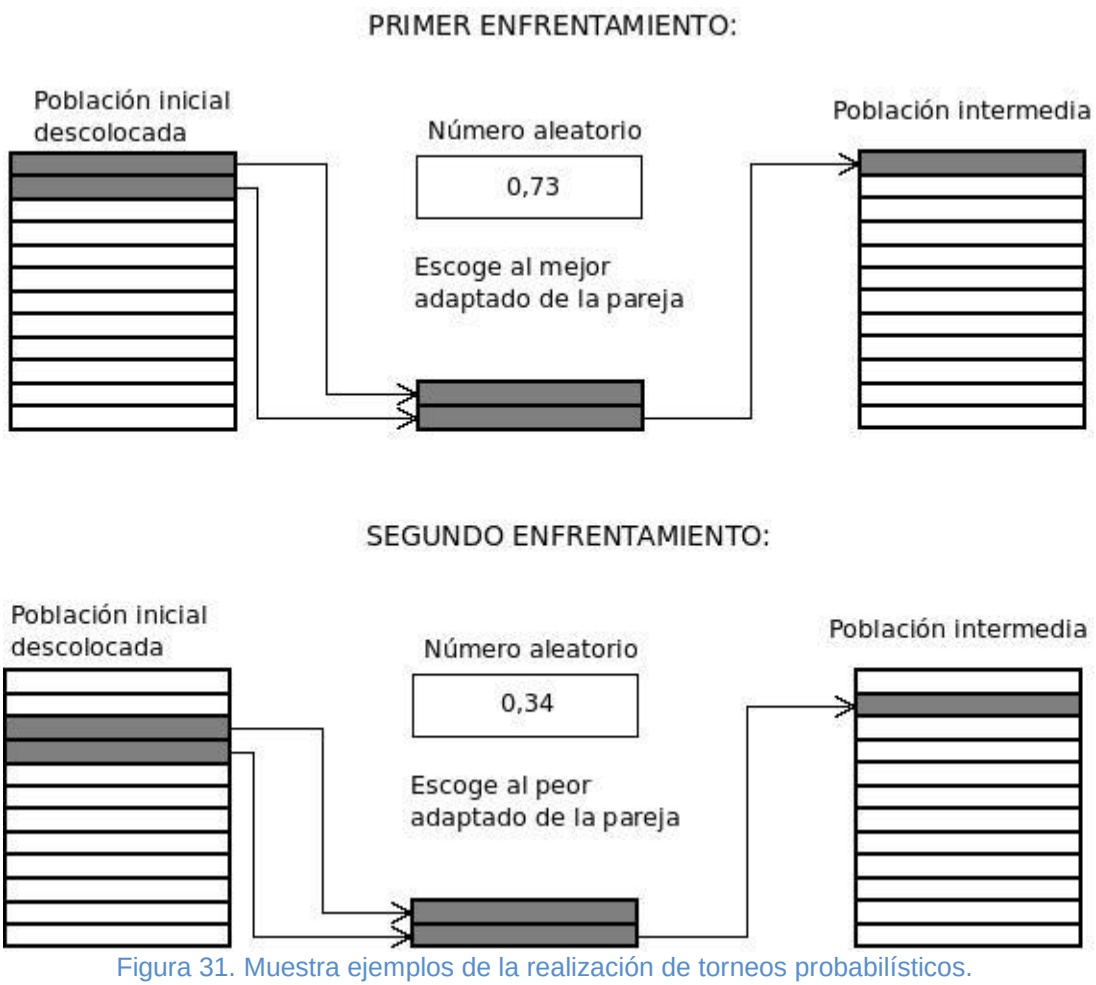


Figura 31. Muestra ejemplos de la realización de torneos probabilísticos.

3.4.5 Operadores genéticos destinados a crear nuevos individuos

La importancia de este tipo de operadores reside en que con ellos se consigue generar nuevos individuos a partir de los ya existentes. Estos nuevos individuos, en la mayoría de los casos, estarán mejor adaptados que sus predecesores a nuestra función de aptitud y por lo tanto serán mejores soluciones para nuestro problema.

Dentro de este grupo de operadores es necesario dividirlos en dos grandes grupos dependiendo de la parte del individuo en la que actúan, ya que se da la restricción de que los genes encargados de codificar las diferentes operaciones no pueden ser tratados de forma individual sino que deben ser tratados como si la unidad mínima fuera la operación. En ambos casos, la función que realizan será la misma.

3.4.5.1 Operadores específicos para los parámetros del individuo

Como ya hemos visto anteriormente, cada individuo está formado por genes que se dividen en dos grupos. Los operadores explicados a continuación solo realizarán su función sobre los genes que indican que datos forman parte de cada individuo, ya que si se aplicará también a los genes que codifican a las operaciones se podrían dar valores erróneos como explicaremos más adelante. Dentro de este amplio grupo de operadores podemos distinguir los siguientes subgrupos:

3.4.5.1.1 Métodos de cruce

Los operadores de cruce o recombinación (crossover en inglés) son un tipo de operadores muy utilizado en los algoritmos genéticos. Su función es la de crear variaciones o diferencias en los individuos entre las distintas generaciones. Principalmente operan entre dos individuos padres que combinan consiguiendo dos individuos hijos que conservan características de ambos padres. Este operador es equivalente a la reproducción sexual dada en la biología. Para nuestro problema se han implementado los siguientes cruces explicados a continuación. Todos ellos tienen asignada una probabilidad de cruce con la que conseguimos que no todos los individuos que fueron seleccionados previamente sean cruzados.

3.4.5.1.1.1 Cruce simple un punto

Este es uno de los métodos más sencillos de cruce. Simplemente se selecciona un punto intermedio (que divide la trama de genes de los padres en dos partes iguales) y a partir de ese punto todos los genes serán intercambiados entre los padres. De este modo creamos 2 individuos hijos que contienen la mitad de genes de cada uno de sus padres.

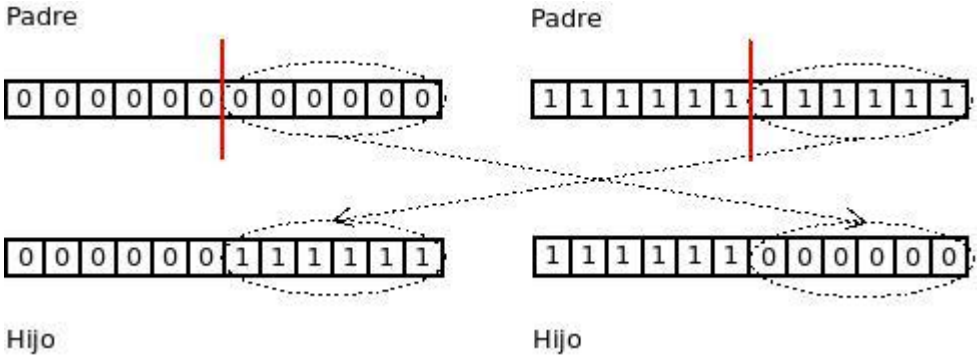


Figura 32. Ejemplo de un cruce simple en un solo punto.

3.4.5.1.1.2 Cruce simple dos puntos

Este cruce es similar al descrito anteriormente. La diferencia es que en este caso se seleccionan dos puntos dentro de cada padre por lo que la trama de genes de cada padre queda dividida en tres partes iguales. A continuación, se intercambia la parte que queda contenida entre ambos puntos (parte central) de cada padre. De este modo obtenemos 2 individuos hijos que contienen genes de ambos padres, más concretamente un 66% de genes de uno de sus padres y un 33% del otro.

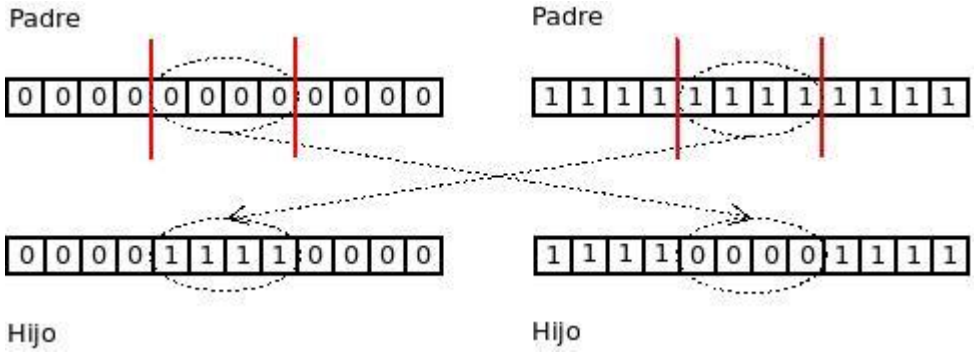


Figura 33. Ejemplo de un cruce simple en dos puntos.

3.4.5.1.1.3 Cruce uniforme

Este cruce es un poco más complejo que los anteriores. En este caso los padres cruzarán los genes uno a uno utilizando para ello una nueva probabilidad llamada factor de cruce. De este modo, los descendientes estarán formados por los genes de sus padres pero estos serán escogidos

de forma aleatoria y no por tramas de genes consecutivos de cada uno de sus padres.

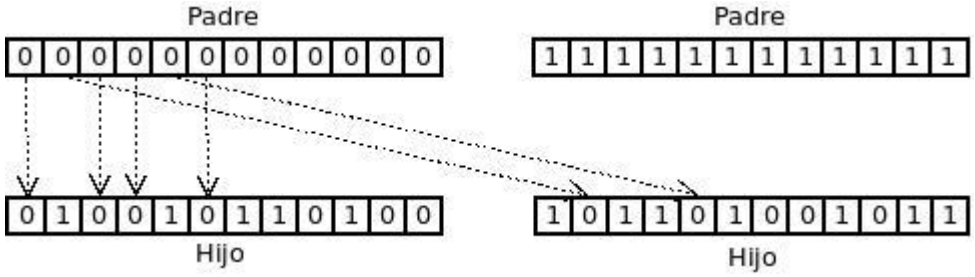


Figura 34. Ejemplo de un cruce uniforme.

3.4.5.1.2 Método de mutación

El operador de mutación, al igual que el de cruce, cumple la función de crear variaciones entre los individuos provocando diferencias entre las distintas generaciones. También realiza la acción de mantener la diversidad entre la población ya que puede introducir cambios en los genes de los individuos que no estaban en generaciones anteriores consiguiendo que estos nuevos individuos cubran otras regiones del espacio de búsqueda, antes no cubiertas, que con otros operadores como el de cruce no se podrían realizar. Tiene su equivalencia en la mutación biológica.

El funcionamiento de este operador es simple, se trata de modificar un gen (un bit 0, 1) transformándolo en el inverso. Este operador, al igual que los operadores de cruce, tiene asignada una probabilidad de mutación. De este modo, conseguimos que no toda la población sufra mutaciones sino que dependiendo de esta probabilidad haya individuos que no sufran ninguna transformación y otros que si las sufran.

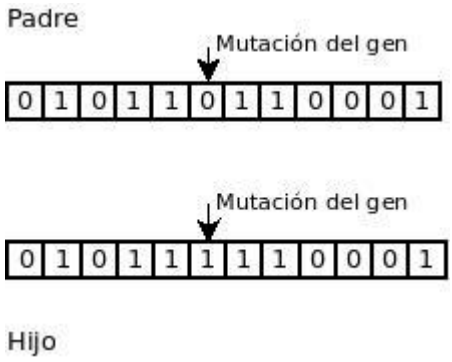


Figura 35. Ejemplo de la mutación de un gen concreto.

Además, los individuos que se haya determinado que han de ser mutados, tendrán asignada otra probabilidad llamada factor de mutación que determinará los genes del individuo que serán transformados. Por lo tanto, el operador primero escoge que individuos de la población han de ser mutados

y a continuación recorre las cadenas de genes de los individuos elegidos y muta varios de estos genes dependiendo del factor de mutación que se le haya asignado.

De este modo conseguimos que se muten varios genes al azar en cada individuo que se ha decido mutar, mientras otros individuos no reciban ninguna mutación.

3.4.5.1.3 Método de inversión

Es equivalente a la inversión cromosómica dada en la biología. Su objetivo, al igual que la mutación es ampliar el espacio de búsqueda ya que al cambiar el orden de varios de los genes o bits se consigue crear un individuo nuevo que contiene características del padre pero también otras características nuevas. Por lo tanto, también se encarga de mantener la diversidad entre la población.

Dependiendo de una probabilidad dada (probabilidad de inversión), selecciona al azar un número concreto de genes consecutivos dentro del individuo e invierte sus posiciones.

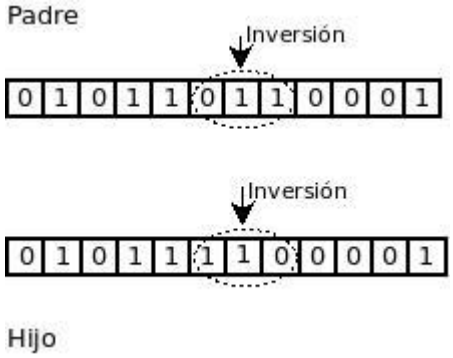


Figura 36. Muestra la inversión producida en un individuo.

3.4.5.2 Operadores específicos para las operaciones

Dadas las características de nuestro problema, tenemos la restricción de que no podemos aplicar los operadores de cruce, mutación, inversión descritos anteriormente a los genes que forman las operaciones, ya que no se pueden tratar cada uno de los genes de forma independiente porque están agrupados de diferentes formas para poder codificar las operaciones como explicamos en la sección (3.4.1 Representación de los individuos). Como se quiere evitar la creación de individuos no válidos, es necesario tratar a estos genes como grupos y no individualmente. Si se aplicaran los operadores descritos previamente en los genes de las operaciones, no se controlaría la restricción y un cambio en un gen podría producir valores erróneos o fuera del rango de nuestro problema.

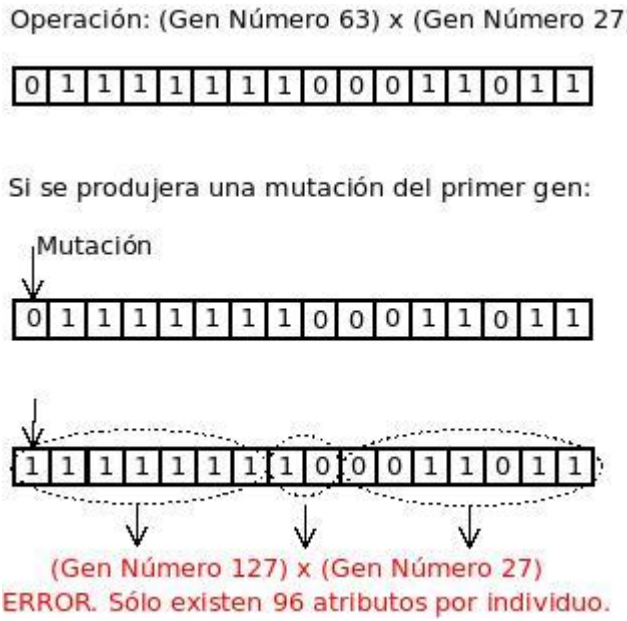


Figura 37. Error al usar la mutación simple en las operaciones.

Por ejemplo como vemos en la imagen anterior, si al operador 63 se le cambiara el primer gen daría como resultado el operador 127 caso imposible para nuestro problema ya que cada individuo como mucho cuenta con 96 atributos.

Para atajar este problema, nosotros controlamos esta restricción creando nuevos operadores a base de realizar adaptaciones a varios de los operadores descritos anteriormente, de este modo tenemos unos operadores nuevos específicos para la parte genética que codifica las operaciones con lo que conseguimos que todos los genes del individuo puedan irse modificando y adaptando al problema.

Los operadores creados para ser utilizados con la parte genética del individuo que codifica las operaciones son los siguientes:

3.4.5.2.1 Mutación de las operaciones

En esencia se basa en el mismo concepto que la mutación normal. La trasformación de un gen en su inverso con una determinada probabilidad dada. La diferencia es que en vez de mutar un solo gen mutaremos una operación al completo (16 genes) teniendo especial cuidado ya que como explicamos antes, las operaciones están formadas por 3 agrupaciones de genes diferentes (operando1, operador y operando2). Independientemente de esto, el resultado del operador será la transformación de una operación en otra totalmente nueva (creada de forma aleatoria). Esta transformación se hará con una determinada probabilidad que se llamara factor de mutación de operaciones.

Por lo tanto, el operador recorre la cadena de genes del individuo, mutando las operaciones según el factor de mutación de operaciones. En el caso de que se mute una operación, se elimina la que existía y se crea otra nueva asignando un nuevo operando1, un nuevo operador y un nuevo operando2 de forma totalmente aleatoria (en el caso del operador se seguirá teniendo en cuenta las probabilidades de aparición de cada uno de los tipos).

De este modo, si tenemos una multiplicación del gen 7 con el gen 35 y resulta que el algoritmo decide mutarla, se eliminaría esa multiplicación y se crea una nueva operación con dos nuevos operandos cogidos al azar y un nuevo operador (suma, resta, división ó multiplicación) siguiendo las probabilidades indicadas.

Este operador cumple la función de crear variaciones entre los individuos provocando diferencias entre las distintas generaciones. También con él conseguimos mantener la diversidad entre la población ya que puede introducir cambios en las operaciones de los individuos que no estaban en generaciones anteriores, consiguiendo que estos nuevos individuos cubran otras regiones del espacio de búsqueda antes no cubiertas.

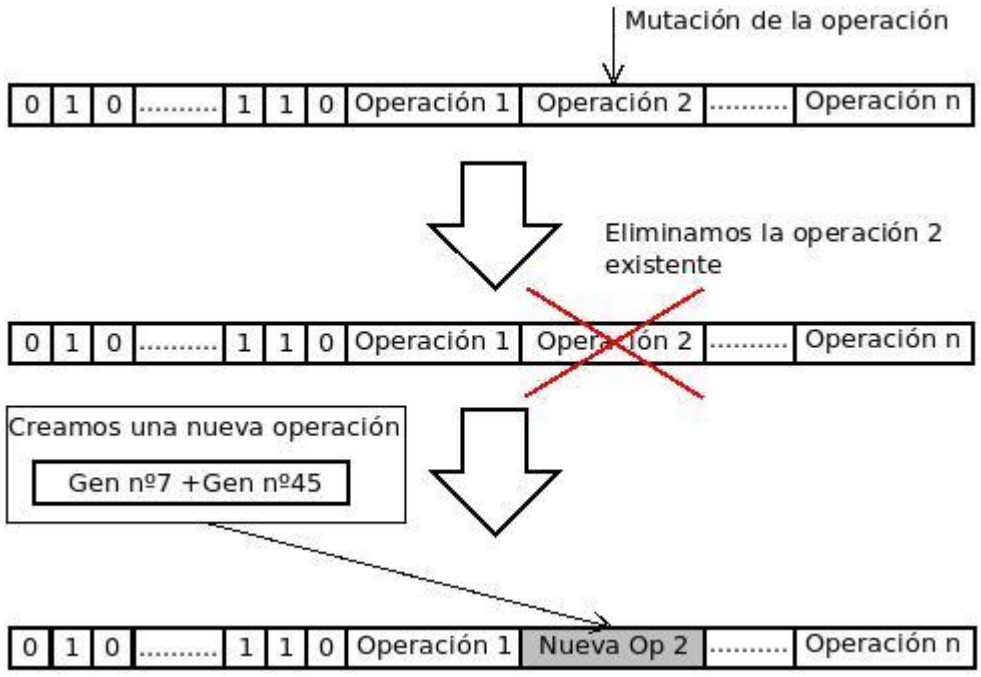


Figura 38. Muestra la mutación de una operación concreta.

Este operador se puede aplicar de forma independiente o como complemento a la mutación normal explicada anteriormente como veremos en la sección 3.4.5.3 Modos de uso de los operadores.

3.4.5.2.2 Cruce uniforme de las operaciones

Este cruce, al igual que los cruces anteriores explicados, opera entre dos individuos padres que combina consiguiendo dos individuos hijos que conservan características de ambos. En este caso, solo realizará estas acciones en la parte de las operaciones.

Su función es la de crear variaciones o diferencias en los individuos entre las distintas generaciones.

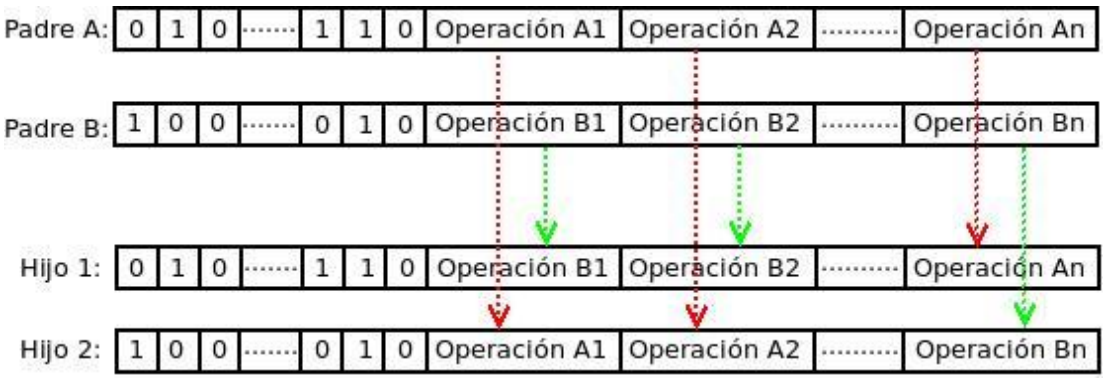


Figura 39. Ejemplo de un cruce uniforme de operaciones.

Está basado en el cruce uniforme pero aplicando un ligero cambio para evitar que de valores no válidos. Este cambio consiste en que, en vez de cruzar un solo gen dado el factor de cruce de operaciones, cruzaremos una operación al completo (formada por 16 genes). De este modo, cada uno de los descendientes contendrá operaciones de cada uno de sus padres escogidas de forma aleatoria.

Este método se podría utilizar como complementario a los cruces de genes anteriores o como un operador independiente como veremos en la sección 3.4.5.3 Modos de uso de los operadores.

3.4.5.3 Modos de uso de los operadores

Existen varias formas de utilizar estos operadores explicados en las secciones anteriores, ya que tanto los operadores de la parte de las operaciones como los de la parte de los atributos pueden ser usados de forma complementaria entre ellos como de forma independiente. A continuación pasamos a explicar ambas formas de utilización.

3.4.5.3.1 Juntos

Si utilizamos los operadores de las operaciones de forma conjunta a los operadores de los atributos o entradas de la red estos irán unidos de modo que si se decide cruzar o mutar a un individuo se realizará la mutación

o el cruce del individuo al completo (atributos y operaciones). Por ejemplo en el caso del cruce, si la probabilidad de cruce determina que hay que cruzar ese individuo se cruzará los genes de ese individuo (usando el método de cruce que se haya determinado) dependiendo del factor de cruce correspondiente y a continuación se cruzarán las operaciones dependiendo del factor de cruce de las operaciones.

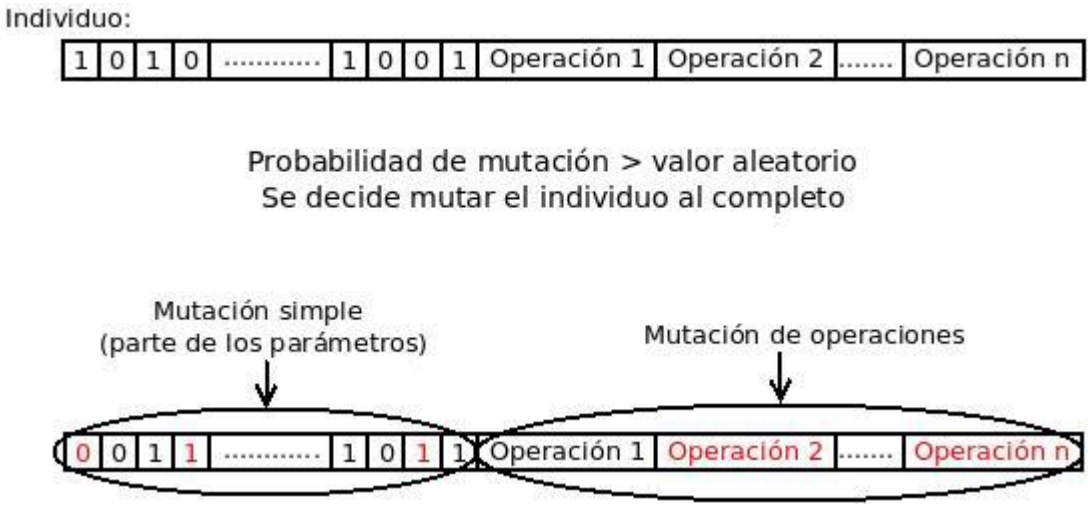


Figura 40. Operadores usados de forma conjunta.
(En rojo las mutaciones producidas. El caso de cruce será igual.)

Usando este modo de utilizar los distintos operadores genéticos, los distintos operadores destinados a generar nuevos individuos serán sólo tres:

1) Cruce conjunto (Cruce genes+ Cruce de operaciones uniforme).
2) Mutación conjunta. (Mutación genes + mutación de operaciones).
3) Inversión.

Este será el modo de uso que más utilicemos durante la experimentación ya que nos parece el más útil.

3.4.5.3.2 Separados

Al utilizarse los operadores de forma independiente, cada uno de los individuos previamente seleccionados podrá sufrir las modificaciones en su cromosoma de cualquiera de los operadores, de varios de ellos, de ninguno, de todos, etc. En este caso, que se cruce los genes de un individuo no significa que se vayan a cruzar sus operaciones, o por el contrario, si se cruzan sus operaciones no significa que vayan a ser cruzados sus genes. Ocurrirá lo mismo en el caso de la mutación. Por lo tanto, cada uno de los operadores tendrán asignada una probabilidad y será esta la que designe a que individuos afectará cada uno de ellos.

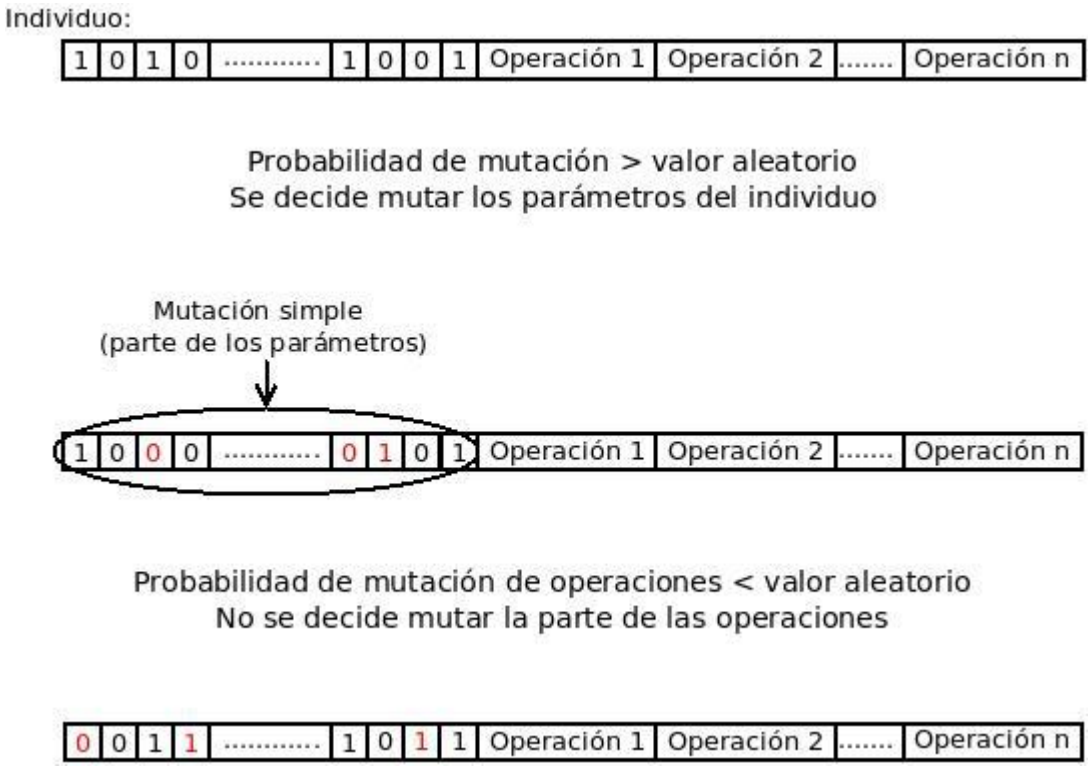


Figura 41. Operadores usados de forma independiente.
(En rojo las mutaciones producidas. La imagen anterior solo muestra una de las cuatro opciones posibles. En este caso, se produce una mutación de genes pero no una mutación de operaciones. El funcionamiento en el caso de cruce será igual.)

Usando este modo de utilizar los distintos operadores genéticos, los distintos operadores genéticos destinados a generar nuevos individuos son cinco:

1) Cruce.
2) Mutación.
3) Inversión.
4) Cruce de operaciones uniforme.
5) Mutación de operaciones.

Todos ellos se ejecutarán de modo independiente y tendrán probabilidades de ejecución diferentes como vimos en la imagen anterior lo que implicará que la ejecución de uno de ellos no afectará a la ejecución o no de los demás. Un individuo puede verse modificado por uno, varios o incluso todos los operadores.

3.4.6 Métodos de reemplazo

Una vez aplicados el resto de operadores, (selección, cruce, mutación, etc.) obtendremos dos poblaciones de individuos diferentes, la de

los individuos padres, y otra nueva de individuos hijos. El operador de reemplazo se ocupa de realizar una selección de los mejores individuos con el fin de que estos pasen a la población inicial de la siguiente generación, pero además se debe ocupar de evitar que la población acabe siendo homogénea manteniendo una diversidad genética. De este modo, con un buen operador de reemplazo conseguiremos promocionar a los mejores individuos e ir mejorando poco a poco la población en cada generación, consiguiendo que nuestra población se vaya adaptando cada vez más a nuestra función de aptitud, y por lo tanto obtengamos mejores soluciones a nuestro problema pero sin perder la diversidad de los individuos.

Este reemplazo se puede realizar de varias maneras, para el problema que nos ocupa hemos implementado los siguientes:

3.4.6.1 Reemplazo generacional

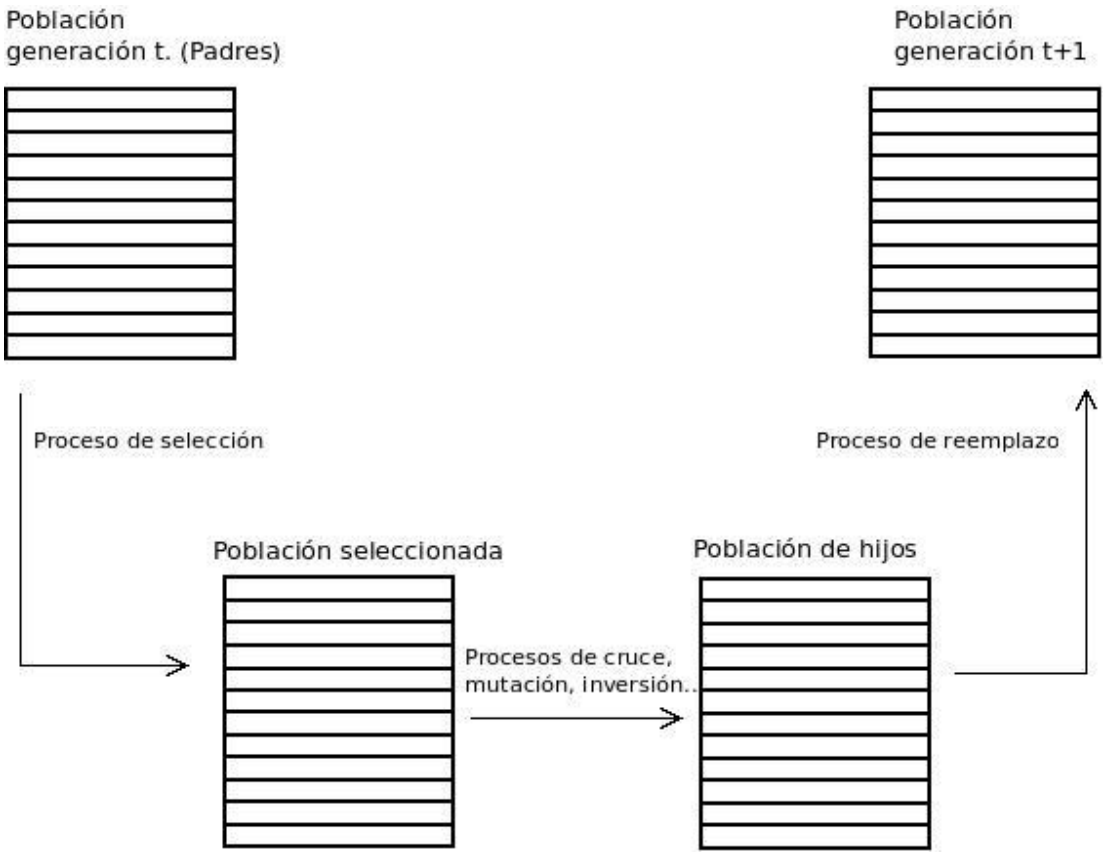


Figura 42. Ejemplo de reemplazo generacional.

Este es el método de reemplazo más simple, tras realizar todas las operaciones la población de individuos hijos (creados durante esa generación) sustituye a la totalidad de los padres. Uno de los principales

inconvenientes de este reemplazo es que al eliminar a todos los padres independientemente de su nivel de adaptación se pueden perder individuos muy adaptados.

3.4.6.2 Reemplazo estacionario

En este caso, en vez de sustituir a la totalidad de los padres por sus descendientes, se hace que un porcentaje de padres permanezcan en la nueva población. Este porcentaje debe ser bajo ya que si es alto la población no evolucionaría o podría acarrear otro tipo de problemas.

Con este método, conseguimos que los *n* mejores padres permanezcan en la nueva población y el resto de individuos sean ocupados por los mejores hijos. Es necesario tener especial cuidado con este porcentaje ya que un porcentaje elevado nos podría llevar a una convergencia prematura debido a que estaríamos manteniendo a los mejores individuos padres y además los pocos hijos que entren en la población probablemente sean descendientes de estos mismos padres.

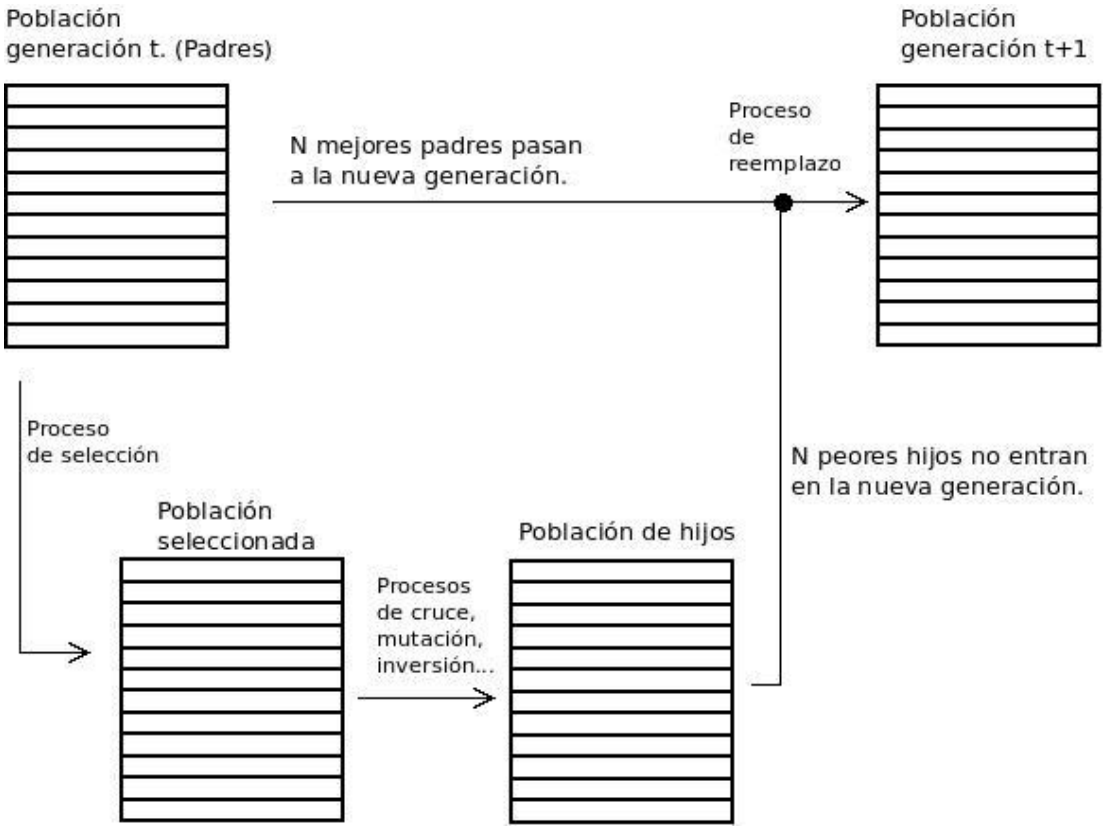


Figura 43. Ejemplo de un reemplazo estacionario.

3.4.6.3 Reemplazo estacionario con elementos aleatorios

Con este método, lo que intentamos conseguir es que no sólo los mejores individuos, sean padres o hijos, pasen a la nueva población sino que también pasen a esta nueva población padres e hijos elegidos al azar.

Así conseguimos que haya una mayor diversidad en la población evitando problemas de uniformidad en la población. Eso sí, para que este reemplazo sea verdaderamente útil a nuestro objetivos conviene que el número de hijos en la nueva población sea mayor que el de padres para conseguir que la población vaya evolucionando generación tras generación y no se estanque y además también es necesario que los porcentajes de los individuos que son elegidos al azar sean menores que los que están mejor adaptados.

Por último decir que no hay ningún tipo de limitación a la hora de elegir los individuos al azar y estos podrán ser elegidos más de una vez. Incluso los mejores individuos que ya hayan sido seleccionados tendrán opciones de volver a ser elegidos al azar.

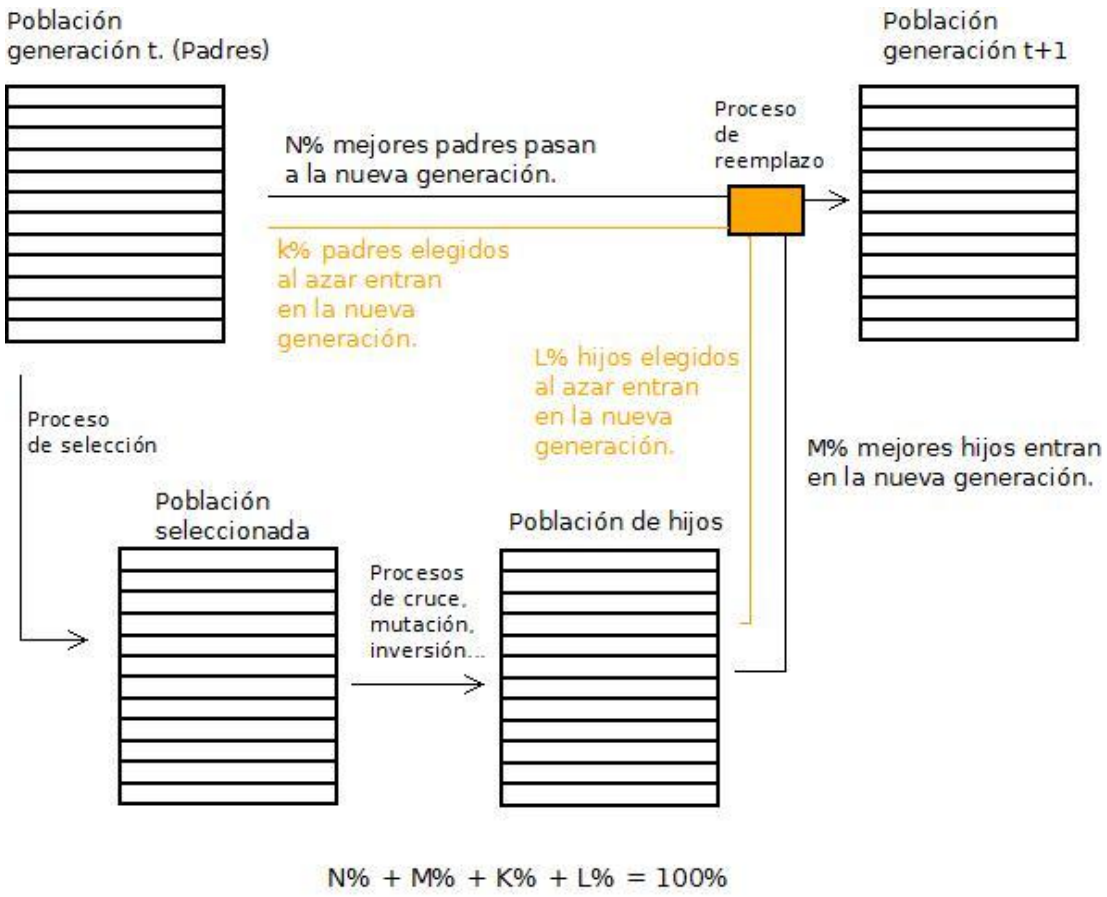


Figura 44. Ejemplo de reemplazo estacionario con elementos aleatorios.

