UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD IZTAPALAPA

DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

Definición de una red neuronal perceptron por medio de un programa evolutivo con cromosomas de longitud variable para la solución de problemas de clasificación

Maestría en Ingeniería Biomédica

Alma E. Martínez Licona

T E S I S

Asesor: Dr. John Goddard Close

Sinodales: Dr. Angel Kuri

M. en I. Oscar Yáñez Suárez

México D.F., Mayo 2002

INDICE

INTRODUCCIÓN

Un problema existente en la actualidad es el de poder clasificar un conjunto de objetos de acuerdo a sus características para tener un mejor conocimiento de éstos, y poder manejarlos de la mejor manera posible.

Dicho problema se encuentra en diversas áreas donde el hombre se desenvuelve, por ejemplo, en el ambiente médico, el poder caracterizar una enfermedad y a su vez clasificarla ayuda a obtener un diagnóstico médico el cual no es fácil si el paciente tiene varios síntomas que no pertenecen a una sola enfermedad. El poderle proporcionar al médico una herramienta que pueda indicarle un diagnóstico lo más aproximado posible a la(s) enfermedad(es) del paciente no implica sustituir su labor sino ayudarlo a una mejor comprensión de la información que obtiene de sus pacientes, ayudándolo, por ejemplo en caso de cansancio, a contemplar toda la información sin omitir ningún detalle.

Otra área importante donde surge este problema es en el reconocimiento automático del habla donde se requiere hacer una clasificación de la señal de voz por medio de frecuencias que trasmiten la mayor energía acústica, estas frecuencias son conocidas como formantes, las más importantes para la caracterización de la voz son las primeras cuatro denominadas F0, F1, F2 y F3 las cuales dan información como el sexo del hablante, la entonación, etc.

Si buscamos en todas las áreas posibles se encontrarán objetos de estudio que puedan clasificarse de alguna manera por medio de sus características; el problema que se presenta son los grandes volúmenes de datos a manejar que hacen de esta labor una tarea difícil a realizar por un ser humano. De aquí la importancia de definir un clasificador automático que ayude a los seres humanos a realizar dicho trabajo.

Las redes neuronales artificiales perceptron multicapa han demostrado ser buenos algoritmos para resolver problemas de clasificación [1], sin embargo esto da lugar a otro problema, ya que existen numerosas arquitecturas. Se tendría que definir qué red neuronal se ocuparía y qué características tendría para la solución de un tipo de problema específico.

Si se toma como base [1] la red neuronal artificial queda definida como un perceptron multicapa de una sola capa oculta, ahora el problema es el número de nodos ocultos, así como los pesos de las conexiones. El número de nodos de entrada y de salida queda definido con el número de atributos y de clases de los objetos a clasificar.

Es aquí donde entran los programas evolutivos para la definición de una red neuronal artificial, el presente trabajo es una propuesta de un programa evolutivo para la definición de redes neuronales tipo perceptron multicapa (de una sola capa oculta) con cromosomas de longitud variable.

Cada individuo de la población del programa evolutivo representa una arquitectura de red neuronal inicializada de forma aleatoria, cada nodo del cromosoma representa un nodo oculto de la red neuronal y como el número de nodos ocultos es aleatorio al inicializar la población esto define el nombre de cromosomas de longitud variable, por medio de la aplicación de operadores genéticos (mutaciones) se trata de encontrar la arquitectura de una

red neuronal adecuada para la solución del problema de clasificación que se le proporcione al programa evolutivo.

El programa evolutivo se encargará entonces de encontrar el número de nodos ocultos de la red neuronal perceptron así como los pesos de las conexiones de todos los nodos de la red.

Es importante aclarar la terminología a usar, en este trabajo se menciona que se codifica cada red neuronal en un cromosoma utilizando éste termino tal como lo maneja Michalewicz, en su libro "Genetic Algorithms + Data Structures = Evolution Programs"[2], donde define a cada individuo de una población con el nombre de cromosoma, sin embargo, en la literatura se asigna el término genotipo a la codificación de cada solución potencial y de fenotipo a la decodificación o evaluación de ésta. En el presente escrito se maneja el termino de cromosoma sin hacer diferencia entre estos.

También es importante mencionar que existen varias metodologías para resolver dicho problema. Las máquinas de vector soporte son otra alternativa para la definición de una red neuronal perceptron de una capa oculta, la idea que se implementa en esta metodología es la siguiente: Se mapea el vector de entrada x a un espacio Z de dimensión mayor a través de un mapeo no lineal, construyendo un hiperplano el cual hace una separación mejor de las clases. (se consideran dos clases únicamente)

Las máquinas de soporte vectorial utilizan un kernel (seleccionado de acuerdo al problema) para hacer el cambio de espacio, para el problema de la definición de la red neuronal se define el kernel de la siguiente manera[3]:

$$k(x,x_i) = S[v(x \cdot x_i)+c] \qquad \text{fórmula 1}$$

donde S(u) es una función sigmoide, el kernel sigmoide tanh(vu)+c con $|u| \leq 1$ satisface la condición Mercer (la cual garantiza una posible expansión del espacio actual al nuevo) sólo para algunos valores de los parámetros de v,c, para estos valores de los parámetros uno puede construir máquinas de soporte vectorial implementando las reglas

$$f(x,\alpha) = \text{sig}\{\sum_{i=1}^{N} \alpha_i S(v(x \cdot x_i) + c) + b\}$$

Las α se obtienen al aplicar la fórmula 1

Utilizando la técnica descrita automáticamente se encuentra lo siguiente:

1. El número de nodos ocultos corresponde al números de los vectores soporte
2. Los pesos de la red neuronal que van de la capa de entrada a la capa oculta corresponde a los x_i
3. Los pesos de la red neuronal que van de la capa oculta a la capa de salida son los valores de α

Sin embargo el presente trabajo se enfocará al primer método mencionado y su organización es la siguiente: el primer capítulo denominado Antecedentes describe

detalladamente en que consiste el problema de clasificación mencionado al principio de esta sección, de ahí la necesidad de hablar de los clasificadores automáticos y de las características de las bases de datos que se les proporciona a dichos clasificadores, se comienzan a definir un conjunto de conceptos que se irán manejando en el desarrollo del trabajo y se describen los problemas que existen con las bases de datos con las que se trabajan, problemas tales como la falta de información en algunos valores de los atributos que caracterizan a un objeto o la poca información que existe para la clasificación, es decir que los datos no son representativos o que estos se traslapan ocasionando dificultad en su clasificación. La siguiente sección aborda la teoría de los algoritmos genéticos y la programación evolutiva así como la descripción de los operadores genéticos existentes y la diferencia entre estos dos métodos.

El capítulo 2 llamado Metodología describe el problema de la definición de una red neuronal perceptron multicapa para ser aplicada en la solución de problemas de clasificación (visto en el primer capítulo) y se mencionan cuatro trabajos representativos de cómo los autores aplicaron los algoritmos genéticos y la programación evolutiva para la definición de una red neuronal, en este mismo capítulo se hace una propuesta de un programa evolutivo describiendo como influyeron los trabajos mencionados en su definición, se hace una descripción detallada de los operadores genéticos ocupados, y la forma de codificación de la red neuronal al cromosoma del programa evolutivo.
En este capítulo se mencionan las alternativas probadas que no proporcionaron un mejoramiento significativo en el programa pero que ayudaron a la evolución de éste.

El último capítulo, el capítulo 3, es el de resultados y conclusiones donde se hace una discusión del desempeño del algoritmo.

Al final se incluyen cuatro apéndices:

El apéndice A describe brevemente la teoría de las redes neuronales artificiales tipo perceptron, que incluye cómo funcionan, introduciendo conceptos como lo es la función de transferencia, medidas de aptitud de la red, etc. se menciona ejemplos de datos linealmente separables y cuáles no y se describe el algoritmo de retropropagación para el ajuste de los pesos de la red, así como un conjunto de aplicaciones.

En el apéndice B se describen seis bases de datos que se seleccionaron para probar la eficiencia del algoritmo propuesto, dos de estas bases de datos son muy conocidas y utilizadas en la prueba de algoritmos de clasificación (iris e indios pima), cuatro de las bases de datos están relacionadas con el área de ingeniería biomédica (indios pima, desórdenes cardíacos, proteínas y Peterson Barney) solo una base de datos es seleccionada aleatoriamente y se refiere a la clasificación de vinos. Estas bases de datos reúnen ciertas características entre las que sobresalen las siguientes, existen resultados publicados de un porcentaje de acierto de clasificación obtenido por medio de otros algoritmos, esto es útil para hacer una comparación con los porcentajes obtenidos con el algoritmo propuesto, algunas bases de datos tienen datos traslapados haciendo difícil la tarea de separación de clases, es decir, no son las clases linealmente separables, existen bases de datos con datos perdidos, es decir, hay valores de algunos atributos desconocidos, estas bases de datos varían en el número de ejemplos, así como en el número de nodos de entrada y de salida.

En el apéndice C se encuentran los valores de los pesos de las conexiones de las arquitecturas encontradas por el programa evolutivo de cromosomas de longitud variable para cada base de datos descrita.

Finalmente en el apéndice D se proporciona el código en C del programa evolutivo con una breve descripción para el usuario.

Cabe mencionar que este trabajo ha sido publicado en la Revista Mexicana de Ingeniería Biomédica, en el número 1, Vol XXII, año 2001, pág 4-11. Aceptado como artículo de investigación original. Dicha revista se encuentra actualmente en el padrón de excelencia de CONACYT.

Además se están aplicando técnicas de paralelización al algoritmo propuesto para comparar su desempeño con dichas técnicas.

Referencias

[1] Mohamad H. Hassoun, "Fundamentals of Artificial Neural Networks", The MIT Press 1995.
[2] Zbigniew Michalewicz , "Genetic Algorithms + Data Structures = Evolution Programs" 3ra ed., Edit. Springer, 1996, pp 15.
[3] Vladimir N. Vapnik. "The nature of Statistical learning theory", 2"ed., Edit.Springer. 1999, pp 146.