3.4.7 Condición de término

Como ya explicamos antes, lo ideal sería que el algoritmo genético terminase su ejecución una vez que haya alcanzado la solución óptima pero dado que ésta solución óptima no siempre se conoce, utilizaremos como condición de parada para nuestro algoritmo el número de generaciones. De este modo, el algoritmo siempre ejecutará el número de generaciones que le digamos independientemente de los resultados que obtengamos. Por ello, deberemos tener cuidado al elegir este número de generaciones ya que si es bajo o muy bajo, la población prácticamente será la inicial y no habrá dado tiempo a que el algoritmo por si solo haya buscado las mejores soluciones en el espacio de búsqueda. Si por el contrario es muy alto, a simple vista no tendremos ningún problema salvo que el tiempo de ejecución aumentará y que es posible que con determinados operadores tengamos el problema de que la población se termine pareciendo mucho.

3.4.8 Otros aspectos o parámetros a tener en cuenta

Además de todo lo explicado anteriormente, es necesario hablar de los siguientes parámetros del algoritmo genético de los que no hemos tenido posibilidad de hablar anteriormente.

- **Tamaño de la población:** Es el número de individuos que formarán la población. Es importante ya que, si el número de individuos es bajo existe un mayor riesgo de que el algoritmo converja prematuramente. Un tamaño de la población grande en cambio, ayudará a evitar problemas de convergencia al proveer al algoritmo de un mayor número de soluciones por ronda.

- **Número de operaciones:** Ya habíamos hablado de las operaciones a la hora de describir la forma de representar a los individuos pero no hablamos de la cantidad de estas que formarán parte de cada individuo. Como habíamos comentado, los individuos estarán formados por entradas de datos normales y por operaciones. Con este parámetro, podremos elegir el número de operaciones que formará parte de los individuos. En principio, no hay ningún problema en que este número de operaciones sea alto o bajo y será la experiencia la que nos diga que número de operaciones es el más adecuado. Eso sí, a la hora de crear a la población todos los individuos contendrán el mismo número de operaciones.

3.5 Clasificador

En esta sección y sus correspondientes subsecciones hablaremos sobre los diferentes aspectos del clasificador. Empezaremos explicando cuál será su función de este dentro de nuestra solución, que tipo de red neuronal elegimos para su composición y el porqué, y también describiremos los diferentes modelos de clasificador que implementamos con el fin de tratar de descubrir cuál de ellos es el que mejor se adapta a nuestro problema.

De esta forma quedará todo explicado para que más tarde, dentro ya del capítulo de Experimentación podamos ver el comportamiento de cada uno de los clasificadores propuestos y podamos elegir uno de ellos para nuestro fin.

3.5.1 ¿Cuál será la función y el funcionamiento del clasificador?

Como ya explicamos, el objetivo principal es el conseguir separar las tres clases diferentes de patrones de los que disponemos consiguiendo la mayor tasa de aciertos posible.

Dentro de nuestra solución, como ya hemos visto, para que el algoritmo genético descrito anteriormente funcione correctamente es necesario implementar una función de fitness que determine cómo de adaptados están los individuos a nuestro problema. Ya vimos que teníamos planteadas varias funciones fitness para nuestro problema pero todas tenían en común varios aspectos:

- Se necesitaba saber el número de entradas a la red que tenía cada individuo para su valoración.

- También ambas funciones fitness necesitaban ver el comportamiento producido por esas entradas. Para ello es necesario que la red se entrene con cada individuo y los datos del BCI destinados para este entrenamiento y devuelva el porcentaje de error producido al validar este entrenamiento.

Por lo tanto, la función de este clasificador será el devolver el porcentaje de error producido por cada uno de los individuos de la población. Esa información, como ya vimos será tratada por la función fitness del algoritmo genético.

En cuanto al funcionamiento interno del ciclo de aprendizaje del clasificador este es el siguiente. A la entrada del clasificador nos llegará un individuo concreto como los descritos anteriormente. Utilizaremos los registros del fichero de entrenamiento para entrenar el clasificador para ese individuo con lo que conseguiremos que los pesos y el umbral de los perceptrones que forman parte del clasificador se vayan adaptando a nuestro problema. Esto se conseguirá introduciendo sólo los parámetros y las operaciones que indica ese individuo al clasificador (Figura 46). Posteriormente, y una vez terminado el entrenamiento, se comprobará el porcentaje de error producido por el clasificador (conocido como error de validación). Esto se hará utilizando los pesos y umbral obtenidos tras el entrenamiento anterior y utilizando de nuevo los registros de datos de entrenamiento. Evidentemente al igual que antes sólo utilizaremos para ello los parámetros y las operaciones que indica ese individuo al clasificador. Estos dos pasos anteriores, entrenamiento y comprobación del porcentaje de error, se repetirán un número concreto de veces con el fin de conseguir el menor porcentaje de error posible adaptando lo mejor posible los distintos elementos de los perceptrones. Ya veremos cómo determinamos este número de ciclos o repeticiones en el capítulo de Experimentación.

Una vez terminado este ciclo de aprendizaje, el clasificador devolverá el error que ha producido ese individuo. En el siguiente esquema podemos ver el funcionamiento explicado anteriormente de forma gráfica:

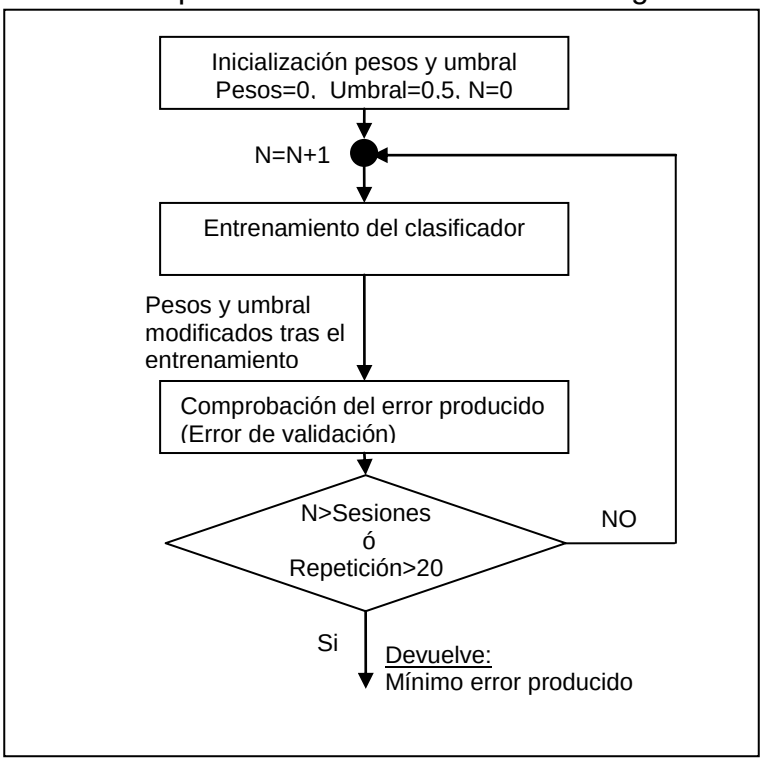


Figura 45. Esquema del entrenamiento del clasificador.
("Sesiones" indicará el número de iteraciones (entrenamiento, comprobación) que queremos que se produzcan en cada ciclo de aprendizaje.)

En cuanto a la fase de test, el clasificador además de realizar todo lo anterior, utilizará después otros conjunto de datos que se encuentran en otro fichero aparte con los que no se ha entrenado nunca (fichero de test) y sin hacer ningún tipo de modificación de pesos o umbral comprobará el error que produce el clasificador para esos nuevos datos (error de test). De este modo podremos ver el comportamiento del clasificador con los parámetros y operaciones de un individuo concreto al introducir datos ajenos a los que utilizamos en el entrenamiento. Esta será la mejor forma de comprobar la adaptación de la red.

Para terminar, en la siguiente imagen podemos ver como se forma la entrada de datos de la red neuronal. (Como se introducen sólo los parámetros y las operaciones que indica un individuo concreto).

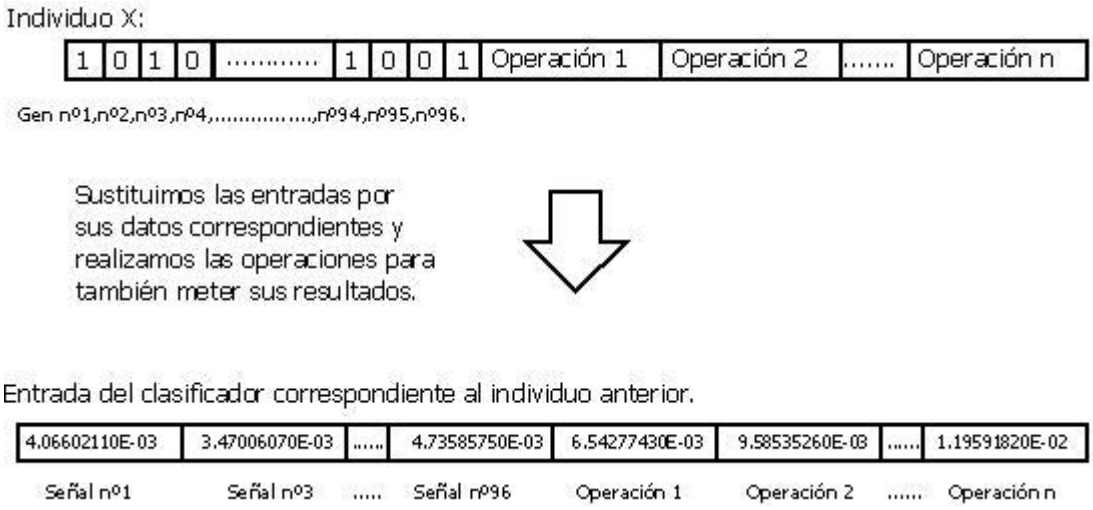


Figura 46. Formación de la entrada del clasificador a partir de un individuo.

Simplemente se cambiarán las entradas por los datos correspondientes (las entradas solo serán los genes con valor 1) y realizará las operaciones que tiene descritas el individuo para incluir también los resultados de estas. Una vez hecho esto, estos datos serán introducidos a la red neuronal propiamente dicha.

3.5.2 ¿Por qué elegimos perceptrones para el clasificador?

Como ya hemos explicado, nos es necesario implementar un sistema que nos permita clasificar las tres clases de datos existentes. Para realizar esta tarea, no disponemos de ningún tipo de modelo programable que sea capaz de identificar las clases, si bien, de lo único que disponemos es de un

amplio conjunto de datos de ejemplo con la entrada correspondiente y la salida que esta provoca. En la sección 2.3.5 Características de las redes de neuronas artificiales y utilidad vimos que una de las principales características de las redes de neuronas era precisamente eso, el ser capaz de solucionar problemas a partir de un gran conjunto de ejemplos, dada su capacidad de aprendizaje, adquiriendo las propiedades necesarias a través de estos ejemplos. Esta característica principalmente y otras muchas explicadas en esa sección hacían que las redes neuronales fueran efectivas para su uso en la clasificación de datos entre otros muchos campos. Es por ello por lo que decidimos que el uso de una red neuronal para solucionar este problema era una buena solución.

De entre todos los tipos de redes de neuronas que creímos conveniente analizar, elegimos el perceptrón simple. Las razones por las que escogimos este tipo de red frente a otras son las siguientes: El perceptrón simple es un sistema menos complejo que el multicapa al necesitar el segundo una o varias capas ocultas que, además de aumentar en gran medida el tiempo computacional, hacen necesario implementar un algoritmo de propagación para rectificar los pesos de todas las capas. Este algoritmo de aprendizaje será muchísimo más complejo que el del perceptrón simple. Asimismo no teníamos la certeza de que un clasificador formado por perceptrones multicapa fuera a mejorar el resultado obtenido por uno formado por perceptrones simples al que además de entradas de datos simples le hemos añadido otro tipo de entradas formadas por operaciones como ya vimos anteriormente.

Otro tipo de red neuronal que podríamos haber usado, y por la que al final no nos decantamos, es la Adaline. La estructura de esta red es similar a la del perceptrón simple pero con la diferencia de que, en vez de tener una salida binaria, su salida es real. Para nuestro caso, es más útil contar con una salida binaria, como la del perceptrón simple, a la que podemos asociar directamente con una clase o un patrón de datos, que una salida real que, si bien, este tipo de salida real nos serviría para cuantificar cuánto nos hemos equivocado en la clasificación, sería más complejo realizar la asociación entre la salida de la red y los distintos patrones de datos.

3.5.3 Diferentes clasificadores planteados

Para realizar las funciones de clasificación explicadas anteriormente, se pueden usar varios tipos de clasificadores utilizando combinaciones de perceptrones simples. Nuestro objetivo es analizar algunas de las opciones disponibles y realizar diferentes pruebas con ellos con el fin de encontrar la que mejor se adecua a nuestro problema. Como es de entender, existen otras muchas formas de implementar un clasificador para el caso que nos ocupa, evidentemente era imposible implementarlas y probarlas todas.

Hemos realizado las pruebas sólo con aquellos clasificadores que nos parecen más obvios o que creemos que nos iban a dar mejor resultado. Los dos clasificadores implementados y analizados son los explicados a continuación.

3.5.3.1 Clasificador con dos perceptrones simples

Dado que el problema de clasificación de los datos del BCI al que nos enfrentamos tiene tres clases diferentes, para conseguir un clasificador usando dos perceptrones necesitaríamos codificar las distintas clases de la siguiente manera (00, 01, 10). De esta forma, un primer perceptrón se encargaría de separar una clase de las otras dos y el otro perceptrón separaría las dos clases restantes. En el caso de que ambos perceptrones se activasen (11) la salida sería errónea ya que no coincidiría con ninguna salida correcta. El esquema de la red es mostrado a continuación:

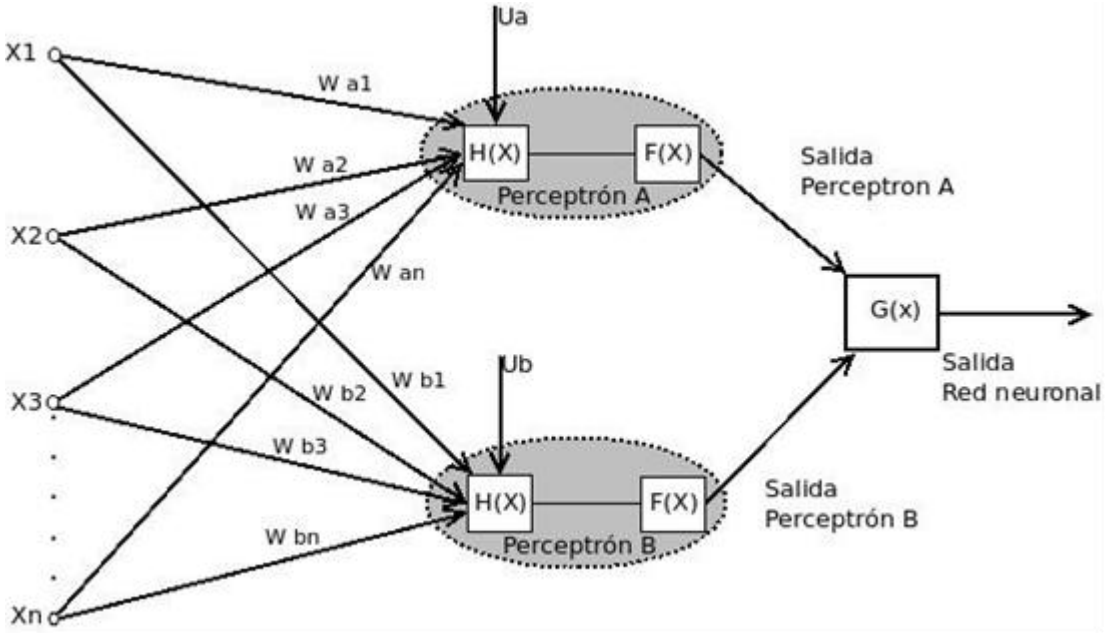


Figura 47. Esquema del clasificador formado por dos perceptrones.

Ambos perceptrones reciben la misma entrada de datos pero se entrenan de forma independiente. El primer perceptrón lo hace con el primer bit de la solución y el segundo perceptrón con el segundo bit. En ambos casos usan la regla de aprendizaje de Windrow-Hoff (que es similar a la regla de aprendizaje explicada en la sección "2.3.6.1 Algoritmo de aprendizaje del perceptrón simple" pero adaptada para casos en cuya salida es 0 ó 1):

$$w_i(t+1) = w_i(t) + \alpha(d(x) - y(x)).x_i$$
$$U_i(t+1) = U_i(t) + \alpha(d(x) - y(x))$$

Ecuación 14. Regla de aprendizaje de Windrow-Hoff

Dónde:

d (x): Es la salida que se desea que tenga el perceptrón.
y (x): La salida devuelta por el perceptrón.
w_i (t+1): Nuevos pesos del perceptrón hallados.
w_i (t): Pesos anteriores.
x_i : Entrada del perceptrón.
U_i (t): Umbral anterior.
U_i (t+1): Nuevo umbral hallado.
α: Razón de aprendizaje.

La función H(x) que aparece en el esquema anterior es la función de entrada de cada uno de los perceptrones y será la misma que explicamos en secciones anteriores. La función de activación de ambos perceptrones (F(x) en la Figura 47) es una función de tipo umbral. Ambos perceptrones usan la misma y asignan a la salida del perceptrón un 0 ó un 1 de la siguiente forma (Explicado detalladamente en la sección 2.3.6 El perceptrón simple):

$$F(x)\begin{cases} 0 & \text{Si } H(x) \le 0 \\ 1 & \text{Si } H(x) > 0 \end{cases} \quad \text{ó dicho de otra forma} \quad F(x)\begin{cases} 0 & Si \quad \sum_{i=0}^{n}(x_i.w_i)+U \le 0 \\ 1 & Si \quad \sum_{i=0}^{n}(x_i.w_i)+U > 0 \end{cases}$$

Ecuación 15. Fórmula de activación desarrollada de los perceptrones.

Una vez que los dos perceptrones hayan calculado de forma independiente su salida, G(x) las recibirá y simplemente ordenará ambos bits. A continuación vemos un ejemplo del funcionamiento de este clasificador. En este ejemplo, la salida final del clasificador es “10”.

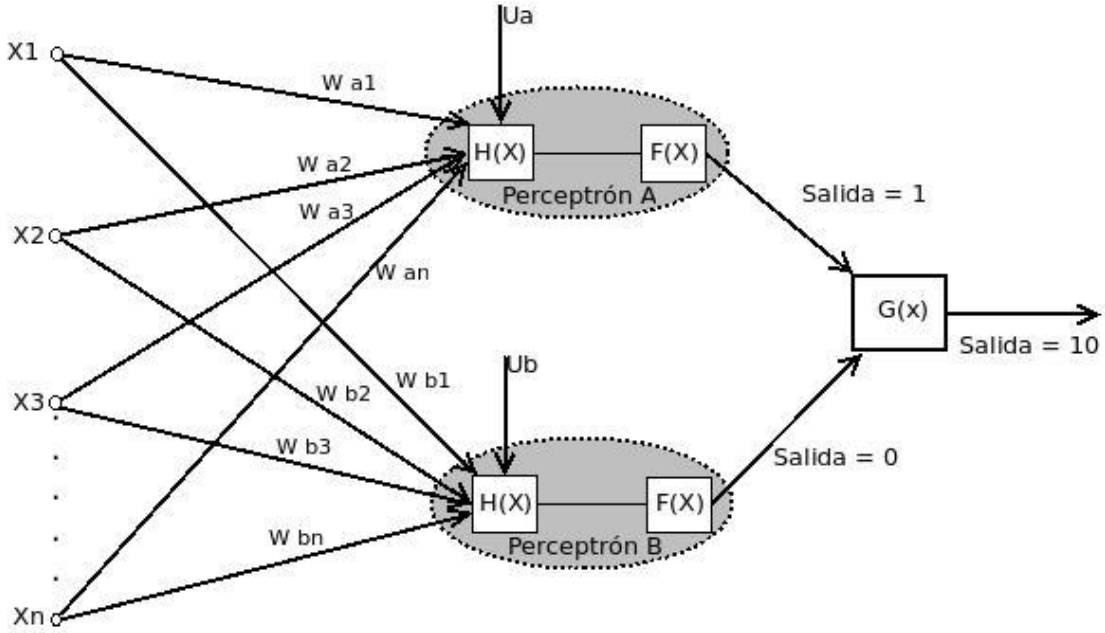


Figura 48. Ejemplo de funcionamiento del clasificador de 2 perceptrones.

3.5.3.2 Clasificador con tres perceptrones simples

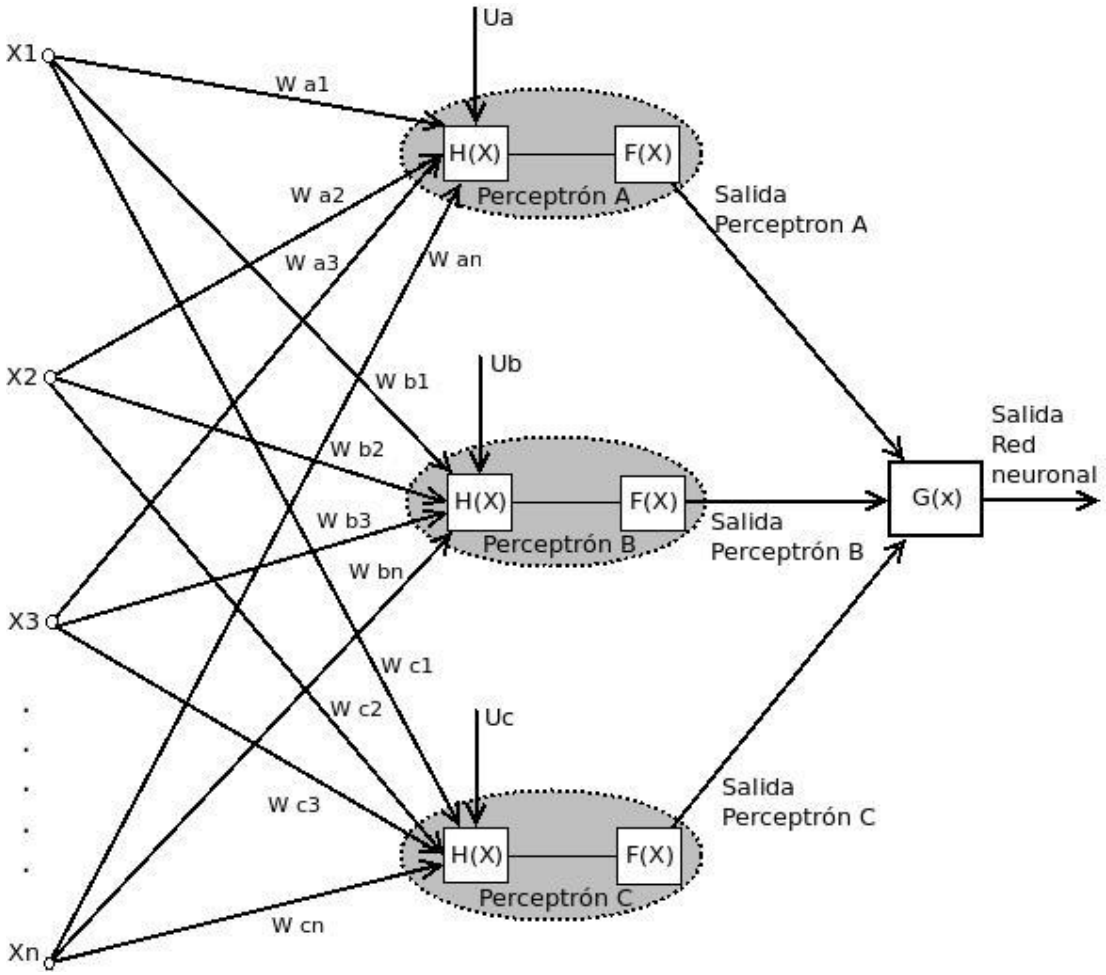


Figura 49. Esquema del clasificador formado por tres perceptrones.

Otro método para crear un clasificador para nuestros tres tipos de datos diferentes es utilizando un perceptrón simple para cada patrón de datos. En este caso, cada perceptrón se encargará de discriminar una clase de datos de las otras dos restantes. Por lo tanto, se tienen las siguientes codificaciones válidas para las distintas clases (001, 010, 100). Sencillamente, si un perceptrón se activa (su salida es 1), significará que los datos de la entrada pertenecen a la clase que discrimina ese perceptrón (siempre y cuando no se active más de uno).

Las restantes salidas posibles (000, 011, 110, 101, 111) no representan ninguna clase y por lo tanto son salidas erróneas que habrá que corregir mediante el proceso de aprendizaje. Estas salidas están dando casos imposibles como que la entrada recibida pertenece a más de una clase diferente o que no pertenece a ninguna.

En la imagen anterior, veíamos el esquema general de este clasificador. En definitiva, como podemos ver guarda muchas similitudes con el clasificador explicado anteriormente.

El funcionamiento de este nuevo clasificador también será similar al del clasificador anterior con algunas pequeñas diferencias ya que ahora en vez de dos perceptrones son tres. El entrenamiento de cada uno de los perceptrones se realizará con la misma regla de aprendizaje Windrow-Hoff explicada en el apartado anterior y la función de entrada y la de activación de cada uno de los perceptrones también será la misma que la explicada en el clasificador anterior y en la sección 2.3.6 El perceptrón simple. Al igual que antes, cada uno de los tres perceptrones de los que disponemos ahora se entrenará con un bit de la solución.

En la siguiente imagen vemos un ejemplo del funcionamiento de esta red, en la que tras recibir la entrada la salida obtenida es “001”.

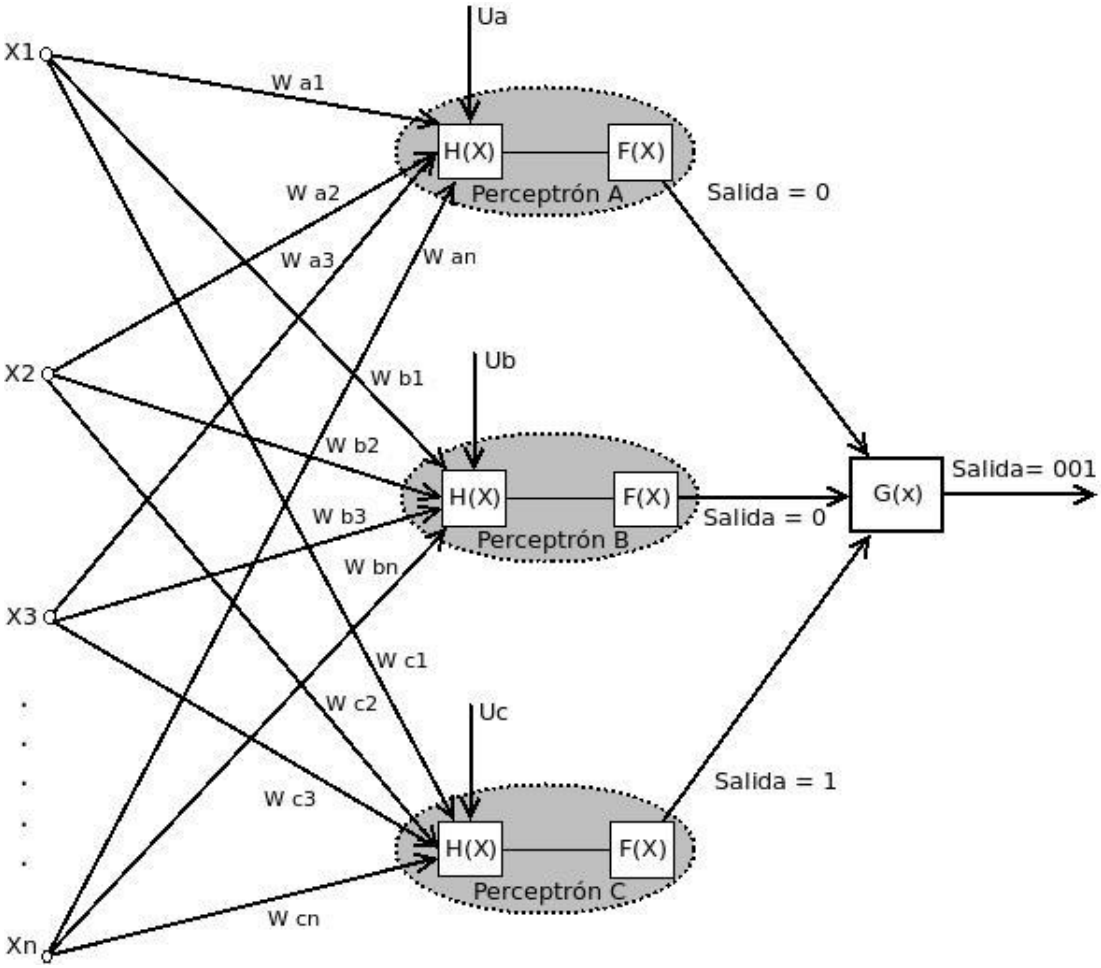


Figura 50. Ejemplo de funcionamiento de la red de tres perceptrones.

Capítulo 4

Experimentación

4.1 Introducción

En este capítulo se detallan y analizan las pruebas llevadas a cabo sobre nuestro sistema. De este modo las diferentes decisiones que tomemos no sólo estarán basadas en nuestros conocimientos, sino también estarán justificadas con los resultados obtenidos de ellas. Además, una vez que hayamos conseguido resolver nuestras dudas sobre la elección del clasificador y sobre otros aspectos menores, realizaremos un conjunto representativo de pruebas con el fin de ver el funcionamiento de toda nuestra solución y los resultados obtenidos.

Por lo tanto, en este capítulo primero realizaremos las pruebas necesarias para determinar cuál de los dos clasificadores explicados anteriormente se adapta y da mejores resultados para nuestro problema concreto. Además también definiremos el número de ciclos de aprendizaje correcto. Posteriormente, elaboraremos un conjunto de pruebas con el fin de ver los resultados que obtiene el algoritmo genético con el clasificador elegido, ver el comportamiento de éste al variar entre los diferentes operadores de cruce, selección, reemplazo, tipo de representación, diferentes tamaños de la población, etc.

4.2 Experimentación previa

Antes de probar la totalidad del algoritmo genético debemos determinar algunos aspectos necesarios para su correcto funcionamiento. Para ello realizaremos unas sencillas pruebas con el fin de intentar ver cuál de los de los dos clasificadores es el que mejor se adapta a nuestro problema, estudiaremos que valor dar a la razón de aprendizaje de los

perceptrones, veremos cuál es el número de operaciones ideal para los individuos, etc. Todos estos aspectos nos ayudarán a mejorar los resultados obtenidos por nuestra solución.

4.2.1 Elección del clasificador

Para comprobar cuál de los distintos clasificadores explicados en el punto anterior se ajustaba mejor al problema de la clasificación de los datos del BCI y ofrecía mejor resultado, se realizaron unas sencillas pruebas con cada uno de ellos variando el número de iteraciones de los perceptrones o lo que es lo mismo, el número de ciclos entrenamiento-validación que explicamos anteriormente.

Para la realización de estas pruebas se escogieron los ficheros obtenidos de las sesiones realizadas por el sujeto 1. El fichero con las sesiones 1 ,2 y 3 se utilizarán durante el entrenamiento de la red o "train" y el fichero con la sesión 4 se usará para comprobar la validez de la red tras la realización de los ciclos de entrenamiento ("test"). Antes de realizarse las pruebas los ficheros de los datos serán completamente desordenados. Dado que sólo se trataba de comprobar la validez de los distintos clasificadores no será necesario realizar ninguna selección de atributos ni se generarán operaciones para realizar la prueba, sino que solamente serán introducidos los 96 atributos de cada medición.

Una vez hechas estas pruebas, decidiremos cuál será el clasificador que utilizaremos en el algoritmo genético a partir del resultado obtenido y a partir de otras variables como por ejemplo el tiempo del cómputo. También veremos cuál es el número óptimo de iteraciones de los perceptrones para cada clasificador. De este modo, no sólo tendremos el clasificador que mejor se adapta a nuestro problema (o que menos porcentaje de error produce al intentar clasificar los datos) de los explicados anteriormente sino que también sabremos algunos de sus parámetros necesarios para su óptima utilización.

4.2.1.1 Prueba del clasificador de 2 perceptrones

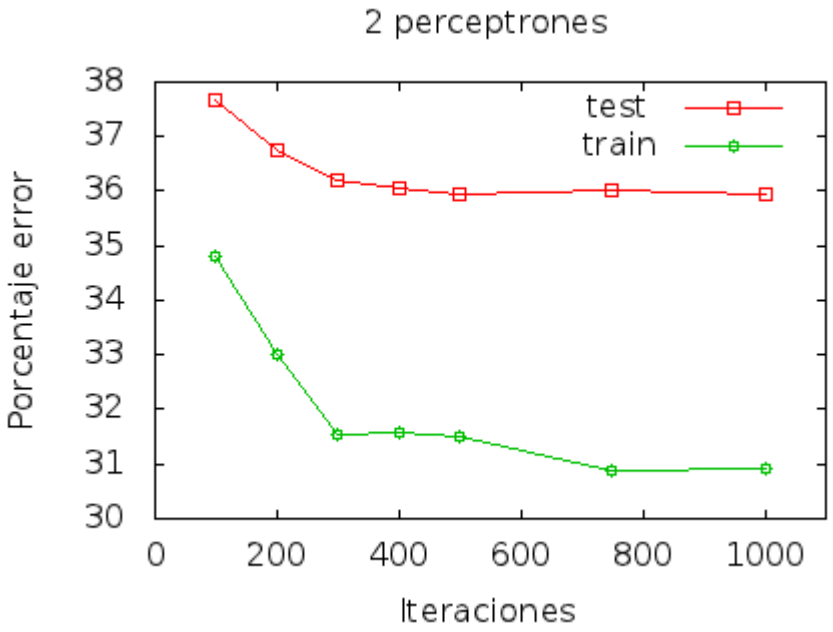
A continuación analizaremos el funcionamiento del clasificador explicado en la sección 3.5.3.1 Clasificador con dos perceptrones simples. Los valores de las gráficas y tablas que veremos a continuación no son los resultados de una sola prueba, sino que son la media matemática de los resultados obtenidos en 100 pruebas independientes entre sí (100 ejecuciones del clasificador). Con ello tratamos de evitar efectos raros que podrían producirse, ya que si sólo lanzásemos una ejecución para cada caso

podría resultar que ésta diera unos resultados anormalmente bajos o altos por causas del azar o por cualquier otro factor. Al tratarse de la media de 100 de pruebas obtenemos una visión más fiable del comportamiento del clasificador.

En la siguiente tabla y en la siguiente gráfica veremos los resultados obtenidos variando el número de iteraciones del ciclo de aprendizaje para nuestro clasificador de dos perceptrones. Distinguiremos los resultados entre los obtenidos en la fase de "Train" o de entrenamiento de la red y entre la fase de "Test" o comprobación del funcionamiento:

Iteraciones	Porcentaje del error producido fase "Train"	Porcentaje del error producido fase "Test"
100	34.791027	37.678368
200	33.002178	36.744862
300	31.532574	36.212330
400	31.583868	36.068783
500	32.024410	36.548233
750	30.884686	36.001709
1000	30.912518	35.929508

Tabla 9. Resultados obtenidos del clasificador 2 perceptrones.



Gráfica 1. Resultados clasificador 2 perceptrones.

Como se puede observar en los resultados a mayor número de iteraciones o ciclos de aprendizaje parece ser que salvo alguna excepción, los errores producidos tienden a ir disminuyendo.

A continuación vemos el tiempo de ejecución del clasificador que se incrementa cuanto mayor sea el número de ciclos de aprendizaje.

Iteraciones	Tiempo (segundos)
100	3,29
200	6,16
300	9,01
400	11,91
500	14,75
750	21,84
1000	28,91

Tabla 10. Clasificador de 2 perceptrones (Iteraciones/Tiempo).

Las tareas que realiza en ese tiempo son: lectura de todos los registros del fichero, preparación de los datos (selecciona las señales a usar) y la ejecución del clasificador en modo 'train' las iteraciones indicadas.

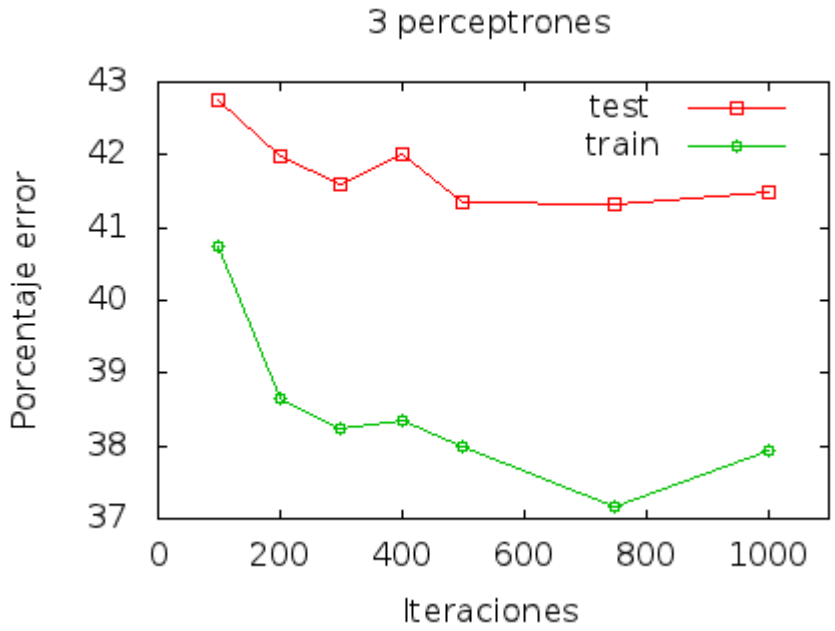
4.2.1.2 Prueba del clasificador de 3 perceptrones

En cuanto al clasificador explicado en la sección 3.5.3.2 Clasificador con tres perceptrones simples, los resultados que hemos obtenido son los siguientes: (al igual que en el caso anterior los valores de las gráficas y tablas son la media de 100 ejecuciones diferentes). Variando el número de iteraciones del ciclo de aprendizaje:

Iteraciones	Porcentaje del error producido fase "Train"	Porcentaje del error producido fase "Test"
100	40.738888	42.739727
200	38.654922	41.976894
300	38.234512	41.603584
400	38.354862	42.010845
500	37.980343	41.354454
750	37.162422	41.307934
1000	37.948997	41.495724

Tabla 11. Resultados obtenidos del clasificador 3 perceptrones.

Como podemos ver en la siguiente gráfica:



Gráfica 2. Resultados clasificador 3 perceptrones.

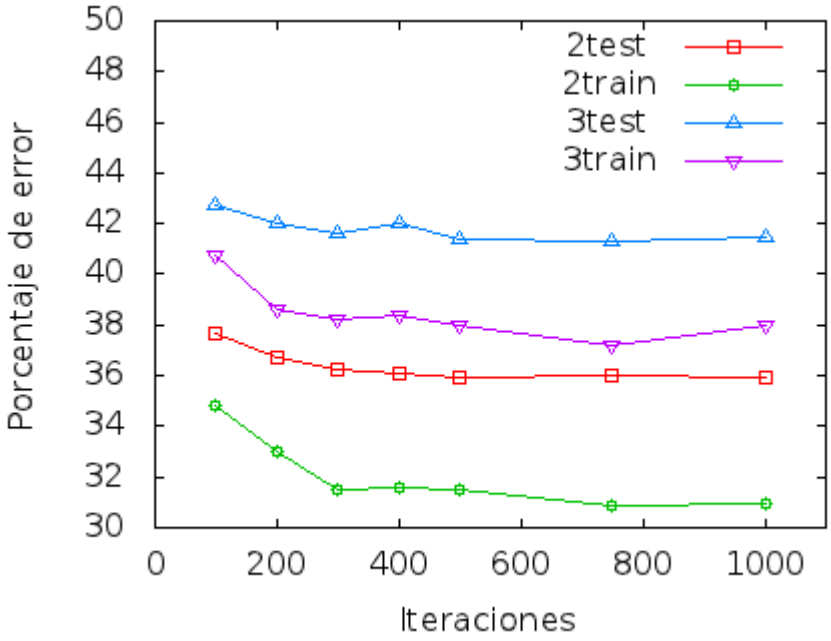
Y en la siguiente tabla vemos el tiempo de ejecución dependiendo el número de iteraciones:

Iteraciones	Tiempo (segundos)
100	4,69
200	8,96
300	13,14
400	17,35
500	21,49
750	32,14
1000	42,73

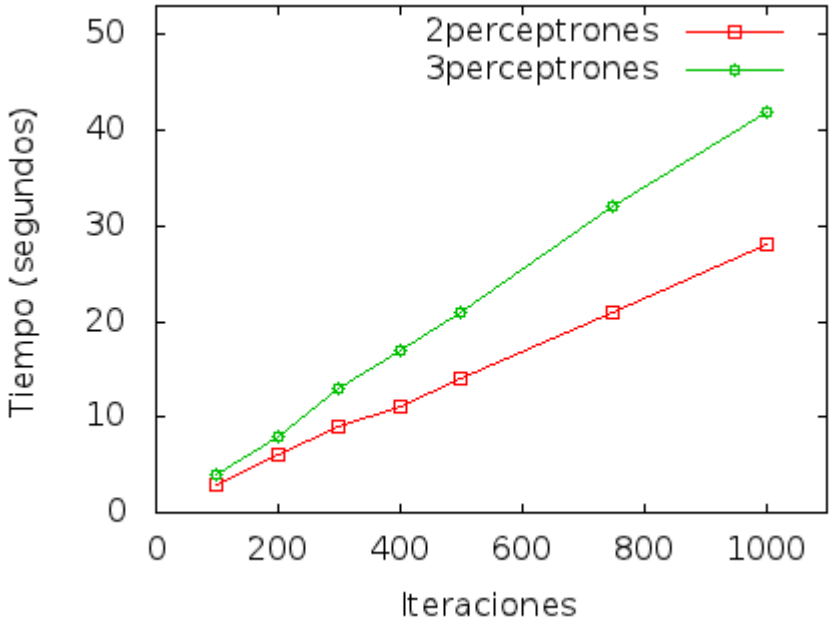
Tabla 12. Clasificador de 3 perceptrones (Iteraciones/Tiempo).