PROBLEMA DE CLASIFICACIÓN

El problema de clasificación consiste en agrupar conjuntos de objetos de acuerdo a sus características, los grupos de objetos con características comunes forman una clase. Este problema se encuentra en diferentes áreas (diagnóstico médico, reconocimiento automático del habla, etc) con grandes volúmenes de datos a manejar, lo cual hace imposible su clasificación de forma manual, es por esto que se han diseñado métodos y herramientas para apoyar el proceso de clasificación buscando que éste se haga de forma automática. En este capítulo se explica el problema de clasificación y se define qué es un clasificador automático. (La información de este capítulo es tomada de la cita [1,2], en las citas [3,4] se encuentran conceptos de clasificación aplicados específicamente a una red neuronal)

Problema de clasificación

En la actualidad se cuenta con una gran cantidad de información en diferentes áreas tales como medicina, botánica, etc., la cual debe ser ordenada de alguna forma para obtener un mejor provecho de ésta. El objetivo es darle al hombre la posibilidad de manipular los datos según sus necesidades, dentro de cada una de las áreas en las que se mueve. Para ello se analizan los objetos en cuestión y se agrupan de acuerdo a sus características comunes, éstas reciben el nombre de atributos. Por ejemplo, en el diagnóstico médico, el objeto a analizar sería cada una de las enfermedades y sus atributos son las características que la definen como los rangos de temperatura, los rangos de presión, niveles de azúcar, etc. Al encontrarse un objeto no analizado se compara con los que ya lo están y se asigna al conjunto de objetos con los que más similitud tiene con sus características. De esta forma un objeto puede pertenecer a varios grupos dependiendo de la característica que se esté evaluando. También puede darse el caso en el que al intentar asignar el objeto a un grupo de los previamente formados, se encuentre que no es lo suficientemente común a los criterios de ningún grupo como para poder pensar que pertenece a él; entonces se asigna a uno de nueva creación. (Por ejemplo cuando se encuentra una nueva enfermedad, como es el caso del SIDA que tiene poco tiempo de estudio comparada por ejemplo con la tuberculosis). Estos grupos reciben el nombre de clases y el proceso de agrupación que se basa en atributos comunes se conoce con el nombre de clasificación. Regresando al ejemplo del diagnóstico médico, cada clase es el nombre específico de cada enfermedad, como puede ser diabetes, tuberculosis, poliomelitis, etc.

Otro ejemplo sería un sujeto, considerando a una persona como objeto, los atributos a utilizar para su clasificación depende del interés específico de ésta, es decir, si lo que nos interesa es hacer un censo económico los atributos a considerar serían: el monto total de ingresos, el monto total de egresos, el tipo de vivienda que habita, el número de dependientes económicos, si tiene automóvil o no, etc. Si por el contrario se desea hacer una clasificación racial, interesarán sus atributos tales como: color de su piel, color de pelo, estatura, rasgos antropológicos etc.

La necesidad de clasificar es más compleja en algunas ciencias dado el volumen y complejidad de objetos a clasificar teniendo así que recurrir a la creación especifica de áreas que se dediquen exclusivamente a la clasificación de los objetos que estudia la ciencia en cuestión, Un ejemplo de esta situación es la taxonomía

Taxonomía: (del griego: arreglo y ley [1]) Parte de la historia natural que trata de la clasificación sistemática de los seres, basándose en las diferencias que existen entre ellos; las clasificaciones taxonómicas que se aplican a los animales son:

1. Reino y subreino
2. Tronco, subtronco e infratronco
3. Supertipo, tipo y subtipo
4. Superclase, clase, subclase e infraclase
5. Superorden orden y suborden
6. Superfamilia, familia y subfamilia
7. Género
8. Especie y subespecie
9. Variedad.

Un ejemplo de clasificación que la taxonomía hace tomando como objeto de estudio al hombre es la siguiente:

Atributo	**Valor**
Reino:	Animal
Subreino:	Metazoarios
Tronco:	Eumetazoontes
Subtronco:	Heteraxonios
Infratronco:	Eucelonios
Supertipo:	Enterocelos
Tipo:	Cordados
Subtipo:	Vertebrados
Clase:	Mamíferos
Subclase:	Terios
Infraclase:	Euterios
Orden:	Primates
Suborden:	Antropoides
Superfamilia:	Hominoideos
Familia:	Homínidos
Género:	Homo
Especie:	Homo Sapiens

Retomando el ejemplo del diagnóstico médico que se menciono anteriormente, este se basa en gran parte en un proceso de clasificación. Dado un conjunto de síntomas se busca aquella enfermedad o grupos de enfermedades que tengan entre sus atributos de clasificación el conjunto de síntomas del caso en estudio; entre mas síntomas se conozcan más preciso será el diagnóstico. Por ejemplo, los principales síntomas de la diabetes son: altos niveles de azúcar en la sangre y en la orina, urinación frecuente, apetito inusual, sed excesiva, pérdida de peso, debilidad y cansancio, irritabilidad y cambios de ánimo,

[1] Diccionario enciclopédico ilustrado. Selecciones del Readers Digest, Décima Edición, 1978

sensación de malestar en el estómago y vómitos, infecciones frecuentes, vista nublada, cortaduras y rasguños que no curan o que curan lentamente, picazón o entumecimiento en los pies e infecciones recurrentes en la piel, la encía o la vejiga.

Dada la importancia que tiene el proceso de clasificación, es necesario diseñar métodos y herramientas que lo apoyen para lograr mayores y mejores aplicaciones.

Clasificador automático

Este tipo de clasificador realiza la tarea de clasificación basándose en los atributos que hacen particular a un objeto en el universo; el agrupamiento de atributos similares forma las clases. La clasificación automática se enfrenta a dos problemas. El primer problema puede enunciarse de la siguiente forma: dado un conjunto de objetos y su conjunto de atributos, formar las clases en las cuales estos puedan clasificarse. Asimismo el segundo problema consiste en: dado un conjunto de clases y un objeto específico encontrar a qué clase pertenece el objeto en cuestión. Dado que existen varias bases de datos ya clasificadas y que son usadas para probar la eficiencia de clasificadores automáticos, es este último punto el de mayor interés para este trabajo por lo que se definirá como el problema de la clasificación automática. El clasificador para este punto consta de una regla de clasificación o de decisión.

Una forma de definir una regla de clasificación o de decisión es la siguiente: es aquella regla que permite decidir a qué clase debe asignarse un objeto nuevo. Ésta debe ser tan estricta que no permita que el objeto a clasificar quede en dos clases distintas, es decir, el conjunto universo deberá poderse fraccionar en clases de forma tal que la unión de todas ellas sea igual al universo, pero deberá asegurar que no haya un elemento tal que pueda pertenecer a dos clases al mismo tiempo.

Escrito en otros términos, sean C_1, C_2, C_3...C_n clases, entonces
$C_1 \cup C_2 \cup C_3 \cup ... \cup C_n = U$, donde U es el universo
$C_i \cap C_j = \varnothing \ \forall \ i \neq j$ y
$C_i \ \forall$ i es un conjunto no vacío. Es decir se debe formar una partición

Un clasificador es una regla de clasificación o de decisión. En la figura 1 se muestra lo que se espera de un clasificador automático, es decir, que sí a esa regla de decisión se le da un conjunto de objetos, esta da como resultado la clase a la que pertenece cada uno. Dado que el problema de clasificación se presenta en diferentes áreas tales como diagnóstico médico, reconocimiento automático del habla, tipos climáticos, etc., y que se presentan grandes volúmenes de datos a manejar, se buscan métodos y herramientas para apoyar el proceso de clasificación buscando que éste se haga de forma automática.
De ahí el nombre de clasificador automático, ya que si se define la regla de decisión de forma adecuada esta dará resultados con un alto porcentaje de aciertos en su clasificación, el problema es como definirla.

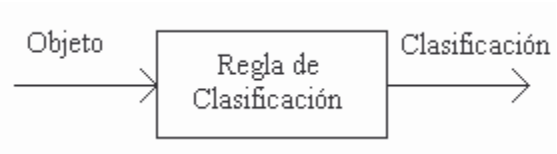


Fig. 1 Clasificador Automático

Es necesario definir de manera más detallada cada uno de los conceptos involucrados en la tarea de un clasificador.

Atributos

Son las características, cualidades o propiedades que se consideran relevantes de un objeto para el fin del estudio. Existen atributos del tipo nominal, discreto y continuo. Cuando el conjunto de valores de los atributos es del tipo no numérico como raza, país de origen, sexo etc. se dice que es un atributo nominal, si por el contrario son cuantitativos éstos se dividen en continuos y discretos. Ejemplos de atributos cuantitativos discretos son: el número de hijos de una familia, el número de tiempos en un motor, el número de materias de una carrera etc., y algunos ejemplos de atributos cuantitativos continuos son: la estatura de una persona, la temperatura de un cuerpo, el peso de un cuerpo, etc.
Estos atributos se utilizan para determinar cuando dos objetos pertenecen a una misma clase.

Clases

Son los conjuntos de objetos con atributos similares, para el caso de estudio se supone que el conjunto de clases es un conjunto finito. Al final el clasificador automático da como resultado a que clase pertenece el objeto que se le dio para clasificar de acuerdo al valor de sus atributos.

El conjunto de datos

Los datos de entrada del clasificador son los objetos que están definidos por un conjunto de atributos y los valores que cada uno de los objetos tiene en cada uno de estos atributos expresados en forma de vector. Para entrenar al clasificador por lo general se utiliza un 70% de la base de datos y el resto se usa para su prueba. Es importante señalar que el conjunto de datos de entrada para el clasificador puede estar incompleto, tener ruido, ser difíciles de clasificar, etc. estos problemas se discutirán más adelante dentro de este mismo capítulo.

Regla de Decisión

El clasificador está definido por un algoritmo o regla de decisión, que permite asignar un nuevo objeto a la clase a la cual pertenece. El algoritmo describe una función, $f: X \rightarrow C$, que

mapea el conjunto de características *X* en el conjunto finito de clases *C*. Los subconjuntos así formados definen una partición del conjunto. Las regiones que separan una partición de otra se les denomina fronteras de decisión (Fig. 2).

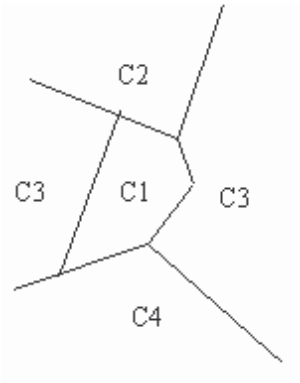


Fig.2 Fronteras de decisión

La calibración

La calibración del clasificador recibe el nombre de entrenamiento y los ejemplos usados para configurarlo forman el conjunto de entrenamiento. Al primer problema de la clasificación automática que se mencionó con anterioridad se le conoce como aprendizaje no supervisado; éste consiste en tratar de asociar clases a un grupo de ejemplos. Al segundo problema de la clasificación automática se le llama aprendizaje supervisado debido a que el clasificador aprende a asociar su categoría correcta a cada vector ya clasificado en el conjunto de entrenamiento.

Problemas que surgen con las bases de datos

El conjunto de datos que se le proporciona al clasificador, por lo general cuenta con una serie de problemas que se deben de tomar en cuenta antes de tratar de clasificarlos. Estos problemas nos lleva a aplicarles algún tipo de preprocesamiento de no ser así es posible que el clasificador nunca obtenga un porcentaje de clasificación aceptable para estas bases de datos.