4.2.1.3 Clasificador elegido y número de ciclos de aprendizaje

En las dos siguientes gráficas vemos el comportamiento de ambos clasificadores de forma conjunta:



Gráfica 3. Comparación entre los dos clasificadores (Error/Iteraciones)



Gráfica 4. Comparación entre los dos clasificadores (Tiempo/Iteraciones).

Como vemos en la Gráfica 3, de los dos clasificadores analizados el que mejor resultado da en cuanto a porcentajes de error es el clasificador de dos perceptrones. Independientemente del número de ciclos de aprendizaje o de iteraciones que usemos este clasificador será siempre mejor, tanto para la fase de “train” como para el “test” (entre un 7% en algunos casos mejor para train y un 5-7% mejor para test). Además de ello, en la Gráfica 4 vemos como el clasificador de dos perceptrones también consigue ser mejor en cuanto a tiempo de ejecución. Esto último era lo esperado ya que tiene un

perceptrón menos que el otro y por lo tanto el clasificador realizará menos operaciones internas.

Por ello elegimos el clasificador de dos perceptrones como clasificador para nuestro algoritmo genético. A partir de ahora será éste el que utilizaremos para todas y cada una de las pruebas. Además, utilizaremos este clasificador con el número de ciclos de aprendizaje o iteraciones de 300, ya que si vemos los resultados obtenidos antes en 4.2.1.1 Prueba del clasificador de 2 perceptrones o en la Gráfica 3, un número de iteraciones menor a éste nos dará resultados en cuanto al porcentaje de error peores (3% de error más para "train" y alrededor del 1,5% más para "test"). En cambio si aumentamos el número de iteraciones más allá de 300 obtendremos porcentajes de error que son sólo ligeramente inferiores. Dado que está disminución del error no es muy significativa, (0,65% mejor en "train", 0,29% mejor en "test") y para conseguir esta reducción es necesario aumentar el número de iteraciones a 750 o 1000 y esto conlleva que se triplique el tiempo de ejecución optamos por dejar el número de iteraciones a 300 ya que no merece la pena triplicar el tiempo de ejecución para obtener esa ligerísima mejora.

4.2.2 Razón de aprendizaje para el clasificador

Otro aspecto que podemos regular del clasificador es la razón de aprendizaje de los perceptrones. Como vimos en "2.3.6.2 Razón de aprendizaje en el perceptrón simple", la razón de aprendizaje era un parámetro de la regla de aprendizaje de los perceptrones y su fin es que los cambios realizados en los pesos y en el umbral de los perceptrones sean menos bruscos consiguiendo que el hiperplano discriminante no oscile demasiado durante el período de adaptación de los perceptrones. Este parámetro podía variar entre valores cercanos a 0 hasta 1. Para nuestro caso concreto, realizamos diversas pruebas con el fin de ver la forma en la que afectaba este parámetro al resultado pero los resultados obtenidos no eran notorios.

4.2.3 Operaciones

Otro aspecto que podemos tratar de analizar es el error producido por la red al variar entre las diferentes opciones que nos ofrecen las operaciones. De este modo podremos determinar si el uso de éstas nos será de ayuda en la clasificación, trataremos de determinar cuál es el número óptimo de operaciones que deberían contener los individuos, veremos sí

existe diferencias en el comportamiento dependiendo del tipo de operación de que se trate, etc.

Por todo ello, aquí explicamos brevemente el comportamiento que hemos observado de todo nuestro programa (clasificador y algoritmo genético) al variar el número o el tipo de las operaciones con el fin de que sirva de ayuda en pruebas posteriores.

En lo que se refiere a individuos formados por atributos y un conjunto de operaciones de diferentes tipos, los resultados en la ronda inicial son ligeramente superiores a los obtenidos en individuos en los que todas las operaciones eran del mismo tipo (salvo en el caso de las divisiones). Esta diferencia es muy notable en muchos casos, pero otro aspecto a tener muy en cuenta es que según avanza la ejecución del algoritmo, éste consigue en varias generaciones contrarrestar en cierta medida este efecto ya que los operadores de selección y reemplazo consiguen encontrar las mejores operaciones en varias rondas disminuyendo el error hasta porcentajes más normales, si bien sigue siendo algo más alto.

Dado que es imposible probar todas las configuraciones posibles del algoritmo genético, a continuación veremos los resultados de un sencillo ejemplo donde se pueden observar estas diferencias que hemos comentado. Se trata de una prueba con una población 15 individuos, 30 rondas en las que los individuos tienen 25 operaciones cada uno:

Tipo de operaciones	Generación inicial	Generación 15	Generación 30
Suma	33.138931	27.916666	27.618416
Resta	31.572947	27.251143	27.149820
Multiplicación	38.045216	28.289639	28.138250
División	46.232273	33.167427	31.902231
Mezcla de tipos	37.943897	29.227457	28.579027

Tabla 13. Algoritmo genético variando el tipo de operaciones.

En definitiva, si bien es cierto que introduciendo un conjunto de operaciones de diferentes tipos en los individuos éstos tienen un porcentaje de error mayor, el algoritmo genético consigue ir encontrando las mejores operaciones disminuyendo el error hasta resultados aceptables. Aún así se observa que individuos cuyas todas operaciones son sumas, restas o multiplicaciones consiguen mejores resultados en las primeras generaciones debido posiblemente a que al tener todos las operaciones del mismo tipo los resultados son similares y la entrada de la red es mucho más uniforme, al contrario de lo que ocurre cuando mezclamos tipos.

Otro efecto que se observa es que los individuos en los que todas las operaciones son iguales (sumas y restas sobre todo) es que sus resultados en todas las ejecuciones no varían mucho y desde la población inicial se obtienen resultados relativamente buenos, en cambio, al hacer que las operaciones de los individuos sean de diferentes tipos se observa como los resultados en la generación inicial son muy diferentes entre sí, pero según pasan las generaciones en casi todas las ocasiones los resultados tienen a igualarse y disminuir.

En cuanto al número de operaciones óptimo, al entrar el algoritmo genético y sus operadores en juego desaparece la tendencia de aumentar el error según aumenta el número de operaciones y se ha observado que con individuos con un conjunto de operaciones mixto (de diferentes tipos), el número óptimo está sobre las 20 operaciones, llegando a tener resultados iguales o incluso sensiblemente mejores que si no empleásemos ninguna operación, cosa que no habíamos observado hasta ahora. Más allá de 40 operaciones el error aumenta y además aumenta el tiempo de ejecución al haber más entradas en el clasificador. En cambio, se observa que en los casos en los que todas las operaciones son sumas o restas, se producen muy buenos resultados independientemente del número de operaciones escogido. A continuación vemos un ejemplo de una misma ejecución con 100 operaciones en la que variamos el tipo:

Tipo de operaciones	Generación inicial	Generación 15	Generación 30
100 Sumas	35.842831	28.211121	26.933204
100 restas	30.176672	26.708406	26.724239
100 multiplicaciones	42.915398	32.760891	31.445669
100 divisiones	41.008739	31.110056	29.391460
Mezcla de tipos (100)	42.739361	32.733028	32.771023

Tabla 14. Ejecución de individuos con 100 operaciones variando el tipo.

4.2.3.1 Conclusiones finales sobre las operaciones

Por lo tanto, y tras ver todo lo anterior las conclusiones a las que hemos llegado son que nuestra primera idea de que los individuos contuvieran operaciones de cualquier tipo sin ningún tipo de control ajeno eligiendo los tipos de operaciones en igualdad de condiciones parece no ser la más beneficiosa (por lo menos en lo visto hasta ahora en ejecuciones con pocas generaciones) ya que hace comenzar a los individuos con un error superior, si bien este error acaba reduciéndose al cabo de las generaciones.

Aún así no dejaremos de lado esta forma de crear las operaciones y trataremos de ver si esto se sigue observando para casos con poblaciones mayores y un mayor número de generaciones, dado que las pruebas que hemos realizado hasta ahora no cubren todas las opciones posibles (cosa casi imposible). Tampoco nos olvidaremos de crear individuos variando el porcentaje de cada tipo de operación o desechando las divisiones (que parecen ser las más perjudiciales).

Otra conclusión a la que llegamos es que parece conveniente usar un número de operaciones entre 15-30. Este parece ser el número de operaciones óptimo para nuestro caso. Nos hemos dado cuenta que al realizar ejecuciones con el algoritmo genético al completo, un número de entre 15-30 operaciones hace que consigamos porcentajes de error sobre el 26%, cosa que también se produce usando solamente atributos pero en menor medida. Si aumentamos el número de operaciones más allá de 30, el error aumenta ligeramente además de aumentar el número de entradas a la red con el consiguiente aumento de tiempo de ejecución. Caso especial era el de los individuos formados solamente por sumas o restas ya que parecían devolver el mismo error independientemente del número de operaciones.

4.3 Pruebas generales del algoritmo genético

A continuación realizaremos un conjunto de pruebas lo más extenso posible con el fin de intentar ver cuáles son los resultados obtenidos por nuestro algoritmo y a su vez determinar que operadores genéticos y cuáles son las condiciones que más se adaptan al problema y por lo tanto que mejores resultados da. Comenzaremos con pruebas sencillas e iremos poco a poco aumentando la complejidad. En cada una de estas pruebas, mostraremos una tabla con la configuración del algoritmo que hemos usado, las gráficas que creamos necesarias así como pequeñas explicaciones cuando lo creamos necesario. Cada una de estas pruebas evidentemente ha sido repetida 10 veces con el fin de analizar la tendencia que se producida y evitarnos resultados obtenidos fruto de la casualidad. Todas estas pruebas han sido hechas usando los dos siguientes ficheros de datos:

Fichero train= train_test1_subject1_psd010203.arff (10528 registros)
Fichero test= test2_subject1_psd04.arff (3504 registros)

Como ya indicamos anteriormente usaremos el clasificador formado por dos perceptrones simples en todas las ocasiones y un número de operaciones cercano a 20 para la mayoría de los experimentos.

Dadas las características de nuestro sistema y por todo lo explicado anteriormente, nos centraremos más en realizar pruebas usando la representación por atributos y la función de fitness 1 (la cual sólo evaluaba el error producido en la red dejando a un lado el número de entradas), dado que esa era la configuración con la que pensábamos afrontar el problema desde el inicio y además es la que creemos que nos traerá mejores resultados. Tras esto, también añadiremos un bloque de pruebas con la representación por canales con el fin de valorar todas las opciones y ver si realmente estamos en lo cierto y esta representación no nos ofrece unos resultados tan válidos como la otra. Por último incluiremos un conjunto de pruebas usando la función de fitness 2 (disminución del número de entradas).

4.3.1 Pruebas usando la representación por atributos

Todas las pruebas que realicemos a continuación en este bloque tienen en común que usan la representación por atributos o genes por la cual los individuos estarán formados por 96 genes que indicarán los atributos escogidos más otro conjunto de genes que codificarán las diferentes operaciones.

Como ya hemos explicado antes, cada una de estas pruebas que veremos a continuación ha sido ejecutada 10 veces con el fin de poder valorar mejor el comportamiento de éstas. Las gráficas que aparecen en todas las pruebas, salvo que se indique lo contrario, muestran una ejecución concreta y nos las 10 a la vez. Esto se ha hecho así para facilitar la compresión de la gráfica. Independientemente de esto, siempre hemos buscado los casos que mejor representaban a la totalidad de las pruebas con el fin de reflejar el verdadero comportamiento del algoritmo.

4.3.1.1 Prueba 1

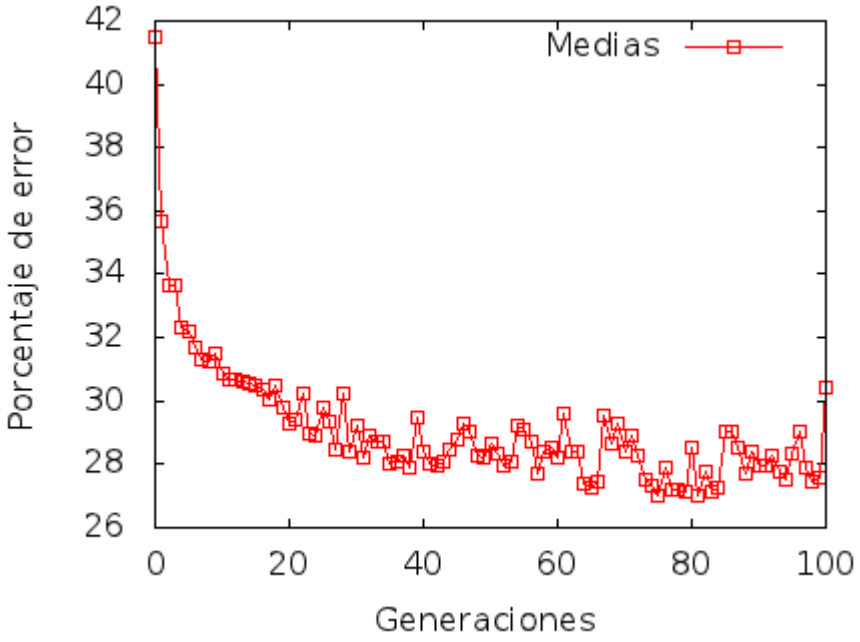
En esta pequeña prueba se puede observar como generación tras generación la población va disminuyendo el error obtenido y a la vez también aumenta ligeramente el porcentaje de entradas a la red. Este efecto se observa en todas las ejecuciones que hemos realizado de esta prueba. Otro efecto observado es que la población tiende a concentrarse y a converger, en algunos casos llegando a ser preocupante ya que alrededor del 50% de la población acaba siendo igual. Esto es debido a los operadores genéticos elegidos así como al pequeño tamaño de la población.

La configuración de esta prueba es la siguiente:

CONFIGURACIÓN
Tamaño de la población = 12 individuos
Número de operaciones por individuo= 15
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección jerárquica • Cruce uniforme y cruce de operaciones uniforme Probabilidad de cruce = 0.30 Factor de cruce = 0.15 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.30 Factor de mutación = 0.15 Factor de mutación operación = 0.15 • Inversión: Probabilidad de inversión = 0.15 • Reemplazo estacionario. Hijos: 90 Padres: 10
Función fitness 1. Solo participa el error de la red

Tabla 15. Configuración prueba 1.

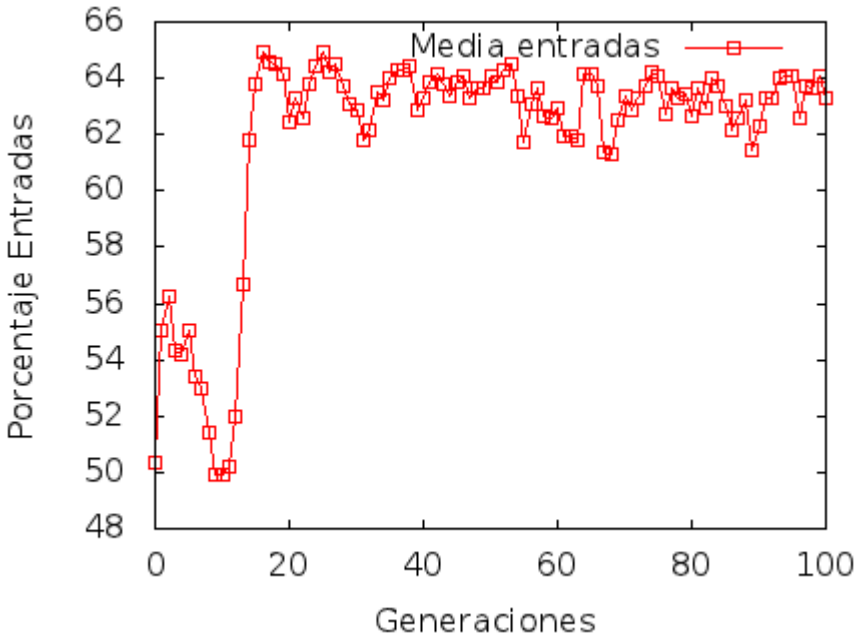
Se observa que en todos los casos, la generación 100 ronda sobre el 26,9%-28% de error de entrenamiento, por lo que se ha conseguido reducir el porcentaje de error significativamente con respecto a la población inicial de la que partíamos.



Gráfica 5. Evolución del porcentaje de error (Prueba 1).

En la gráfica podemos ver la media de error producido en cada una de las generaciones. Se observa una disminución importante en las primeras generaciones y a continuación se estabiliza y disminuye más ligeramente. A partir de la generación 20 al algoritmo le cuesta encontrar individuos mejor preparados y por lo tanto la media disminuye más lentamente. En la mayoría de los casos la diferencia entre los mejores individuos de la generación 100 y de la generación 25 ó 30 es de sólo un 1% de error de mejora.

Y como ya dijimos, la función de fitness 1 no controla el número de entradas a la red. Al no controlarse este factor tiende a aumentar llegando a rondar el 64% de atributos en los individuos.



Gráfica 6. Evolución del porcentaje de entradas (Prueba 1).

En cuanto a los resultados del test, éstos varían entre el 29% al 33%. Este error de test siempre es superior al error producido durante el entrenamiento.

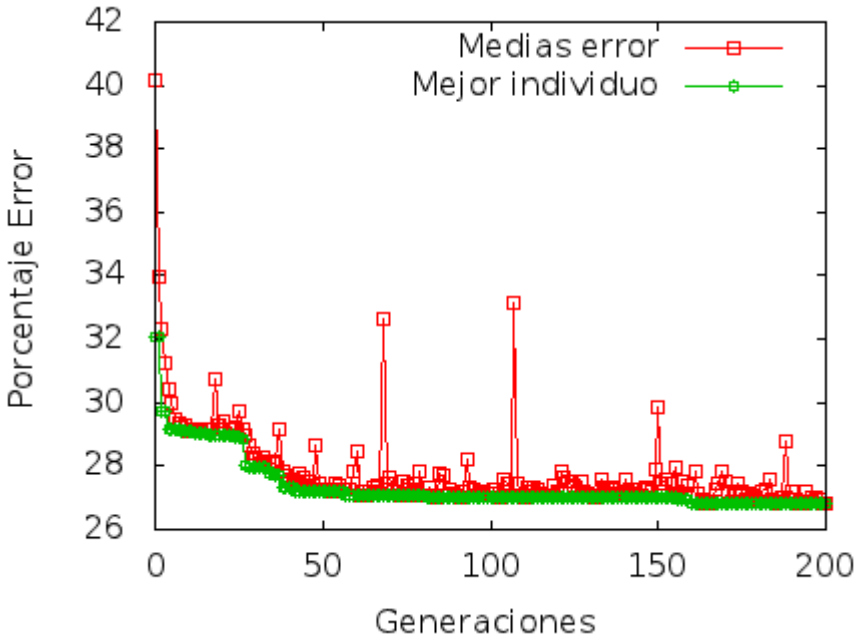
4.3.1.2 Prueba 2

CONFIGURACIÓN
Tamaño de la población = 12 individuos
Número de operaciones por individuo= 15
Número de rondas del algoritmo = 200
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección jerárquica • Cruce uniforme y cruce de operaciones uniforme Probabilidad de cruce = 0.30 Factor de cruce = 0.15 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.30 Factor de mutación = 0.15 Factor de mutación operación = 0.15 • Inversión: Probabilidad de inversión = 0.18 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 16 Padres elegidos al azar: 0 • Mejores Hijos: 75 Hijos elegidos al azar: 9
Función fitness 1. Solo participa el error de la red

Tabla 16. Configuración prueba 2.

Esta prueba es similar a la anterior (prueba 1) pero con algunos ligeros cambios. El cambio realizado más importante es el aumento del número de rondas del algoritmo con el fin de ver si al doblar el número de rondas se conseguía reducir el error aún más. En ninguna de las ejecuciones se ha conseguido que éste disminuya más de lo que lo hizo anteriormente, más bien lo que se ha conseguido es que el algoritmo, al llegar a porcentajes de error sobre el 26,5%- 27%, se estanque y no encuentre la forma de seguir disminuyendo el error por lo que la población acaba convergiendo aún más. Si se observa la siguiente gráfica, podemos ver que el mejor individuo encontrado en la ronda 50 es el mismo que el de la ronda 150 por lo que el algoritmo no avanza prácticamente nada salvo unas centésimas. Alrededor de la ronda 160 hay un pequeño salto y aparece un individuo un 0,5% mejor. Además si observamos ambas líneas de la gráfica (mejor individuo y media) nos damos cuenta que la cercanía que hay entre ambas tiene que ser debido a que la totalidad de la población tiene que tener la misma valoración o ser muy similar lo que nos hace pensar que la población no está evolucionando y es necesario aumentar las probabilidades de cruce, aumentar la población o utilizar un método selectivo menos agresivo.

En otras ejecuciones realizadas se ha observado casos similares a éste y nunca hemos conseguido individuos mejores a 26,5%, siempre el algoritmo al acercarse a esa cifra la disminución se produce de centésimas a centésimas en el caso de que llegue a producirse.



Gráfica 7. Media de error y mejor individuo en cada ronda (Prueba 2).

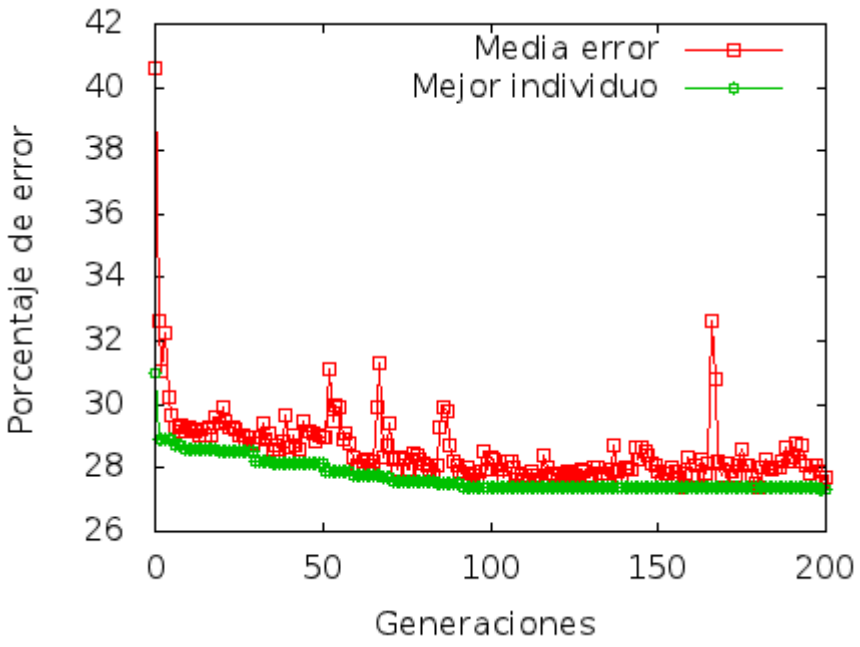
Otro cambio realizado con respecto a la prueba anterior es la realización de un reemplazo estacionario pero introduciendo elementos aleatorios. Aún así la población sigue tendiendo a converger, quizás también por culpa del mayor número de rondas, por una baja probabilidad de cruce,

un método de selección que prima fuertemente a los mejor preparados y el pequeño tamaño de la población que manejamos por el momento.

Los resultados obtenidos en el test son similares al caso anterior. Hay algún caso en el que aparece un individuo con un 29% de error en el test pero por lo general los individuos tienen un porcentaje de error en el test superior al 30%. Se observa también que el individuo que mejor valoración tiene en el entrenamiento no tiene porque ser el mejor en el test. El porcentaje de entradas simples (atributos) aumenta hasta el 55%-60%.

Viendo los resultados anteriores, volvimos a repetir esta prueba sustituyendo la selección jerárquica por una selección por ruleta con el fin de ver cómo afectaba la modificación. Los resultados obtenidos tras este cambio son similares a los que habíamos obtenido. Después del descenso drástico al inicio del error, a continuación se estabiliza y se va reduciendo poco a poco. Eso sí, tras el cambio, se observa que la población converge más lentamente. Los porcentajes de error en el test son similares a los de la prueba anterior.

En la siguiente gráfica donde se muestra la media de error de cada ronda, se puede observar que hay mayor variabilidad de la media que antes debido a que no converge tanto, además la distancia entre el mejor individuo y la media de los individuos es algo ligeramente mayor lo que indica que la población no tiende a parecerse tanto y hay mayores diferencias, aunque siguen siendo pocas.



Gráfica 8. Media de error y mejor individuo tras el cambio (Prueba 2).

Independientemente de lo anterior vemos como no se producen grandes avances y la ronda 200 sigue obteniendo valores similares a la

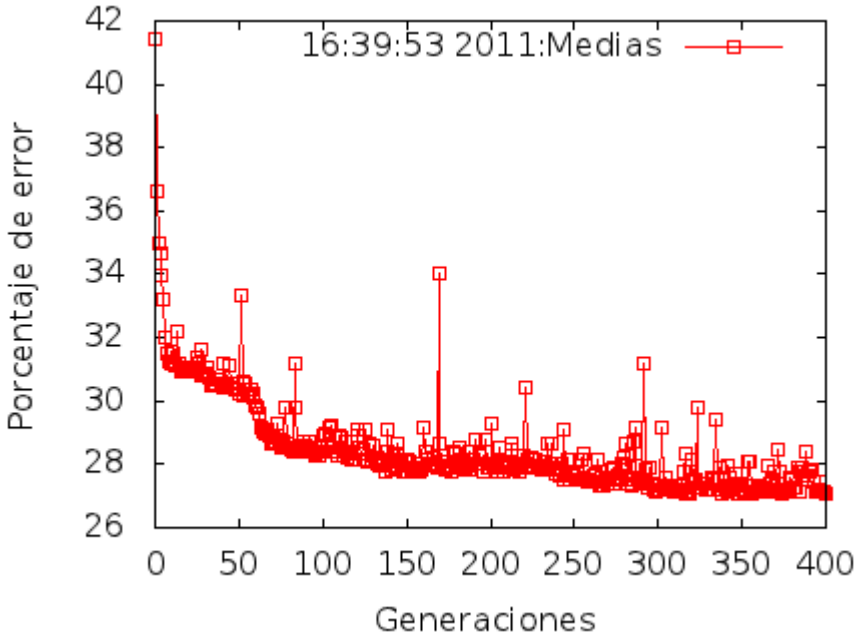
ronda 100. Llegando a valores cercanos a 27% la población se estanca y no se consiguen mejoras.

Los resultados del test realizado a la última generación (ronda 200), dan unos resultados que varían entre 29,7% y el 31,5%. En todos los casos, el error es superior al que se producía en el entrenamiento.

4.3.1.3 Prueba 3

CONFIGURACIÓN
Tamaño de la población = 12 individuos
Número de operaciones por individuo= 15
Número de rondas del algoritmo = 400
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección jerárquica. • Cruce uniforme y cruce de operaciones uniforme Probabilidad de cruce = 0.30 Factor de cruce = 0.15 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.30 Factor de mutación = 0.15 Factor de mutación operación = 0.15 • Inversión: Probabilidad de inversión = 0.18 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 16 Padres elegidos al azar: 0 Mejores Hijos: 75 Hijos elegidos al azar: 9
Función fitness 1. Sólo participa el error de la red.

Tabla 17. Configuración prueba 3.



Gráfica 9. Evolución de la media de error (Prueba 3).

Tras los resultados de la prueba anterior decidimos repetirla aumentando el número de rondas a 400 con el fin de ver si se consigue mejorar los resultados al aumentar las rondas. Pero como podemos observar, los resultados obtenidos son similares a los conseguidos en la prueba 2 y parece que este mayor número de rondas no nos hace conseguir mejores resultados. Se sigue estancando al llegar a valores cercanos a 26,5%-27% de error en el entrenamiento y no consigue mejorar esos porcentajes, con lo que nos podemos llegar a pensar que este es el límite de error del clasificador y no conseguiremos disminuirlo. En cuanto a los resultados del test, siguen siendo similares a los anteriores con valores sobre el 30%.

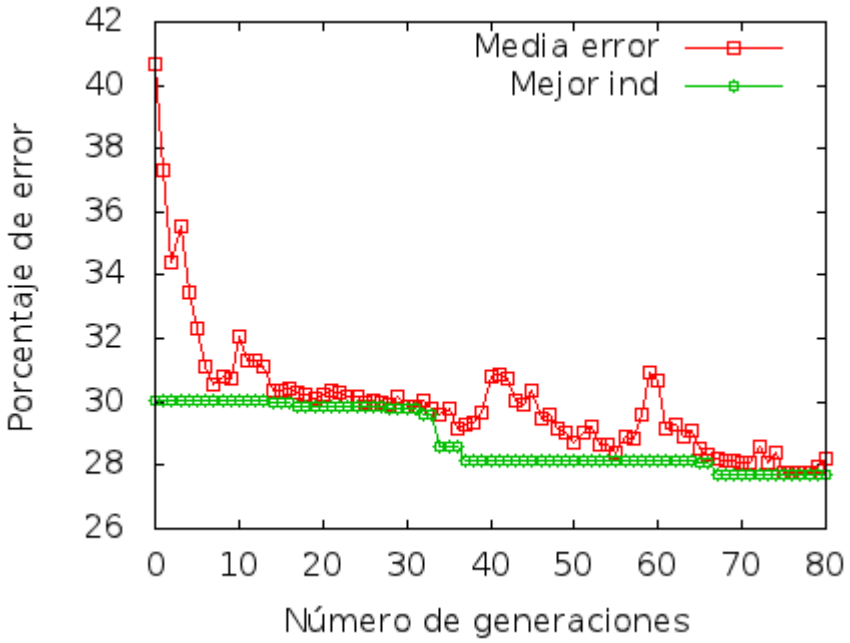
Y, al igual que antes, también cambiamos el método de selección por una selección por ruleta, repetimos una vez más la prueba (con sus 10 ejecuciones) y observamos que seguíamos obteniendo los mismos resultados que en la prueba 2 independientemente de que hayamos doblado el número de rondas. Es más, este aumento de rondas supuso que la población se concentrase aún más en un punto sin mejorar sus resultados. Por lo tanto, en ambos casos podemos determinar que un aumento de rondas no supone que el algoritmo siga disminuyendo el error sino que a simple vista parece que esta disminución se estanca al llegar a valores cercanos al 27% de error en el entrenamiento independientemente del número de rondas que queden por ejecutar.

4.3.1.4 Prueba 4

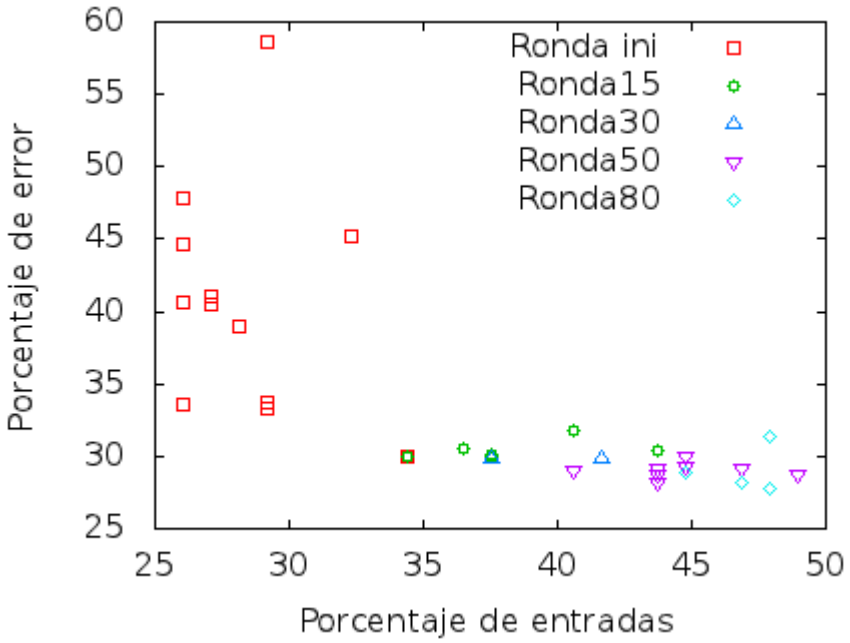
CONFIGURACIÓN
Tamaño de la población = 12 individuos
Número de operaciones por individuo= 20
Número de rondas del algoritmo = 80
Porcentaje de atributos con los que comienza la población= 0.30
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por el método de la ruleta. • Cruce de un punto y cruce de operaciones uniforme Probabilidad de cruce = 0.60 Factor de cruce operación = 0.40 • Mutación y mutación de operaciones Probabilidad de mutación = 0.20 Factor de mutación = 0.10 Factor de mutación operación = 0.25 • Reemplazo estacionario: Hijos: 90 Padres: 10
Función fitness 1. Solo participa el error de la red

Tabla 18. Configuración prueba 4.

El principal cambio realizado en esta prueba con respecto a las anteriores es que en esta hemos inicializado la población al 0,3 lo que indica que los individuos partirán con un menor número de atributos. Los resultados del entrenamiento son los siguientes:



Gráfica 10. Mejor individuo y media error en cada ronda (Prueba 4).



Gráfica 11. Evolución de la población (Prueba 4).

Esta gráfica nos sirve para poder observar como independientemente del porcentaje de entradas con el que fue inicializado la población está tiende a aumentarlo en su búsqueda de los mejores individuos.

Una vez más vemos como el algoritmo consigue reducir el porcentaje de error y va buscando mejores individuos ronda a ronda. La diferencia con respecto a otras pruebas es que los porcentajes de error con los que comenzábamos eran peores a los de otras pruebas dado el menor número de atributos. Esta población inicial ha condicionado al algoritmo haciendo

que le costara más que en otras ocasiones conseguir individuos por debajo del 30% de error en el entrenamiento.

Otro aspecto a tener en cuenta es que aunque iniciásemos el porcentaje de entradas al 30%, en todas las ejecuciones de esta prueba ha habido un aumento significativo de éste hasta rondar el 50% e incluso superarlo, debido a que en casi todas las ocasiones siempre los individuos con mayor porcentaje de entradas tenían un porcentaje de error menor. De hecho, en la mayoría de ellas se ha observado que no han empezado a aparecer individuos con un porcentaje de error menor al 30% si no llegaban a tener un porcentaje de entradas cercanas al 50%. A pesar de todo, en esta prueba los resultados obtenidos han sido ligeramente superiores a los anteriores ya que los mejores individuos obtenidos han rondado el 28% en el entrenamiento y en cuanto al test han obtenido resultados cercanos al 32%-34%.

Otro aspecto que se observa es que hay mayor variación de los resultados entre las 10 ejecuciones distintas que hemos realizado de esta prueba. En algunas ejecuciones la población se ha mantenido con un 40% de entradas y un error sobre el 30%. Sólo aquellas ejecuciones que han ido evolucionando y han conseguido que sus individuos obtengan un porcentaje de entradas sobre el 50% obtienen porcentajes de error cercanos al 28% en el entrenamiento.

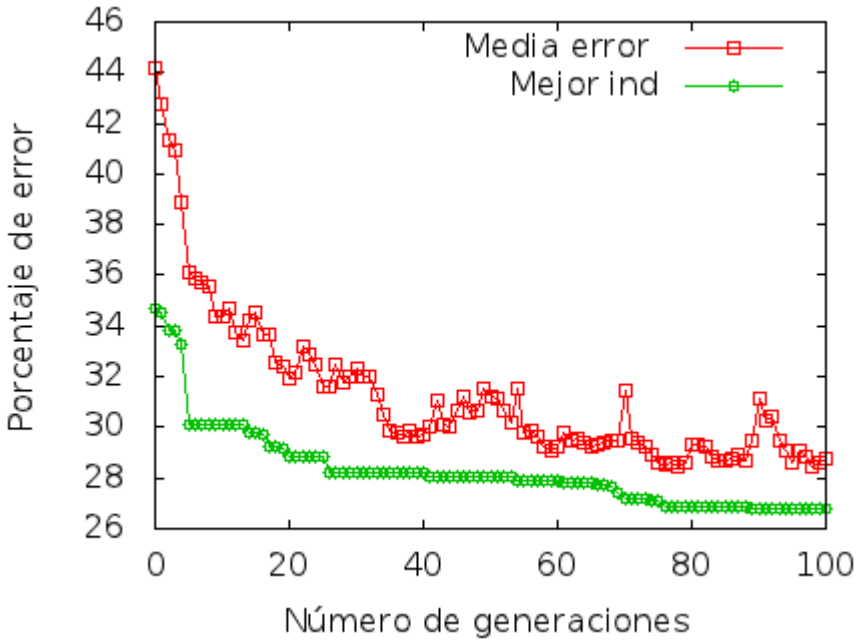
Además hicimos pruebas cambiando la razón de aprendizaje del clasificador a 0,1. Independientemente de este cambio los resultados obtenidos son muy parecidos por no decir iguales que los de la prueba anterior.

4.3.1.5 Prueba 5

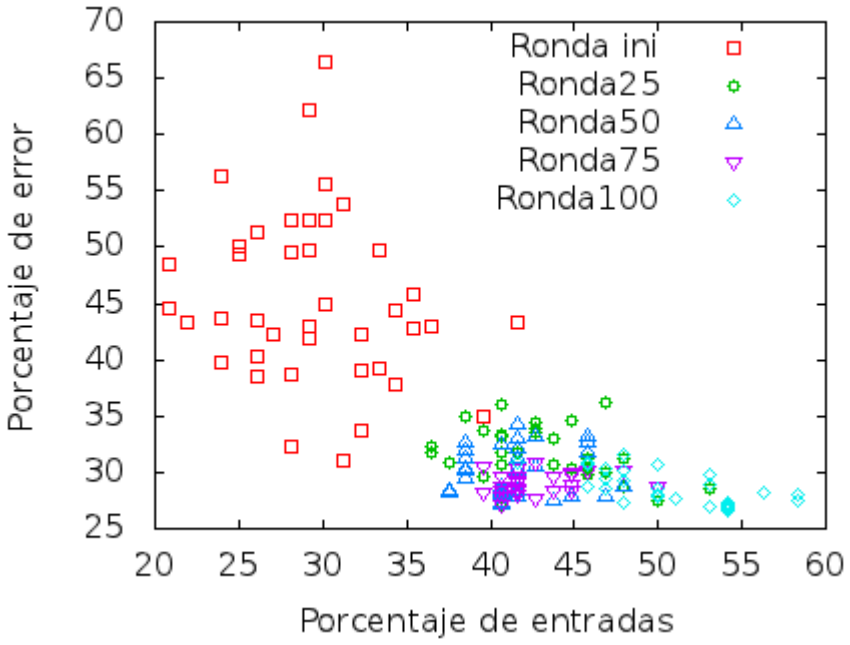
CONFIGURACIÓN
Tamaño de la población = 40 individuos
Número de operaciones por individuo= 20
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.30
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por el método de la ruleta. • Cruce de un punto y cruce de operaciones uniforme Probabilidad de cruce = 0.60 Factor de cruce operación = 0.40 • Mutación y mutación de operaciones Probabilidad de mutación = 0.20 Factor de mutación = 0.10 Factor de mutación operación = 0.25 • Reemplazo estacionario. Hijos: 90 Padres: 10
Función fitness 1. Solo participa el error de la red
Clasificador: Razón= 0.1, Sessions= 300.

Tabla 19. Configuración prueba 5.

Esta prueba que mantiene la misma configuración que la prueba 4 salvo por el incremento significativo de la población hasta los 40 individuos y el aumento del número de rondas hasta 100. Con ello queremos ver si se siguen observamos los mismos efectos anteriores.



Gráfica 12. Mejor individuo y media de error en cada ronda (Prueba 5).



Gráfica 13. Evolución de la población (Prueba 5).
(Muestra el aumento del porcentaje de entradas según avanzan las rondas)

Estos cambios realizados no han conseguido cambiar mucho lo que vimos en la prueba anterior. Seguimos viendo como la población inicial parte de con un porcentaje de error alto (sobre el 34% los mejores individuos) y

ronda a ronda va disminuyendo hasta el 26%-27%. Esta disminución de error va acompañada de un aumento del porcentaje de entradas. Si la población no aumentase su porcentaje de entradas es prácticamente imposible que consiguiera llegar a esos porcentajes de error. Independientemente de esto, vemos que se consigue conservar la diversidad en las distintas generaciones evitando la concentración de la población en un punto.

En cuanto a los porcentajes de error en el test, están siempre por encima del 30% (entre el 30% y el 33%).

4.3.1.6 Prueba 6

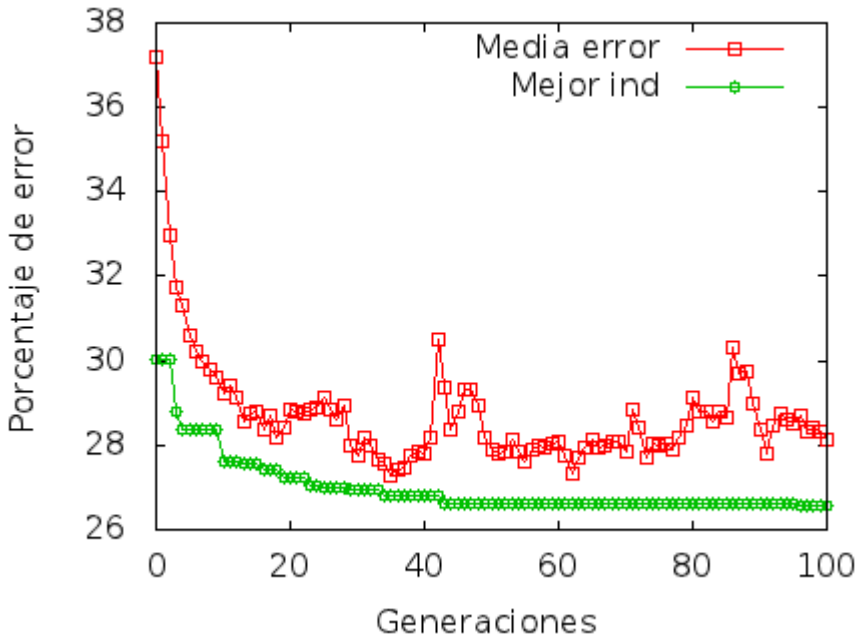
CONFIGURACIÓN
Tamaño de la población = 30 individuos
Número de operaciones por individuo= 20
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0,70
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por torneos probabilística. • Cruce de dos puntos y cruce de operaciones uniforme Probabilidad de cruce = 0.60 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.25 Factor de mutación = 0.15 Factor de mutación operación = 0.25 • Inversión: Probabilidad de inversión= 0.15 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 10 Padres elegidos al azar: 6 Mejores Hijos: 74 Hijos elegidos al azar: 10
Función fitness 1. Solo participa el error de la red
Clasificador: Razón= 0,1 Sessions= 300.

Tabla 20. Configuración prueba 6.

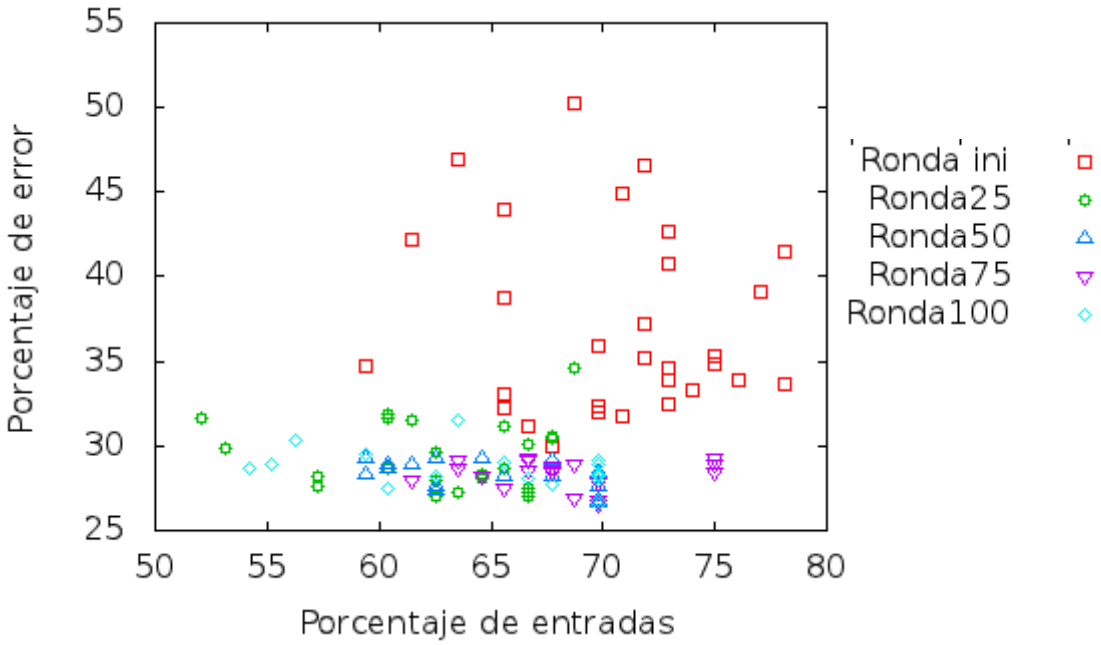
En esta prueba el principal cambio es que inicializamos la población con un 70% de atributos. Esto nos permitirá ver un efecto que no habíamos observado hasta ahora.

Como vemos tanto en las dos gráficas, la evolución del error durante las distintas rondas es igual a la de los casos anteriores, una disminución pronunciada al inicio y a continuación se estabiliza en torno a valores sobre el 26%-27% donde empieza a disminuir poco a poco. El principal cambio es que, al contrario de lo observado anteriormente en este caso la disminución del error no va acompañada de un aumento del porcentaje de entradas, al contrario se produce una ligera disminución.

En cuanto al test, se observa como los resultados del test son como un 5 ó 6% superiores a los obtenidos en la última ronda del entrenamiento (31% en adelante).



Gráfica 14. Mejor individuo y media de error en cada ronda (Prueba 6).



Gráfica 15. Evolución de la población (Prueba 6).

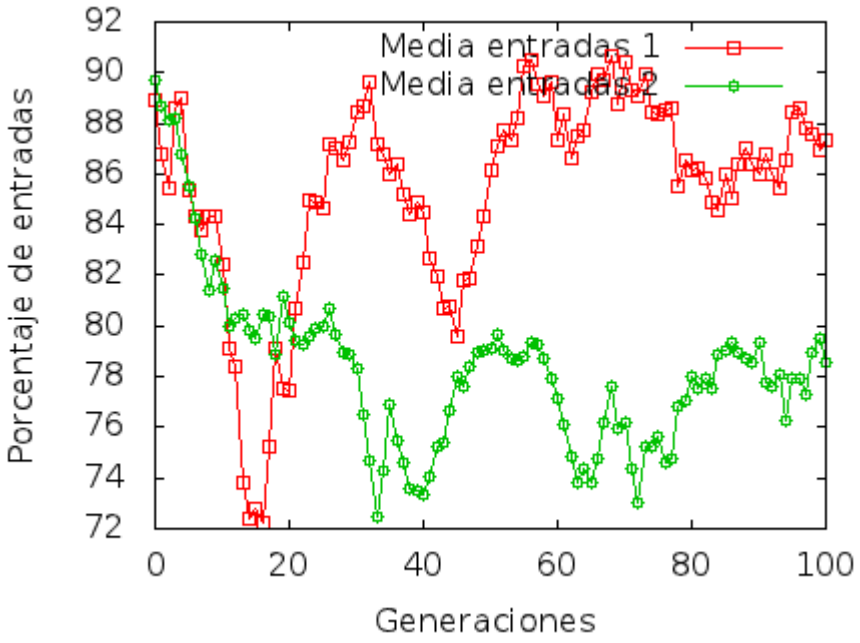
(En esta última gráfica observamos cómo se produce una disminución de las entradas a medida que aumentan las rondas)

4.3.1.7 Prueba 7

CONFIGURACIÓN
Tamaño de la población = 30 individuos
Número de operaciones por individuo= 20
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.90
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por torneos probabilística. • Cruce de dos puntos y cruce de operaciones uniforme Probabilidad de cruce = 0.55 Factor de cruce operación = 0.25 • Mutación y mutación de operaciones Probabilidad de mutación = 0.25 Factor de mutación = 0.15 Factor de mutación operación = 0.25 • Inversión: Probabilidad de inversión= 0.15 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 10 Padres elegidos al azar: 6 Mejores Hijos: 74 Hijos elegidos al azar: 10
Función fitness 1. Solo participa el error de la red

Tabla 21. Configuración prueba 7.

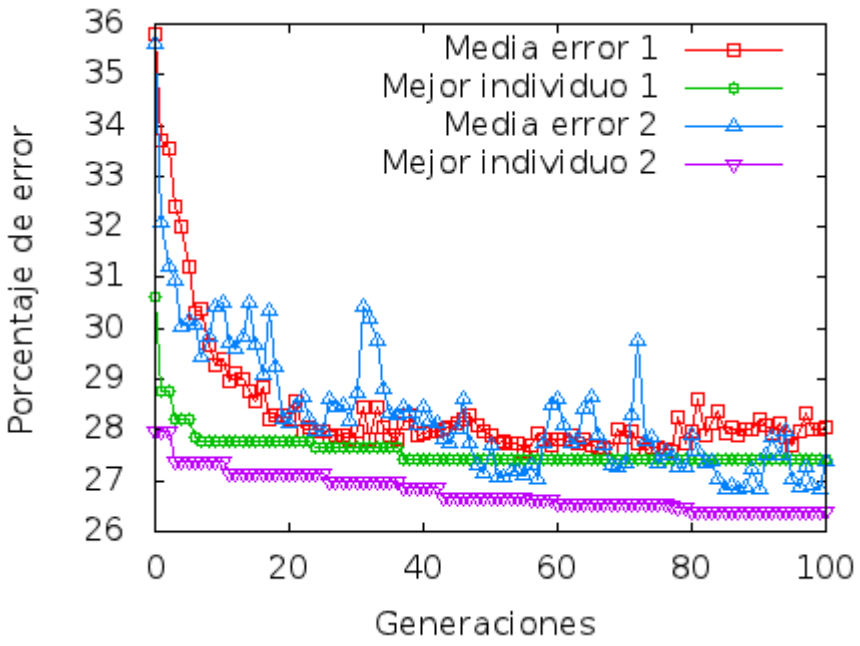
En esta prueba comenzamos con una población de individuos que contienen el 90% de atributos.



Gráfica 16. Media de entradas 2 ejecuciones (Prueba 7).

En la gráfica anterior se observan dos ejecuciones distintas de la prueba y la media de entradas en cada una de las rondas del algoritmo. Vemos como existe una mayor variabilidad que en otras pruebas ya que algunas ejecuciones disminuyen el porcentaje de entradas hasta el 75% otras se quedan en el 90% y esta variabilidad del porcentaje de entradas

afecta al resultado. Esta vez, al contrario de lo que ocurría con poblaciones inicializadas con otros porcentajes, para conseguir los mejores resultados es necesario disminuir el número de entradas como vemos en la gráfica de abajo donde el mejor individuo de la ejecución 1 mejora los resultados del mejor individuo de la ejecución 2 así como la media de error de la población 1 es también ligeramente inferior.



Gráfica 17. Comparación de mejores individuos y media (Prueba 7).

Llegados a la última ronda, el mejor individuo de la ejecución1 obtiene una valoración de 27,4%, en cambio el mejor individuo de la ejecución2 obtiene un 26,3%.

Pero en lo que respecta a los resultados obtenidos en el test, nos llevamos una grata sorpresa con los datos obtenidos en la ejecución 1. En el entrenamiento había obtenido resultados ligeramente peores a los de la ejecución 2 pero en la realización del test obtiene unos magníficos resultados nunca vistos hasta ahora consiguiendo porcentajes de error incluso inferiores al 26% (25.62% el mejor individuo del test). Es la primera vez que se observa unos porcentajes de error en el test inferiores a los del entrenamiento, así como es la primera vez que observamos un porcentaje de error en el test tan bajo. Esto sólo fue observado en la ejecución 1 y no se ha podido encontrar el motivo de tales resultados. Ninguna de las restantes ejecuciones dio esos porcentajes de test y estuvieron rondando siempre el 30% de error. Por ello deberemos seguir realizando pruebas con el fin de encontrar el motivo que hizo obtener estos buenos resultados.

4.3.1.8 Conclusiones relativas al porcentaje de entradas

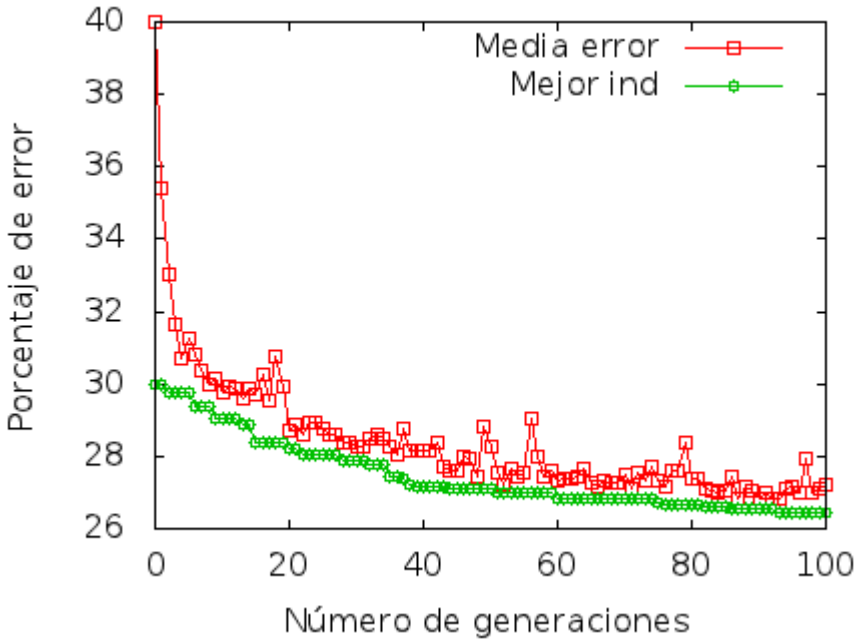
Las pruebas anteriores nos sirven de ejemplo y nos ayudan a explicar uno de los efectos observados a lo largo de todo el conjunto de pruebas relativo al porcentaje de entradas. Este efecto es que el algoritmo genético, en su búsqueda por los mejores individuos, siempre suele acabar con poblaciones de individuos con entre un 60%-75% de porcentaje de entradas o atributos. En la inmensa mayoría de los casos en los que hemos inicializado la población con un 50% de atributos o incluso menos, las poblaciones de las rondas finales han acabado rondando el 60-65%. Podíamos pensar entonces que el algoritmo siempre hace que la población aumente sus atributos, pero lo sorprendente fue que al inicializar poblaciones al 70% o incluso al 90% se observa en varias ocasiones el efecto contrario, una disminución de atributos hasta llegar a valores sobre 60-65% en los casos inicializados al 70% y rondando el 80% en algunos casos del 90% lo cual indica que el algoritmo en la gran mayoría de los casos siempre tiende a seleccionar individuos en la franja entre el 60-75% de atributos, que casualmente son los que tienen un porcentaje de error en el entrenamiento sobre el 26%-27% (Error mínimo obtenido hasta ahora). Es casi imposible ver a individuos con porcentajes de error en el entrenamiento sobre el 26% con un porcentaje de entradas inferior al 50% o superior al 70%.

4.3.1.9 Prueba 8

CONFIGURACIÓN
Tamaño de la población = 50 individuos
Número de operaciones por individuo= 15
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección jerárquica • Cruce uniforme y cruce de operaciones uniforme Probabilidad de cruce = 0.30 Factor de cruce = 0.15 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.30 Factor de mutación = 0.15 Factor de mutación operación = 0.15 • Inversión: Probabilidad de inversión = 0.15 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 8 Padres elegidos al azar: 0 Mejores Hijos: 90 Hijos elegidos al azar: 2
Función fitness 1. Solo participa el error de la red

Tabla 22. Configuración prueba 8.

Esta prueba es similar a las anteriores pero con un ligero aumento de la población hasta los 50 individuos. El fin de este aumento y del uso de un reemplazo menos selectivo es que se mantenga una diversidad mínima en la población, dado que el método que estamos usando para la selección (jerárquica) prima fuertemente a los individuos mejor preparados.



Gráfica 18. Mejor individuo y media de error en cada ronda (Prueba 8)

En la gráfica, observamos como evidentemente la población mejora ronda a ronda hasta cierto punto, si bien también vemos que la ronda 75 y la ronda 100 contienen individuos muy similares y las mejoras son mínimas. Esto es debido a que como ya hemos comentado, una vez que la población se acerca a valores de error sobre el 26%-27% le es muy costoso encontrar individuos mejores y si los encuentra, la mejora es de centésimas a lo sumo. Según pasan las rondas se puede ver como los individuos tienden a ser cada vez más similares aunque debido a que la población es más amplia que en otras ocasiones (consta de 50 individuos) se mantiene la diversidad. Otro aspecto que se observa es el aumento del porcentaje de entradas con respecto a la inicialización del 50% que hicimos.

En cuanto al porcentaje de error en el test, se obtienen alrededor de un 5% de más comparado con los resultados obtenidos en el entrenamiento, lo que hace que la población obtenga valores sobre 30%-32%. También vemos cómo en multitud de ocasiones el mejor individuo en el entrenamiento no tiene porqué ser el mejor en el test.

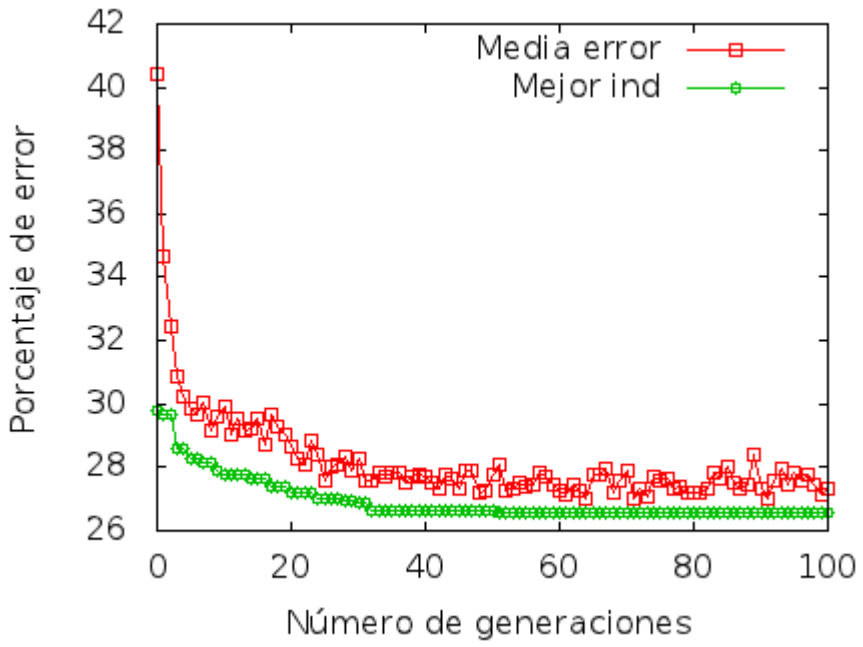
Pero además de esto, repetimos esta misma prueba utilizando una selección por torneos probabilísticos con el fin de ver si existía alguna variación, pero los resultados no variaron demasiado, la única diferencia es que con el cambio se produjo un aumento de diversidad de la población.

4.3.1.10 Prueba 9

CONFIGURACIÓN
Tamaño de la población = 50 individuos
Número de operaciones por individuo= 0
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: • Selección jerárquica • Cruce uniforme Probabilidad de cruce = 0.30 Factor de cruce = 0.15 • Mutación Probabilidad de mutación = 0.30 Factor de mutación = 0.15 • Inversión: Probabilidad de inversión = 0.15 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 8 Padres elegidos al azar: 0 Mejores Hijos: 90 Hijos elegidos al azar: 2
Función fitness 1. Solo participa el error de la red

Tabla 23. Configuración prueba 9.

Esta prueba es igual a la anterior pero sin usar ninguna operación. Al no usar operaciones, el número de entradas al clasificador es menor y por ello el tiempo de ejecución es ligeramente inferior. El fin de no utilizar operaciones es ver si se observa algún cambio en los resultados como ya vimos en secciones anteriores o si por el contrario, al aumentar el tamaño de la población y el número de rondas del algoritmo se atenúa este efecto.



Gráfica 19. Mejor individuo y media de error en cada ronda (Prueba 9).

En cuanto a los resultados, observamos que no parece existir variación. La población desciende su porcentaje de error ronda a ronda hasta llegar a porcentajes cercanos al 26%-27% en el entrenamiento, quizás la única diferencia sea que en esta prueba parece existir una mayor pérdida de la diversidad de la población concentrándose en torno a un punto.

Los resultados del test nos devuelven valores sobre 31%-33%, lo que es ligeramente superior a la prueba anterior pero este aumento puede ser debido a otros aspectos como el azar de la población, etc.

4.3.1.11 Prueba 10

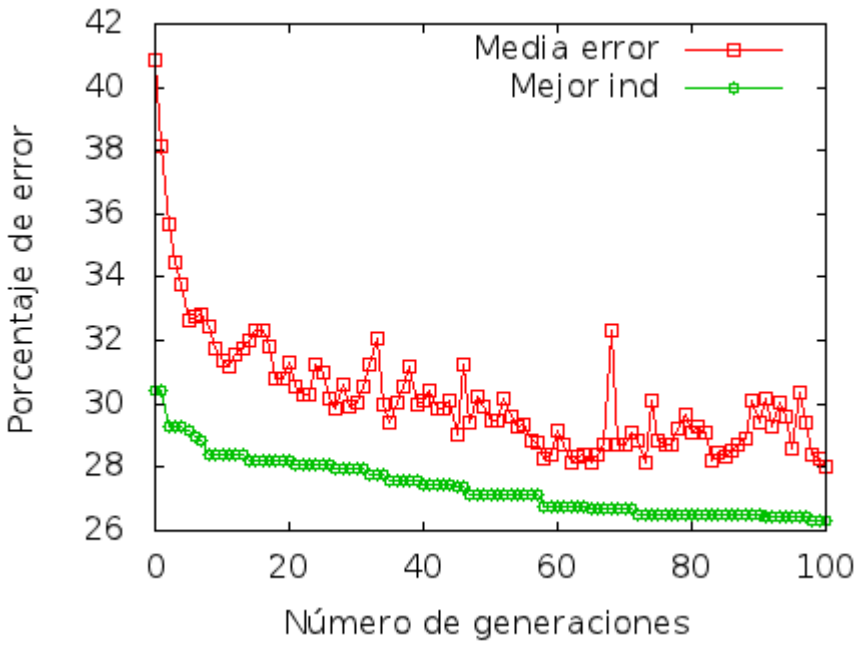
CONFIGURACIÓN
Tamaño de la población = 50 individuos
Número de operaciones por individuo= 15
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección torneos probabilística • Cruce uniforme y cruce de operaciones uniforme Probabilidad de cruce = 0.60 Factor de cruce = 0.35 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.40 Factor de mutación = 0.15 Factor de mutación operación = 0.25 • Inversión: Probabilidad de inversión = 0.25 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 10 Padres elegidos al azar: 4 Mejores Hijos: 74 Hijos elegidos al azar: 12
Función fitness 1. Solo participa el error de la red

Tabla 24. Configuración prueba 10.

Similar a la prueba anterior con la principal distinción del método de selección. En esta ocasión hemos utilizado la selección por torneos probabilística, método quizás menos “agresivo” que el usado anteriormente. También hemos aumentado los porcentajes del reemplazo estacionario relativos a los individuos escogidos al azar (tanto padres como hijos).

Estos cambios como vemos en la gráfica, nos hacen mantener una población más diversificada permitiendo grandes diferencias entre individuos pertenecientes a la misma ronda. Pero esto no nos sirve para mejorar nuestros resultados, si bien ahora los individuos de una misma ronda no están tan concentrados entre ellos pero en cambio siguen sin producirse mejoras y la ronda 75 y la ronda 100 contienen individuos con la misma preparación. Esto se produce porque seguimos sin conseguir sobrepasar la barrera del 26% de error en el entrenamiento y, dado que no somos capaces

de superar este valor, el algoritmo no consigue mejorar los resultados haciendo que los individuos de rondas posteriores sean prácticamente iguales y con la misma valoración. La única variación que parece surgir es que una vez llegados a esos porcentajes se empiezan a seleccionar individuos con un mayor porcentaje de entradas. En este caso se observan individuos en la ronda 100 con un porcentaje de entradas cercano al 74%, casi un 25% más de lo que fueron inicializados.



Gráfica 20. Mejor individuo y media de error en cada ronda (Prueba 10).

En cuanto al test, los porcentajes como en la mayoría de los casos están por encima de los obtenidos en el entrenamiento. En este caso, la población de la ronda final contiene un porcentaje de error en el test entre el 29%-33%. Aparecen varios individuos cercanos al 29%, (29.19%, 29.40%, etc.) en algunas de las ejecuciones.

4.3.1.12 Prueba 11

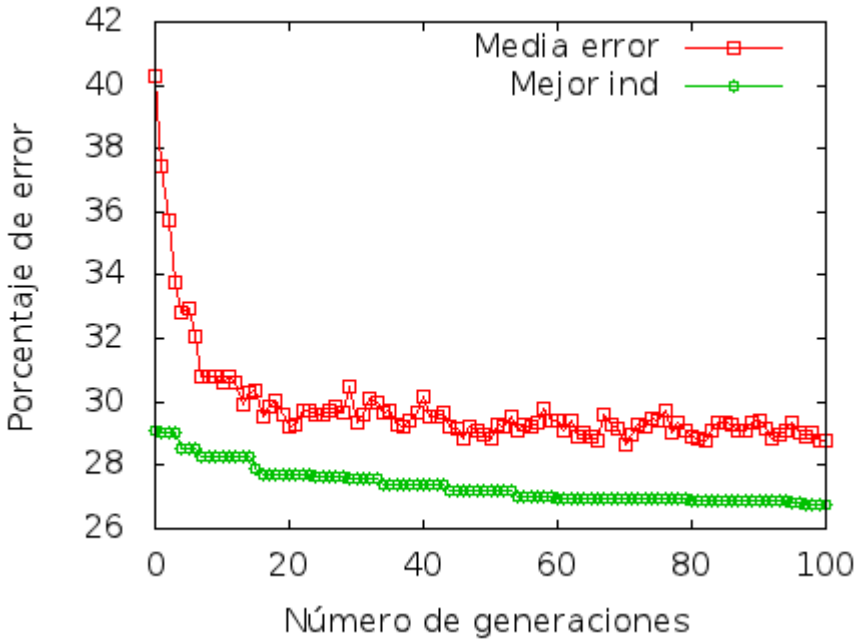
Igual que la anterior pero sin ninguna operación en sus individuos. Parece ser que la falta de operaciones hace que se pierda diversidad en la población (por otro lado previsible ya que los individuos contienen menos características que los definen) consiguiendo poblaciones ligeramente más centradas alrededor de un punto.

De igual manera, seguimos observando como el algoritmo mejora ronda a ronda los resultados rápidamente hasta que llega al 27% de error, donde se estanca y pasa a disminuir muy lentamente. Los resultados de la última ronda del entrenamiento están sobre el 27% y el 30% de error.

Tabla con la configuración de la prueba:

CONFIGURACIÓN
Tamaño de la población = 50 individuos
Número de operaciones por individuo= 0
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: • Selección torneos probabilística • Cruce uniforme Probabilidad de cruce = 0.60 Factor de cruce = 0.35 • Mutación Probabilidad de mutación = 0.40 Factor de mutación = 0.15 • Inversión: Probabilidad de inversión = 0.25 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 10 Padres elegidos al azar: 4 Mejores Hijos: 74 Hijos elegidos al azar: 12
Función fitness 1. Solo participa el error de la red

Tabla 25. Configuración prueba 11.



Gráfica 21. Mejor individuo y media de error en cada ronda (Prueba 11).

En cuanto al test, aparecen algunos individuos con un 29% de error (29,12% mejor individuo de todas las ejecuciones realizadas) pero en todas las ejecuciones la mayoría de la población se encuentra sobre el 30%-32%.

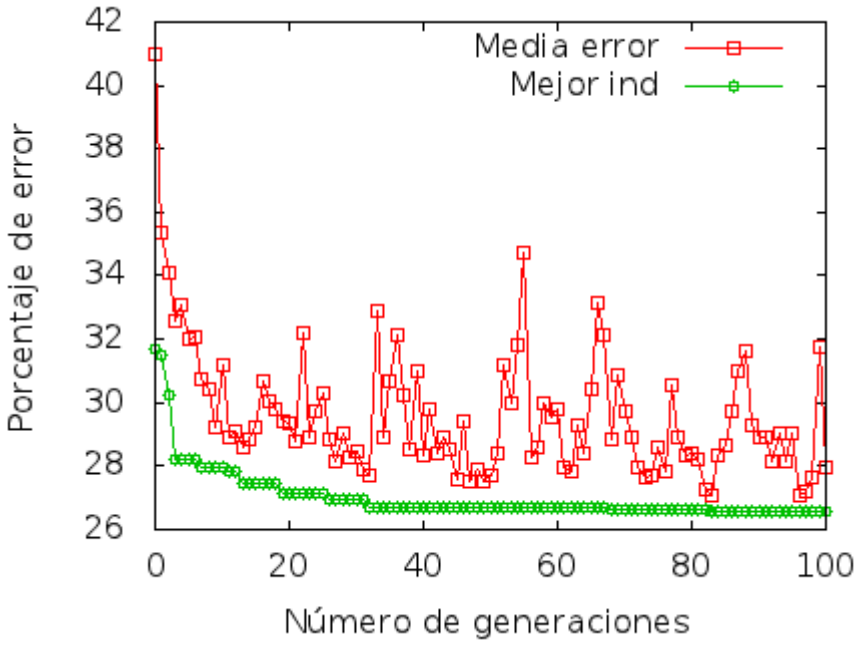
4.3.1.13 Prueba 12

En lo que respecta a los resultados, siguen la tendencia que se ha observado en el resto de las pruebas realizadas hasta ahora, descenso

acusado en las primeras rondas hasta alcanzar valores cercanos al 27% de error y a partir se ralentiza, disminuyendo unas pocas décimas, todo ello acompañado con un aumento del porcentaje de entradas en los individuos.

CONFIGURACIÓN
Tamaño de la población = 50 individuos
Número de operaciones por individuo= 20
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección torneos determinista. Tamaño torneo= 2 • Cruce de un punto y cruce de operaciones uniforme Probabilidad de cruce = 0.60 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.40 Factor de mutación = 0.15 Factor de mutación operación = 0.25 • Inversión: Probabilidad de inversión = 0.35 • Reemplazo estacionario. Hijos: 90 Padres: 10
Función fitness 1. Solo participa el error de la red

Tabla 26. Configuración prueba 12.



Gráfica 22. Mejor individuo y media de error en cada ronda (Prueba 12).

La diferencia en esta prueba es una mayor variabilidad en la media del error en cada ronda producida por las altas probabilidades de los operadores genéticos que nos generan descendientes con multitud de cambios con respecto a sus padres. Estas altas probabilidades podrían ser peligrosas ya que generan grandes cambios perdiendo los rasgos de sus antecesores (los individuos de rondas anteriores) pero dado que utilizamos un reemplazo estacionario evitaremos perder a los mejores individuos. Otra consecuencia de esto es que al generarse generaciones con tantos cambios

evitaremos que la población acabe convergiendo en un punto. Aún con todo, seguimos obteniendo individuos cercanos al 26,5% de error en el entrenamiento como vemos en la gráfica 22. En cuanto al test, los resultados obtenidos varían entre el 30%-32% de error.

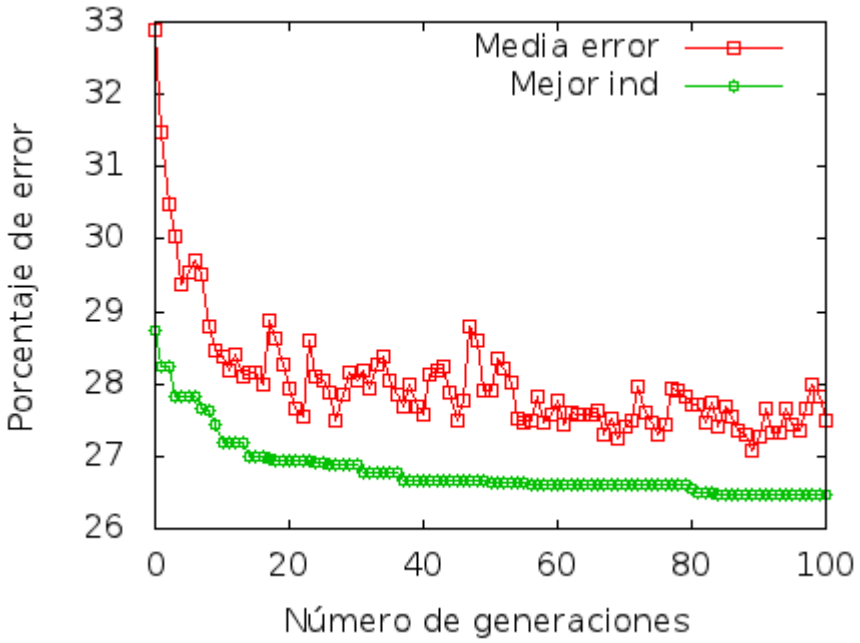
4.3.1.14 Prueba 13

CONFIGURACIÓN
Tamaño de la población = 50 individuos
Número de operaciones por individuo= 20 Porcentaje sumas= 40 porcentaje restas= 40 Porcentaje multiplicación= 20 porcentaje división= 0
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0,5
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección torneos determinista. Tamaño torneo= 2 • Cruce de un punto y cruce de operaciones uniforme Probabilidad de cruce = 0.60 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.40 Factor de mutación = 0.15 Factor de mutación operación = 0.25 • Inversión: Probabilidad de inversión = 0.35 • Reemplazo estacionario. Hijos: 90 Padres: 10
Función fitness 1. Solo participa el error de la red

Tabla 27. Configuración prueba 13.

Esta prueba es muy similar a la prueba 12 anterior, la única distinción existente es que en esta prueba por primera vez hemos variado la probabilidad de aparición de los operadores de las operaciones dejando de generarse completamente al azar. De este modo desechamos las divisiones que como vimos en la experimentación previa parecían no favorecer al resultado.

Como vemos en la gráfica, los resultados obtenidos de inicio son visiblemente mejores a los que habíamos obtenido hasta ahora, en la generación inicial surgen individuos con un porcentaje de error sobre el 29% y varios más que rondan el 30% algo que no se había observado hasta ahora. En cambio, en todas las ejecuciones realizadas para esta prueba vemos como a medida que pasan las rondas los resultados no mejoran y siguen estancados al llegar a valores alrededor del 27%, con lo cual, y por mucho que partamos de poblaciones iniciales mejores, no conseguimos reducir el error. La población está por encima del 32% de error en el test. En otras ejecuciones han surgido individuos sobre el 30%.



Gráfica 23. Mejor individuo y media error por ronda (Prueba 13)

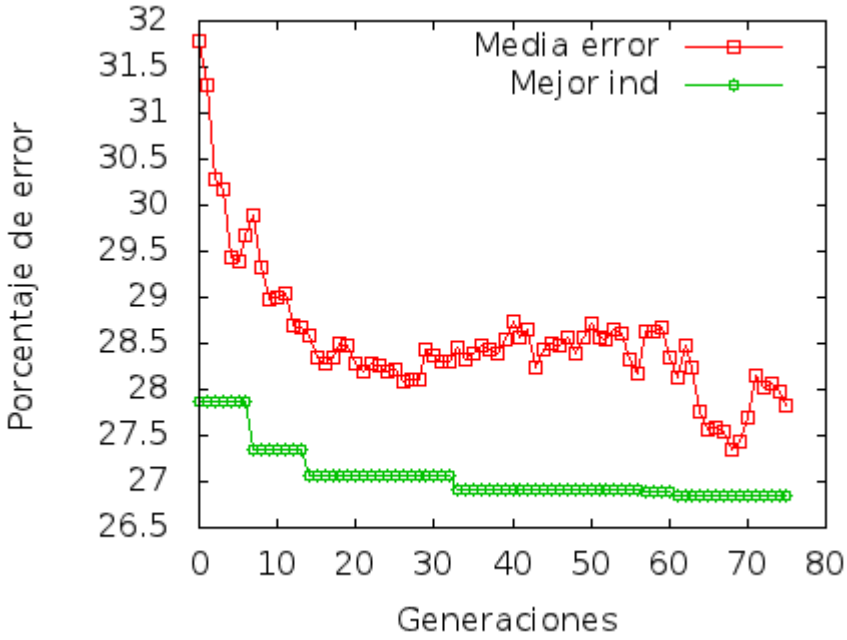
4.3.1.15 Prueba 14

CONFIGURACIÓN
Tamaño de la población = 40 individuos
Número de operaciones por individuo= 25 Porcentaje sumas= 50 porcentaje restas= 50 Porcentaje multiplicación= 0 porcentaje división= 0
Número de rondas del algoritmo = 75
Porcentaje de atributos con los que comienza la población= 0,5
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por método de la ruleta • Cruce dos puntos y cruce de operaciones uniforme Probabilidad de cruce = 0.55 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.25 Factor de mutación = 0.15 Factor de mutación operación = 0.25 • Inversión: Probabilidad de inversión = 0.15 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 10 Padres elegidos al azar: 5 Mejores Hijos: 80 Hijos elegidos al azar: 5
Función fitness 1. Solo participa el error de la red

Tabla 28. Configuración prueba 14.

En esta prueba se crean sólo sumas y restas dejando a un lado las divisiones y multiplicaciones. Con ello conseguimos comenzar con una población muy buena en la que los mejores individuos tienen un porcentaje de error en el entrenamiento de 28%. Pero al igual que en pruebas anteriores este buen comienzo no se traduce en una mejora de los

resultados ya que tras 75 rondas del algoritmo el mejor individuo de la ronda 75 en el caso que vemos en la gráfica obtiene un 26,84% de error en el entrenamiento, lo cual es una mejoría de alrededor de un 1%. En todas las ejecuciones se producen casos similares y aunque el algoritmo comience con una muy buena población esto no es suficiente para que el error disminuya más allá del 26%.



Gráfica 24. Mejor individuo y media error por ronda (Prueba 14)

La principal diferencia entre la ronda inicial del algoritmo y la ronda 75 es que en la ronda 75 los individuos están más agrupados y obtienen todos un porcentaje de error inferior al 29% en el entrenamiento. En cuanto al test, los resultados superan los del entrenamiento y están sobre el 31%-34%.

4.3.1.16 Prueba 15

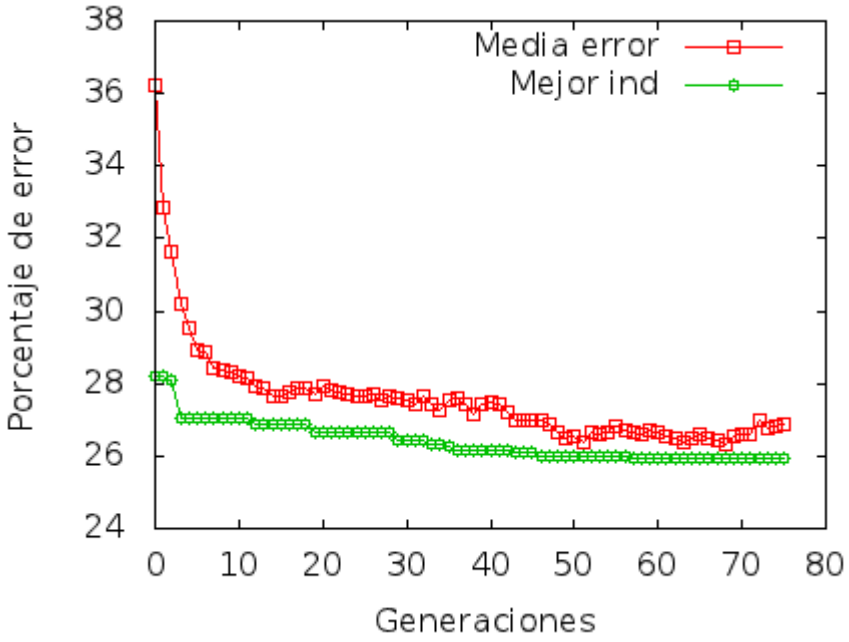
En esta prueba desechamos las divisiones. En la gráfica siguiente vemos como desde el inicio aparecen individuos muy buenos hecho que se ha observado en prácticamente todas las ejecuciones realizadas de esta prueba. Independientemente de esto, y debido a estos buenos individuos surgidos en rondas muy tempranas, a las pocas rondas de iniciarse la ejecución el algoritmo llega a porcentajes de error sobre el 27% ralentizándose la disminución del error hasta llegar al 26%-27%. Esto ha sido común salvo en una de las ejecuciones en la que sorprendentemente surgen dos individuos que obtienen una valoración de 25,93% y 25,97% en el entrenamiento, valores que no habíamos sido capaces de observar en ninguna de las pruebas anteriores. Si bien esta disminución no es muy grande y se trata de unas pocas decimas con respecto a los resultados

anteriores, es importante ya que es la primera vez que rompemos la barrera del 26% en el entrenamiento a pesar de considerarse fruto del azar.

Configuración de la prueba:

CONFIGURACIÓN
Tamaño de la población = 40 individuos
Número de operaciones por individuo= 25 Porcentaje sumas= 33 Porcentaje restas= 34 Porcentaje multiplicación= 33 Porcentaje división= 0
Número de rondas del algoritmo = 75
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por torneos probabilísticos. • Cruce uniforme y cruce de operaciones uniforme Probabilidad de cruce = 0.55 Factor de cruce = 0.15 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.20 Factor de mutación = 0.15 Factor de mutación operación = 0.25 • Inversión: Probabilidad de inversión = 0.15 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 10 Padres elegidos al azar: 5 Mejores Hijos: 80 Hijos elegidos al azar: 5
Función fitness 1. Solo participa el error de la red

Tabla 29. Configuración prueba 15.