A continuación se describen algunos problemas típicos en las bases de datos:

1. Datos incompletos, es decir, el valor de uno o varios atributos de uno o varios objetos pueden no estar presentes; esto dificulta la labor del clasificador por lo que se deben tomar acciones previas a la introducción de los datos al clasificador, como por ejemplo eliminar la información redundante o incompleta. Una forma de lograrlo es estimándola mediante la obtención del promedio de los demás sujetos de la muestra.

Un ejemplo de esto se encuentra en la base de datos de los indios pima, donada por V. Sigillito y la cual es usada para estimar el desempeño de clasificadores automáticos. Esta base de datos representa la información de un estudio con mujeres de herencia pima y mayores a 20 años para descubrir si mostraban signos de diabetes, según criterios de la

Organización Mundial de la Salud. Consta de 8 variables numéricas representando atributos como edad, número de embarazos, la concentración de glucosa plasma, y un índice de masa corporal. Los promedios para estas variables son 33.2, 3.8, 120.9, 32.2. La base de datos consta de 728 casos en total, donde 500 mujeres están clasificadas como sanas y 268 como enfermas, el problema de esta base de datos es que faltan algunos valores.

Este tipo de problemas surge a menudo en estudios médicos por varias razones, tal vez por error no se capturo el valor de alguna medida, o no se pudo realizar algún estudio o prueba con algún paciente, o hubo pérdidas de análisis por parte de algún laboratorio, etc.

Esto nos lleva a decidir cómo manejar los datos faltantes, como se mencionó al principio de la sección, podría omitirse, aunque esto nos lleva a otro problema, cuánta información se omitiría y de los 728 ejemplos al final cuántos ejemplos quedarían, si se omiten demasiados tal vez la base de datos no sea tan representativa como se desearía o si los datos faltantes son en su mayoría de la clase enferma, esto sesga mucho los resultados ya que existen más ejemplos sanos y el clasificador aprendería más una clase sobre la otra. Otra solución es estimar sustituyendo el promedio de los valores de los otros datos en la misma clase del atributo faltante.

2. Cuando se tiene una mezcla de tipos de datos en los valores de los atributos, es decir, cuando hay variables de tipo numéricas, nominales y ordinarias cuyas unidades físicas no están relacionadas es necesario hacer una codificación adecuada para uniformar los datos, esta codificación muchas veces marca una diferencia en los porcentajes de aciertos de un clasificador. Este tipo de problema se encuentra muy a menudo, una forma de resolver estos problemas es la siguiente: Los valores de los atributos, aún siendo nominales, se pueden sustituir por valores numéricos, es decir, se pueden codificar. Esta acción facilita su interpretación al poder asociar un vector con los valores correspondientes de los atributos. Por ejemplo, si requerimos clasificar la población mexicana por su estado de nacimiento un método práctico es numerar cada uno de los estados para después hacer referencia al estado de nacimiento de un individuo a través de este número.

3. Normalización, para facilitar la tarea de un clasificador, a menudo es necesario normalizar o escalar los valores de los atributos. La normalización consiste en ajustar los valores de los atributos en un rango de valores acotado, frecuentemente entre 0 y 1. Una forma para normalizar los datos es:

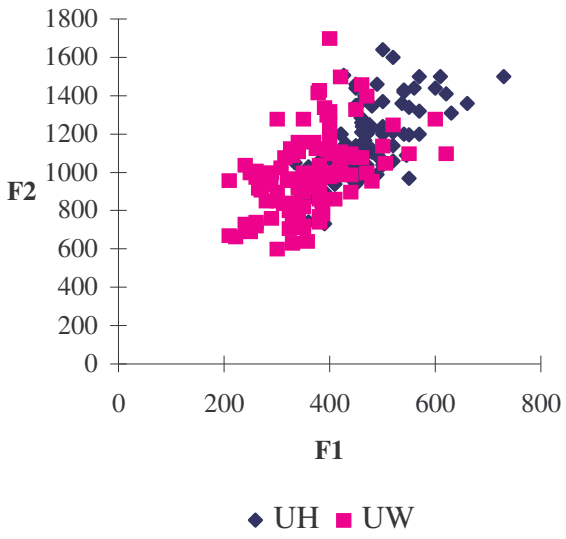
$$\frac{(X - X_{\min})}{X_{\max} - X_{\min}}$$

donde X es el valor a normalizar, y X_{min} y $X_{máx}$ son los valores mínimos y máximos respectivamente del atributo.

4. Que la base de datos sea representativa. Es importante considerar al momento de tomar un conjunto de datos, que la muestra sea lo suficientemente representativa y no que se encuentre sesgada hacia un valor en específico. Por ejemplo, si el 99% de los ejemplos

pertenece a una sola clase y la regla de decisión es aquella que siempre asigna un nuevo ejemplo a la primera clase, la clasificación resultará en una tasa de error del 1 %, siendo una cifra muy engañosa con respecto a la situación real. Si tenemos el primer problema y decidimos eliminar la información faltante esto nos ocasionaría el problema que estamos definiendo, por lo tanto es muy importante tomar en cuenta todas las acciones que decidamos hacerle a la base de datos y ver que consecuencias nos ocasionarían.

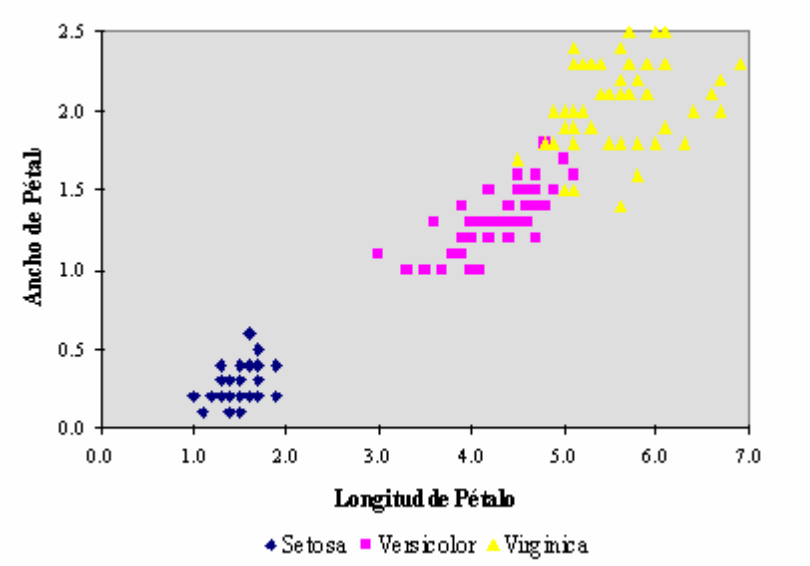
5. Ruido en la señal, en la Ingeniería Biomédica se trabaja con una serie de señales para ser clasificadas, una de ellas es la señal de la voz, la forma en como se caracteriza es por medio de frecuencias que trasmiten la mayor energía acústica, estas frecuencias son conocidas como formantes, existen varias formantes pero las más importantes para la caracterización de la voz son las primeras cuatro denominadas F0, F1, F2 y F3 las cuales nos dan información como el sexo del hablante, la entonación, etc. En 1952 Peterson y Barney usaron un conjunto de datos que consiste en dos repeticiones de diez vocales por 76 hombres, mujeres, y niños dando un total de 1520 ejemplos. Las cuatro primeros formantes de cada una de las vocales fueron tomados como los atributos, y las diez vocales como las clases. En la gráfica 1 se representan solo dos formantes (F1 y F2) para las vocales UH y UW, en esta gráfica se observa bastante variabilidad en las frecuencias y traslape entre ellas ocasionando una mayor dificultad para el clasificador ya que debido al traslape se observa que no son linealmente separables y debido a la variabilidad de los datos es probable que se encuentre la señal con ruido. El ruido de una señal, en este caso de voz, puede ser ocasionado por varias razones, ya sea por la forma en que se tomaron las muestras, por un mal funcionamiento del micrófono o mal aislamiento de la señal teniendo ruido de fondo, etc



Gráfica 1. Representación de las formantes F1 y F2 de la base de datos de Peterson y Barney

Este ejemplo muestra la dificultad para caracterizar un sonido vocal ‘típico’, y advierte sobre la complejidad del reconocimiento automático del habla, y de la clasificación automática en general.

6. Cuando los datos se traslapan, este problema más que de la base de datos es para el clasificador ya que existen muchas bases de datos que no son linealmente separables y las fronteras de decisión están muy pegadas o traslapadas provocando confusión en el clasificador para realizar su tarea, es por esto la importancia de diseñar técnicas o herramientas robustas para clasificación. Un ejemplo de este tipo de datos es el que se mencionó anteriormente (base de datos de Peterson y Barney) otro ejemplo lo es la base de datos de la planta Iris. En 1936 el famoso estadístico Fisher, estudió un conjunto de datos sobre la planta Iris que es quizá el más conocido en el campo del reconocimiento de patrones. El conjunto consiste de 150 ejemplos divididos en tres clases, cada clase consta de 50 ejemplos, las clases representan las clasificaciones de iris setosa, iris versicolor, e iris virgínica. Cada ejemplo tiene cuatro atributos numéricos reales que corresponden al ancho y longitud del sépalo y pétalo. En la gráfica 2 se muestra la localización de los atributos del pétalo.



Gráfica 2. Dispersión de los datos del pétalo de la base de datos de la planta Iris

En este caso la clase de la planta iris setosa puede ser separada de las otras dos clases por un hiperplano, es decir, es linealmente separable de las otras clases, mientras que se observa que en las clases versicolor y virginica no se pueden separar tan fácilmente.

Estos ejemplos dan una idea de lo complejo que es definir la regla de decisión para lograr una buena clasificación, sobre todo si las bases de datos con las que se esta trabajando tienen más de un problema que considerar, lo cual sucede a menudo en datos de situaciones cotidianas.

Desempeño de un clasificador

Anteriormente se mencionó la calibración de un clasificador, en esa parte se mencionó que se debe entrenar al clasificador antes de ser usado con casos nuevos, para realizar este trabajo, el conjunto de datos con el que se esta trabajando se divide aleatoriamente en dos

subconjuntos, un conjunto de entrenamiento y otro de prueba. Es común asociar dos terceras partes del conjunto de datos al entrenamiento y el resto al de prueba. El conjunto de entrenamiento es utilizado para configurar al clasificador, y se calcula una tasa de error para obtener una medida de la eficiencia del clasificador, es decir, que tan bien clasifica. Una forma de estimar esta tasa de error es por medio de la siguiente fórmula la cual se denomina tasa de error de la muestra de prueba:

$$Tasa\_de\_error = \frac{Núm.\_de\_errores}{Núm.\_de\_casos}$$

Un segundo método llamado *"dejar uno fuera"* consiste en configurar el clasificador con *m-1* de los *m* elementos del conjunto de datos y realizar la prueba con el dato restante; este proceso se repite el número de veces de datos que se tengan, y al final se promedian las tasas de error. El hecho de tener que repetir la clasificación *m* veces implica una gran cantidad de cálculos; no obstante con este método se obtienen mejores resultados. Una generalización al método se denomina *"validación cruzada"* y consiste en tomar el conjunto total de datos y dividirlo en un número *n* de subconjuntos de tamaño similar para posteriormente aplicarles a cada uno el método de *"dejar uno fuera"*, así solamente se realiza una clasificación en cada subconjunto. Los errores encontrados en cada conjunto se promedian para obtener la tasa de error.