Gráfica 25. Mejor individuo y media error por ronda (Prueba 15)

En cuanto a los resultados del test, no se observan mejorías y estas pocas décimas que hemos conseguido mejorar en el entrenamiento no se traducen en el resultado del test.

4.3.1.17 Conclusiones sobre las operaciones

Los cambios o diferencias son sobre todo que en las pruebas en las que no hemos utilizado operaciones con respecto a sus homólogas con operaciones (prueba 8 con operaciones y 9 sin ellas, y prueba 10 con operaciones y 11 sin ellas) en el sentido que las primeras parecían dar una mayor diversidad a la población pero, en lo relativo a los porcentajes de error, no se observaron diferencias significativas en los resultados.

La mayor diferencia quizás no esté en usar operaciones o no sino en que al usarlas e iniciar las operaciones se escoja los porcentajes de cada uno de los operadores y no se deje ese aspecto al azar. Como hemos podido observar en las pruebas 13, 14 y 15 en los que modificamos estos porcentajes y desechamos las divisiones, los resultados que se obtenían de inicio eran mucho mejores, con lo que conseguíamos partir con una población inicial muy buena, pero también nos sirvió para descubrir que aún partiendo de una buena población inicial esto no nos iba a servir para reducir el error en las generaciones finales más allá de lo que parece nuestro límite (26% en el entrenamiento), ya que al llegar a los mismos porcentajes de error el algoritmo se estancaba. Por ello, y salvo en un solo caso, para lo único que nos sirvió fue para conseguir llegar en un menor número de rondas al 26% de error.

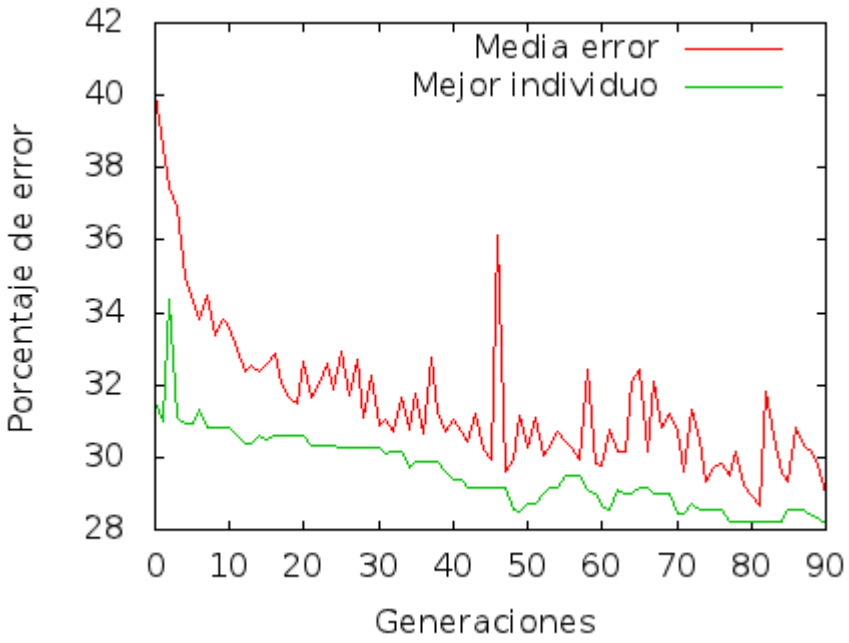
4.3.1.18 Prueba 16

CONFIGURACIÓN
Tamaño de la población = 15 individuos
Número de operaciones por individuo= 25
Número de rondas del algoritmo = 90
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por torneos determinista. Tamaño torneos=2 • Cruce dos puntos y cruce de operaciones uniforme Probabilidad de cruce = 0.50 Factor de cruce operación = 0.35 • Mutación y mutación de operaciones Probabilidad de mutación = 0.20 Factor de mutación = 0.25 Factor de mutación operación = 0.35 • Inversión: Probabilidad de inversión = 0.20 • Reemplazo generacional.
Función fitness 1. Sólo participa el error de la red.

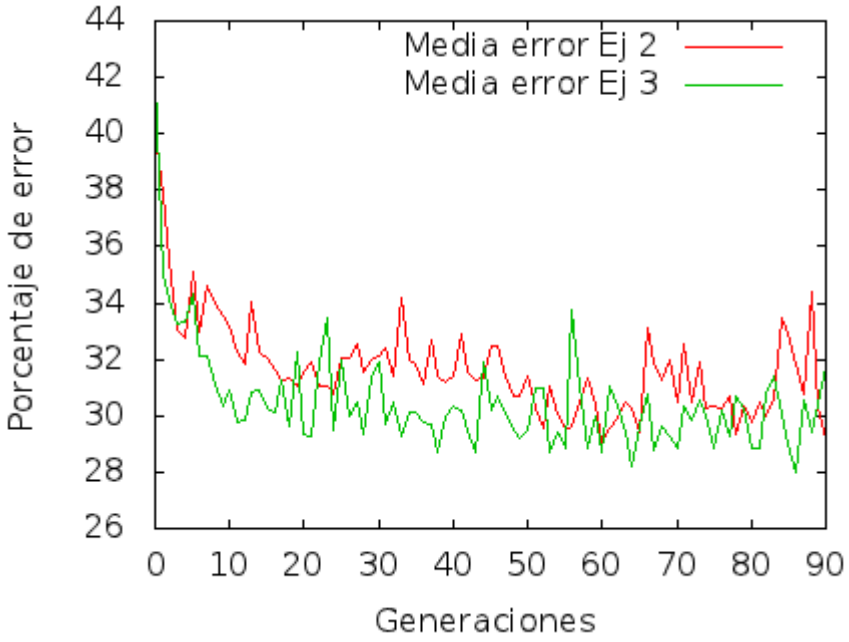
Tabla 30. Configuración prueba 16.

En esta prueba, hay muchos cambios con respecto a las pruebas anteriores, pero el cambio más importante es el del tipo de reemplazo. Es la primera ocasión en la que utilizamos un reemplazo generacional lo que conlleva a que la totalidad de los individuos padres sean sustituidos por los

individuos hijos sin importar el nivel de adaptación de éstos. El resultado de este reemplazo es el que se observa en la gráfica 26. Al ser sustituida la población entera, se pierden los mejores individuos de rondas anteriores. Además el algoritmo ya no nos muestra la típica gráfica donde generación tras generación la media de error de la población es similar a la generación anterior y el mejor individuo siempre es mejor o por lo menos igual a las generaciones anteriores sino que ahora va dando saltos.



Gráfica 26. Mejor individuo y media de error por ronda (Prueba 16).



Gráfica 27. Evolución porcentaje error, varios ejemplos (Prueba 16).

Existe una mayor variabilidad entre las diferentes ejecuciones de una prueba al contrario de lo que ocurría en la mayoría de los casos anteriores

dado que ahora se depende más del azar. En la gráfica 27 vemos dos ejemplos de esta misma prueba donde se observa la diferencia existente entre ambas y también la diferencia con el caso anterior.

Otra diferencia es que existe una mayor variabilidad de la población y los individuos de una misma ronda están más dispersos y no tan concentrados en un punto como en otras ocasiones, tampoco se ha producido un aumento del porcentaje de entradas y se mantienen con un porcentaje similar al que fueron inicializados. En cuanto al porcentaje de error en el entrenamiento, los mejores individuos obtenidos en esta prueba han rondado el 28% (error ligeramente superior al de las pruebas anteriores). En cambio, los resultados del test se mantienen sobre el 31%-34%.

4.3.1.19 Prueba 17

CONFIGURACIÓN
Tamaño de la población = 75 individuos
Número de operaciones por individuo= 10
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección torneos probabilística • Cruce de un punto y cruce de operaciones uniforme Probabilidad de cruce = 0.40 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.25 Factor de mutación = 0.20 Factor de mutación operación = 0.25 • Inversión: Probabilidad de inversión = 0.15 • Reemplazo generacional.
Función fitness 1. Solo participa el error de la red

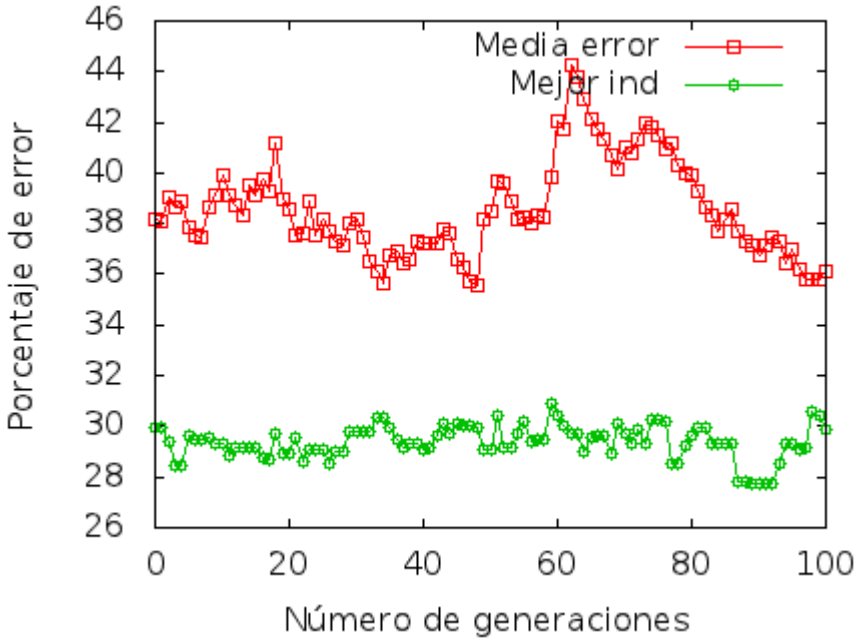
Tabla 31. Configuración prueba 17.

Al igual que la prueba anterior, el principal cambio y el que más influirá en los resultados es el reemplazo generacional. Este será el responsable de los siguientes resultados:

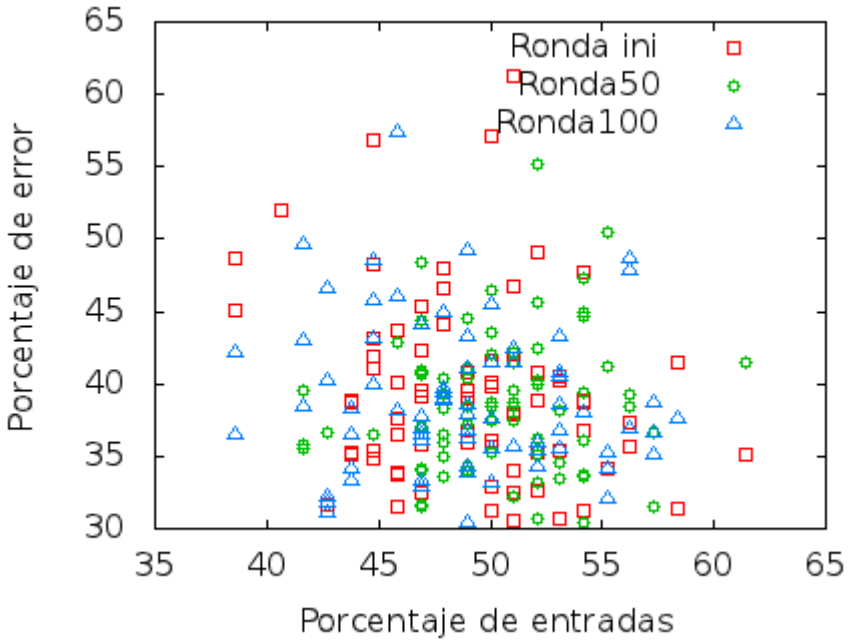
Como vemos en la gráfica 28, al reemplazarse la totalidad de la población se eliminan los mejores individuos de las rondas anteriores por lo que el algoritmo, en vez de centrarse en una dirección en su búsqueda de los mejores individuos e ir descendiendo poco a poco el porcentaje de error, va dando pequeños bandazos al variar los individuos en cada ronda.

Y a continuación en gráfica 29 vemos la comparación de varias generaciones. Se puede observar como al contrario de lo que ocurría en otras pruebas, en esta no se ve una mejoría entre las rondas iniciales y las posteriores. En cambio se observa una “nube de puntos” donde se entremezclan las distintas generaciones de la población y donde los individuos mejor preparados rondan el 30% de error en el entrenamiento,

resultado pésimo comparado con otras pruebas, lo que indica que no ha habido una mejoría en las 100 rondas generadas con respecto a la inicial y no ha servido de mucho la prueba ya que no ha podido realizar mejoras en la población.



Gráfica 28. Mejor individuo y media de error por ronda (Prueba 17).



Gráfica 29. Evolución de la población (Prueba 17).

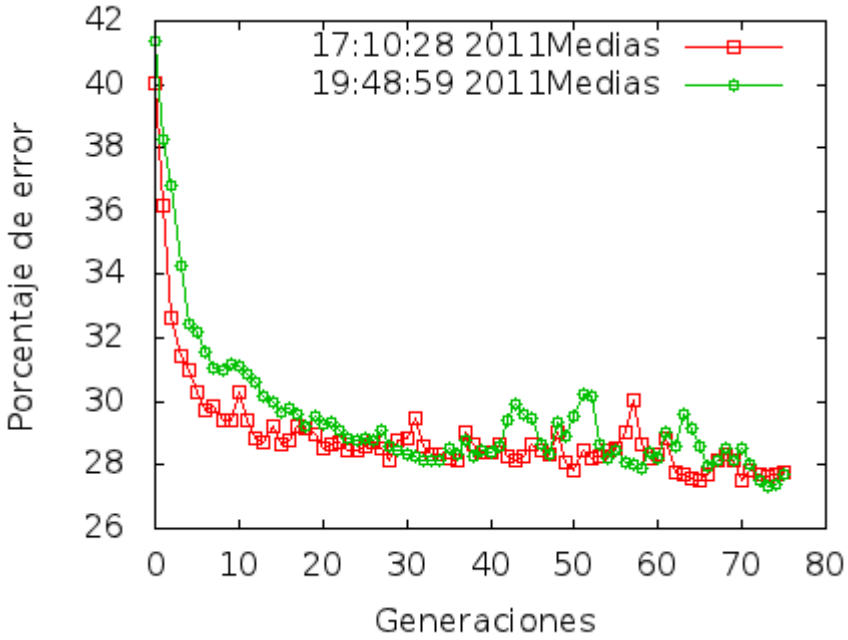
En cuanto al test, la variabilidad es más amplia debido a que los distintos individuos de la ronda 100 son muy diversos entre sí pero esto no ha hecho que aparecieran individuos con un buen porcentaje de error en el test siendo los resultados peores con respecto a las demás pruebas.

4.3.1.20 Prueba 18

CONFIGURACIÓN
Tamaño de la población = 25 individuos
Número de operaciones por individuo= 15
Número de rondas del algoritmo = 75
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por método de la ruleta • Cruce uniforme y cruce de operaciones uniforme Probabilidad de cruce = 0.30 Factor de cruce = 0.15 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.30 Factor de mutación = 0.15 Factor de mutación operación = 0.15 • Inversión: Probabilidad de inversión = 0.18 • Reemplazo estacionario. Hijos: 85 Padres: 15
Función fitness 1. Solo participa el error de la red.

Tabla 32. Configuración prueba 18.

En esta sencilla prueba vemos como, al contrario de lo que ocurría en las pruebas anteriores con el reemplazo generacional, varias ejecuciones de la misma prueba siguen con ciertos matices la misma evolución y dan resultados parecidos. En la gráfica de más abajo vemos 2 ejecuciones de esta misma prueba:



Gráfica 30. Evolución del porcentaje de error, 2 ejemplos (Prueba 18).
(Linea verde, ejemplo1, línea roja ejemplo 2)

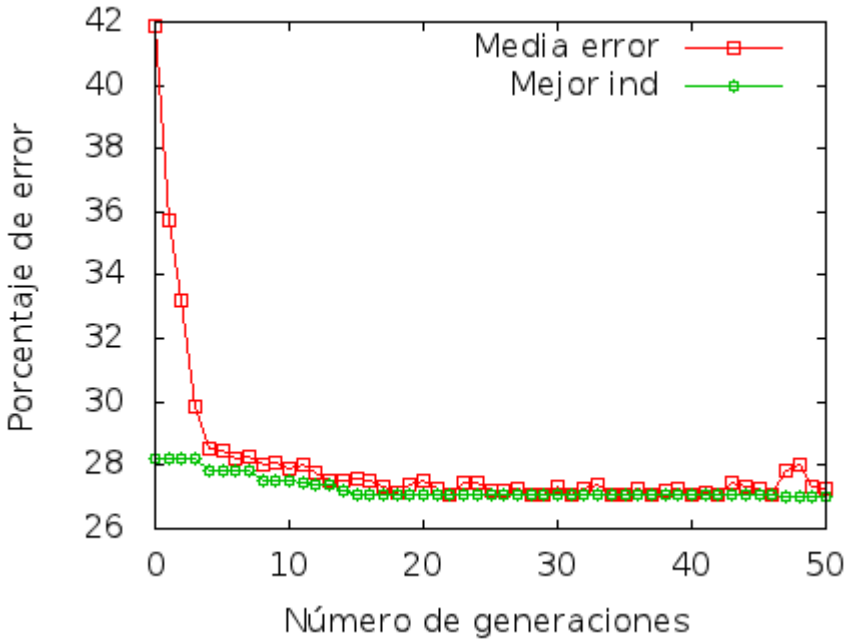
Se observa como existe una progresión ya que según aumenta el número de generación o ronda el porcentaje de error en el entrenamiento

disminuye a la vez que aumenta el porcentaje de entradas. De todas las ejecuciones realizadas, los mejores individuos de la ronda 75 tienen una valoración del 26,82% en el entrenamiento, el resto de los individuos de esa ronda están entre el 27% y el 29%. Los resultados de test están sobre el 30%-32,5% de error.

4.3.1.21 Prueba 19

CONFIGURACIÓN
Tamaño de la población = 25 individuos
Número de operaciones por individuo= 0
Número de rondas del algoritmo = 50
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: • Selección torneos determinista. Tamaño torneos = 3 • Cruce uniforme Probabilidad de cruce = 0.30 Factor de cruce = 0.50 • Mutación Probabilidad de mutación = 0.25 Factor de mutación = 0.12 • Inversión: Probabilidad de inversión = 0.25 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 16 Padres elegidos al azar: 8 Mejores Hijos: 68 Hijos elegidos al azar: 8
Función fitness 1. Solo participa el error de la red

Tabla 33. Configuración prueba 19.



Gráfica 31. Mejor individuo y media de error por ronda. (Prueba 19)

En esta pequeña prueba de 25 individuos y 50 rondas, además de no usar ninguna operación, utilizamos un reemplazo estacionario con elementos

aleatorios en los que los individuos aleatorios y los mejores padres tienen una pequeña aunque importante aparición. En cuanto a los resultados, vemos como conseguimos reducir el error pero también como a medida que pasan las generaciones la población tiende a concentrarse en un punto.

Los resultados del entrenamiento en todas las ejecuciones han estado sobre el 27%-28% de error y en todas ellas, prácticamente todos los individuos de la población tenían el mismo valor. Esto se debe a lo reducido del tamaño de la población, al método de selección empleado que no es el más adecuado para una población tan pequeña (ni mucho menos el tamaño de los torneos) así como al resto de la combinaciones de operaciones dado que la baja probabilidad de cruce no ayuda a generar diferencias en los individuos con los anteriores.

4.3.1.22 Prueba 20

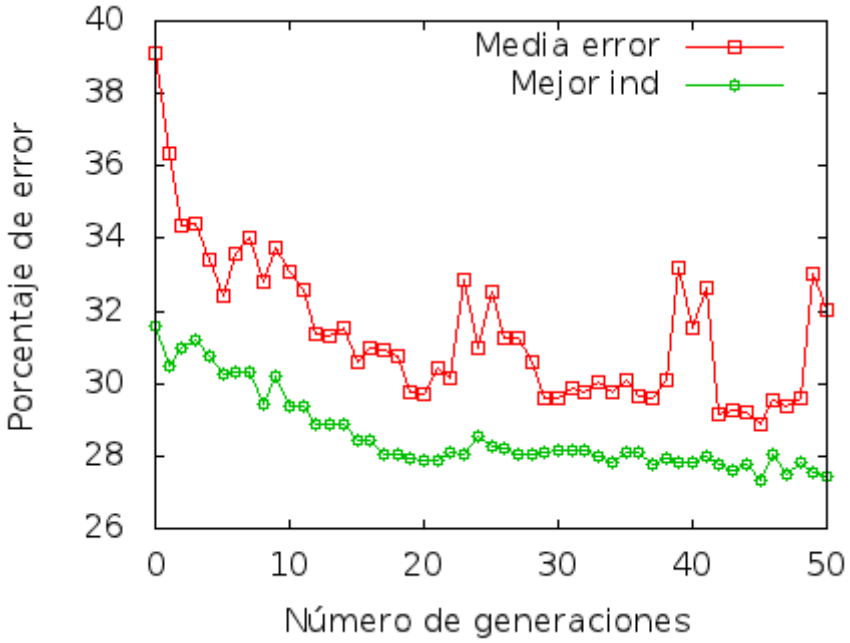
CONFIGURACIÓN
Tamaño de la población = 30 individuos
Número de operaciones por individuo= 20
Número de rondas del algoritmo = 50
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección jerárquica. • Cruce dos puntos y cruce de operaciones uniforme Probabilidad de cruce = 0.60 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.40 Factor de mutación = 0.15 Factor de mutación operación = 0.25 • Inversión: Probabilidad de inversión = 0.35 • Reemplazo generacional.
Función fitness 1. Solo participa el error de la red

Tabla 34. Configuración prueba 20.

En esta prueba volvemos a utilizar un reemplazo generacional e intentamos compensarla con una selección jerárquica con el fin de que, aunque perdamos a los individuos padres mejor preparados los hijos creados sean descendientes de estos. Las probabilidades de cruce, mutación e inversión son más altas de lo común.

En cuanto a los resultados obtenidos, vemos como se consigue mejorar los resultados poco a poco independientemente de la pérdida de los mejores individuos (en torno al 27,5% de error) aunque éstos sean un poco peores a los obtenidos en otras pruebas. Las poblaciones son más heterogéneas fruto del reemplazo y de los altos porcentajes de los operadores. En cuanto al test, el porcentaje de error se encuentra entre el 30% y el 33,5%. Fruto de este reemplazo, existe una mayor variabilidad en los resultados entre las diferentes ejecuciones realizadas. El ejemplo

mostrado en la gráfica representa los efectos producidos en la mayoría de los casos.



Gráfica 32. Mejor individuo y media error por ronda (Prueba 20)

4.3.1.23 Conclusiones sobre el reemplazo

Con las pruebas anteriores podemos hacernos una idea sobre los tres métodos de reemplazo y su comportamiento. El aspecto más significativo y que más hace notar las diferencias es la preservación de los mejores individuos de la población anterior (padres). Como hemos podido observar en las pruebas 16 y 17 donde el algoritmo usaba un reemplazo generacional, los resultados eran visiblemente peores, sobre todo en la prueba 17. Al no permanecer una parte de los mejores individuos padres parece que se pierde el objetivo haciendo que las sucesivas poblaciones se expandan por el espacio de búsqueda en vez de centrarse en las zonas más prometedoras. Esto tiene su justificación ya que al depender directamente de la totalidad de los individuos hijos sin importar su fitness, dependeremos de que las mutaciones, cruces e inversiones utilizadas en la creación de nuevos individuos sean beneficiosas o no para nuestro fin. Por ello, se considera más útil usar uno de los reemplazos estacionarios implementados siempre y cuando se ajusten bien los parámetros (tamaño de la población, porcentajes de mutación, cruce, etc.) y el porcentaje de padres escogidos sea bajo para favorecer la evolución de la población.

El reemplazo estacionario y el reemplazo estacionario con elementos aleatorios dan ambos buenos resultados, si bien el segundo, dadas sus características, consigue mantener una mayor diversidad en las poblaciones futuras al introducir en las nuevas poblaciones individuos (tanto padres como

hijos) escogidos al azar y sin importar su nivel de aptitud. Pero al igual que antes, habrá que tener cuidado y seleccionar un conjunto bajo de individuos con estas características ya que si no podríamos hacer que ocurran efectos parecidos a los que surgían con un reemplazo generacional.

4.3.1.24 Conclusiones globales

Tras los resultados conseguidos en el conjunto de pruebas realizado, hemos llegado a la una serie de conclusiones sobre los diferentes aspectos que engloba el algoritmo genético. Vamos a dividir estas conclusiones en puntos según el tema que traten:

Resultados obtenidos:

Como hemos podido ver, el porcentaje de error de entrenamiento, independientemente de los parámetros del algoritmo, nunca hemos conseguido reducirlo más allá del 25,6%, siendo puntual la aparición de individuos con un porcentaje menor al 26%. En cuanto al porcentaje de error en test, los individuos siempre rondan el 30% con la aparición en ocasiones de individuos con un 29% o incluso con un 28,9%. Sólo en una sola ocasión, en una ejecución de la prueba 7 se vieron resultados en el test del 25.62% pero los resultados de esta ejecución pueden ser debidos al azar ya que no hemos sido capaces de obtenerlos en otras ocasiones y sólo surgieron en un caso puntual. Ambos errores, tanto en entrenamiento como en test, se consideran buenos ya que, gracias a la selección de atributos llevada a cabo por el genético, hemos conseguido mejorar el error de clasificación del perceptrón simple de manera aislada. Los resultados obtenidos en la Experimentación previa (4.2.1.1 Prueba del clasificador de 2 perceptrones) eran de 31.53% en el entrenamiento y de 36.21% en el test, por lo tanto en ambos casos se ha conseguido una mejoría sobre el 6% o incluso superior.

Independientemente de esta mejora, también las pruebas nos han servido para conocer los límites del clasificador, y con ellas hemos visto cómo llegado a cierto punto el algoritmo se estanca (en torno al 26%-27%) y no consigue disminuir el error en el entrenamiento. Sin importar las rondas que tenga por delante o la cantidad de individuos, el sistema jamás consigue reducir ese error más allá del 25,6%. Este hecho nos lleva a pensar que hemos llegado al límite de nuestro clasificador y que no seremos capaces de mejorar estos resultados a no ser que lo sustituyamos por otro más eficiente.

En la siguiente tabla podemos observar todas las pruebas realizadas, su configuración más relevante y los resultados obtenidos (los porcentajes de error en entrenamiento y test son la media de todas las ejecuciones realizadas en cada una de las pruebas):

Nº	Pob	Rond	Nº Op	Sel	Mutación		Cruce			Reempl.	%Ent	%Test
					P	F	T	P	F			
1	12	100	15	2	0.3	0.15	3	0.3	0.15	2	28.53	31,84
2	12	200	15	2	0.3	0.15	3	0.3	0.15	3	27,75	32.29
3	12	400	15	2	0.3	0.15	3	0.3	0.15	3	27.31	30.76
4	12	80	20	1	0.2	0.1	1	0.6		2	28.97	32.97
5	40	100	20	1	0.2	0.1	1	0.6		2	28.92	31.56
6	30	100	20	3	0.25	0.15	2	0.6		3	28.15	32.92
7	30	100	20	3	0.25	0.15	2	0.55		3	27.71	**29.86**
8	50	100	15	2	0.3	0.15	3	0.3	0.15	3	27.48	31.76
9	50	100	0	2	0.3	0.15	3	0.3	0.15	3	27.54	32.52
10	50	100	15	3	0.4	0.15	3	0.6	0.35	3	28.91	30.77
11	50	100	0	3	0.4	0.15	3	0.6	0.35	3	30.68	31.85
12	50	100	20	4	0.4	0.15	1	0.6		2	28.39	31.54
13	50	100	20	4	0.4	0.15	1	0,6		2	28.62	33.59
14	40	75	25	1	0.25	0.15	2	0.55		3	27.82	32.11
15	40	75	25	3	0.2	0.15	3	0.55	0.15	3	**26.99**	33.23
16	15	90	25	4	0.2	0.25	2	0.5		1	29.97	33.50
17	75	100	10	3	0.25	0.2	1	0.4		1	38.64	38.59
18	25	75	15	1	0.3	0.15	3	0.3	0.15	2	27.76	31.98
19	25	50	0	4	0.25	0.12	3	0.3	0.5	3	27.97	32.06
20	30	50	20	1	0.4	0.15	2	0.6	0.2	1	32.06	34.20

Tabla 35. Resumen de las pruebas realizadas.

Donde:

- Nº: Número de prueba.
- Pob: Población.
- Rond: Rondas del algoritmo.
- Nº Op.: Número de operaciones.
- Sel (método de selección): 1 Selección por método de la ruleta, 2 Selección jerárquica, 3 Selección por torneos probabilísticos, 4 Selección por torneos deterministas.
- P: Probabilidad de mutación o cruce según el caso.
- F: Factor de mutación o cruce según el caso.
- T (tipo de cruce): 1 Cruce un punto, 2 Cruce dos puntos, 3 Cruce uniforme.
- Reempl (Tipo de reemplazo usado): 1 Reemplazo generacional, 2 Reemplazo estacionario, 3 Reemplazo estacionario con elementos aleatorios.

Operadores del algoritmo y características

Tamaño de la población: Generalmente, a un mayor número de individuos existe un mayor número de oportunidades de encontrar unos mejores resultados (ya que el espacio de búsqueda es mayor) además de disminuir las probabilidades de convergencia prematura. Sin embargo, cuanto más grande sea el tamaño de la población, más se incrementará el tiempo de ejecución. Con poblaciones entre 30 y 70 individuos conseguimos que la población mantenga una diversidad mínima que de buenos resultados sin que el tiempo de ejecución sea muy elevado.

Número de rondas. En cuanto al número de rondas, éste dependerá de la configuración de cada prueba. El número de rondas ha de ser suficiente para que el algoritmo tenga oportunidades de buscar y generar nuevos individuos, ya que con pocas rondas los individuos dependerán mucho del azar con el que creamos a la población inicial con la que comencemos. Nos hemos percatado que a partir de 75 rondas se puede alcanzar el mínimo porcentaje de error obtenido.

Operaciones: Resumiendo lo concluido en apartados anteriores, decir que nuestra idea inicial de crear individuos con operaciones de diferentes tipos parece errada ya que conviene elegir el porcentaje de cada uno de los tipos de operaciones desechando las divisiones. El número óptimo de operaciones es entre 15-30, con más de 30 aumenta considerablemente el tiempo de ejecución sin que se produzcan mejoras en los resultados.

Porcentaje de atributos de la población. Ya tratamos este tema anteriormente en 4.3.1.8 Conclusiones relativas al porcentaje de entradas. Recordar que si el algoritmo genético se ejecuta con los parámetros adecuados no importará mucho con qué porcentaje de entradas sea inicializada la población ya que éste en su búsqueda de los mejores individuos lo modificará con el paso de las rondas. Otro aspecto a tener en cuenta es que en muchas ocasiones los individuos que han prosperado y estaban mejor adaptados al problema se situaban en la franja que va desde el 55% al 70% de porcentaje de entradas. Por todo ello, en la mayoría de estas pruebas inicializamos la población al 50% ya que parece ser un buen punto de partida.

Métodos de selección. Todos los métodos de selección son óptimos pero, como es lógico, dependiendo de la prueba concreta y del tipo de resultados que se busquen se adaptarán mejor o peor. En la selección por torneos deterministas conviene que el tamaño de los torneos sea lo más pequeño posible con el fin de evitar que sólo prosperen los individuos mejor adaptados y la población converja. Por ello, para poblaciones menores a 100 individuos no conviene que el tamaño de los torneos sea mayor de 4 ó 5. A continuación se muestra una clasificación según la presión que ejercen cada uno de ellos sobre la población:

Selección jerárquica. Este puede ser el método más selectivo. Favorece a los individuos con mejor preparación dejando con pocas opciones de prosperar a los individuos menos preparados. Cuanto mayor sea la población menos opciones tendrán de prosperar los peores individuos.

Selección por torneos deterministas. Este método lo hemos situado en segundo lugar pero podría estar en cualquier posición dado que se puede graduar el tamaño de los torneos y con ello la fuerza que ejercerá. Si elegimos un tamaño de torneo muy bajo con poblaciones muy grandes, los individuos peor situados tendrán algunas opciones de seguir en

poblaciones futuras en cambio, si el tamaño de los torneos es alto para la población de que disponemos, prácticamente los individuos que seleccionemos serán los mejor preparados con el riesgo de acabar con una población muy similar y que converja prematuramente.

Selección por ruleta. Con este método estaremos dando un mayor número de oportunidades de ser seleccionados a los individuos mejor preparados, sin embargo los individuos peor situados seguirán teniendo opciones de promocionar.

Selección por torneos probabilísticos. Similar al método por torneos deterministas pero con la diferencia de que no siempre escogerá al individuo mejor preparado del torneo sino que en ocasiones seleccionará al peor. Con ello la población escogida contendrá una mezcla heterogénea de individuos.

Métodos de cruce. Es importante que probabilidad de cruce de los individuos no sea baja con el fin de que se crucen un número importante de individuos y así existan diferencias entre los individuos padres y los individuos hijos, de lo contrario la población no evolucionaría. El cruce uniforme dispone de un "factor" regulado que indica la probabilidad de cada uno de los genes de ser cruzado. En la mayoría de los casos hemos usado un factor del 25%. Los tres tipos de cruces implementados han demostrado su utilidad, si bien el más competente es el cruce uniforme dado que los dos descendientes hijos creados por este método mantienen un conjunto de genes totalmente aleatorio y desordenado de cada uno de sus padres (al contrario de lo que ocurre en el cruce un punto o de dos puntos).

Métodos de mutación e inversión. Ambos métodos son muy útiles para ensanchar el espacio de búsqueda ya que se encargan de introducir nuevas características a la población. Los resultados son buenos siempre y cuando las probabilidades de mutación e inversión no sean demasiado grandes, de este modo en cada ronda sólo sufren mutaciones e inversiones un pequeño conjunto de la población dejando al resto sin sufrirlas. Además en el caso de la mutación, debemos usar un "factor" bajo para que sólo se muten unos pocos atributos en cada individuo seleccionado ya que en caso contrario se formaría un individuo completamente nuevo perdiendo las propiedades de sus antecesores.

Operadores para las operaciones. Existen dos operadores: cruce uniforme y mutación. Evidentemente el comportamiento de éstos se asemeja al de sus semejantes que operan con atributos. El cruce intercambia las operaciones de los individuos y en el caso de la mutación consigue crear una nueva operación abriendo el espacio de búsqueda. Su importancia reside en que provocan variación en las operaciones y por tanto no dependeremos del azar con el hayan sido creadas. Los factores de mutación y cruce convienen que no sean excesivamente altos con el fin de no realizar demasiados cambios a

la vez en las operaciones y permitir así que perduren las operaciones de sus antecesores que eran buenas.

Métodos de reemplazo. De este operador ya se habla en 4.3.1.23 Conclusiones sobre el reemplazo. El aspecto más importante es que el reemplazo generacional sustituye a la totalidad de los individuos padres por sus descendientes perdiendo a individuos muy buenos, hecho un poco perjudicial. Los reemplazos estacionario y estacionario con elementos aleatorios parecen mejores ya que permiten conservar a estos individuos. Es importante ajustar bien sus parámetros.

4.3.2 Pruebas utilizando la representación por canales

Las pruebas comprendidas dentro de esta sección han sido realizadas utilizando la representación por canales de los individuos. Este pequeño conjunto de pruebas se realizaron con el fin de ver las diferencias surgidas al realizar el cambio de representación de los individuos.

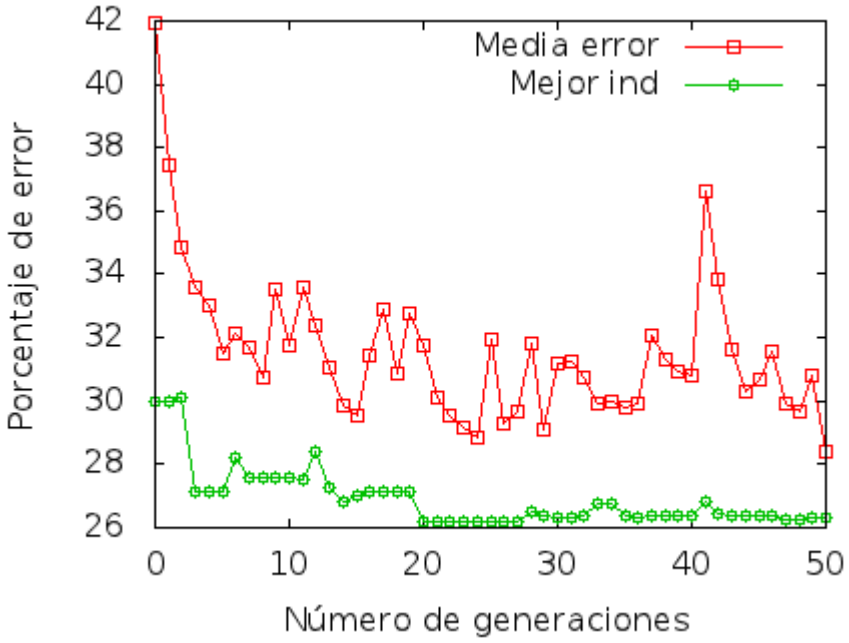
4.3.2.1 Prueba 21

CONFIGURACIÓN
Tamaño de la población = 30 individuos
Número de operaciones por individuo= 20
Número de rondas del algoritmo = 50
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Canales.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección jerárquica. • Cruce dos puntos y cruce de operaciones uniforme Probabilidad de cruce = 0.60 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.40 Factor de mutación = 0.15 Factor de mutación operación = 0.25 • Inversión: Probabilidad de inversión = 0.35 • Reemplazo generacional.
Función fitness 1. Solo participa el error de la red

Tabla 36. Configuración prueba 21.

Los parámetros empleados son los mismos que en la prueba 20. La representación por canales no variará tanto en los porcentajes de error sino en la forma de conseguirlos. Como vemos en la siguiente gráfica, los porcentajes de error en el entrenamiento son similares a los que se obtenían con la otra representación y los mejores individuos de la última ronda varían dependiendo la ejecución. En algunas de ellas hay errores de 26,29% y en

otras de 28,4% pero el cambio reside en que se produce un aumento desmesurado del porcentaje de entradas hasta llegar prácticamente al 100%, lo que indica que es necesario escoger prácticamente todos los canales para conseguir obtener estos resultados.



Gráfica 33. Mejor individuo y media del error en cada ronda (Prueba 21)

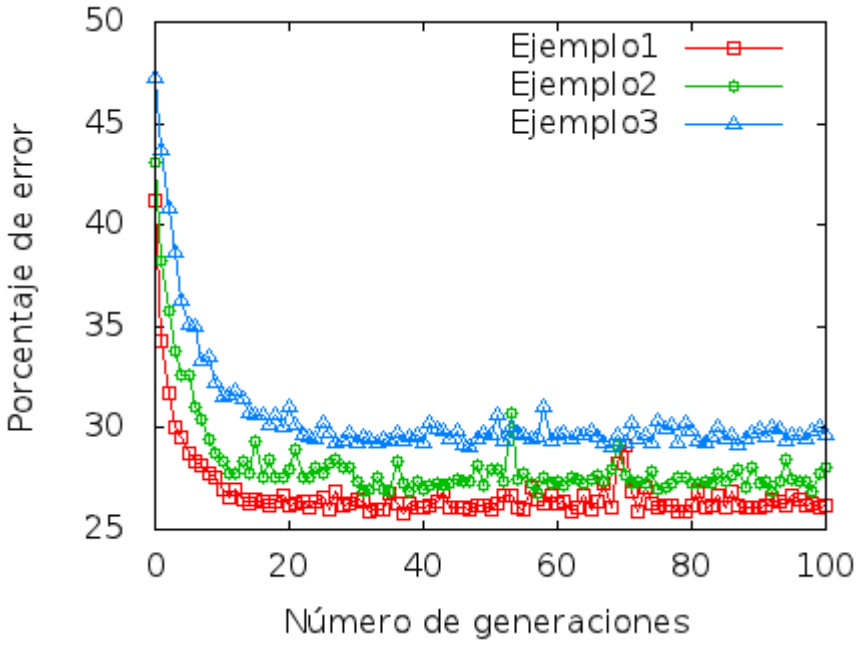
En cuanto al test, los resultados siguen siendo más elevados que en el entrenamiento y están sobre 32%-34% en los mejores casos.

4.3.2.2 Prueba 22

CONFIGURACIÓN
Tamaño de la población = 50 individuos
Número de operaciones por individuo= 15
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Canales.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección jerárquica • Cruce uniforme y cruce de operaciones uniforme Probabilidad de cruce = 0.30 Factor de cruce = 0.15 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.30 Factor de mutación = 0.15 Factor de mutación operación = 0.15 • Inversión: Probabilidad de inversión = 0.15 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 8 Padres elegidos al azar: 0 Mejores Hijos: 90 Hijos elegidos al azar: 2
Función fitness 1. Solo participa el error de la red

Tabla 37. Configuración prueba 22.

Usa los mismos parámetros que la prueba 8. A continuación vemos 3 ejecuciones más representativas de las 10 realizadas que nos servirán para ver las similitudes y diferencias que guardan entre ellas:

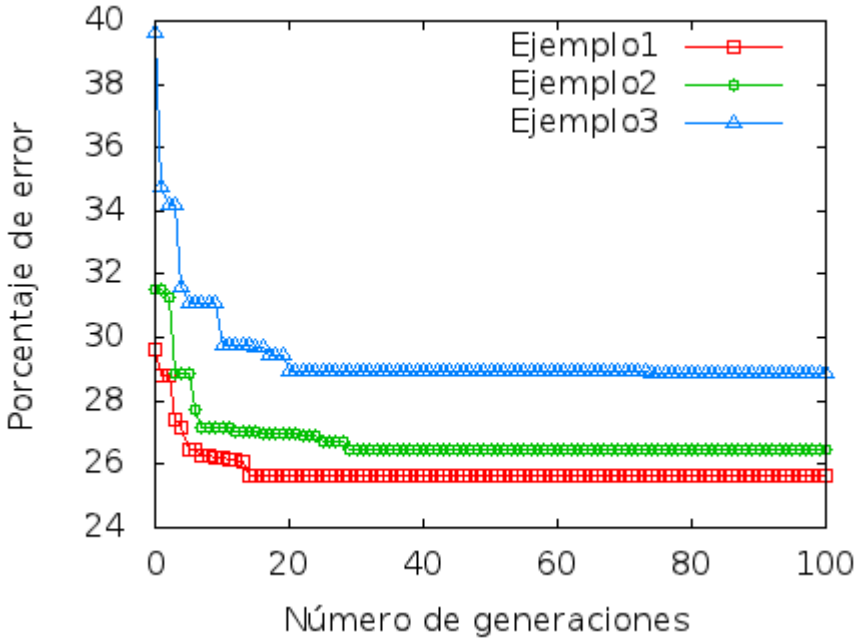


Gráfica 34. Media de error por ronda, varios ejemplos (Prueba 22)

En la gráfica 34 vemos como evoluciona la media del error en cada uno de los 3 casos. La curva de las gráficas es similar, la única diferencia es que cada una de ellas consigue bajar hasta un determinado punto. El ejemplo 1 (en rojo) es la que consigue disminuir más el error llegando al 26,5% de media, en cambio el ejemplo 3 (azul) disminuye menos siendo algo superior al 30%. Con lo cual vemos que, aunque las tres ejecuciones evolucionan de forma similar, cada una es capaz de reducir el error hasta cierto punto.

En la gráfica 35 vemos cuales son los mejores individuos en cada una de las rondas y la sorpresa viene de la ejecución 1 (en rojo) ya que su mejor individuo está por debajo del 26% (25,61%) y aparece en rondas tempranas. En cambio el ejemplo 3 se queda por encima del 28%. El ejemplo 2 se mantiene en valores cercanos al 26%. En todos los casos el algoritmo consigue que estos individuos aparezcan en rondas tempranas y a continuación se estanca sin encontrar mejores individuos.

Otro detalle que se observa es que el ejemplo 1 y el 2 mantienen porcentajes de entradas muy similares, en cambio el ejemplo 3 tiene una media de entradas un 10% menor. Es sorprendente ver como casualmente este ejemplo era el que tenía un porcentaje de error mayor.



Gráfica 35. Mejores individuos por rondas, varios ejemplos (Prueba 22)

En cuanto al test, los resultados varían como es lógico, pero siempre están por encima de los obtenidos en el entrenamiento. En el ejemplo 1 los resultados del test están sobre el 32% y el 34% siendo el mejor caso de 31,99%, en el ejemplo 2 sobre el 34% y en el ejemplo 3 entre el 32% y 35%.

4.3.2.3 Prueba 23

CONFIGURACIÓN
Tamaño de la población = 50 individuos
Número de operaciones por individuo= 25 Porcentaje sumas= 33 porcentaje restas= 34 Porcentaje multiplicación= 33 porcentaje división= 0
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.20
Representación usada: Canales.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por método de la ruleta • Cruce dos puntos y cruce de operaciones uniforme Probabilidad de cruce = 0.50 Factor de cruce operación = 0.30 • Mutación y mutación de operaciones Probabilidad de mutación = 0.20 Factor de mutación = 0.15 Factor de mutación operación = 0.25 • Inversión: Probabilidad de inversión = 0.10 • Reemplazo estacionario. Porcentaje padres = 90 Porcentaje hijos = 10
Función fitness 1. Solo participa el error de la red

Tabla 38. Configuración prueba 23.

En esta prueba además de la representación por canales hemos seleccionado las operaciones con el fin de conseguir unos mejores resultados. También inicializamos los individuos sólo con el 20% puesto que esta representación tiende a aumentar las entradas de forma desmesurada.

Se comienza en todos con una buena población inicial y aún así el algoritmo consigue reducir el porcentaje de error de los individuos poco a poco hasta llegar al 26,4% en los mejores casos. El descenso se produce principalmente en las primeras rondas y hasta alcanzar el 27%, a partir de ahí la reducción es mucho más lenta. Este comportamiento es común para todas las ejecuciones.

En lo que se refiere al aumento de entradas, independientemente de que hayamos inicializado los individuos al 20%, con el paso de las rondas éste aumenta hasta que en la ronda número 100, con una media del 80%, varios individuos utilizan el 100% de los canales. Es decir, de nuevo es necesario emplear casi la totalidad de los canales.

En el test se produce un ligero aumento del porcentaje de error con respecto a lo obtenido en el entrenamiento y los mejores individuos obtienen un 31,9%.

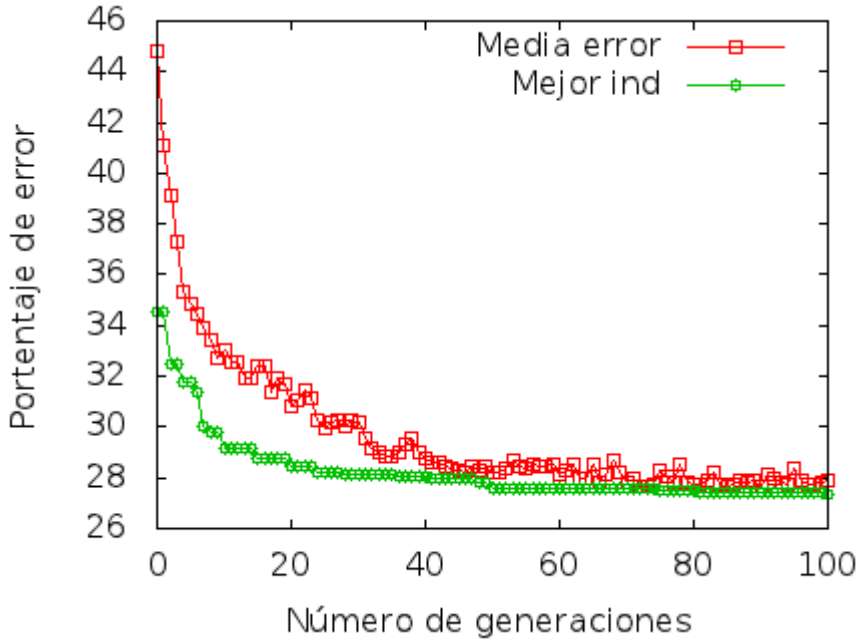
4.3.2.4 Prueba 24

CONFIGURACIÓN
Tamaño de la población = 40 individuos
Número de operaciones por individuo= 20 Porcentaje sumas= 35 Porcentaje restas= 40 Porcentaje multiplicación= 25 Porcentaje división= 0
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.10
Representación usada: Canales.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por torneos probabilísticos. • Cruce uniforme y cruce de operaciones uniforme Probabilidad de cruce = 0.50 Factor de cruce = 0.15 Factor de cruce operación = 0.30 • Mutación y mutación de operaciones Probabilidad de mutación = 0.15 Factor de mutación = 0.15 Factor de mutación operación = 0.30 • Inversión: Probabilidad de inversión = 0.10 • Reemplazo estacionario. Mejores Padres: 10 Padres elegidos al azar: 5 Mejores Hijos: 75 Hijos elegidos al azar: 10
Función fitness 1. Solo participa el error de la red

Tabla 39. Configuración prueba 24.

Al igual que la prueba anterior, en esta prueba seleccionaremos las operaciones con el fin de obtener los mejores resultados posibles. En esta

ocasión utilizaremos una población de 40 individuos e inicializaremos la población con sólo un 10% de los canales.



Gráfica 36. Mejor individuo y media de error por ronda (Prueba 24)

Como vemos en la gráfica anterior, independientemente de que el número de canales con los que hemos inicializado la población es muy bajo, desde un comienzo se produce una disminución del error, claro está que esta disminución del error se produce a la vez que se seleccionan a individuos con un mayor número de canales. En las últimas rondas obtenemos individuos que están sobre el 27,4% de error en el entrenamiento y sobre el 63% de porcentaje de entradas, con lo que observamos que el algoritmo ha encontrado individuos mucho más preparados a costa de incluir un mayor número de canales. Además, la población acaba pareciéndose mucho y convergiendo en un punto. En cuanto al porcentaje de error en el test, vemos en como en todos los casos los resultados rondan el 32% de error en adelante.

4.3.2.5 Aspectos sobre la representación por canales

En cuanto a resultados se refiere, los porcentajes de error en entrenamiento y test son similares a los obtenidos en la representación por atributos (sobre el 26% en el entrenamiento y algo superiores en el test). Además se sigue observando el estancamiento del algoritmo al llegar a esos valores. Sin embargo, para conseguir estos porcentajes de error son necesarias un mayor número de entradas, llegando a casos en los que se necesitan seleccionar el 100% de los canales. La única explicación encontrada es que el algoritmo tiende a seleccionar a los individuos con un mayor número de entradas (en este caso las entradas están agrupadas en

canales) dado que éstos son los individuos que obtienen unos mejores porcentajes de error, lo cual nos hace llegar a la conclusión de que no hay un canal o canales que introduzca ruido sino que todos los canales parecen ser importantes y sirven para mejorar el resultado.

A continuación, y a modo de resumen mostramos una tabla con las pruebas realizadas, su configuración más relevante y los resultados:

Nº	Pob	Rond	Nº Op	Sel	Mutación		Cruce			Reempl.	%Ent	%Test
					P	F	T	P	F			
21	30	50	20	2	0.4	0.15	2	0.6		1	36.62	40.18
22	50	100	15	1	0.3	0.15	3	0.3	0.15	3	27.96	33.79
23	50	100	25	1	0.2	0.15	2	0.5		2	**27.67**	33.64
24	40	100	20	3	0.15	0.15	3	0.5	0.15	2	27.88	**33.23**

Tabla 40. Resumen de las pruebas realizadas

Donde:

- Nº: Número de prueba.
- Pob: Población.
- Rond: Rondas del algoritmo.
- Nº Op.: Número de operaciones.
- Sel (método de selección): 1 Selección por método de la ruleta, 2 Selección jerárquica, 3 Selección por torneos probabilísticos, 4 Selección por torneos deterministas.
- P: Probabilidad de mutación o cruce según el caso.
- F: Factor de mutación o cruce según el caso.
- T (tipo de cruce): 1 Cruce un punto, 2 Cruce dos puntos, 3 Cruce uniforme.
- Reempl (Tipo de reemplazo usado): 1 Reemplazo generacional, 2 Reemplazo estacionario, 3 Reemplazo estacionario con elementos aleatorios.

En cuanto a los operadores genéticos, un detalle a tener en cuenta es que, dada las características de esta representación, operadores como la mutación, inversión y cruce afectarán muchísimo más al resultado puesto que una mutación de un canal equivale a 12 mutaciones de atributos con la representación anterior. El cruce uniforme tendrá un efecto similar al de la mutación. La inversión producirá grandes cambios al afectar a un número de genes mayor, etc. Por todo ello hay que tener especial cuidado con la aplicación de estos operadores y analizar bien la nueva situación disminuyendo el factor de mutación, de cruce o la probabilidad de inversión. De esto modo se evita que los cambios producidos de una generación a otra sean demasiados bruscos y se pierdan las características de los individuos anteriores por las cuales habían sido escogidos. En cambio, operadores de selección o reemplazo, así como aspectos relativos a las operaciones o número de rondas, no parecen verse afectados por el cambio de representación.

Con esto se deduce que nuestro clasificador no es capaz de reducir el error más allá de los límites encontrados. Del mismo modo, esta

representación disminuye o limita las posibilidades al reducir las opciones del porcentaje de entradas al obligar a que operadores como la mutación, cruce o inversión se hagan obligatoriamente por grupos de 12 atributos (canales) en lugar de aplicarse a un atributo solo. Dado que no parece haber ninguna razón que nos haga desechar algún canal concreto o que nos haga pensar que hay canales mejor preparados para realizar la clasificación que otros, nos parece más adecuado el uso de la representación por atributos.

4.3.3 Pruebas con la función fitness 2

En este bloque de pruebas utilizaremos la segunda función de fitness la cual tenía en cuenta el porcentaje de entradas en la valoración de los individuos (3.4.3.2 Función de aptitud mejorada (suma ponderada de objetivos)). De este modo se primará a los individuos con un número de entradas menor, que si bien no era uno de nuestros objetivos principales, la reducción de entradas puede ser importante ya que con ella se consigue una disminución del tiempo de ejecución. Dentro de este bloque incluiremos pruebas usando la representación por atributos y otras pruebas usando la representación por canales.

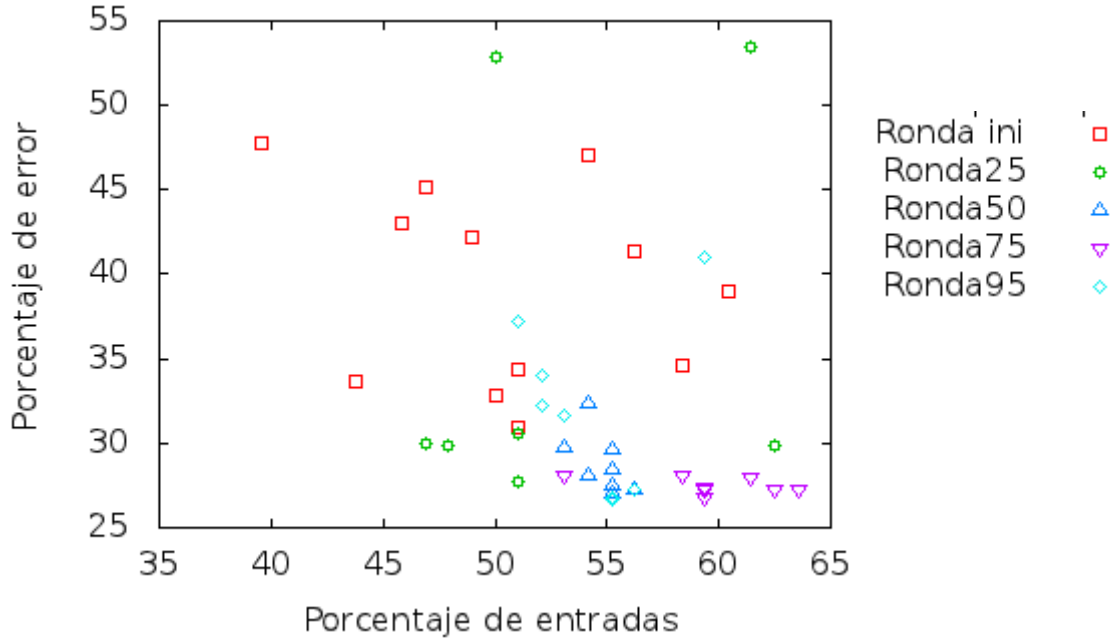
4.3.3.1.1 Prueba 25

CONFIGURACIÓN
Tamaño de la población = 12 individuos
Número de operaciones por individuo= 15
Número de rondas del algoritmo = 95
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por método de la ruleta • Cruce uniforme y cruce de operaciones uniforme Probabilidad de cruce = 0.30 Factor de cruce = 0.15 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.30 Factor de mutación = 0.15 Factor de mutación operación = 0.15 • Inversión: Probabilidad de inversión = 0.18 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 16 Padres elegidos al azar: 0 Mejores Hijos: 75 Hijos elegidos al azar: 9
Función fitness 2. Peso error red: 0.95 Peso entradas:0.05

Tabla 41. Configuración prueba 25.

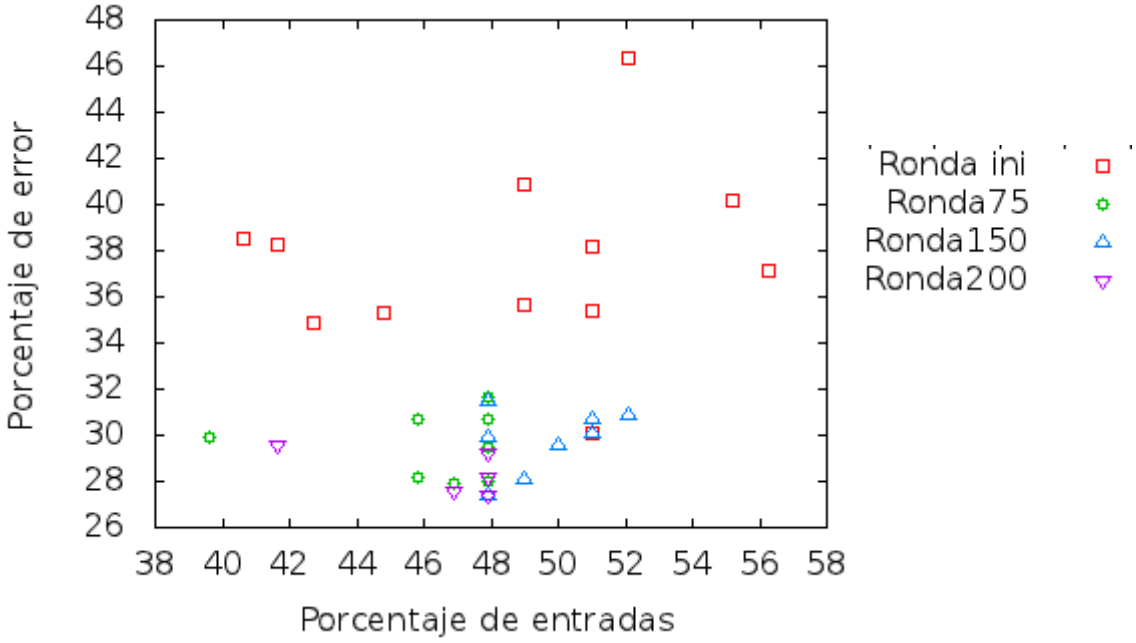
En esta prueba utilizaremos unos pesos de 0.95 para el error y de 0.05 para el porcentaje de entradas, así la mayoría de la valoración seguirá estando en manos del porcentaje de error. En la gráfica vemos cómo el porcentaje de entradas sigue aumentando a la vez que disminuye el error

pero también se aprecia cómo los individuos de la ronda 95 disminuyen este porcentaje con respecto a los de la ronda 75. Los resultados en el entrenamiento de la última ronda están sobre el 26,69% en adelante, y en cuanto al test, los mejores individuos obtienen una valoración del 30,5%.



Gráfica 37. Evolución de la población (Prueba 25)

Con el fin de ver mejor los efectos producidos en la prueba de esta función fitness, decidimos repetirla con un número de rondas aún mayor (200). El peso de las entradas sigue siendo solamente de 0,05 pero es lo suficiente para hacer que el algoritmo no aumente el porcentaje de entradas y éstas se mantengan en valores cercanos a las que fueron inicializadas.



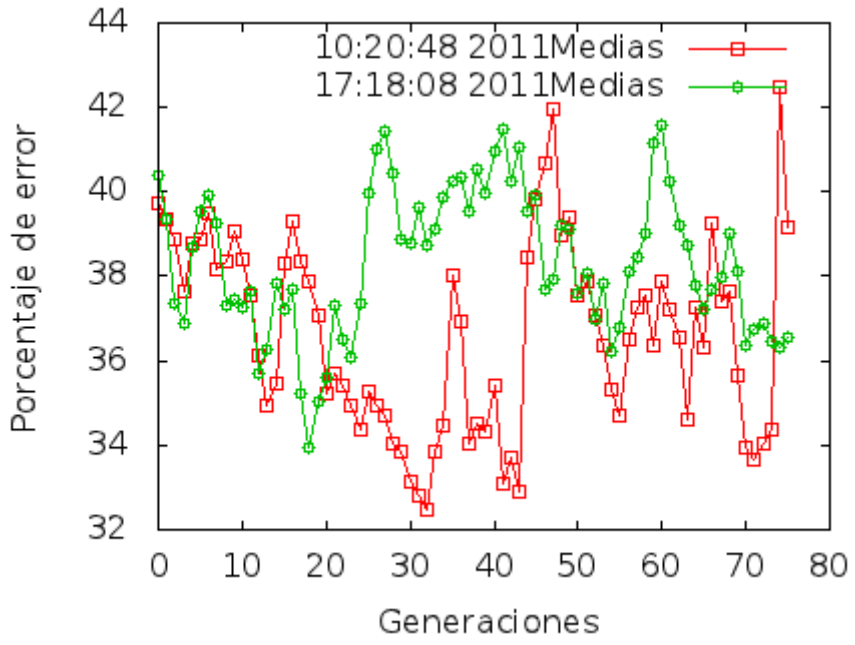
Gráfica 38. Evolución de la población, 200 rondas. (Prueba 25)

Visualmente se observa este efecto de forma más clara en la gráfica 38 donde vemos cómo según avanzan las generaciones éstas disminuyen el porcentaje de error sin aumentar el porcentaje de entradas o, lo que es lo mismo, se desplazan solamente hacia la parte baja de la gráfica. En cuanto al porcentaje de error en el entrenamiento, éste sigue quedándose estancando una vez alcanza valores cercanos, para en este caso, al 27%. En todas las ejecuciones realizadas, los mejores individuos de la última ronda reciben una valoración de cómo mucho 27,4% en el entrenamiento y de 31,3% en el test.

4.3.3.1.2 Prueba 26

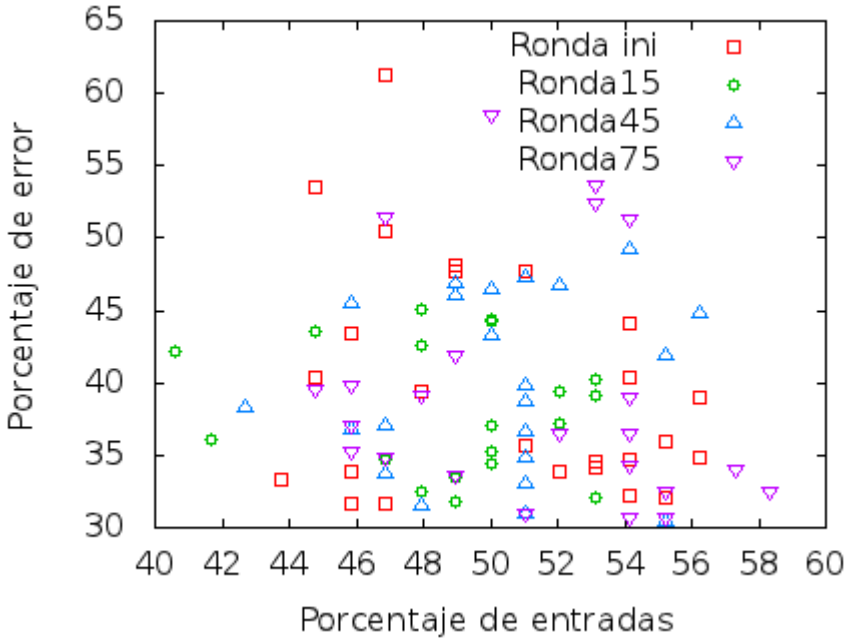
CONFIGURACIÓN
Tamaño de la población = 25 individuos
Número de operaciones por individuo= 15
Número de rondas del algoritmo = 75
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por método de la ruleta • Cruce uniforme y cruce de operaciones uniforme Probabilidad de cruce = 0.30 Factor de cruce = 0.15 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.30 Factor de mutación = 0.15 Factor de mutación operación = 0.15 • Inversión: Probabilidad de inversión = 0.18 • Reemplazo generacional.
Función fitness 2. Peso de entradas= 0.05 Peso del error= 0.95

Tabla 42. Configuración prueba 26.



Gráfica 39. Media de entradas, 2 ejemplos (Prueba 26)

En esta prueba la novedad reside en que hemos usado también un reemplazo generacional. Los resultados obtenidos son similares a los de otras pruebas en las que hemos usado este tipo de reemplazo. Como podemos observar en la gráfica 39, con este tipo de reemplazo no hay ningún patrón y en cada una de las ejecuciones los resultados varían notablemente fruto de la sustitución completa de la población por sus hijos. La población no evoluciona ronda a ronda y los resultados de todas ellas son similares a los de la ronda inicial. En la siguiente gráfica vemos varias generaciones y sus resultados.



Gráfica 40. Evolución de la población (Prueba 26)

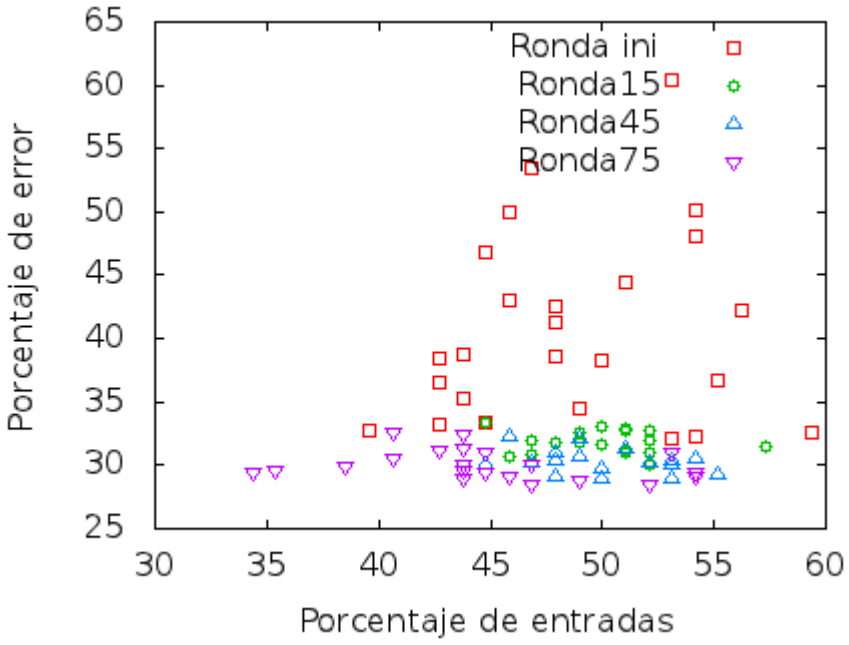
4.3.3.1.3 Prueba 27

CONFIGURACIÓN
Tamaño de la población = 25 individuos
Número de operaciones por individuo= 15
Número de rondas del algoritmo = 75
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por método de la ruleta • Cruce uniforme y cruce de operaciones uniforme Probabilidad de cruce = 0.30 Factor de cruce = 0.15 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.30 Factor de mutación = 0.15 Factor de mutación operación = 0.15 • Inversión: Probabilidad de inversión = 0.18 • Reemplazo estacionario. Hijos: 85 Padres: 15
Función fitness 2. Peso de entradas= 0.1 Peso del error= 0.9

Tabla 43. Configuración prueba 27.

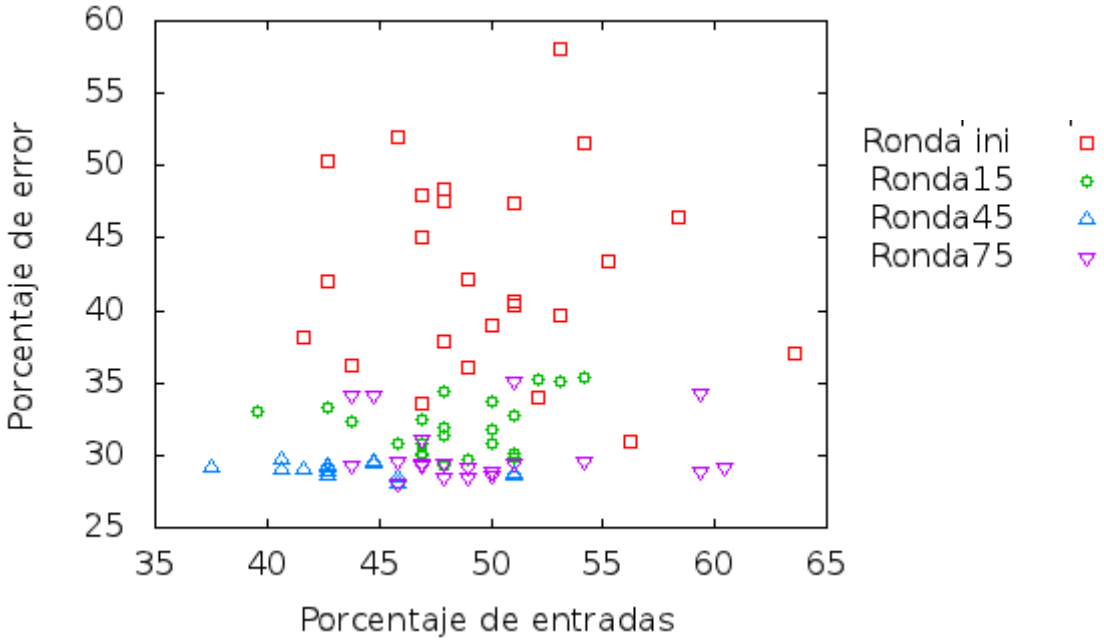
En esta prueba hemos cambiado el reemplazo generacional por un reemplazo estacionario y además utilizamos un peso para las entradas de 0.10 y un peso para el error producido de 0.90, salvo estos dos aspectos el resto de los parámetros y operadores son iguales a los de la prueba 26.

En lo que a resultados se refiere, vemos que el cambio de los pesos ha hecho que el porcentaje de entradas de los individuos obtenga más importancia haciendo que se valore más este objetivo. Esto produce un descenso del número de entradas en algunos casos hasta el 35%. Además de este descenso, el porcentaje de error también se consigue reducir pero al contrario de lo que ocurría en otras pruebas éste solo se reduce en la totalidad de las ejecuciones hasta el 28%, dado que quizás se centre demasiado en la búsqueda de individuos con un menor porcentaje de entradas dejando a un lado la reducción del error. En cuanto al test, el error producido está entre el 31% y el 33% en todos los casos analizados salvo unos pocos individuos que superan estos valores.



Gráfica 41. Evolución de la población (Prueba 27)

Además, repetimos esta misma prueba variando el porcentaje de los pesos de la función fitness (peso para las entradas de 0.05 y un peso para el error producido de 0.95) con el fin de que éstos valoren menos el porcentaje de entradas y ver si ello afectaba a los resultados. Con el cambio de pesos la población se queda con unos porcentajes de entradas algo superiores (sobre el 45% en la última ronda para la mayoría de las ejecuciones), resultados ligeramente superiores a los obtenidos en la prueba anterior. En cuanto a los porcentajes de error, los mejores individuos en el entrenamiento obtienen un 27,5% y un 31% de error, pequeña mejoría comparado con el caso anterior.



Gráfica 42. Evolución de la población cambio pesos (Prueba 27)

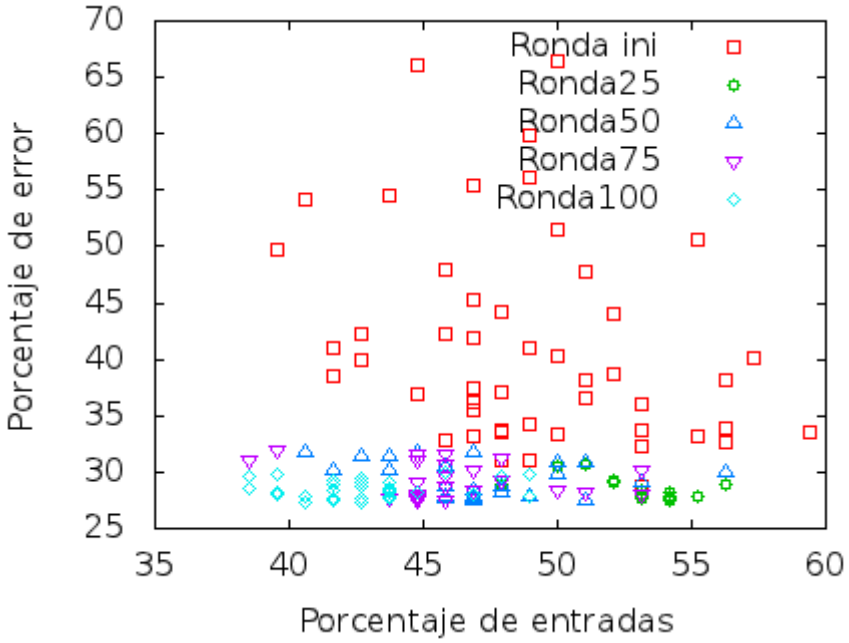
4.3.3.1.4 Prueba 28

CONFIGURACIÓN
Tamaño de la población = 50 individuos
Número de operaciones por individuo= 15
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Genes.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección jerárquica • Cruce uniforme y cruce de operaciones uniforme Probabilidad de cruce = 0.30 Factor de cruce = 0.15 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.30 Factor de mutación = 0.15 Factor de mutación operación = 0.15 • Inversión: Probabilidad de inversión = 0.15 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 8 Padres elegidos al azar: 0 Mejores Hijos: 90 Hijos elegidos al azar: 2
Función fitness 2. Peso entradas= 0.05 Peso error= 0.95

Tabla 44. Configuración prueba 28.

Los parámetros son como los de la prueba 8 salvo por la diferencia de la función fitness. En la gráfica 43 observamos como la evolución de las distintas rondas hace que inicialmente los individuos se desplacen hacia la parte baja de la gráfica (disminuyendo el porcentaje de error) para a continuación las nuevas generaciones se desplacen hacia la izquierda de la

gráfica (reduciendo así el porcentaje de entradas). Los resultados están sobre el 27% de error en entrenamiento (27,29% mejor individuo).



Gráfica 43. Evolución de la población (Prueba 28)

En cuanto a los resultados obtenidos en el test, los resultados varían entre el 30% y el 32% siendo el mejor individuo encontrado de 30,30%.

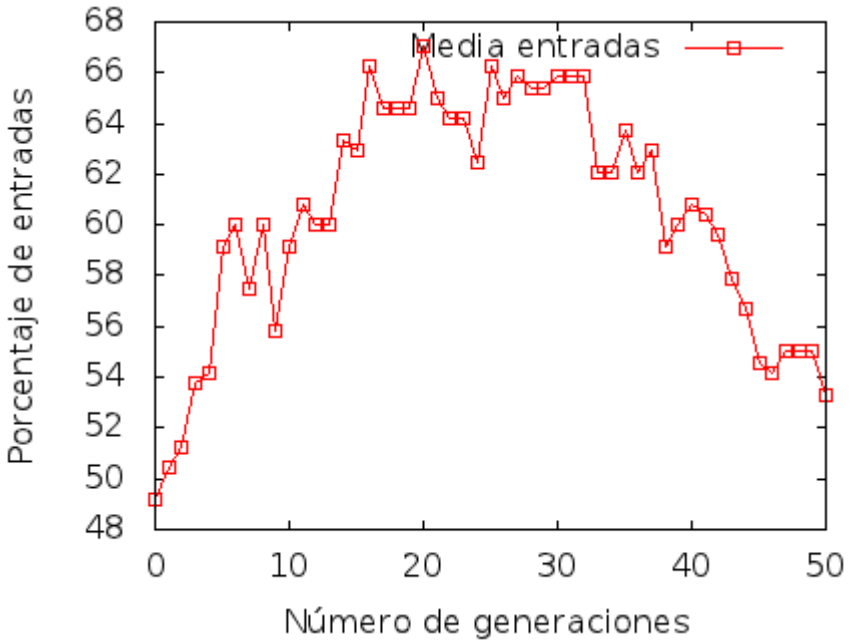
Como observación habría que añadir que todas las ejecuciones de esta prueba han seguido el mismo esquema, pero en alguna de ellas la población ha llegado al 26,62% de error de entrenamiento manteniéndose en el 52% de porcentaje de entradas, en otras la población no ha conseguido bajar del 28%. Las ejecuciones en las que los individuos obtenían un menor porcentaje de error suelen ser aquellas que tienen mayor porcentaje de entradas.

4.3.3.1.5 Prueba 29

En esta ocasión utilizaremos la representación por canales con la función de fitness 2. Le hemos dado a los pesos de la función los valores de 0.95 para el error y de 0.05 para las entradas. Con ello pretendemos evitar el efecto que se producía en las anteriores pruebas en las que usábamos esta representación dónde el porcentaje de entradas se nos disparaba hasta el máximo posible y ver si evitando este efecto conseguimos unos porcentajes de error similares o por lo menos aceptables. El resto de la configuración de la prueba es exactamente igual que la de la prueba 21. En la siguiente gráfica vemos la evolución del porcentaje de entradas en cada una de las generaciones de la prueba:

CONFIGURACIÓN
Tamaño de la población = 30 individuos
Número de operaciones por individuo= 20
Número de rondas del algoritmo = 50
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Canales.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección jerárquica. • Cruce dos puntos y cruce de operaciones uniforme Probabilidad de cruce = 0.60 Factor de cruce operación = 0.20 • Mutación y mutación de operaciones Probabilidad de mutación = 0.40 Factor de mutación = 0.15 Factor de mutación operación = 0.25 • Inversión: Probabilidad de inversión = 0.35 • Reemplazo generacional.
Función fitness 2. Peso error = 0.95 Peso de entradas = 0.05

Tabla 45. Configuración prueba 29.

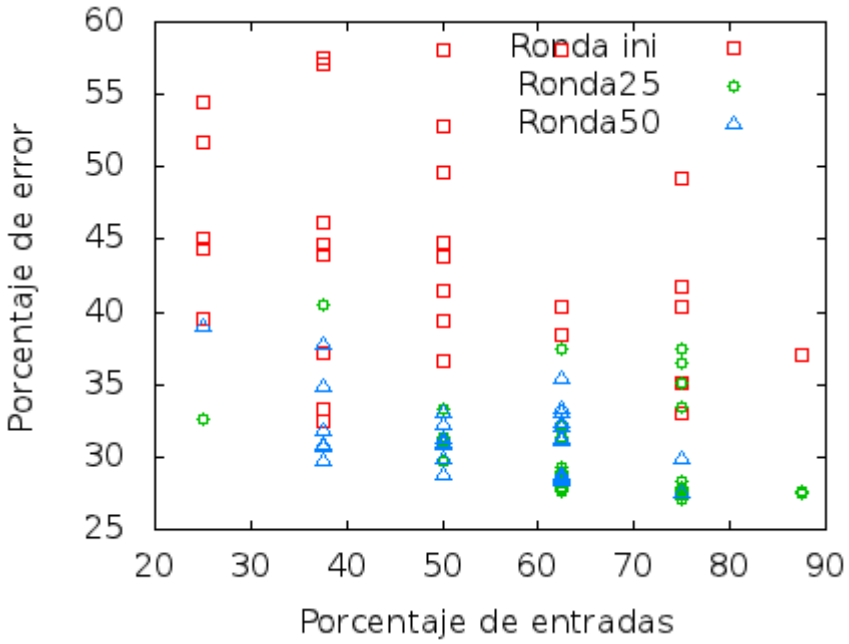


Gráfica 44. Media de entradas por generación (Prueba 29)

Este porcentaje de entradas parece aumentar en un principio pero, en este caso concreto que estamos analizando, sobre la ronda 30 inicia un descenso hasta valores cercanos a los que fueron inicializados. En cuanto a los porcentajes de error, en gráfica 45 se observa cómo ha ido descendiendo el porcentaje de error, además los individuos con mayor porcentaje de entradas parecen tener una ligera ventaja y conseguir unos errores ligeramente inferiores. También vemos con detalle a individuos en la ronda 25 con muy buen porcentaje de error pero que no han prosperado dado que tenían un mayor número de entradas y por ello han sido “penalizados” por la función fitness. Estos individuos estaban sobre el 27%.

Los mejores individuos de la ronda 50 obtienen una valoración de 28,63% en el entrenamiento, en cuanto al test los resultados rondan el 30%-31% siendo el mejor individuo de 29,79%.

En el resto de ejecuciones se producen casos similares, cuando empleamos el reemplazo generacional la variabilidad entre las distintas ejecuciones de una misma prueba es mayor. Sin embargo, este reemplazo no es recomendable pues con el uso de esta configuración no logramos evitar la pérdida de individuos muy válidos.