Existen varios métodos para medir la eficiencia de un clasificador, por ejemplo contabilizar el número de aciertos y errores o en el caso de las redes neuronales para clasificación calcular el error cuadrático medio ECM de la siguiente forma:

$$ECM=\sum(y_i-d_i)^2$$

Donde y_i es la salida de la red y d_i es la salida deseada.

Con estas técnicas de medición se trata de ver el poder de generalización que tiene el clasificador, es decir, una vez calibrado o entrenado que tan bien clasifica los casos nuevos (que no estaban en el grupo de entrenamiento)

Como se observa, el problema de clasificación no es fácil de resolver, involucra una serie de consideraciones que hay que definir para lograr un buen clasificador automático. En los capítulos siguientes se explicarán técnicas de Inteligencia Artificial y de acuerdo a éstas se dará una propuesta de un clasificador automático.

Referencias

[1] Notas no publicada de reconocimiento de patrones autor Dr. John Goddard.
[2] Tom M. Mitchell , "Machine Learning", McGraw Hill International, 1997, pp 54-56.
[3] Russ Eberhart, Pat Simpson, Roy Dobbins, "Computational Intelligence PC Tools", Edit. AP Professional, 1996, pp. 62, 72, 83, 84, 116 .

[4] Simon Haykin “Neural Networks” Macmillan, 1994, cap. 4, pp 106-113.

ALGORITMOS GENÉTICOS Y PROGRAMACIÓN EVOLUTIVA

Los algoritmos genéticos y la programación evolutiva así como las redes neuronales artificiales, son ramas importantes dentro del área de la Inteligencia Artificial. Los dos primeros son conceptos que se forman por un conjunto de algoritmos que comparten la misma base conceptual de simular la evolución de una población; cada individuo de la población representa una solución potencial a un problema y cada solución representa un cromosoma. La evolución se simula usando operadores genéticos tales como selección, cruzamiento y mutación. En los capítulos previos se explicó el problema de clasificación y se mencionó que una red neuronal perceptron podría ser una buena solución al problema, sin embargo la definición de la arquitectura de la red que se va a ocupar es también otro problema a resolver. Existen varios trabajos que han demostrado que los algoritmos genéticos así como la programación evolutiva pueden definir la arquitectura de una red neuronal para la solución de problemas de clasificación. En este capítulo se explicarán qué son los algoritmos genéticos y la programación evolutiva.

Breve semblanza de la evolución natural

El principio fundamental de la selección natural como principio evolutivo fue formulado por Charles Darwin mucho tiempo antes de descubrir los mecanismos genéticos. En su obra, *El origen de las especies*, publicada en 1859, Darwin defendió el principio de la evolución mediante la selección natural cuyos puntos importantes son los siguientes: cada individuo tiende a transmitir rasgos a su hijo, sin embargo, la naturaleza produce individuos con rasgos diferentes, los individuos más adaptados, aquellos que poseen los rasgos más favorables, tienden a tener más hijos que aquellos con rasgos no favorables, conduciendo así, a la población, como un conjunto de especies hacia la obtención de rasgos favorables. Durante largos períodos se puede acumular la variación, produciendo especies completamente nuevas cuyos rasgos las hacen especialmente adaptadas a nichos ecológicos particulares[1].

En 1865 G. Mendell descubre los principios básicos de transferencia de los factores hereditarios de los padres a su descendencia. Sin embargo, la genética fue completamente desarrollada por T. Morgan y sus colaboradores quienes probaron experimentalmente que los cromosomas son los principales transportadores de la información hereditaria y que los genes, los cuales presentan los factores hereditarios, están entrecruzados sobre un cromosoma[2].

Algoritmos evolutivos

Los algoritmos evolutivos son un conjunto de algoritmos basados en la evolución natural, existen diferentes tipos de algoritmos evolutivos, los cuales se conocen como, algoritmos genéticos, programación evolutiva y estrategias evolutivas. La diferencia entre los algoritmos evolutivos puede estar definida en las diferentes representaciones o formas de codificación, esquemas de selección y operadores de búsqueda. Por ejemplo, los algoritmos genéticos por lo general usan como operadores de búsqueda el cruzamiento como principal operador y como secundario a la mutación (aunque a veces suele sólo aplicarse el cruzamiento) mientras que la programación evolutiva solamente usa mutación.

Los algoritmos genéticos hacen énfasis a la evolución genética mientras que los programas evolutivos ponen más énfasis a la evolución de la conducta (comportamiento adaptivo). La respuesta de cual algoritmo es el mejor para ser aplicado depende del problema a resolver[3].

Existen dos características que distinguen a estos algoritmos de otros algoritmos de búsqueda:

1. Que todos ellos se basan en una población que representa a un conjunto de soluciones potenciales a un problema
2. Que existe comunicación e intercambio de información entre los individuos de la población, esa comunicación e intercambio de información son el resultado de los operadores de búsqueda que son aplicados a los individuos. Estos operadores de búsqueda son conocidos como operadores genéticos dentro de los algoritmos genéticos [3].

A continuación se hará una descripción de los algoritmos genéticos y la programación evolutiva sin dejar de mencionar las estrategias evolutivas así como los operadores de búsqueda que cada una de estas técnicas utiliza.

Descripción general de los algoritmos genéticos, programación evolutiva y estrategias evolutivas

Los algoritmos genéticos fueron desarrollados por un investigador de la Universidad de Michigan llamado John Holland quien estaba conciente de la importancia de la selección natural, y a fines de los 60's desarrolló una técnica que permitió incorporarla en un programa de computadora. A la técnica que inventó Holland se le llamó originalmente "planes reproductivos", pero se hizo popular bajo el nombre "algoritmo genético" tras la publicación de su libro "Adaptation in Natural and Artificial Systems"[4] en 1975 [5]. El concepto atrás de los algoritmos genéticos es la evolución natural; estos algoritmos buscan modelar algunos fenómenos naturales como la herencia genética y la lucha de supervivencia. La lucha de supervivencia implica que cada especie se adapte de la mejor manera a su ambiente el cual va cambiando con el tiempo haciéndose cada vez más complejo. La herencia genética que cada especie tiene debe mejorar de una generación a otra ya que se deben desarrollar características que permitan a la nueva especie una mejor adaptación a su medio ambiente para que pueda sobrevivir en él. Esta herencia se encuentra en los cromosomas de la especie. Por ejemplo, en el reino animal, hay especies cuyo color de piel ha ido evolucionando con el tiempo de tal forma que se mimetizan en el medio ambiente en el que viven y pasan desapercibidos a otras especies que podrían devorarlos, o crean nuevos mecanismos de defensa para su supervivencia[2].

Otra técnica muy semejante a los algoritmos genéticos es la llamada programación evolutiva. La programación evolutiva fue desarrollada por Lawrence Fogel, en un trabajo donde dirigió el proceso evolutivo en el sentido de desarrollar la habilidad de predecir cambios en un ambiente (1966). El ambiente fue descrito como un conjunto de símbolos (de un alfabeto finito) y el algoritmo evolutivo produce como salida un nuevo símbolo. Esta técnica se basa en los principios de la evolución natural, principalmente en los cambios que

tiene cada individuo para su supervivencia, el principal operador genético es la mutación[2].

La programación evolutiva esta basada es la evolución, razón por la cual las estrategias evolutivas están basadas sobre la evolución de la evolución (Rechemberg, 1994). Rechemberg llega a esa conclusión basado en que los procesos biológicos han sido optimizados por la evolución y la evolución es un proceso biológico, entonces la evolución ha sido optimizada por ella misma.

Aunque las estrategias evolutivas utilizan mutación y cruzamiento (ambas) tienen pequeñas diferencias con respecto a los algoritmos genéticos o la programación evolutiva.

La teoría detrás del concepto de mutación y cruzamiento en estrategias evolutivas resulta en una mejora del fitness solo si aterriza dentro de una ventana definida de evolución. Por otro lado el cruzamiento se hace seleccionando los M mejores individuos como padres para producir L hijos, con uniones de dos grupos de individuos, no se usa Selección para definir M. Las estrategias evolutivas no ocupan elitismo como es el caso de los algoritmos genéticos[6].

La importancia de estas técnicas radica en que existen una serie de problemas interesantes para los cuales los algoritmos tradicionales no son factibles para su solución. Muchos de estos problemas son problemas de optimización, sin embargo, métodos como los mencionados han sido aplicados dando buenos resultados.

Un problema clásico es el de colocar N reinas en un tablero de ajedrez de N x N cuadros sin que se den jaque, en donde se trataría de encontrar una posición de las piezas de entre las $N^2!(N^2-N)!/N!$ combinaciones diferentes, que en caso de un tablero real de N=8 serían más de cuatro mil millones de combinaciones, para este problema pueden encontrarse algoritmos convencionales con tiempo de resolución proporcionales a N!.

La utilización de estas técnicas se basan en la idea de fijar el objetivo del problema mediante una expresión matemática, denominada función de costo o función objetivo, la cual hay que minimizar. Dando así un valor menor a las soluciones malas al problema[7].

Otros problemas típicos de optimización se mencionan a continuación:

El problema del agente viajero, el cual consiste en dadas N ciudades que tiene que visitar un vendedor, encontrar el camino más corto para, partiendo de una de ellas, visitarlas todas sin pasar más de una vez por cada una y volviendo finalmente a la ciudad de partida. La complejidad radica en la gran cantidad de posibles caminos, muchos de ellos de longitud similar. Para N ciudades, el número de rutas alternativas es de N!/2N[7].

El problema del emparejamiento ponderado, el cual se plantea como el emparejar entidades en función de determinado criterio ponderable buscando su combinación óptima. Por ejemplo: si una empresa contrata a cuatro empleados para cuatro de sus sucursales, de tal forma que la empresa se hace cargo de los gastos de desplazamiento de los empleados, trataría de realizar la asignación sucursal-empleado que consiga que la suma de distancias

entre el domicilio de cada empleado y su sucursal sea la menor posible. No siempre el criterio utilizado en los problemas de emparejamiento es el de la distancia entre entidades; en general se determinará algún aspecto que pueda ser cuantificado y que permita valorar los costos de cada opción[7].

A continuación se describirá cada una de estas técnicas (algoritmos genéticos programación evoluitva y estrategias evolutivas).

Algoritmos genéticos

Como se mencionó anteriormente, los algoritmos genéticos se basan en los principios de la evolución mediante la selección natural y toman un vocabulario semejante a la genética natural. En los algoritmos genéticos se habla de una población formada por un conjunto de individuos, genotipos, o cromosomas. En la genética natural una especie está formada por un conjunto de cromosomas; el hombre por ejemplo cuenta con 46 cromosomas. En los algoritmos genéticos un individuo es representado por un cromosoma.

Los cromosomas se representan por un conjunto de genes, en donde cada gen controla la herencia de una o varias características.

Cada individuo o cromosoma en la población de los algoritmos genéticos, representa una solución potencial al problema que se desea resolver. La forma de representar a cada individuo es determinada de forma externa por el usuario. La codificación de estas posibles soluciones potenciales, se hace por medio de una representación binaria, el motivo principal es que teóricamente facilita el análisis de las soluciones del problema y proporcionan una forma elegante de aplicar los operadores genéticos, sin embargo, esta representación tiene sus desventajas, cuando es aplicada a problemas numéricos de alta precisión, es decir, al manejar variables reales la precisión es proporcional al de decimales utilizados. La determinación de esta precisión determina la longitud de la cadena binaria o del cromosoma que representa la cual es fija, ya que es la misma longitud para cada uno de los individuos de la población, recordando que cada individuo es representado por una cadena binaria. (Más adelante se retomará esto por medio de un ejemplo)

Una vez definida la población, queda también definido un espacio de soluciones potenciales; el proceso de evolución, donde se aplican los operadores genéticos, corre sobre la población de individuos o cromosomas para hacer una búsqueda de la mejor solución al problema a través de este espacio.

Tal búsqueda requiere balancear dos objetivos:

explorar la mejor solución y
explorar el espacio de búsqueda

Existen varias estrategias de búsqueda, a continuación se dan dos ejemplos seleccionados arbitrariamente, sin dejar de mencionar que existen más.

"Hillclimbing" es un ejemplo de una estrategia de búsqueda, el cual no explora todo el espacio de la misma, pero si intenta explorar las mejores soluciones al problema para encontrar la óptima[2].

Otro ejemplo es la búsqueda aleatoria (random), estrategia que explora todo el espacio de búsqueda, ignorando explotar solo aquellas regiones en el espacio que tienden a ser buenas candidatas a la solución buscada[2].

Dentro de este tipo de algoritmos se debe definir un parámetro de detención, es decir, cuántas veces se debe aplicar el proceso de evolución a la población para poder encontrar la solución óptima al problema. Esto no es sencillo de definir, una aproximación consiste en determinar un número máximo de generaciones de acuerdo a una ardua tarea de experimentación o detener el algoritmo cuando la población se haya estabilizado, es decir, cuando todos o la mayoría de los individuos tengan la misma aptitud (fitness), hablando en términos de problemas de optimización, la población se estabiliza cuando ya no se pueda minimizar más la función objetivo que se persigue.

El fitness es una función de evaluación definida sobre los cromosomas que indica la aptitud de cada uno, es decir, que tan bueno es cada cromosoma como solución al problema. En adelante se hará mención a la aptitud como la función de fitness del algoritmo.

Operadores genéticos

Los operadores genéticos que se aplican en estos algoritmos son: selección, cruzamiento y mutación. A continuación se explicarán estos operadores de forma general ya que dependiendo del problema a veces se hacen algunas variaciones. (Información basada de las citas [2,5].)

Métodos de selección

La reproducción es una versión artificial de la selección natural. El primero y posiblemente el más reconocido trabajo es debido a DeJong en 1975. El cual consiste en dos modelos, uno en preservar al mejor cromosoma (elitismo, el cual se mencionará más adelante en esta misma sección), el otro consiste en reducir los errores estocásticos de la rutina de selección (modelo del valor esperado). Este último se hace introduciendo un contador para cada cromosoma *v* el cual sé inicializa con el valor:

$$f(v)/\overline{f}$$

donde *f* es la función de fitness y se va decrementando por 0.5 o 1 cuando el cromosoma es seleccionado para reproducción con cruzamiento o mutación respectivamente y el denominador de la ecuación es el promedio de la función de fitness sobre todos los cromosomas. Cuando el contador del cromosoma toma un valor abajo de cero el cromosoma no esta disponible para la selección.

Otro método esta basado en la incorporación de pesos artificiales: Los cromosomas son seleccionados proporcionalmente a su rango. Estos métodos están basados en la creencia de

que la causa común de la convergencia prematura es la presencia de super individuos los cuales son mucho mejor que el fitness promedio de la población, tales super individuos tienen un gran número de descendencia e impiden a otros individuos contribuir con alguna descendencia en las siguientes generaciones. En pocas generaciones un super individuo puede eliminar material cromosomal deseable y causar una rápida convergencia a lo óptimo.

Hay muchos métodos para asignar un número de descendencia basada en un rango. Por ejemplo, Backer tomo un valor (MAX) definido por el usuario, como el límite superior para el valor esperado de descendencia y una curva lineal a través de MAX tal que el área bajo la curva igualara al tamaño de la población. De forma que se pueda fácilmente determinar la diferencia entre los números esperados de descendencia entre individuos adyacentes. Por ejemplo, para MAX=2.0 y el tamaño de la población igual a 50 la diferencia entre los números esperados de descendencia entre individuos adyacentes puede ser de 0.04 (tamaño de la población/MAX), otra posibilidad es tomar un parámetro q definido por el usuario y definir una función lineal, por ejemplo,

$$prob(rango) = q - (rango - 1)r$$

o una función no lineal, por ejemplo,

$$prob(rango) = q * (1 - q)^{rango-1}$$

donde q es la presión de selección de un individuo

Ambas funciones regresan la probabilidad de que un individuo que esta ranqueado sea seleccionado en una selección simple, (si el rango=1 significa que es el mejor individuo, si el rango=tamaño de la población es el peor individuo).

Ambos esquemas permiten a los usuarios influir en la selección de la presión del algoritmo. En el caso de las funciones lineales el requerimiento

$$\sum_{i=1}^{tam\_pob} prob(i) = 1$$

Implica que

$$q = r(tamaño\_población - 1)/2 + 1/tamaño\_población$$

si r=0 (por lo tanto q=1/tamaño_población) hay una presión de no selección en todo: todos los individuos tienen la misma probabilidad de selección de otra forma si

$$q - r(tamaño\_población - 1) = 0$$

entonces r=2/(tamaño_población(tamaño_población-1)) y q=2/tamaño_población proporciona la máxima presión de selección, es decir, si una función lineal es seleccionada para proveer la probabilidad para el ranqueo individual, un simple parámetro q, el cual

varia entre 1/tamaño_población y 2/tamaño_población puede controlar la presión de selección del algoritmo.

Para la función no lineal q no depende del tamaño de la población, los valores mayores de q implican fuerte presión selectiva del algoritmo, el valor de q se encuentra dentro del rango entre 0 y 1.

Un método de selección adicional es el de selección por torneo, el cual combina la idea de ranqueo en una eficiente e interesante forma, este método selecciona algún número k de individuos y selecciona el mejor de este conjunto de k elementos en la siguiente generación. Este proceso es repetido tantas veces como el tamaño de la población, es claro que los valores mayores de k incrementan la presión selectiva de este procedimiento, un valor típicamente aceptado para muchas aplicaciones es k=2.

Otro proceso clásico de la reproducción se comporta como una rueda de ruleta donde cada ranura tiene tamaños diferentes proporcionales al valor de fitness, es decir, de acuerdo a su aptitud. La técnica es llamada *roulette-wheel parent selection* que selecciona el candidato a ser reproducido. La técnica puede ser implementada mediante un algoritmo como se describe a continuación:

1. Se suman los fitness de todos los miembros de la población y el resultado recibe el nombre de fitness total.
2. Se genera *n*, un número aleatorio entre 0 y el total del fitness.
3. El algoritmo regresa el primer miembro de la población cuyo fitness acumulado sea mayor o igual al número *n* aleatorio, el fitness acumulado se calcula adicionando los fitness de los miembros de la población precedente para cada miembro.

El siguiente ejemplo ilustra la técnica de la ruleta. Considere 6 cromosomas con un valor total de fitness igual a 50, el correspondiente peso en la ruleta se muestra en la figura 1. Se generan números aleatorios entre 0 y 50 y se escoge el primer cromosoma que se encuentra con un fitness acumulado igual o mayor al generado aleatoriamente. Los miembros con el fitness más alto tendrá mayor posibilidad de sobrevivir.

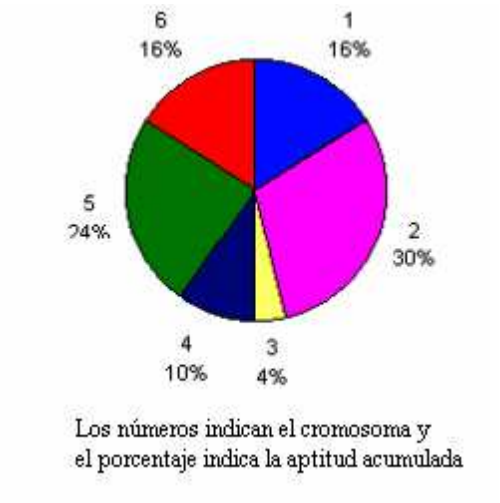


Fig. 1 Método de selección

En la tabla 1 se encuentran los datos del ejemplo

No. de cromosoma	Cadena cromosomica	Fitness	% con respecto al total	Fitness acumulado
1	01110	8	16	8
2	11000	15	30	23
3	00100	2	4	25
4	10010	5	10	30
5	01100	12	24	42
6	00011	8	16	50

Tabla 1 Datos del resultado del ejemplo

El fitness acumulado se calculó acumulando los fitness de cada miembro, el cromosoma 1 tiene fitness igual a 8, por lo tanto su fitness acumulado es de 8, el cromosoma 2 tiene fitness igual a 15 más 8 del primero da su fitness acumulado igual a 23, el cromosoma 3 tiene fitness 2 más 23 del cromosoma 2 da su fitness acumulado igual a 25, etc.