Gráfica 45. Evolución de la población (Prueba 29)

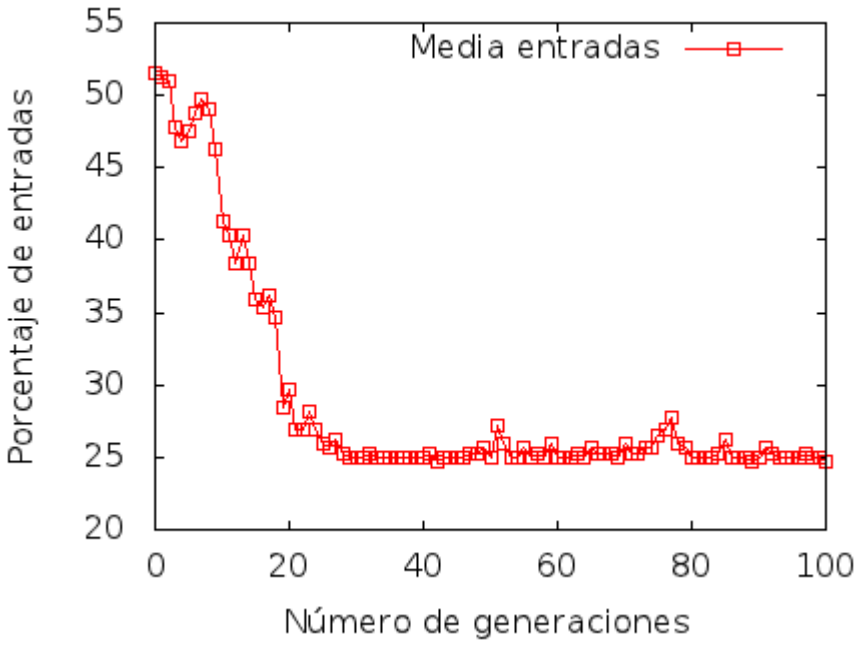
4.3.3.1.6 Prueba 30

Ahora se realiza una selección de operaciones. Se espera obtener unos buenos resultados desde la población inicial, como ya ha ocurrido en casos anteriores, y a partir de ahí ver si los resultados consiguen evolucionar favorablemente.

En todas las ejecuciones de esta prueba se observan reducciones drásticas del porcentaje de entradas tal y como se ve en la gráfica 46 hasta que acaba rondando el 25%. Esto es gracias a la función fitness 2. Pero esta reducción del porcentaje de entradas no sería competente si no va acompañada de una reducción del error hasta porcentajes aceptables. En esta prueba, parece que hemos conseguido los dos objetivos y, como vemos en la gráfica 47, también se produce una reducción del error hasta obtener unos porcentajes de error en el entrenamiento del 27% (27,68% el mejor individuo en todas las ejecuciones).

CONFIGURACIÓN
Tamaño de la población = 40 individuos
Número de operaciones por individuo= 25 Porcentaje sumas= 35 Porcentaje restas= 40 Porcentaje multiplicación= 25 Porcentaje división= 0
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.5
Representación usada: Canales.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por torneo probabilística. • Cruce uniforme y cruce de operaciones uniforme Probabilidad de cruce = 0.55 Factor de cruce = 0.15 Factor de cruce operación = 0.30 • Mutación y mutación de operaciones Probabilidad de mutación = 0.15 Factor de mutación = 0.15 Factor de mutación operación = 0.30 • Inversión: Probabilidad de inversión = 0.10 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 10 Padres elegidos al azar: 5 Mejores Hijos: 75 Hijos elegidos al azar: 10
Función fitness 2. Peso error = 0.95 Peso de entradas = 0.05

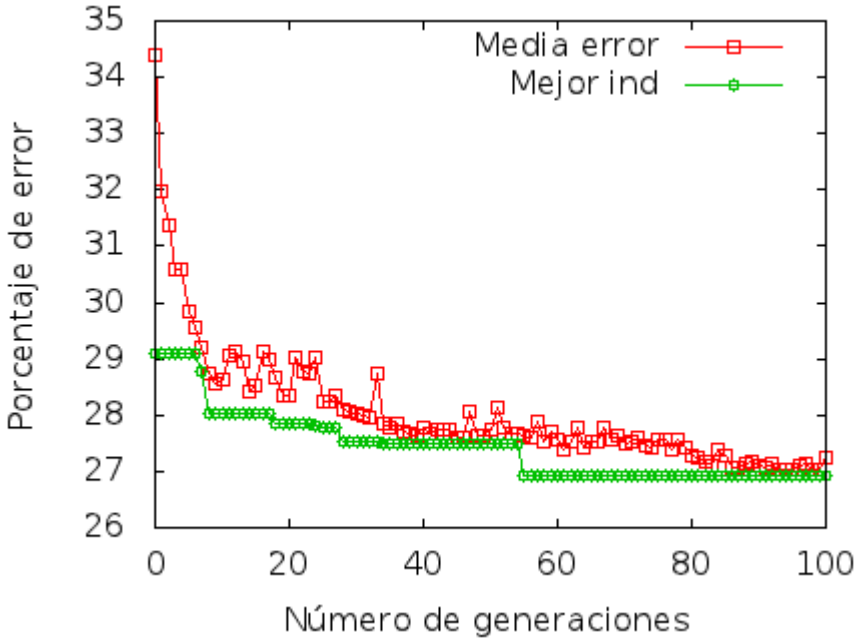
Tabla 46. Configuración prueba 30.



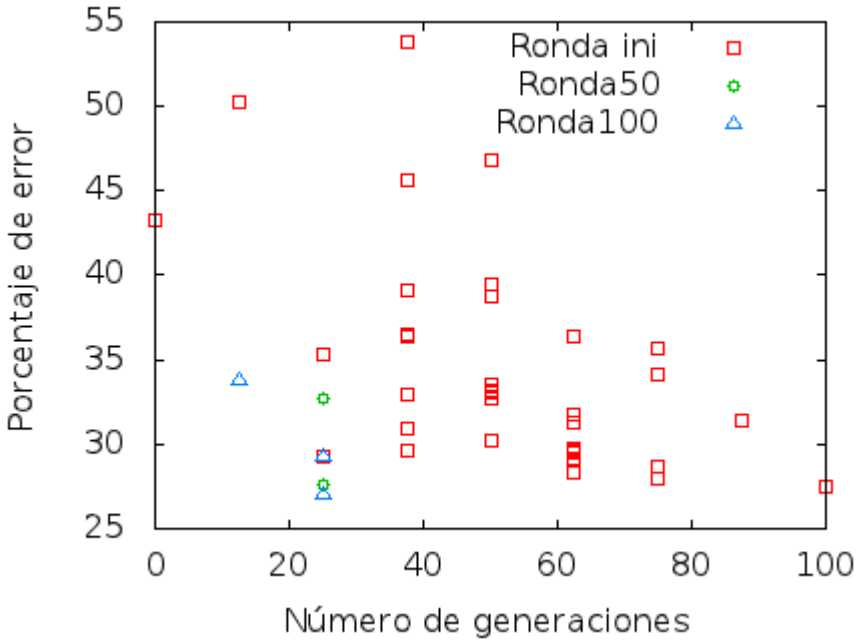
Gráfica 46. Media de entradas por ronda (Prueba 30)

Sin embargo, como muestra la gráfica 48, en la mayoría de los casos la población converge en un punto, lo que significa que se pierde la heterogeneidad en la población y que prácticamente los 40 individuos de la población son iguales.

En cuanto al test, los resultados están siempre entre el 30% y el 32%.



Gráfica 47. Mejor individuo y media de error por ronda (Prueba 30).

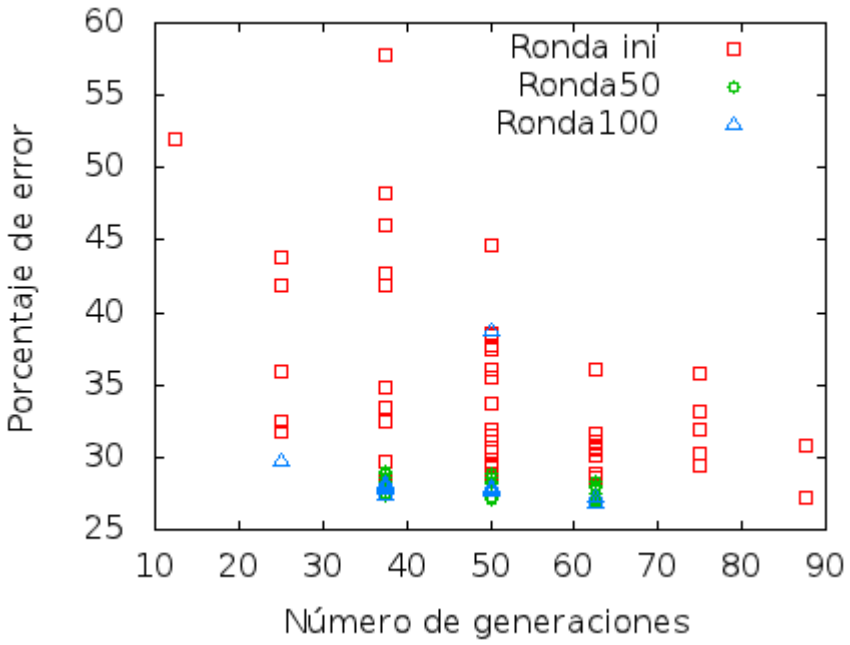


Gráfica 48. Evolución de la población (Prueba 30)

4.3.3.1.7 Prueba 31

CONFIGURACIÓN
Tamaño de la población = 50 individuos
Número de operaciones por individuo= 25. Porcentaje sumas= 45 Porcentaje restas= 45 Porcentaje multiplicación= 10 Porcentaje división= 0
Número de rondas del algoritmo = 100
Porcentaje de atributos con los que comienza la población= 0.50
Representación usada: Canales.
Operadores genéticos: (EJECUCCIÓN CONJUNTA) • Selección por método de la ruleta. • Cruce dos puntos y cruce de operaciones uniforme Probabilidad de cruce = 0.60 Factor de cruce operación = 0.30 • Mutación y mutación de operaciones Probabilidad de mutación = 0.25 Factor de mutación = 0.15 Factor de mutación operación = 0.30 • Inversión: Probabilidad de inversión = 0.15 • Reemplazo estacionario con elementos aleatorios. Mejores Padres: 12 Padres elegidos al azar: 8 Mejores Hijos: 70 Hijos elegidos al azar: 10
Función fitness 2. Peso de entradas = 0.05 Peso del error = 0.95

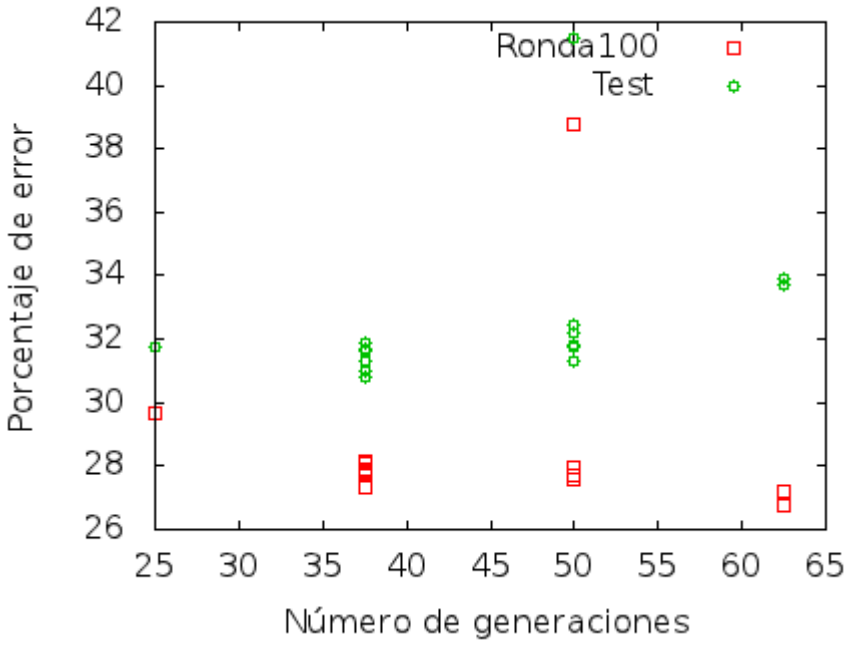
Tabla 47. Configuración prueba 31.



Gráfica 49. Evolución de la población (Prueba 31)

Como vemos en la gráfica 49, volvemos a obtener unos resultados buenos ya que reducimos el error ronda a ronda hasta llegar al 27% (27,30% el mejor individuo en todas las ejecuciones realizadas) en la ronda 100 y además conseguimos que el porcentaje de entradas no se dispare (la mayoría de la población tiene un 37,5% de entradas y algunos individuos un 50%). También parece que en esta prueba se ha conservado algo más la

heterogeneidad en la población consiguiendo que ésta no sea completamente igual en las últimas rondas.



Gráfica 50. Entrenamiento y test de la última ronda (Prueba 31)

En lo que se refiere a los resultados obtenidos en el test, la mayoría de las ejecuciones se encuentran entre el 30% y el 32% habiendo sólo algunos individuos aislados que superan estos valores.

4.3.3.1.8 Impresiones sobre la función de fitness 2

El análisis de la experimentación llevada cabo indica que esta función de fitness puede ser muy útil, sobre todo en el caso de la representación por canales. Usándola con esta representación hemos conseguido evitar el aumento de entradas que se producía con la anterior función fitness manteniendo unos porcentajes de error similares, este hecho es favorable ya que reducimos de forma importante el tiempo de ejecución. En cuanto a la representación por atributos, los resultados son muy similares a los obtenidos aunque en algunos casos los porcentajes de error han sido ligeramente superiores dado que esta función, comparada con la anterior, resta algo de importancia a este aspecto.

Pero como ya se comentó anteriormente, este pequeño conjunto de pruebas sólo nos ha servido para intuir los resultados que se dan, se considera necesario un estudio mayor sobre el tema que se propone para líneas futuras.

La tabla que mostramos a continuación contiene las pruebas realizadas en este apartado y sus datos y resultados más importantes:

Nº	Pob	Rond	Nº Op	Sel	Mutación		Cruce			Reempl.	%Ent	%Test
					P	F	T	P	F			
25	12	95	15	1	0.3	0.15	3	0.3	0.15	3	30.40	30.53
26	25	75	15	1	0.3	0.15	3	0.3	0.15	1	37.85	38.81
27	25	75	15	1	0.3	0.15	3	0.3	0.15	2	29.49	32.07
28	50	100	15	2	0.3	0.15	3	0.3	0.15	3	28.29	31.47
29	30	50	20	2	0.4	0.15	2	0.6	0.2	1	31.64	33.41
30	40	100	25	3	0.15	0.15	3	0.55	0.15	3	**27.59**	**31.31**
31	50	100	25	1	0.25	0.15	2	0.6	0.3	3	27.83	31.52

Tabla 48. Resumen de las pruebas realizadas

Donde:

- Nº: Número de prueba.
- Pob: Población.
- Rond: Rondas del algoritmo.
- Nº Op.: Número de operaciones.
- Sel (método de selección): 1 Selección por método de la ruleta, 2 Selección jerárquica, 3 Selección por torneos probabilísticos, 4 Selección por torneos deterministas.
- P: Probabilidad de mutación o cruce según el caso.
- F: Factor de mutación o cruce según el caso.
- T (tipo de cruce): 1 Cruce un punto, 2 Cruce dos puntos, 3 Cruce uniforme.
- Reempl (Tipo de reemplazo usado): 1 Reemplazo generacional, 2 Reemplazo estacionario, 3 Reemplazo estacionario con elementos aleatorios.

Capítulo 5

Presupuesto

5.1 Introducción

En este capítulo calcularemos los costes y la duración de la realización del proyecto. En él detallaremos las distintas fases en que se divide el proyecto: su duración tanto global como de cada fase concreta, los medios necesarios para su desarrollo, los costes tanto de personal como materiales de la realización de este proyecto, etc.

Dividiremos este capítulo en cuatro secciones diferentes:

- Duración global del proyecto, las diferentes tareas o actividades que lo componen y que han sido necesarias realizar así como la duración de éstas.
- Costes personales del proyecto.
- Costes del material empleado para su realización.
- Coste global del proyecto.

5.2 Duración del proyecto y fases de desarrollo

Como podemos ver en las tablas e imágenes siguientes, el proyecto comenzó el día 9 de Noviembre de 2009 y ha finalizado el día 25 de Noviembre de 2011. Durante todo este tiempo, no ha existido una jornada de trabajo estable ya que por motivos laborales no siempre se pudo dedicar el

mismo tiempo al proyecto, en ocasiones unas semanas empleábamos una jornada de 4 horas al día para su realización y en cambio en otras semanas nos era imposible realizar tarea alguna. Además de esto durante los meses de Julio y Agosto no se realizó ninguna tarea o actividad relacionada con el proyecto. Por todo ello, las horas empleadas para la realización han sido 895 horas.

En la siguiente tabla podemos ver las distintas fases en que se divide el proyecto realizado, las fechas de inicio y fin de cada una de las fases así como las horas de trabajo destinadas a ellas. Como veremos más adelante, estas fases estarán formadas a su vez por las diferentes tareas necesarias para realizar el proyecto.

Código	Nombre de la fase	Comienzo	Fin	Trabajo
1	Inicio del proyecto	lun 09/11/09	lun 09/11/09	0 horas
2	Análisis	lun 09/11/09	jue 17/12/09	53 horas
3	Diseño de la solución	jue 17/12/09	mar 23/03/10	77 horas
4	Desarrollo de la solución	lun 11/01/10	mié 10/11/10	325 horas
5	Experimentación y evaluación de la solución	lun 01/02/10	mié 30/03/11	190 horas
6	Desarrollo de la documentación	mié 30/03/11	vie 25/11/11	250 horas
7	Fin de proyecto	vie 25/11/11	vie 25/11/11	0 horas

Tabla 49. Fases en las que se subdivide el proyecto, duración y trabajo.

De modo más gráfico, las distintas fases del proyecto y su orden son:

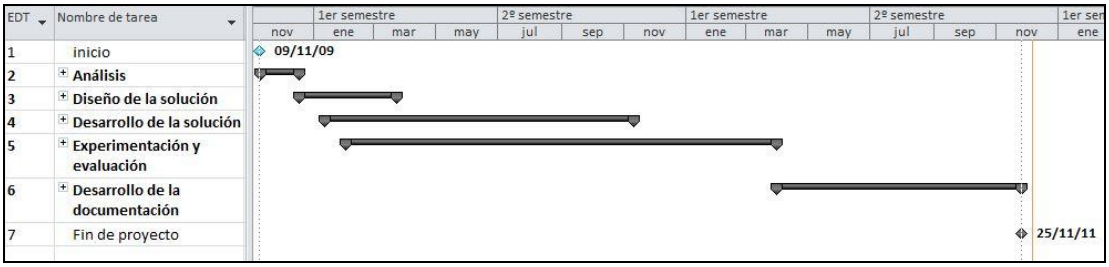


Figura 51. Fases del proyecto y período de tiempo que estuvieron activas

CAPÍTULO 5 - PRESUPUESTO

El conjunto de tareas que forman las fases de desarrollo del proyecto es:

Código	Nombre de la tarea	Comienzo	Fin
1	Inicio del proyecto	lun 09/11/09	lun 09/11/09
2	**Análisis**	**lun 09/11/09**	**jue 17/12/09**
2.1	Estudio del problema	lun 09/11/09	jue 19/11/09
2.2	Búsqueda de información	jue 19/11/09	mar 01/12/09
2.3	Estudio posibles soluciones	mar 01/12/09	jue 10/12/09
2.4	Elección parámetros solución	jue 10/12/09	jue 17/12/09
3	**Diseño de la solución**	**jue 17/12/09**	**mar 23/03/10**
3.1	Diseño general solución	jue 17/12/09	vie 01/01/10
3.2	Diseño de varios clasificadores	vie 01/01/10	vie 08/01/10
3.3	Elección del clasificador	vie 26/02/10	vie 05/03/10
3.4	Diseño operadores algoritmo genético	vie 05/03/10	mar 23/03/10
4	**Desarrollo de la solución**	**lun 11/01/10**	**mié 10/11/10**
4.1	Implementación de los clasificadores	lun 11/01/10	vie 29/01/10
4.2	Implementación algoritmo genético	mié 24/03/10	jue 14/10/10
4.3	Unión del clasificador al algoritmo genético	mar 02/11/10	mié 10/11/10
5	**Experimentación y evaluación de la solución**	**lun 01/02/10**	**mié 30/03/11**
5.1	Comprobación funcionamiento clasificadores	lun 01/02/10	vie 05/02/10
5.2	Pruebas clasificador 1	vie 05/02/10	mar 16/02/10
5.3	Pruebas clasificador 2	mié 17/02/10	vie 26/02/10
5.4	Comprobación operadores algoritmo genético	jue 14/10/10	lun 01/11/10
5.5	Comprobación funcionamiento alg. genético	mié 10/11/10	jue 09/12/10
5.6	Pruebas globales	jue 09/12/10	mié 30/03/10
6	**Desarrollo de la documentación**	**mié 30/03/11**	**vie 25/11/11**
6.1	Búsqueda de documentación especifica	mié 30/03/11	jue 14/04/11
6.2	Textos	jue 14/04/11	vie 21/10/11
6.3	Ilustraciones	lun 24/10/11	vie 25/11/11
7	Fin de proyecto	vie 25/11/11	vie 25/11/11

Tabla 50. Tareas del proyecto desglosadas por fases.

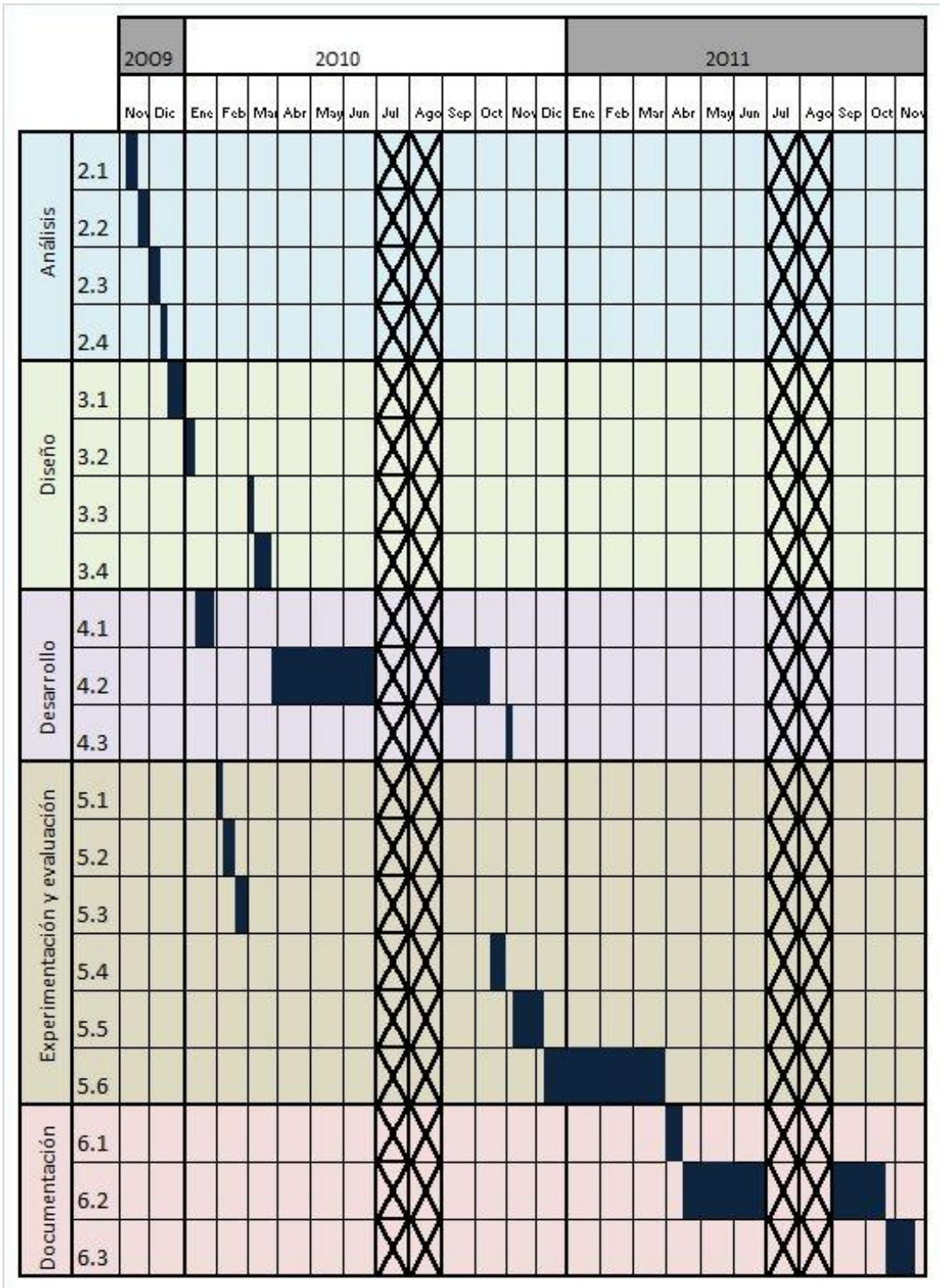


Tabla 51. Tareas del proyecto ordenadas en el tiempo.

En la tabla anterior vemos el orden en el que se realizaron las tareas y la duración de éstas en el calendario. A simple vista podemos observar la línea que sigue el desarrollo del proyecto. Una vez que finalizamos la fase de

análisis de todo el proyecto, realizamos el diseño, implementación y las pruebas de la parte del clasificador. A continuación, realizamos el diseño, implementación y las pruebas del algoritmo genético. Después unimos ambas partes obteniendo nuestra solución completa a falta de la experimentación o conjunto de pruebas necesarias para analizar los resultados conseguidos y la realización de la documentación de todo el proyecto. El esquema del desarrollo global es el siguiente:

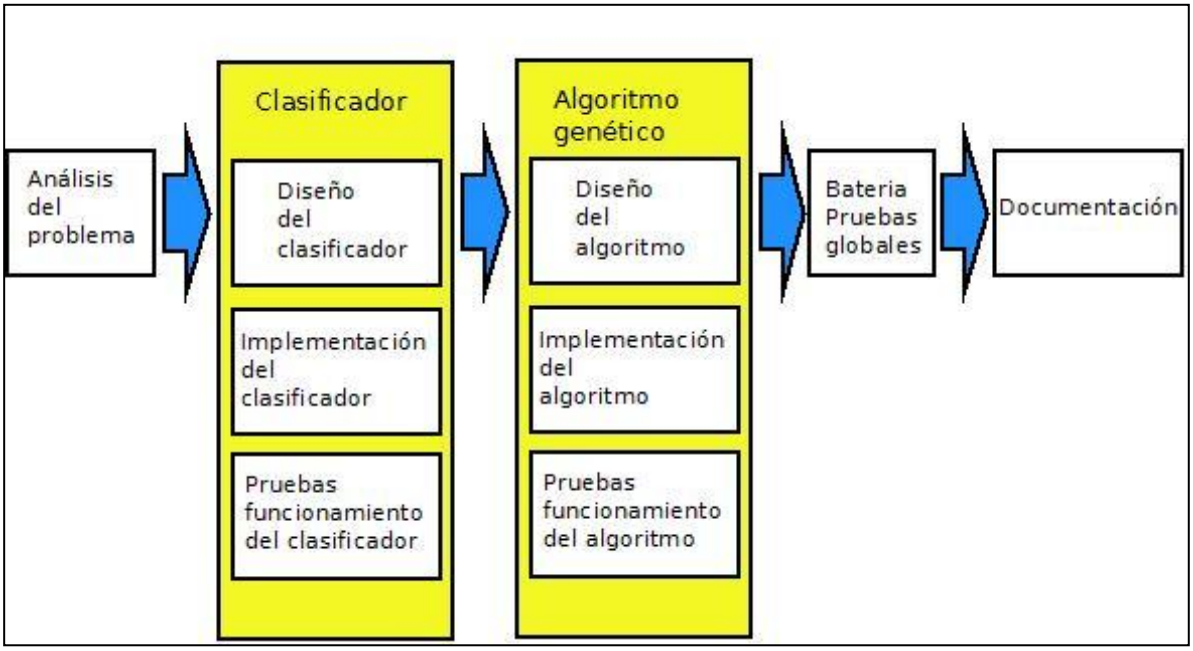


Figura 52. Esquema con las fases de desarrollo del proyecto.

5.3 Costes personales

Nombre recurso	Costes por fases					Coste total
	Análisis	Diseño	Desarrollo	Experimentación	Documentación	
Ingeniero técnico informático	1325 €	1925 €	8125 €	4750 €	6250 €	22375 €

Tabla 52. Costes personales del proyecto por fases y global.

(En la última fila podemos observar los costes personales de cada una de las fases en las que se subdivide el proyecto. La última columna de la derecha marca los costes totales personales del proyecto completo).

En lo relativo a los costes personales del proyecto, dado que esté ha sido realizado por una sola persona, y se ha estimado un coste para el ingeniero técnico informático de 25 € /hora trabajada los costes serán de 22.375 € como hemos visto en la tabla anterior.

5.4 Costes materiales

En cuanto a los costes materiales para la realización del proyecto, principalmente hemos usado dos recursos diferentes.

- El primero de los recursos materiales es un ordenador portátil (Pentium i3 2.27GHz 4GB de RAM) usado a lo largo de todas las fases del proyecto para su realización. En cuanto a los costes de éste, dado que su precio fue de 599€ y estimamos que tendrá una vida útil de 5 años, para poder amortizarlo apuntaremos un gasto de 119,8 € al año. El uso de las aplicaciones privativas utilizadas (Microsoft Word, Microsoft Proyect, etc.) estarán incluidas dentro de esta cantidad.

- El segundo recurso material fue el servidor de la universidad que se usó ocasionalmente para la realización de algunas pruebas. Para este servidor estimamos una cantidad de 150 € al mes en concepto de uso.

Nombre recurso	**Costes por fases**					**Coste total por recurso**
	Análisis	Diseño	Desarrollo	Experimentación	Documentación	
Ordenador portátil	13,14€	11,03€	43,29€	55,51	58,85	181,82 €
Servidor	0	0	0	150 €	0	150 €
Coste total por fases	13,14€	11,03€	43,29€	205,51€	58,85€	331,82 €

Tabla 53. Costes materiales desglosados por fases y global.

La última columna (columna azul) marca los costes totales de cada uno de los recursos. La última fila (fila amarilla) marca los costes en cada fase de todos los recursos materiales usados. La intersección de ambas (casilla verde), marca el coste total de todos los recursos materiales usados a lo largo de todo el proyecto).

5.5 Coste total

Sumando los costes personales y los costes materiales obtenemos como resultado los costes totales del proyecto:

Costes totales por fases					**Coste total del proyecto**
Análisis	Diseño	Desarrollo	Experimentación	Documentación	
1338,14€	1936,03€	8168,29€	4955,51€	6308,85€	22706,82 €

Tabla 54. Costes totales del proyecto por fases y global.

Por lo tanto, los costes totales del proyecto realizado serán de **22.706,82** Euros sin impuestos.

Capítulo 6

Conclusiones y líneas futuras

6.1 Introducción

El contenido de este capítulo versará sobre las conclusiones a las que hemos llegado tras el análisis del conjunto de pruebas realizadas en el capítulo anterior y las posibles mejoras o campos de investigación futuros que creemos que podrían llegar a mejorar los resultados de nuestro trabajo.

6.2 Conclusiones

A lo largo del capítulo 4 ya fuimos comentando las diferentes conclusiones a las que íbamos llegando en aquellos aspectos y detalles del algoritmo que creímos convenientes. Nos faltaría, pues, agruparlas y, desde una perspectiva global, ver todos aquellos objetivos cumplidos, los resultados obtenidos, la mejor manera de llegar a ellos, etc.

El aspecto más importante es que, ejecutando nuestro algoritmo genético de forma conjunta con el clasificador hemos conseguido nuestro principal objetivo, que no era otro que reducir el porcentaje de error. En la experimentación previa, (más concretamente en 4.2.1.1 Prueba del clasificador de 2 perceptrones) observamos cómo ejecutando solamente el clasificador y sin seleccionar las entradas (introduciendo los 96 atributos sin más) la media de resultados era de 31.53% en el entrenamiento y de 36.21% en el test. Estos resultados han sido los que hemos mejorado notablemente ya que usando nuestro algoritmo genético de forma conjunta con el clasificador hemos conseguido errores tanto de entrenamiento como de test un 6% menores. En los mejores casos se han obtenido errores de 26% en el entrenamiento y de 29%-30% en el test (algunos casos puntuales han mejorado incluso estos resultados). Esta mejoría conseguida ha sido gracias

al funcionamiento del algoritmo genético que es capaz de buscar las mejores soluciones para nuestro problema desechando aquellas entradas que no nos eran útiles e introducían ruido.

Otro aspecto del que nos hemos percatado durante la realización de éste proyecto es que parecía imposible disminuir los porcentajes de error más allá del 25,6% en el entrenamiento o del 28,9% en el test, salvo en un solo caso anecdótico que no podemos tener en cuenta. A pesar de que el clasificador cumple con nuestro objetivo, en multitud de ocasiones hemos podido ver cómo el algoritmo parecía estancarse y no conseguía proseguir su descenso del error. Esto nos hace pensar que el no conseguir mejores resultados ocurre por una limitación del clasificador utilizado y que dadas sus características no es capaz de realizar una mejor clasificación mejorando así estos porcentajes. En 6.3 Líneas futuras expondremos posibles formas de afrontar esto.

Respecto a aspectos algo más técnicos o parámetros internos de nuestro algoritmo genético (comentados detenidamente en las conclusiones del capítulo 4) es importante destacar las diferencias producidas en los resultados al variar la representación de las soluciones de nuestro problema. Como vimos, disponíamos de una representación por atributos y otra por canales. Ambas parecen dar resultados similares en cuanto a porcentajes de error se refiere pero existen diferencias notables en otros aspectos. En la representación por canales se ve incrementado notoriamente el porcentaje de entradas necesarias para obtener resultados competentes con los de la otra representación, esto conlleva un aumento en el tiempo de ejecución con respecto a la representación por atributos. Además, operadores como la mutación, cruce e inversión se comportan de forma más “agresiva” afectando más al resultado y produciendo cambios mayores en los individuos. Si a este hecho le añadimos que los resultados no revelan que exista un canal o canales específicos que perjudique la clasificación, creemos que esta representación por canales limita nuestras opciones así como el espacio de búsqueda pues, al agrupar las entradas por lotes, obliga a desechar a todos los genes del bloque o a ninguno, a mutarlos a todos o a ninguno, etc. Por estas razones anteriores se piensa que la representación por atributos es mucho más viable.

Y por último comentar que, aunque no formara parte de nuestros planes ni de nuestros objetivos, la curiosidad nos hizo probar nuestro algoritmo con la función de fitness 2 (la cuál pondera el error producido al clasificar al individuo y el porcentaje de entradas de éste). Dado que se sale de nuestros objetivos no hemos podido llevar a cabo un estudio detallado y sólo hemos realizado un pequeño conjunto de pruebas. Éstas parecen revelar que usando esta función de aptitud se podría contrarrestar el aumento de entradas que producía la representación por canales haciéndola un poco más viable.

6.3 Líneas futuras

No podemos finalizar el proyecto sin antes explicar que aspectos de nuestro trabajo creemos que se podrían mejorar, así como tratar de indicar a los interesados cuáles son los caminos a seguir que creemos más interesantes. A lo largo del proyecto ya hemos ido comentando diferentes opciones que por falta de tiempo no nos fue posible probar o posibles mejoras que surgían fruto de la experiencia que íbamos obteniendo las cuales se salían de los campos que nosotros tratábamos y que para evitar extendernos demasiado no abordábamos. Todas estas ideas, divididas en tres grupos, son las expuestas a continuación:

1. **Mejoras estéticas o de accesibilidad del programa:** Estas mejoras aquí propuestas no modificarán los resultados obtenidos sino que facilitarán el uso del programa:

    - **Interfaz gráfica:** Sería posible mejorar la usabilidad del programa realizando una sencilla interfaz gráfica para la generación de los ficheros de configuración 'conf.txt'. Esto evitaría a los usuarios tener que escribir manualmente las características de nuestro algoritmo en el fichero con lo que se evitaría fallos humanos en la introducción de datos, falta de campos, etc. Además facilitaría mucho la tarea de selección de las diferentes opciones.

    - **Adaptar el programa a otros sistemas operativos:** Se podría realizar unos ligeros cambios en el código como las 'path' de los ficheros o algunos comandos del sistema usados para conseguir que nuestro programa se ejecute correctamente en otros sistemas operativos diferentes al que fue concebido.

2. **Experimentación pendiente:** Los siguientes puntos forman parte de las pruebas que no pudieron ser realizadas totalmente en la experimentación por falta de tiempo en algunos casos y en otros por evitar hacer más extenso y complejo este proyecto. Cualquiera de los puntos pueden ser estudiados en nuestro programa de forma sencilla sin necesidad de realizar ninguna adaptación.

    - **Función de aptitud que tenga en cuenta el número de entradas a la red:** Como vimos, nosotros tratamos ligeramente el tema de la reducción de entradas del clasificador mediante la modificación de la función fitness, pero dado que no era nuestro objetivo principal no nos centramos en ello en la experimentación y sólo hicimos ligeros estudios en uno de los anexos y algunas pruebas. Se propone estudiar más en detalle el uso de esta función e intentar así disminuir el número de entradas necesarias y con ello el tiempo de ejecución.

- **Extracción de nuevos registros de datos:** Sería muy útil extraer nuevos registros de datos por nosotros mismos con el fin de controlar el proceso y a la vez ver si estos resultados son extrapolables a la mayoría de los individuos.

- **Estudio por canales:** En nuestra experimentación nos centramos en tratar de obtener los mejores resultados usando una representación por atributos dándole menor importancia a la representación por canales. Aunque según nuestras pruebas aparentemente no existen mejoras sobre los resultados conseguidos sería interesante comprobar estas afirmaciones con un estudio mucho más detallado.

3. **Posibles vías de mejora de los resultados de error obtenidos:** Las mejoras aquí propuestas confiamos mejoren sustancialmente nuestros resultados. Estas mejoras no están implementadas en nuestra solución y será necesario modificar el código de nuestro programa:

   - **Utilizar otro clasificador**: Una de las conclusiones obtenidas es que hemos llegado al límite de nuestro clasificador. Para poder mejorar los porcentajes de clasificación sería interesante sustituirlo por otro nuevo. Se propone utilizar, por ejemplo, el perceptrón multicapa u otros tipos de redes de neuronas teniendo en cuenta siempre que el tiempo computacional se vería notablemente aumentado.

   - **Algoritmo multiobjetivo:** Se propone el uso de algoritmos genéticos multiobjetivo. De este modo podríamos reducir el error a la vez que el número de entradas utilizadas en la red. Además, dadas las características de estos algoritmos, se hallaría un frente de Pareto evitando la concentración de las soluciones en un punto como nos ocurrió en algunos casos.

Referencias

A continuación aparecen las referencias que han sido citadas a lo largo del proyecto. Estas referencias pertenecen a documentos consultados para poder realizar este proyecto y han sido referenciados en aquellas partes de nuestro proyecto que se han visto fuertemente influenciadas por ellos. Además de ello, también incluiremos referencias en aquellas imágenes que no hemos realizado nosotros mismos indicando aquí el lugar del que fueron extraídas. (Las referencias están ordenadas por orden de aparición).

[1]. "*El lenguaje de programación C: Diseño e implementación de programas*" Félix García Carballeira, Jesús Carretero Pérez, Javier Fernández Muñoz, Alejandro Calderón Mateos, ISBN: 8420531782, (Prentice Hall, 2002).

[2]. "*Interfaz cerebro computadora (ICC) basada en el potencial relacionado con eventos P300: análisis del efecto de la dimensión de la matriz de estimulación sobre su desempeño*", Eliana García Cossio, Gerardo Gabriel Gentiletti, Revista Ingeniería Biomédica, ISSN 1909-9762, volumen 2, número 4, julio-diciembre 2008 págs. 26-33. http://revistabme.eia.edu.co/numeros/4/art/2633%20%28Interfaz%20cerebro%20computadora%29.pdf (Última visita 20/04/2011)

[3]. Página del proyecto BCI de EVANNAI: http://www.uc3m.es/portal/page/portal/grupos_investigacion/evannai/evannai_eng/Research/Projects1/evproj_mlbci (Última visita 20/04/2011)

[4]. "*Redes neuronales artificiales: Un enfoque práctico*", Juan Manuel Corchado, Fernando Díaz, Lourdes Borrajo, Florentino Fernández, ISBN: 8481581453, (Servicio de Publicacións da Universidade de Vigo, 2000).

[5]. http://es.wikipedia.org/wiki/Neurona (Última visita 22/04/2011) "Imagen de una neurona biológica".

[6]. "*Herramientas en GNU/Linux para estudiantes universitarios. Redes Neuronales con GNU/Linux*", Francisco José Palacios Burgos http://www.ibiblio.org/pub/linux/docs/LuCaS/Presentaciones/200304curso-glisa/redes_neuronales/curso-glisa-redes_neuronales-html/index.html (Última visita 24/04/2011)

[7]. "*Aprendizaje Automático: conceptos básicos y avanzados: aspectos prácticos utilizando el software Weka*", Basilio Sierra Araujo, ISBN: 848322318X, (Pearson Prentice Hall, 2006).

[8]. "*Redes de neuronas artificiales: Un enfoque Práctico*", Pedro Isasi Viñuela, Inés M. Garván León, ISBN: 8420540250, (Pearson Prentice Hall, 2004)

[9]. "*Redes Neuronales y Sistemas Borrosos*", Bonifacio Martín de Brío, Alfredo Sanz Molina, ISBN: 8478977430, 3ª Edición (Ra-Ma, 2006)

[10]. "*Analysis and Applications of Artificial Neural Networks*", L.P.J. Veelenturf, ISBN: 013489832X, (Prentice Hall, 1995).