A continuación se muestran los números generados aleatoriamente entre 0 y 50 y los cromosomas que sobrevivieron para la siguiente generación [4]

Números aleatorios	Cromosomas sobreviv.
26	4
2	1
49	6
15	2
40	5
36	5
9	2

Tabla 2

Elitismo

Este método de selección guarda al mejor individuo de entre todas las generaciones, la forma en que selecciona a dicho individuo es la siguiente: en la primera generación, basado en la función de fitness, guarda al mejor individuo, en la siguiente generación encuentra al mejor y al peor individuo, si el mejor de esta generación supera al ya seleccionado se reemplaza al anterior individuo por este, en caso contrario remplaza el peor individuo de la población actual con el mejor de la generación anterior, este proceso se aplica el número máximo de generaciones definidas por el usuario.

Es importante mencionar que con este operador se garantiza la convergencia del algoritmo genético. En el artículo "Convergence Analysis of Canonical Genetic Algorithms"[9] se menciona que las cadenas de Markov son un modelo apropiado para el análisis de los algortimos genéticos y han sido usados [8, 9] para probar la convergencia a la mejor solución dentro de una población llegando al óptimo global usando como operador de selección el elitismo (el mejor individuo sobrevive con probabilidad igual a 1)

Cruzamiento

Una descendencia tiene dos padres y heredan genes de ambos, el principal operador que simula esto es el cruzamiento, en este método se seleccionan a los padres por medio de una probabilidad de cruzamiento, el sitio de cruzamiento es seleccionado aleatoriamente. Véase figura 2.

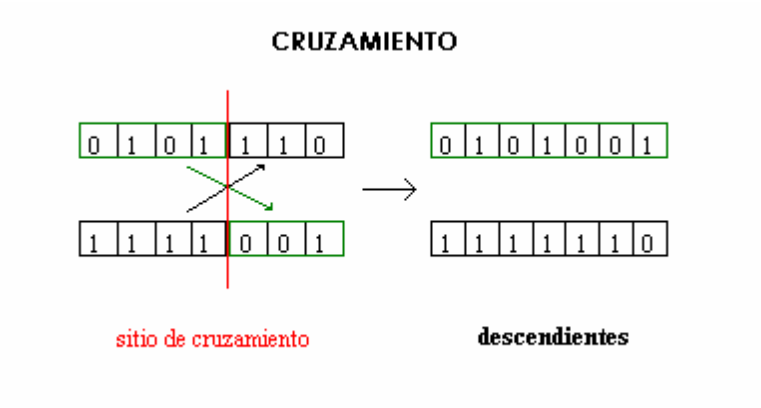


Fig. 2 Ejemplo de un solo punto de cruzamiento

En este ejemplo se observa que el sitio de cruzamiento fue el 4, donde la parte terminal del primer cromosoma es ahora la parte terminal del segundo y la parte terminal del segundo cromosoma pasa a ser ahora la parte terminal del primero.

Existen otros tipos de cruzamiento, por ejemplo están los de 2 puntos de cruzamiento.

Cuando se usan 2 puntos de cruzamiento, se procede de manera similar, pero en este caso el intercambio se realiza en la forma mostrada en la figura 3.

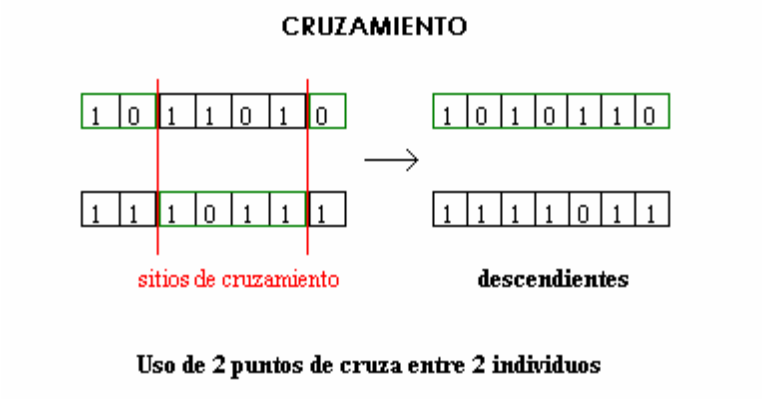


Fig. 3 Cruzamiento

En la figura 3 se usaron 2 puntos de cruzamiento entre 2 individuos. En este caso se mantienen los genes de los extremos, y se intercambian los del centro. También aquí existe la posibilidad de que uno o ambos puntos de cruzamiento se encuentren en los extremos de la cadena, en cuyo caso se hará un cruzamiento usando un solo punto, o ningún cruzamiento, según corresponda. [2 ,3]

Normalmente el cruzamiento se maneja dentro de la implementación del algoritmo genético como un porcentaje que indica con qué frecuencia se efectuará. Esto significa que no todas las parejas de cromosomas se cruzarán, sino que habrá algunas que pasarán intactas a la siguiente generación.

Mutación

Además de la selección y el cruzamiento, existe otro operador llamado mutación, el cual realiza un cambio a uno de los genes de un cromosoma elegido aleatoriamente. Cuando se usa una representación binaria, un bit se sustituye por su complemento (un cero cambia en uno y viceversa). Este operador permite la introducción de nuevo material cromosómico en la población, tal y como sucede con sus equivalentes biológicos.

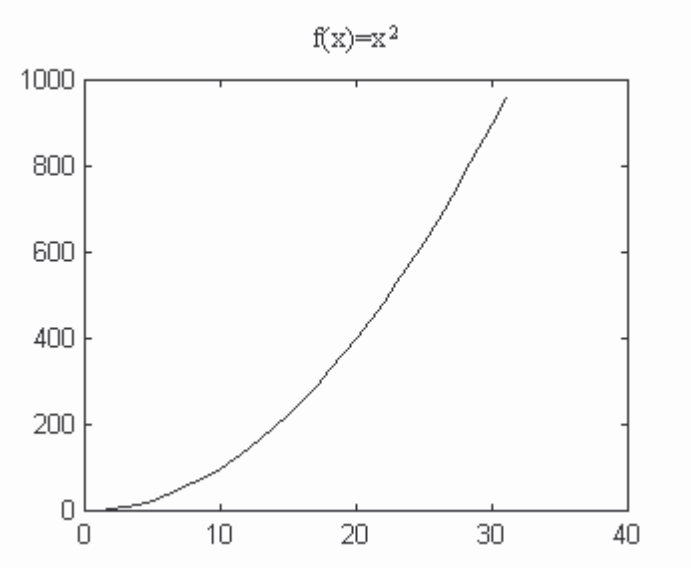
Al igual que el cruzamiento, la mutación se maneja como un porcentaje que indica con qué frecuencia se efectuará, aunque se distingue de la primera por ocurrir mucho más esporádicamente (el porcentaje de cruzamiento normalmente es de más del 60%, mientras que el de mutación normalmente nunca supera el 5%, lo cual no sucede en la programación evolutiva).

Ejemplos

Los siguientes ejemplos están basados de la cita [2]

1. En el primer ejemplo que se muestra, se hace una codificación binaria de números enteros y la aplicación del método de la ruleta descrito anteriormente así como el operador genético denominado cruzamiento.

Considere el problema de maximizar la función $f(x)=x^2$, donde x es un número entero entre 0 y 31, es decir, se debe obtener un x^0 tal que $f(x^0)\geq f(x) \ \forall \ x \in [0,31]$. Graf. 1.



Graf. 1 Maximizar la función $f(x)=x^2$

Para el algoritmo genético se codifica la variable x como un número binario de longitud 5. Por ejemplo, la cadena "11000" representa el valor entero de $x=24$; la función de fitness se define como la función $f(x)$. El proceso de reproducción se muestra en la tabla 3. Es importante notar que la longitud del número binario define el tamaño para todos los cromosomas, el cual es constante.

Primero se inicializa una población de tamaño 4 aleatoriamente, el cruzamiento de la siguiente generación es a partir de la ruleta la cual rueda en 4 tiempos

No. de cromosoma	**Población inicial**	**x**	**Fitness $f(x)=x^2$**	**Pselect $f_i/\sum f$**	**Selección de la ruleta**
1	01001	9	81	0.08	1
2	11000	24	576	0.55	2
3	00100	4	16	0.02	0
4	10011	19	361	0.35	1
		Sum	1034		
		Prom	259		

Tabla 3 Datos del ejemplo

En la selección de la ruleta lo que se ve es cuántas veces cayó la ruleta en cada cromosoma; en el ejemplo, el cromosoma 3 nunca fue seleccionado. En la tabla 4 se muestra el proceso

de cruzamiento, en este caso el proceso de la ruleta seleccionó a los cromosomas que se van a cruzar.

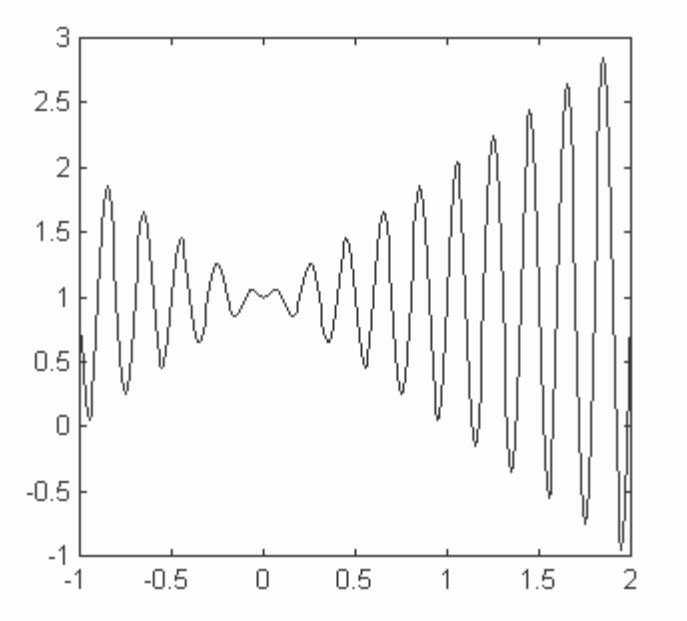
No. de cromos.	Población inicial	Cromosoma con el que se cruza	Sitio de cruzamiento	Nueva población	x	X^2
1	01001	2	4	01000	8	64
2	11000	1	4	11001	25	625
2	11000	4	2	11011	27	729
4	10011	2	2	10000	16	256
					Sum	1674
					Prom	419

Tabla 4

En este ejemplo el cromosoma con el que se cruza es seleccionado de forma aleatoria entre los cromosomas seleccionados por el método de la ruleta.

2. En este ejemplo se aplicarán los pasos descritos en los algoritmos genéticos para un problema de optimización utilizando una codificación binaria para números reales, se explica su codificación y se aplican los operadores genéticos de mutación y cruzamiento descritos anteriormente.

El problema consiste en optimizar la función $f(x)=x\text{sen}(10\pi x)+1$ donde $x \in [-1,2]$.



Graf. 2 Optimizar la función $f(x)=x\text{sen}(10\pi x)+1$

La representación de cada cromosoma se hace de la siguiente manera: cada cromosoma representa cada valor real de la variable x y esta es representada como una cadena, esta cadena es un número binario, el tamaño de la cadena depende de la precisión que requiera el problema, en este caso es necesario 6 dígitos después del punto decimal. El dominio de la variable es de tamaño 3, por lo tanto, el rango [-1,2] debe ser dividido en 3*1000000 tamaños de rangos iguales, esto significa que son necesarios 22 bits para representar a cada cromosoma, es decir,

$$2097152 = 2^{21} < 3000000 \leq 2^{22} = 4194304$$

(todos los cromosomas tienen la misma longitud)

Para mapear la cadena binaria al número real correspondiente basta

1. convertir la cadena binaria de base 2 a base 10, este resultado se denominará x'
2. encontrar su correspondiente número real x con la siguiente fórmula:

   $x = -1 + x' * (3/(2^{22}-1))$

   donde el –1 representa el límite inferior del rango donde corren las x's, el 3 el tamaño del dominio

   Por ejemplo, el cromosoma 1000101110110101000111 representa el número 0.637197
   $x' = (1000101110110101000111)_2 = 2288967$
   y x = -1 + 2288967* 3/4194303 = 0.637197

   Los cromosomas (00000000000000000000) y (11111111111111111111) representan los límites extremos del rango [-1,2]

Al aplicar el algoritmo genético en este problema se realizan los siguientes pasos:

1. Primero se inicializa la población con cromosomas de cadenas binarias de tamaño de 22 bits, estos bits son definidos de forma aleatoria.
2. Se define una función fitness o de evaluación. Para las cadenas binarias la función eval es equivalente a la función f. eval de (v) = f(x) donde el cromosoma v representa al número real x

   La función de evaluación juega el papel del medio ambiente, teniendo así un conjunto de soluciones potenciales en términos de esta función.

   Por ejemplo, se tienen los siguientes tres cromosomas:

   v_1 = (10001011101101010001 11)
   v_2 = (00000011100000000010000)
   v_3 = (11100000001111111000101)

   Corresponden a los valores

$x_1 = 0.637197$
$x_2 = -0.958973$
$x_3 = 1.627888$

así

$eval(v_1) = f(x_1) = 1.586345$
$eval(v_2) = f(x_2) = 0.078878$
$eval(v_3) = f(x_3) = 2.250650$

al maximizar la función se busca al cromosoma con el valor más alto, en este conjunto de cromosomas el cromosoma v_3 es el mejor individuo.

3. Se aplican los siguientes operadores genéticos:

a. Mutación, una forma fácil de mutar es cambiando uno o más genes de un cromosoma, por ejemplo, suponiendo que se seleccionó el quinto gen del cromosoma v_3 para ser mutado, éste por ser igual a 0 se muta cambiando a 1, así

v_3' = (11101**1**00000111111000101)

este cromosoma representa el valor

$x_3' = 1.721638$
$f(x_3') = -0.082257$ ocasionando esta mutación un decrecimiento considerable en el cromosoma v_3 dejando de ser el mejor cromosoma del conjunto anteriormente analizado.

b. Cruzamiento el cual se ilustrará utilizando los cromosomas v_2 y v_3, el punto de cruzamiento se selecciona aleatoriamente asumiéndose que se seleccionó el quinto gen como posición para cruzamiento

v_2 = (00000 | 01110000000010000)
v_3 = (11100 | 00000111111000101)

el resultado del cruzamiento es

v_2' = (00000 | 00000111111000101)
v_3' = (11100 | 01110000000010000)

al evaluar se tiene

$f(v_2') = f(-0.998113) = 0.940865$
$f(v_3') = f(1.666028) = 2.459245$

obteniendo estos cromosomas mejores evaluaciones que la de sus padres.

Los parámetros con los que se obtuvo la optimización de la función fueron los siguientes:

Tamaño de la población = 50
Probabilidad de cruzamiento = 0.25
Probabilidad de mutación = 0.001
Número de generaciones = 150

Los resultados de algunas generaciones se muestran en la siguiente tabla:

Generación	Función de evaluación
1	1.441942
6	2.250003
8	2.250283
9	2.250284
10	2.250363
12	2.328077
39	2.344251
40	2.345087
51	2.738930
99	2.849246
137	2.850217
145	2.850227

Tabla 5 Resultados de la optimización de la función $f(x)=x\text{sen}(10\pi x)+1$

El valor óptimo es x=1.85056 obteniéndose una f(x)=2.850274, como se observa en la tabla 5, en la generación número 145 se obtiene un óptimo muy cercano al real.

Anteriormente se mencionó que la representación binaria tenía sus desventajas, cuando es aplicada a problemas numéricos de alta precisión. Por ejemplo para 100 variables con dominio en el rango [-500, 500] con una precisión de 6 dígitos después del punto decimal, el tamaño de una cadena binaria es de 3000 generando un espacio de búsqueda de 10^{1000} dejando de ser el algoritmo genético una herramienta adecuada para su solución con este tipo de representación.

3. Otro ejemplo típico de optimización es el que se mencionó en la sección de la descripción general de los algoritmos genéticos y la programación evolutiva y es la del agente viajero, recordando el problema, éste consiste en dadas n ciudades que tiene que visitar un vendedor, encontrar el camino más corto para, partiendo de una de ellas, visitarlas todas sin pasar más de una vez por cada una y volviendo finalmente a la ciudad de partida.

En una representación binaria de n ciudades cada ciudad puede ser codificada como una cadena de $\lceil \log_2 n \rceil$ bits, un cromosoma es una cadena de $n*\lceil \log_2 n \rceil$ bits, es decir, que cada cromosoma representa un recorrido para el vendedor viajero. Existen una serie de problemas con los operadores genéticos descritos en los ejemplos anteriores para la solución de este ejemplo, en una mutación puede resultar una secuencia de ciudades la cual no es un recorrido válido ya que se puede tener la misma ciudad dos veces dentro de la misma secuencia, por otro lado, con 20 ciudades, donde se necesitan 5 bits para representar una ciudad, algunas secuencias no corresponden a alguna ciudad válida, por ejemplo 10101, (debido a que solo existen 20 ciudades y 10101 representa la ciudad 21) similares problemas se presentan con el cruzamiento. Para este tipo de casos es necesario modificar el algoritmo de mutación haciendo un algoritmo de rectificación, detectando los resultados no válidos, moviendo a los cromosomas a un espacio de posibles soluciones válidas. Para este problema conviene más una codificación de números enteros que binarios donde los algoritmos de corrección funcionan más fácilmente que con la representación binaria.

Estrategias evolutivas y Programación evolutiva

Las estrategias evolutivas son algoritmos que tratan de imitar los principios de la evolución natural como un método para resolver problemas de optimización. Estas se desarrollaron en Alemania durante los años 60´s.

Al principio las estrategias evolutivas se basaron en una población de un solo individuo además hubo un solo operador genético aplicado en el proceso de evolución denominado mutación. Lo interesante, algo que no tenían los algoritmos genéticos, fue representar a un individuo como una pareja de vectores de números flotantes, es decir, $v=(x,\sigma)$, donde el vector x representa un punto en el espacio de búsqueda y el vector σ representa al vector de la desviación estándar, la mutación se llevaba a cabo remplazando a x por $x^{t+1}=x^t+N(0, \sigma)$ donde $N(0,\sigma)$ es un vector de números aleatorios Gaussianos con media igual a 0 y desviación estándar σ. Esta convención se dio debido a que en la evolución natural ocurren más a menudo pequeños cambios en lugar de cambios drásticos. Se acepta el resultado de la mutación como un nuevo miembro de la población remplazando a su padre siempre y cuando tenga mejor fitness que éste, es decir, sea *f* la función objetivo, la salida (x^{t+1}, σ) reemplaza a su padre (x^t, σ) si $f(x^{t+1}) > f(x^t)$. En otro caso la salida se elimina y la población se mantiene sin cambios. A esta estrategia se le denominó estrategia evolutiva de dos miembros, la razón es que la salida compite con su padre y en ese estado de competencia hay dos individuos en la población. El vector de la desviación estándar σ permanece sin cambios durante el proceso de evolución. Si todos sus elementos del vector estándar son idénticos, i.e., $\sigma = (\sigma \ldots \sigma)$ y el problema de optimización es regular [1] es posible comprobar el teorema de convergencia[2].

[1] Un problema de optimización es regular si la función objetivo *f* es continua, el dominio de la función es un conjunto cerrado.

Teorema de convergencia: Para $\sigma>0$ y dado un problema de optimización regular con $f_{opt} > -\infty$ (minimización) ó $f_{opt} < -\infty$ (maximización)

$p\{\lim_{t\to\infty} f(x^t) = f_{opt}\} = 1$

Este teorema afirma que el óptimo global es encontrado con la probabilidad de uno con un suficiente tiempo de búsqueda sin embargo no provee ninguna pista para la razón de convergencia, (cociente de la distancia cubierta respecto al óptimo y el número de generaciones necesarias)[2].

La estrategia de evolución de múltiples miembros difiere de la estrategia evolutiva de dos miembros en el tamaño de la población, éste es mayor a 1 y se adicionan otras características tales como:

Todos los individuos de la población deben tener la misma probabilidad de ser padres
Poder introducir un nuevo operador llamado recombinación ó cruzamiento

Estas estrategias fueron estudiadas por Ingo Rechenberg y Hans – Paul [2].

En la programación evolutiva también se trabaja con una población de individuos, donde cada individuo representa una solución potencial al problema representada en un cromosoma, sin embargo existen diferencias con los algoritmos genéticos descritos anteriormente principalmente en la forma de codificar estas soluciones potenciales y en el operador genético que se aplica. A continuación se describen de forma paralela solo dos técnicas especificando las semejanzas y diferencias que existen entre ellas en la forma de ejecutar cada paso dentro de su algoritmo.

Pasos de los algoritmos genéticos y los programas evolutivos

Ambas técnicas llevan a cabo 5 pasos importantes:

1. La codificación de soluciones del problema en cromosomas, que a su vez constituyen una población
2. La utilización de una función fitness definida sobre los cromosomas que indica la aptitud de cada uno de estos
3. La inicialización de la población de cromosomas utilizando un procedimiento de inicialización y evaluando cada miembro de la población inicial con la función de evaluación fitness
4. Aplicación de los operadores genéticos
5. Definición del conjunto de parámetros con los que el algoritmo comienza a trabajar, estos son datos como el tamaño de la población, número de generaciones que se van a aplicar los operadores genéticos, probabilidad de mutación o de cruzamiento, etc.