[11]. *"Genetic algorithms + data structures = evolution programs"* Zbigniew Michalewicz, ISBN: 3540553878, (Editorial Springer, 1992).

[12]. "*Evolución de functional link networks con un enfoque multiobjetivo*" Autor: Ángel Carrasco Fernández, Tutores: Ricardo Aler Mur, Inés M. Galván León (Proyecto fin de carrera 13/03/09 Universidad Carlos III de Madrid). http://e-archivo.uc3m.es/handle/10016/6251 (Última visita 9/05/2011)

[13]. "*Using Evolutionary Multiobjective Techniques for Imbalanced Classification Data"* Sandra García, Ricardo Aler, Inés M. Galván
International Conference on Artificial Neural Networks. (ICANN 2010). Lecture Notes in Computer Science págs. 422-427. Thessaloniki, Grece. September, 2010.
También accesible en formato electrónico:
http://e-archivo.uc3m.es/handle/10016/9251 (Última visita 12/05/2011)

[14]. "*Multiobjective Algorithms Hybridization to Optimize Broadcasting Parameters in Mobile Ad-Hoc Networks*" Sandra García, Cristobal Luque, Alejandro Cervantes, Inés M. Galván. International Conference on Artificial Neural Networks (IWANN'09). Lecture Notes in Computer Science, 2009, Volume 5517/2009, págs. 728-735, Salamanca, Spain. June 2009.

[15]. "*Teoría de la optimización*", Vicente J. Novo Sanjurjo, ISBN: 8436240391, (Universidad Nacional de Educación a Distancia 1999)

[16]. "*Programación matemática*", Alejandro Balbas de la Corte, Jose Antonio Gil, 2ªEdición, ISBN: 8472880133, (Editorial AC, 1990)

[17]. Facultad de informática Universidad Nacional de la Plata.
http://weblidi.info.unlp.edu.ar/catedras/neuronales/09_Tecnicas%20de%20Seleccion.pdf (Última visita 21/04/2011)

Anexos

Anexo 1. Cálculo de los pesos para la función de fitness 2

En el caso de que usemos la función fitness 2 o mejorada, necesitaremos dos pesos que ponderen el error producido por la red y el número de entradas de ésta. Dados los objetivos que queremos cumplir y la importancia que le damos a cada uno de ellos, tenemos claro que el reducir porcentaje de error producido por la red es nuestro objetivo más importante y la importancia de reducir el número de entradas a la red será menor ya que como hemos explicado antes, este aspecto ya es conseguido al crear la población. La inclusión del porcentaje de parámetros de cada individuo en la función fitness es por lo tanto simplemente una manera de que se valore positivamente a los individuos que tienen menos parámetros frente a los que tienen más con el fin de que generación tras generación el porcentaje de parámetros de los individuos de la población vaya disminuyendo.

Para conseguir todo ello debemos calibrar estos pesos con el fin de que esta función fitness vaya reduciendo las entradas que no afectan a la clasificación y que poco a poco el número de entradas a la red disminuya (eliminando las entradas que no nos son útiles y que sólo introducen ruido) y nos quedemos sólo con las entradas valiosas. De todas las combinaciones posibles de pesos estudiaremos los dos siguientes casos (W_0=0.1, W_1=0.9 y W_0=0.05, W_1=0.95) ya que creemos que disminuir más la importancia del porcentaje de error dentro de la función fitness hará que se escojan individuos no deseados (primaría a individuos formados por una cantidad muy baja de entradas independientemente del porcentaje de error que tengan) y en cambio, si aumentamos más su importancia hará que los resultados obtenidos por ésta sean iguales a los de la función fitness 1. Dando al peso que pondera el error de la red un valor cercano a 0.9 estamos dando una importancia vital al porcentaje de error sobre el porcentaje de parámetros. El porcentaje de parámetros de cada individuo entrará a formar parte de la aptitud de los individuos pero de forma muy baja con lo que nos aseguramos que las entradas que se irán reduciendo serán las de menos importancia en la clasificación, ya que si se empezaran a quitar entradas importantes el porcentaje de error aumentaría y la valoración del individuo decaería.

Pruebas con W_1= 0.9 W_0= 0.1:

A continuación vemos a tres individuos que reciben la misma valoración sin tener el mismo porcentaje de error ni el mismo porcentaje de parámetros. Estos tres individuos estarán exactamente igual de adaptados y tendrán las mismas opciones de ser seleccionados, reemplazados, etc.

Individuos	% Error red	% Parámetros	Fitness total
Individuo1	26,5	40,5	27,9
Individuo2	26	45	27,9
Individuo3	25,5	49,5	27,9

Tabla 55. Individuos con el mismo fitness (W_1= 0.9 W_0= 0.1).

Si observamos detenidamente la tabla, el individuo 1 recibe la misma valoración o fitness que el individuo 3 teniendo un 1% más de error por el hecho de tener un 9% menos de parámetros. Quizás con esta combinación de pesos seguimos dando una importancia alta al porcentaje de parámetros. Dada nuestra experiencia obtenida tras haber observado el comportamiento del algoritmo y de las diferentes poblaciones obtenidas, un 1% menos de error de la red es una diferencia importante y debería de tenerse más en cuenta. Corremos el riesgo que con esta combinación de pesos, el algoritmo genético ronda a ronda vaya seleccionando a individuos con porcentajes del 1 ó 2% más de error por el mero hecho de tener menos parámetros.

Una diferencia de un 9% de parámetros es importante pero no tanto como para hacer que se igualen individuos que tienen porcentajes de error tan diferentes. Aún así veamos más ejemplos:

Individuos	% Error red	% Parámetros	Fitness total
Individuo1	27,5	40	28,75
Individuo2	27	49,5	29,25
Individuo3	26,5	59	29,75

Tabla 56. Ejemplos de individuos y su valoración (W_1= 0.9 W_0= 0.1).

En la tabla anterior vemos ejemplos aún más claros. Con esta combinación de pesos elegida (W_1= 0.9 W_0= 0.1) el efecto conseguido es el contrario al que esperábamos. La relación se invierte y los individuos con un porcentaje de error menor consiguen un fitness peor que el de individuos con un error mayor. Si bien estos individuos tienen un menor porcentaje de parámetros, esta diferencia debería hacer que reciban fitness similares o

darles una ventaja de décimas, en ningún caso es justificable que consigan un fitness final de un 1% mejor.

Nuestras sospechas quedan probadas. Esta combinación de pesos es demasiado agresiva para nuestro caso. Con ella conseguiríamos una reducción de entradas quizás demasiado drástica a costa de poblaciones de individuos que no mejorarían el error producido por la red.

Pruebas con W_1=0.95 W_0=0.05:

Volvemos a mostrar para este nuevo caso a tres individuos que reciben la misma valoración sin tener el mismo porcentaje de error ni el mismo porcentaje de parámetros.

Individuos	% Error red	% Parámetros	Fitness total
Individuo1	27,5	40	28,125
Individuo2	27	49,5	28,125
Individuo3	26,5	59	28,125

Tabla 57. Individuos con el mismo fitness (W_1= 0.95 W_0= 0.05).

Para este caso, el individuo 1 recibe el mismo fitness que el individuo 3 teniendo un 1% más de error pero a diferencia del caso anterior, éste tiene un 19% menos de parámetros. Un 19% de parámetros ya es un porcentaje de parámetros respetable con el que podemos justificar que reciban el mismo fitness final.

En la siguiente tabla vemos a dos individuos, su fitness final es prácticamente el mismo, si bien, el del individuo 2 es ligeramente mejor. El individuo 2 tiene 20 atributos menos que el individuo 1 (38 frente a 58) y un porcentaje de error un 1% superior. Esta diferencia tan amplia en el número de parámetros justifica que ambos obtengan valoraciones similares.

Individuos	% Error red	% Parámetros	Fitness total
Individuo1	27,5	39,58	28,104
Individuo2	26,5	60,42	28,195

Tabla 58. Ejemplos de 2 individuos y su valoración (W_1= 0.9 W_0= 0.1).

Con esta combinación de pesos, conseguiremos que el porcentaje de parámetros cuente en la valoración del individuo y sea uno de los objetivos que el algoritmo busque en las nuevas poblaciones pero siempre sin olvidarnos que nuestro objetivo principal es la reducción del error producido.

Anexo 2. Manual de usuario

A continuación explicaremos todo lo necesario para que cualquier persona pueda ejecutar nuestro programa de forma correcta.

Como ya hemos explicado antes, el programa necesitaba de unos ficheros de entrada para su correcto funcionamiento y posteriormente devolvía unos ficheros con la salida obtenida por el programa. Explicaremos el formato de todos estos ficheros y además comentaremos otras características del programa como la forma de compilar el código en otros ordenadores, la forma de elegir los operadores que queremos que se ejecuten, cómo modificar las probabilidades de los operadores, el tamaño de la población, cambiar el número de operaciones que contendrán los individuos, etc.

2.1. Archivos y directorios necesarios

Nuestro programa estará formado por los siguientes archivos:

- aleatorizar.c
- alg.c
- operadores.c
- operadores.h
- perceptrones.c
- perceptrones.h
- configuración.h
- makefile

Debe copiar estos archivos en su ordenador en el directorio que haya elegido y compruebe que no falta ninguno. Una vez hecho esto deberá crear dentro de ese directorio tres carpetas con los siguientes nombres (tenga cuidado con las mayúsculas y no escriba ningún acento):

- soluciones
- datosBCI
- configuración

A continuación, guarde en la carpeta ‘datosBCI’ los dos ficheros de datos y cree un fichero con el nombre ‘conf.txt’ dentro de la carpeta ‘configuracion’. En los siguientes apartados veremos cómo compilar estos archivos en su ordenador y el formato que deberá contener tanto los ficheros de datos como el fichero de configuración.

2.2. ¿Cómo compilar y ejecutar el programa?

Desde un ordenador con sistema operativo Linux y el compilador gcc instalado, los comandos de consola necesarios para realizar las siguientes operaciones son:

- Para compilar el programa: make
- Ejecutar el programa: ./algoritmo_genetico
- Para borrar los archivos ejecutables: make clean

(Modificando en el código fuente las llamadas al sistema que contiene y realizando sencillos cambios en las variables que incluyen las direcciones de los ficheros este programa podrá ser utilizado también en Windows).

2.3. Ficheros de datos de entrada usados y su formato

Nuestro programa necesita utilizar dos ficheros de datos para realizar su tarea como ya hemos visto anteriormente. Ambos ficheros contendrán los registros pre-procesados que han sido recogidos por el BCI. El formato de estos ficheros deberá ser el siguiente:

- Cada uno de los registros deberá estar contenido en una línea del fichero de datos.

- Cada registro estará formado por 96 datos decimales o en notación exponencial separados por un espacio o tabulación y 2 números enteros separados por un espacio (0 1, 1 0, 0 0) que codificarán la clase a la que forma parte ese registro.

Este será el único formato de datos que leerá nuestro programa. Por ello deberá tener cuidado y revisar que los ficheros usados lo cumplen ya que en caso contrario el programa podría no conseguir el resultado deseado.

2.4. Fichero de configuración (conf.txt)

El fichero conf.txt será la forma que utilizaremos para comunicarnos con el algoritmo genético. Gracias a este fichero, podremos determinar todos los parámetros y características de la ejecución de nuestro algoritmo. En él es donde almacenaremos toda la configuración del programa. Dada la

cantidad de opciones y modos de ejecución que hemos visto a lo largo de este documento, en un principio podrá parecer que el uso de este fichero es complejo pero una vez que lo hayamos usado un par de veces nos desenvolveremos bien con él. Facilita el uso del programa y nos evita tener que ir modificando las variables del programa directamente de las librerías lo que implicaría tener que compilar el código con cada modificación.

Queda pendiente la realización de una herramienta auxiliar (un formulario o de una interfaz gráfica simple) que facilite aún más la selección de todos estas opciones consiguiendo que el usuario final seleccione todos estos parámetros de una manera más sencilla mediante un par de sencillos clips. Esta pequeña herramienta por falta de tiempo no se pudo realizar.

Los parámetros o campos que deberán seguir el siguiente formato:

Nombre del campo= valor deseado
(Dejando un espacio en blanco entre el símbolo '=' y el valor asignado).

Configuración general de la población.

Los siguientes campos modifican las diferentes opciones explicadas anteriormente sobre varios aspectos del algoritmo genético.

- **Campo tipo_ejecucion**
  Con este campo escogemos la representación que queremos para los individuos. Como ya explicamos, (3.4.1 Representación de los individuos) podíamos escoger entre dos tipos diferentes de representaciones, por ello, los únicos valores posibles son 0 ó 1.
  tipo_ejecucion= 0 Representación de los individuos por atributos.
  tipo_ejecucion= 1 Representación de los individuos por canales.

- **Campo numero_individuos**
  Este campo identificará el número de individuos que formará la población del algoritmo genético. (Explicado en 3.4.8 Otros aspectos o parámetros a tener en cuenta). Este campo deberá ser un número entero positivo mayor de 0.
  Por ejemplo:
  numero_individuos= 68

- **Campo numero_rondas**
  Sencillamente indica el número de rondas que se ejecutará el algoritmo genético, o lo que es lo mismo , el número de generaciones que se producirán.(Explicado en 3.4.7 Condición de término). Este campo deberá ser siempre un número entero no negativo. Por ejemplo:
  numero_rondas= 75

- **Campo porcentaje_unos**
  Probabilidad de que se le asigne un uno a los genes cuando se crea la población. Con este campo indicaremos el porcentaje de atributos o canales con los que queremos que se inicialice la población. Este campo toma valores decimales entre el 0 y el 1 (ambos inclusive).
  porcentaje_unos= 0.5
  porcentaje_unos= 1 (Los 96 atributos entran a formar parte de los individuos).

- **Campo numero_operaciones**
  Con este campo indicaremos al algoritmo genético el número de operaciones que formará parte de cada individuo. (Explicado en 3.4.8 Otros aspectos o parámetros a tener en cuenta). Este campo deberá ser siempre un número entero no negativo. Por ejemplo:
  numero_operaciones=50
  numero_operaciones= 0 (No hay ninguna operación).

En el caso de que el número de operaciones sea superior a 0 podremos elegir el porcentaje de cada uno de los tipos de operaciones de forma que podremos primar un tipo sobre los demás, desechar alguno de los tipos, etc. La única condición que deberemos tener en cuenta es que la suma de los cuatro porcentajes deberá ser siempre 100. Si damos 25 como valor a los cuatro porcentajes todos los operadores tendrán las mismas opciones y serán seleccionadas completamente al azar, si damos a uno de ellos el valor 0 no se crearán operaciones de ese tipo.

- **Campo porcentaje_sumas**
  Con este parámetro indicamos el porcentaje de sumas que queremos que se creen en las operaciones. Este campo es un número entero y deberá estar contenido entre 0 y 100 ambos inclusive.
  Por ejemplo: porcentaje_sumas= 40

- **Campo porcentaje_restas**
  Con este parámetro indicamos el porcentaje de restas. Este campo es un número entero entre 0 y 100 ambos valores inclusive.
  Por ejemplo: porcentaje_restas= 30

- **Campo porcentaje_multiplicacion**
  Con este parámetro indicamos el porcentaje de multiplicaciones que queremos que se creen en las operaciones. Este campo es un número entero entre 0 y 100 ambos inclusive.
  Por ejemplo: porcentaje_multiplicacion= 20

- **Campo porcentaje_division**
  Con este parámetro indicamos el porcentaje de divisiones. Este campo es un número entero entre 0 y 100, ambos valores inclusive.
  Por ejemplo: porcentaje_division= 10

Campos relativos a los ficheros de datos.

Los siguientes campos indican las características de los ficheros de datos que utilizará el programa.

- **Campo nombre_fichero_train**
  Este campo indica el nombre del fichero de datos destinado para el entrenamiento o train de la red. Por ejemplo:
  nombre_fichero_train= train_test1_subject1_psd010203.arff

- **Campo numero_registros_train**
  Esta variable indica al algoritmo genético el número de registros que contiene el fichero de entrenamiento. (Dado que cada registro ocupará una línea, este número debe coincidir con el número de líneas del fichero). Por ejemplo:
  numero_registros_train= 10528

- **Campo nombre_fichero_test**
  Este campo indica el nombre del fichero de datos destinado para el test de la red. Por ejemplo:
  nombre_fichero_test= test2_subject1_psd04.arff

- **Campo numero_registros_test**
  Esta variable indica al algoritmo genético el número de registros que contiene el fichero usado para el test de la red. (Dado que cada registro ocupará una línea, este número debe coincidir con el número de líneas del fichero). Por ejemplo:
  numero_registros_test= 3504

Función de aptitud

- **Campo tipo_fitness**
  Con esta variable elegimos la función de fitness que queremos de las dos que explicados en la sección (3.4.3 Función de aptitud o función de fitness)**.** Los valores son:
  tipo_fitness= 1 Función de fitness que valora sólo el porcentaje de error de la red.
  tipo_fitness= 2 Función ponderada por pesos. (w1.%error+w0.%unos)

En el caso de que escojamos la función fitness 2 necesitaremos además añadir otros dos campos adicionales (los pesos de la función):

- **Campo w0**
  Este campo toma valores decimales entre el 0 y el 1 (ambos inclusive). La única restricción que deberá cumplir es que w0+w1=1.
  Ejemplo: w0= 0.05 Peso que pondera el número de entradas.

- **Campo w1**
  Este campo toma valores decimales entre el 0 y el 1 (ambos inclusive). La única restricción que deberá cumplir es que w0+w1=1.
  Ejemplo: w1= 0.95 Peso que pondera el error de la red

Operadores genéticos

- **Campo operadores_operaciones**
  Con este campo escogemos el modo en el que se ejecutan los operadores de las operaciones. Como ya hemos explicado, (3.4.5.3 Modos de uso de los operadores) podíamos escoger entre dos modos de uso de los operadores, conjuntos, o de forma independiente.
  Los únicos valores posibles son 0 ó 1.
  operadores_operaciones= 1 Se ejecutan de forma conjunta.
  operadores_operaciones= 0 Se ejecutan de forma independiente.

Selección

- **Campo tipo_seleccion**
  Con esta variable elegimos el tipo de selección que queremos de los explicados en la sección (3.4.4 Métodos de selección)**.**
  Los valores son:
  tipo_seleccion= 1 Selección por método de la ruleta.
  tipo_seleccion= 2 Selección jerárquica.
  tipo_seleccion= 3 Selección por torneos probabilísticos.
  tipo_seleccion= 4 Selección por torneos deterministas.

En el caso de que se realice la selección por torneos determinista será necesario utilizar el siguiente campo:

- **Campo tamanio_torneo**
  Número entero que indicará el tamaño de los torneos que se realizarán en este tipo de selección. Este número deberá ser mayor de 2 y menor que el número de individuos de la población (variable "numero_individuos").
  Por ejemplo:
  tamanio_torneo= 7
  tamanio_torneo= 2

Nota: No conviene un tamaño de torneo muy grande ya que entonces seleccionará siempre a los mismos individuos que serán los más aptos y la población convergerá muy rápido.

Mutación

Para poder seleccionar las diferentes opciones relativas a la mutación de los individuos que explicamos anteriormente en la sección 3.4.5.1.2 Método de mutación debemos modificar los siguientes campos:

- **Campo mutaciones**
  Con este campo elegimos si queremos que se aplique el operador de mutación con la población de nuestro algoritmo genético. Las dos únicas opciones son:
  mutaciones= 0 No hay mutación.
  mutaciones= 1 Se realiza la mutación.

En el caso de que se realice la mutación utilizaremos los siguientes campos:

- **Campo probabilidad_mutacion**
  Número decimal entre 0 y 1 (ambos inclusive) que indica la probabilidad que tiene cada uno de los individuos de ser mutado. Por ejemplo:
  probabilidad_mutacion= 0.78

- **Campo factor_mutacion**
  Número decimal entre 0 y 1 (ambos inclusive) que indica la probabilidad de que un gen de un individuo sea mutado. Por ejemplo un valor que podría tomar este parámetro sería:
  factor_mutacion= 0.45

Mutación de operaciones

- **Campo mutar_operaciones**
  Con este campo elegimos si queremos que se aplique el operador de mutación de operaciones a la población de nuestro algoritmo genético (Explicado en 3.4.5.2.1 Mutación de las operaciones).
  Las dos únicas opciones son:
  Mutar_operaciones= 0 No hay mutación de operaciones.
  Mutar_operaciones= 1 Se realiza la mutación de operaciones.

En el caso de que se realice la mutación de operaciones deberemos utilizar los siguientes campos (En el caso de que se selecciones operadores_operaciones=1 no será necesario incluir el campo probabilidad_mutacion_operaciones):

- **Campo probabilidad_mutacion_operaciones**
  Número decimal entre 0 y 1 (ambos inclusive) que indica la probabilidad que tiene cada uno de los individuos de que sus operaciones sean mutadas. Por ejemplo:
  probabilidad_mutacion_operaciones= 0.335

- **Campo factor_mutacion_operaciones**
  Número decimal entre 0 y 1 (ambos inclusive) que indica la probabilidad de que una operación de un individuo sea mutada. Un ejemplo:
  factor_mutacion_operaciones= 0.10

Cruce

Para poder seleccionar las diferentes opciones relativas a los cruces de los individuos que explicamos anteriormente en la sección 3.4.5.1.1 Métodos de cruce debemos modificar los siguientes campos:

- **Campo tipo_cruce**
  Con este parámetro elegimos el tipo de cruce que queremos. Las únicas opciones posibles son las siguientes:
  tipo_cruce= 0 Sin cruce.
  tipo_cruce= 1 Cruce un punto
  tipo_cruce= 2 Cruce dos puntos
  tipo_cruce= 3 Cruce uniforme

En el caso de que hayamos seleccionado uno de los cruces anteriores deberemos rellenar los siguientes campos:

- **Campo probabilidad_cruce**
  Número decimal entre 0 y 1 (ambos inclusive) que indica la probabilidad que tiene una pareja de individuos de ser cruzada. Por ejemplo:
  probabilidad_cruce= 0.41

Y si el hemos seleccionado el cruce uniforme (tipo_cruce= 3) también deberemos añadir este campo al fichero:

- **Campo factor_cruce**
  Número decimal entre 0 y 1 (ambos inclusive) que indica la probabilidad de que un gen de un individuo sea cruzado. Un ejemplo:
  factor_cruce= 0.25

Cruce operaciones.

- **Campo cruce_operaciones**
  Con este campo elegimos si queremos que se aplique el operador de cruce de operaciones a la población de nuestro algoritmo genético (Explicado en 3.4.5.2.2 Cruce uniforme de las operaciones).
  Las dos opciones posibles son:
  cruce_operaciones= 0 No hay cruce de operaciones.
  cruce_operaciones= 1 Se realiza el cruce de operaciones.

En el caso de que se realice el cruce de operaciones deberemos utilizar los siguientes campos (En el caso de que se seleccione operadores_operaciones=1 no será necesario incluir el campo probabilidad_cruce_operaciones):

- **Campo probabilidad_cruce_operaciones**
  Número decimal entre 0 y 1 (ambos inclusive) que indica la probabilidad que tiene cada pareja de individuos de que sus operaciones sean cruzadas. Por ejemplo:
  probabilidad_cruce_operaciones= 0.07

- **Campo factor_cruce_operaciones**
  Número decimal entre 0 y 1 (ambos inclusive) que indica la probabilidad que tiene cada una de las operaciones de ser cruzadas. Por ejemplo:
  factor_cruce_operaciones= 0.15

Inversiones

En lo relativo a la inversión de genes que vimos en la sección 3.4.5.1.3 Método de inversión los campos que aplicamos son que debemos rellenar son los siguientes.

- **Campo inversiones**
  Con este campo elegimos si queremos que se aplique el operador de inversión a la población de nuestro algoritmo genético. Las dos únicas opciones son:
  inversiones= 0 No hay inversión.
  inversiones= 1 Se usa el operador inversión en el algoritmo genético.

- **Campo probabilidad_inversion**
  Número decimal entre 0 y 1 (ambos inclusive) que indica la probabilidad de que un individuo sufra una inversión de varios de sus genes. Un ejemplo de este campo:
  probabilidad_inversion= 0.18

Reemplazo

- **Campo tipo_reemplazo**
  Con esta variable elegimos el tipo de reemplazo que queremos de los explicados (3.4.6 Métodos de reemplazo)**.** Los valores son:
  tipo_reemplazo= 1 Reemplazo generacional
  tipo_reemplazo= 2 Reemplazo estacionario
  tipo_reemplazo= 3 Reemplazo estacionario con elementos aleatorios.

En el caso de que se trate de un reemplazo estacionario (tipo_reemplazo= 2) deberemos añadir el siguiente campo:

- **Campo porcentaje_reemplazo**
  Con este parámetro indicamos el porcentaje de individuos que queremos que sean sustituidos en cada ronda. Este campo es un número entero y deberá estar contenido entre 0 y 100. (Con porcentaje_reemplazo= 100 obtendríamos el mismo resultado que si hubiéramos seleccionado un reemplazo generacional). Por ejemplo:
  porcentaje_reemplazo= 85 (se reemplaza un 85% de la población.)

Y si hemos seleccionado el reemplazo estacionario con elementos aleatorios (tipo_reemplazo= 3), deberemos añadir los cuatro siguientes campos con la condición de que la suma de los cuatro de 100:

- **Campo porcentaje_padres**
  Con este parámetro indicamos el porcentaje de mejores padres que queremos que accedan a la siguiente ronda. Este campo es un número entero y deberá estar contenido entre 0 y 100.
  Por ejemplo: porcentaje_padres= 21

- **Campo porcentaje_padres_ale**
  Con este parámetro indicamos el porcentaje de padres escogidos al azar que queremos que accedan a la siguiente ronda. Este campo es un número entero y deberá estar contenido entre 0 y 100.
  Por ejemplo: porcentaje_padres_ale= 9

- **Campo porcentaje_hijos**
  Con este parámetro indicamos el porcentaje de mejores hijos que queremos que accedan a la siguiente ronda. Este campo es un número entero y deberá estar contenido entre 0 y 100.
  Por ejemplo: porcentaje_hijos= 62

- **Campo porcentaje_hijos_ale**
  Con este parámetro indicamos el porcentaje de hijos escogidos al azar que queremos que accedan a la siguiente ronda. Este campo es un número entero y deberá estar contenido entre 0 y 100.
  Por ejemplo: porcentaje_hijos_ale= 8

El fichero conf.txt quedaría así:

Figura 53. Ejemplo de un fichero de configuración 'conf.txt'.
(Las líneas en azul son comentarios).

2.5. Otras opciones configurables del clasificador.

Además de todas las opciones explicadas anteriormente sobre el algoritmo genético, también podemos modificar algunos campos de nuestro clasificador. La diferencia reside en que para modificar las características del

clasificador es necesario cambiar varias constantes que se encuentran en la librería 'Perceptrones.h' lo que requerirá que para estos cambios surtan efecto tendremos que compilar de nuevo el programa. Se decidió dejar estos campos en la librería en vez de ponerlos en el fichero de configuración para evitar añadir más complejidad a éste ya que las características del clasificador para nuestro problema quedaron determinadas durante la Experimentación y no serán modificadas durante el resto del proyecto salvo en contadas ocasiones.

Los cambios más importantes que podemos realizar en el clasificador son:

- **Cambiar el número de perceptrones del clasificador.** Podemos seleccionar el número de perceptrones que queremos que tenga nuestro clasificador. Tendremos dos opciones: 2 perceptrones o 3 perceptrones como contamos en la sección 3.5.3 Diferentes clasificadores planteados. Por defecto, el clasificador estará formado por 2 perceptrones ya que como vimos en la sección "4.2.1.3 Clasificador elegido y número de ciclos de aprendizaje" este clasificador da mejores resultados. La constante que deberemos modificar para ello es 'N_perceptrones'.

- **Cambiar la razón de aprendizaje.** Podemos cambiar la razón de aprendizaje de los perceptrones. En la sección 2.3.6.2 Razón de aprendizaje en el perceptrón simple, ya hablamos de este campo y de su utilidad. Por defecto este valor será de 1 (1 es igual a un aprendizaje normal sin razón de aprendizaje). El valor de esta constante podrá variar entre 0.0001 y 1. La constante a modificar en la librería perceptrones.h será 'RAZON'.

- **Cambiar el ciclo de aprendizaje de los perceptrones.** Otra constante que podemos modificar a nuestro antojo es el ciclo de aprendizaje de los perceptrones. Podemos cambiar el número de ciclos cambiando la constante 'SESSIONS'. Por defecto, el número de ciclos sería 300.

Recuerde que tras realizar alguno de estos cambios es necesario que compile el código del programa de nuevo introduciendo por consola los siguientes comandos:

- make clean
- make

2.6. Salida obtenida por el programa

Una vez que el programa haya terminado de forma satisfactoria su ejecución nos encontraremos con las siguientes salidas (los archivos se encontrarán dentro de la carpeta 'soluciones' una vez finalizada la ejecución del programa):

- Por pantalla: Todos los resultados obtenidos en cada una de las rondas y los resultados de la ronda de test final incluyendo medias de la población, etc.

- Un fichero donde muestra toda la salida del programa. (guarda en el fichero exactamente lo mismo que se muestra por pantalla) y la configuración de los operadores del algoritmo que han provocado esos resultados. Este fichero tendrá como nombre la fecha del sistema más ':Salida'. Por ejemplo 'Sat Jun 25 17:12:31 2011:Salida'.

- Otro fichero llamado 'Sat Jun 25 17:12:31 2011:Datos' donde se almacenará el porcentaje de error producido y el porcentaje de parámetros de cada individuo en cada ronda. Este fichero nos será útil para hacer gráficas. Además al final de éste fichero se incluirá también las estadísticas y datos del mejor elemento de la última generación (el número de atributos escogidos de cada uno de los canales, el número de operaciones que contiene y el tipo de cada una de ellas, los pesos de los perceptrones para ese individuo, etc.), y el test realizado a la última generación.

- Un último fichero llamado 'Sat Jun 25 17:12:31 2011:Media' donde se guarda la media de entradas de la red , la media del porcentaje de error y la media de fitness que tiene la población, además de los individuos con mejor y el peor fitness en cada generación. De este modo viendo este fichero podremos observar el comportamiento del algoritmo genético a lo largo de las diferentes generaciones.

2.7. Códigos de error del programa y forma de solucionarlos

Dado que la configuración del algoritmo será introducida mediante un fichero de configuración y esté será escrito por el usuario final, se pueden llegar a producir errores al introducir esta configuración por un descuido del usuario, por desconocimiento de las diferentes características de cada parámetro configurable, errores gramaticales, etc. Estos errores provocarán que el algoritmo genético no pueda ejecutarse. Para intentar corregir estos

errores y avisar al usuario de cuál es el fallo o error cometido implementamos unos mensajes de error que aparecerán tanto por pantalla como en el fichero de salida del programa. A la hora de desarrollar los mensajes de error devueltos por el programa hemos intentado que estos fueran lo más explicativos posibles, con el fin de que no sea necesario consultar este apartado cada vez que surja alguno de ellos y estos puedan ser resueltos de manera intuitiva por el usuario del programa. De todas formas, a continuación explicamos más detenidamente cada uno de los mensajes de error que puede devolver el programa con el fin de ayudar en su corrección.

ERROR 1. Los pesos w0 y w1 exceden a 1. La suma de ambos deben no puede superar a 1. Estos pesos w0 y w1 son los utilizados por la función de fitness 2 explicada anteriormente (3.4.3.2 Función de aptitud mejorada (suma ponderada de objetivos)). Como vimos w0 es el peso que pondera las entradas de la red y w1 el que pondera el error producido. Como también vimos ambos deben sumar 1. Cuando se muestre error significará que la suma de ambos pesos es mayor de 1 y se deberá corregir para que el algoritmo genético se ejecute correctamente.

ERROR 2. La variable 'probabilidad_inversion' es errónea. Debe escoger un valor entre 0 y 1. Cuando se muestre este error significará que la probabilidad de inversión introducida es errónea. El valor que deberá tomar este parámetro será un número decimal contenido entre 0 y 1. Ambos valores inclusive. Por ejemplo serían valores correctos: 0.5, 0.2888, 1, 0.

ERROR 3. La suma de 'porcentaje_padres', 'porcentaje_padres_ale', 'porcentaje_hijos', 'porcentaje_hijos_ale' es diferente a 100. En este caso, hemos elegido el reemplazo con elementos aleatorios pero el problema reside en que debemos indicar el porcentaje de cada una de las clases como ya vimos. Evidentemente la suma de los 4 porcentajes deberá dar 100% para que así la nueva población contenga el mismo número de individuos que la antigua. Esta condición es la que no se cumple.

ERROR 4. Variable 'porcentaje_reemplazo' es errónea. Debe escoger un valor entre 0 y 100. En este caso, debemos elegir un porcentaje de individuos que se sustituirán en el reemplazo estacionario. Este porcentaje deberá ser un número entero entre 0 o 100.

ERROR 5. Variable 'tipo_reemplazo' es errónea. Debe escoger un valor entre 1 y 3. Este error significa que el valor elegido para esta variable es erróneo. Deberemos modificar el valor en el fichero ‘conf.txt’ y sustituirlo por un número entero entre el 1 y el 3 ambos inclusive.

ERROR 6. Variable 'tipo_cruce' es errónea. Debe escoger un valor entre 0 y 3. El programa nos está informando de que el valor elegido para esta variable es erróneo. Deberemos sustituirlo por un número entre el 0 y el 3.

ERROR 7. Variable 'factor_cruce_operaciones' es errónea. Debe escoger un valor entre 0 y 1. Este error significará que el valor dado a 'factor_cruce_operaciones' es erróneo. El valor que deberá tomar este parámetro será un número decimal contenido entre 0 y 1 (ambos inclusive).

ERROR 8. Variable 'probabilidad_cruce_operaciones' es errónea. Debe escoger un valor entre 0 y 1. Este error significará que el valor dado a 'probabilidad_cruce_operaciones' es erróneo. El valor de este parámetro deberá ser un número decimal entre 0 y 1. Ambos valores inclusive.

ERROR 9. Variable 'probabilidad_cruce' es errónea. Debe escoger un valor entre 0 y 1. Cuando se muestre este error significará que el valor dado a 'probabilidad_cruce' es erróneo. El valor que deberá tomar este parámetro será un número decimal contenido entre 0 y 1. Ambos valores inclusive.

ERROR 10. Variable 'factor_cruce' es errónea. Debe escoger un valor entre 0 y 1. Al igual que el anterior este error significará que el valor dado a 'factor_cruce' es erróneo. El valor que deberá tomar este parámetro será un número decimal contenido entre 0 y 1. Ambos valores inclusive.

ERROR 11. Variable 'tipo_seleccion' es errónea. Debe escoger un valor entre 1 y 4. Este error nos indicará que el valor elegido para esta variable es erróneo. Deberemos modificar el valor en el fichero 'conf.txt' y sustituirlo por un número entero entre el 1 y el 4 ambos inclusive.

ERROR 12. Variable 'factor_mutacion_operaciones' es errónea. Debe escoger un valor entre 0 y 1. Cuando aparezca este error significará que el valor dado a 'factor_mutacion_operaciones' es erróneo. El valor que deberá tomar este parámetro será un número decimal contenido entre 0 y 1. Ambos valores inclusive.

ERROR 13. Variable 'probabilidad_mutacion_operaciones' es errónea. Debe escoger un valor entre 0 y 1. Cuando se muestre este error significará que el valor dado a 'probabilidad_mutacion_operaciones' es erróneo. El valor que deberá tomar este parámetro será un número decimal contenido entre 0 y 1. Ambos valores inclusive.

ERROR 14. Variable 'probabilidad_mutacion' es errónea. Debe escoger un valor entre 0 y 1. Con la aparición de este error se nos trata de avisar de que el valor dado a 'probabilidad_mutacion' es erróneo. El valor que deberá tomar este parámetro será un número decimal contenido entre 0 y 1. Ambos valores inclusive.

ERROR 15. Variable 'factor_mutacion' es errónea. Debe escoger un valor entre 0 y 1. Este error nos indicará que el valor dado a 'factor_mutacion' es erróneo. Deberemos poner un número decimal contenido entre 0 y 1. Ambos valores inclusive.

ERROR 16. Variable 'porcentaje_unos' es errónea. Debe escoger un valor entre 0 y 1. Cuando se muestre este error significará que el valor dado a ‘num_unos’ es erróneo. El valor que deberá tomar este parámetro será un número decimal contenido entre 0 y 1. Ambos valores inclusive.

ERROR 17. Variable 'operadores_operaciones' es errónea. Debe ser 0 ó 1. La variable ‘operadores_operaciones' sólo puede tomar dos valores (0 ó 1). En el caso de que se muestre este valor significará que esta condición no se cumple.

ERROR 18. Variable 'tipo_ejecuccion' es errónea. Debe ser 0 ó 1. La variable ‘tipo_ejecuccion’ sólo puede tomar dos valores (0 ó 1). En el caso de que se muestre este valor significará que esta condición no se cumple.

ERROR 19. Variable 'numero_individuos' es errónea. Debe ser mayor de 1. Cuando se muestre este error significará que el valor dado a ‘numindividuos’ es incorrecto. Este valor debe ser siempre un número entero mayor de 1.

ERROR 20. Variable 'numero_rondas' es errónea. Debe tener un valor positivo. Este parámetro indica el número de rondas que realizará el algoritmo genético. Este valor nunca podrá contener números negativos ni podrá estar vacío. En el caso de que aparezca este error, habrá que comprobar que el valor que aparece en el fichero conf.txt sea positivo.

ERROR 21. Variable 'numero_operaciones' es errónea. Debe tener un valor positivo. Este parámetro indica el número de operaciones de cada individuo. Este valor nunca podrá contener números negativos ni podrá estar vacío. En caso de que no quiera que los individuos contengan operaciones indíquelo poniendo numero_operaciones= 0.

ERROR 22. Variable 'numero_registros_train' es errónea. Debe tener un valor positivo. Dado que este parámetro es el número de registros o de datos que hay en el fichero destinado al entrenamiento debe ser un número positivo. Si se muestra este error significará que no se encuentra el valor en el fichero conf.txt o que es un número negativo.

ERROR 23. Variable 'numero_registros_test' es errónea. Debe tener un valor positivo. Este error indica que el numero_registros_test no aparece en el fichero conf.txt ó se ha introducido en él un valor no válido. El valor de este campo deberá ser siempre un número entero positivo.

ERROR 24. Variable 'nombre_fichero_train' incorrecta. Compruebe que está en el fichero conf.txt y su valor es correcto. Cuando aparezca este error significará que no se encuentra el nombre del fichero de datos. Compruebe que existe ese campo en conf.txt y que lo ha rellenado.

ERROR 25. Variable 'nombre_fichero_test' incorrecta. Compruebe que está en el fichero conf.txt y su valor es correcto. No se encuentra el fichero destinado al test. Compruebe que está relleno ese campo en conf.txt.

ERROR 26. El fichero de configuración 'conf.txt' no se encuentra en la carpeta 'configuracion'. El programa no localiza el fichero 'conf.txt'. Compruebe que el fichero está dentro de la carpeta y que su nombre es correcto. Además compruebe que el nombre de la carpeta sea el correcto. La carpeta ha de llamarse 'configuracion' (en minúsculas y sin acentos).

ERROR 27. El fichero 'nombre del fichero train' no se encuentra en la carpeta 'datosBCI'. Compruebe que el nombre este bien escrito. El programa no localiza el fichero elegido para realizar el entrenamiento. Compruebe que en 'conf.txt', en la línea 'nombre_fichero_train=' escribió correctamente el nombre del fichero elegido. Además, si con esto no se soluciona, compruebe que el fichero está dentro de la carpeta 'datosBCI' y que el nombre del fichero y de la carpeta está correctamente escrito.

ERROR 28. El fichero 'nombre del fichero test' no se encuentra en la carpeta 'datosBCI'. Compruebe que el nombre este bien escrito. El programa no localiza el fichero de datos elegido para realizar el 'test'. Compruebe que en el fichero 'conf.txt' en la línea 'nombre_fichero_test=' escribió a continuación correctamente el nombre del fichero elegido. Además, compruebe que el fichero está dentro de la carpeta 'datosBCI' y que el nombre del fichero y de la carpeta está correctamente escrito.

Error 29. El fichero 'train aleatorizado' no existe o la ruta ha sido mal introducida. Se produjo un error al aleatorizar. Se ha producido un error en la aleatorización de los datos. Controle que el fichero contiene el número de registros que indico en la variable 'numero_registros_train'. Si se ha indicado un número mayor al que tiene pueden producirse errores.

Error 30. El fichero test aleatorizado' no existe o la ruta ha sido mal introducida. Se produjo un error al aleatorizar. Se ha producido un error en la aleatorización de los datos. Controle que el fichero contiene el número de registros que indicó en el campo 'numero_registros_test').

Error 31. Variable 'tamanio_torneo' es errónea. Debe ser mayor que 2 y menor que 'numero_individuos'. El valor de 'tamaño_torneo' no es válido. Debe comprobar que este valor es un número entero mayor que 2 y menor que el número de individuos de la población ('numero_individuos').

Error 32. La suma de 'porcentaje_sumas', 'porcentaje_restas', 'porcentaje_multiplicacion', 'porcentaje_division' es diferente a 100'. El error viene producido al haber elegido los porcentajes de cada tipo de operación. Evidentemente la suma de los 4 porcentajes deberá dar 100%. Esta condición es la que no se cumple, revise los porcentajes.