Diferencias entre estas técnicas

Son dos diferencias importantes entre estas dos técnicas, las cuales se describen a continuación:

Una diferencia importante es la forma de codificación de las soluciones potenciales del problema, las cuales en la programación evolutiva se realizan mediante estructuras más naturales para el problema y no con números binarios como es el caso de los algoritmos genéticos, además de que éstos últimos mantienen un tamaño fijo en la representación de los cromosomas. Eso usualmente implica que los operadores genéticos tienen que ser diseñados especialmente para el problema.

La otra diferencia tiene que ver con los operadores genéticos que aplican. Se les llama padres a dos individuos de la población que se seleccionan aleatoriamente para aplicarles operadores genéticos para su reproducción o modificación de su composición genética. Los operadores estándar son mutación y cruzamiento. En este punto existe también una diferencia entre las dos técnicas, los algoritmos genéticos generalmente usan operadores de cruzamiento y mutación, dando mayor importancia al cruzamiento, mientras que la computación evolutiva utiliza diferentes formas de mutación.

A continuación se muestran los pseudocódigos de ambas técnicas para especificar de forma más clara las diferencias descritas anteriormente:

Algoritmo genético

```
comienza
  t=0
  inicializa  P(t)  {P es la población en la generación t, se codifica de forma binaria y el ta-}
  evalua P(t)                                                    {maño de los cromosomas es fijo}
  mientras t<núm_generaciones haz
  comienza
    t=t+1
    selecciona padres de P(t-1)
    selecciona a quien se elimina de P(t-1)
    crea P(t): reproduce a los padres seleccionados aplicando principalmente cruzamiento
    evalua P(t)
  termina
termina
```

Programación evolutiva

```
comienza
  t=0
  inicializa  P(t)  {P es la población en la generación t, su representación no es binaria}
  evalua  P(t)
  mientras t<núm_generaciones haz
  comienza
```

```
    t=t+1
    selecciona a quien se elimina de P(t-1)
    crea P(t): aplica mutación
    evalua P(t)
  termina
termina
```

En ambas técnicas se guarda al mejor cromosoma de una generación a otra.

La utilización de estas técnicas permiten una programación en paralelo lo cual agiliza la ejecución de las instrucciones y por lo tanto mejoran los tiempos de ejecución.

Al igual que en el caso de las redes neuronales, el esquema puede parecer excesivamente sencillo comparado con el punto de vista biológico, sin embargo ha permitido soluciones a problemas complejos que no son fáciles de resolver con otras técnicas.

Existen algunos métodos heurísticos que guían al usuario en la selección apropiada de las estructuras de datos y operadores genéticos para un problema particular.

Hay algunos métodos disponibles para definir una población inicial: una es la inicialización de forma aleatoria o usar la salida de algún algoritmo determinístico para inicializar la población. Hay también algunas reglas heurísticas generales para la definición de valores de los parámetros usados en estos algoritmos; para muchas aplicaciones de algoritmos genéticos el tamaño de la población oscila entre 20 y 100 individuos, la probabilidad de cruzamiento entre 0.65 y 1 y la probabilidad de mutación entre 0.001 y 0.01, algunas reglas heurísticas sirven para variar los valores de estos parámetros durante el proceso de evolución[2].

Ventajas y desventajas con respecto a otras técnicas de búsqueda

Tanto para los algoritmos genéticos como para la programación evolutiva, estos algoritmos tienen ventajas y desventajas las cuales se mencionan a continuación:

•No necesitan conocimientos específicos sobre el problema que intentan resolver.
•Operan de forma simultánea con varias soluciones, en vez de trabajar de forma secuencial como las técnicas tradicionales.
•Cuando se usan para problemas de optimización -maximizar una función objetivo- resultan menos afectados por los máximos locales (falsas soluciones) que las técnicas tradicionales.
•Resulta sumamente fácil ejecutarlos en las modernas arquitecturas masivamente paralelas.
•Usan operadores probabilísticos, en vez operadores determinísticos de las otras técnicas.
•Pueden tardar mucho en converger, o no converger en absoluto, dependiendo en cierta medida de los parámetros que se utilicen -tamaño de la población, número de generaciones, etc.[5].

Referencias

[1] Patrick Henry Winston. “Inteligencia Artificial”, Edit. Addison Wesley Iberoamericana. 1992.
[2] Zbigniew Michale wicz ,”Genetic Algorithms + Data Structures = Evolution Programs” 3ra ed., Edit. Springer. 1996.
[3] Xin Yao, “An Overview of Evolutionary Computacion. Chinese Journal of Advanced Software Research” Vol 3 No. 1 1996.
[4] Holland, J.H., "Adaptation in Natural and Artificial Systems", University of Michigan Press, 1975.
[5] http://www.fciencias.unam.mx/rebvista/soluciones/N17/Coello2.html.
[6] Russ Eberhart, Pat Simpson, Roy Dobbins. “Computational Intelligence PC Tools, AP Professional” 1996.
[7] José R. Hilera, Víctor J. Martínez. “Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones” Addison Wesley Iberoamericana. 1995.
[8] A. E. Eiben, E.H.L. aarts and K.M. Van Hee, “Global convergence of genetic algorithms: A Markov chain analysis, in Parallel Problem Solving from nature”, H.P. Schwefel and R. Männer (Eds.), Berlin and Heidelberg: Springer, 1991, pp4-12.
[9] Günter Rudolph, “Convergence Analysis of Canonical Genetic Algorithms”, 1992.

DEFINICIÓN DE UNA RED NEURONAL PARA CLASIFICACIÓN POR MEDIO DE UN PROGRAMA EVOLUTIVO

Una vez descrito el problema de clasificación y la teoría de los algoritmos genéticos, en este capítulo se explicará el problema de la definición de la arquitectura de una red neuronal (perceptron multicapa, véase apéndice A) para ser empleada en la solución de problemas de clasificación y se dará una propuesta para resolver dicho problema utilizando un programa evolutivo con cromosomas de longitud variable. Este programa evolutivo puede encontrar el número adecuado de nodos ocultos, (aquel número mínimo de nodos para la solución del problema, optimizando así el tiempo de ejecución del algoritmo en una PC tradicional), así como los pesos de todas las conexiones para definir por completo la arquitectura de la red. Se supone únicamente que el número de nodos de entrada y de salida están fijos. Los operadores que se usan en el programa evolutivo son selección y dos formas de mutación.

Problema para definir la arquitectura de una red neuronal perceptron multicapa

Es importante tomar conciencia de los aspectos que involucra la definición de la arquitectura de un perceptron multicapa para entender lo difícil que es determinar dicha arquitectura para un problema dado, estos aspectos se mencionan a continuación:

1. Número de nodos de entrada así como los de salida
2. Número de capas ocultas
3. Número de nodos ocultos para cada capa oculta
4. Pesos que corresponden a cada conexión

Para la solución del problema de clasificación, algunos de estos aspectos se definen automáticamente. El número de nodos de entrada queda determinado por el número de atributos que aparecen en el problema a resolver y el número de nodos de salida lo define el número de clases y la forma de codificar esas clases. Esto quiere decir que si existen dos clases se puede trabajar con un nodo de salida en lugar de dos, donde la salida de la red determina si pertenece o no a una clase. En el caso de que sean más clases, tal vez cada nodo de salida represente una clase y al final se "prenda" el nodo que corresponda a la clase que la red neuronal perceptron encontró como solución al conjunto de valores de atributos que se le proporcionó.

Con respecto al número de capas ocultas, se ha demostrado que las redes neuronales del tipo perceptron, con una capa oculta tienen la capacidad de distinguir entre conjuntos de patrones complejos [1], por lo tanto queda cubierta esta decisión. Las conexiones entre los nodos quedan definidas por ser un perceptron multicapa de la siguiente manera: se conecta cada nodo de la capa de entrada contra todos los nodos de la capa oculta y cada nodo de la capa oculta contra todos los nodos de salida.

Queda entonces la determinación de los nodos ocultos. Una forma común de proceder es mediante un proceso de acierto y error, intentar con diferentes valores y ver cuál perceptron da mejores resultados, después de encontrar sus pesos en cada caso. Para encontrar los pesos numéricos habitualmente se hace uso del algoritmo de retropropagación. Este algoritmo se basa en el método de gradiente descendente y requiere una función de transferencia diferenciable, así como de fijar ciertos parámetros como la tasa de aprendizaje.

Esta forma de determinar el número de nodos ocultos es muy engorrosa, se podría pensar en escoger un número "grande" de nodos ocultos, sin embargo, esto ocasionaría un problema importante. El propósito del perceptron es fungir como un clasificador para predecir si un ejemplo pertenece o no a una clase. Lo importante entonces es lo que se llama "su poder de generalización" sobre los casos nuevos, es decir, qué tan bien clasifica sobre casos no vistos previamente. Cuando el número de nodos ocultos es demasiado grande, puede ocurrir un fenómeno donde el perceptron clasifica muy bien sobre los ejemplos conocidos que se usaron para determinar los pesos, pero no sobre nuevos ejemplos. Lo idóneo sería adoptar otro enfoque en la determinación de un perceptron que podría especificar todos los parámetros involucrados simultáneamente, tanto la arquitectura como los pesos numéricos.

El problema con los valores de los pesos es el siguiente: se trata de cambiar los pesos de una sola distribución para asegurar un aprendizaje uniforme, debido a que los datos estándar dan valores positivos y negativos se busca que los pesos estén entre $-\mathbb{W} < w < +\mathbb{W}$, si $\mathbb{W}$ es muy pequeña la función de activación en los nodos ocultos será muy pequeña y esto servirá para implementar un modelo lineal, el cual no siempre resuelve el problema, por otro lado si $\mathbb{W}$ es muy grande los nodos ocultos se pueden saturar, lo que se sugiere trabajar en un rango entre –1 y 1 porque estos valores ayudan a resolver las situaciones mencionadas.[2]

Existen varios métodos para resolver estos problemas, con respecto al número de nodos ocultos se podría utilizar algún algoritmo de poda para eliminar nodos ocultos redundantes, para la definición de pesos, se puede usar el decaimiento de pesos (weight decay) método que penaliza pesos grandes, este método trata de que los pesos converjan a pequeños valores**.** Los pesos excesivamente grandes pueden causar en las unidades ocultas una salida demasiado desigual y en las unidades de salida pueden causar una salida fuera del rango de los datos si la salida de la función de activación no se obliga a que este en el mismo rango de los datos[3]

En la introducción se mencionó la opción de usar máquinas de soporte vectorial para la definición de la arquitectura de la red neuronal, como un ejemplo de que existen varias alternativas a la solución del problema que se presenta, sin embargo, por la experiencia que se tiene en algunos métodos de inteligencia artificial y ver los resultados de otros autores al respecto se decidió utilizar un programa evolutivo.

El algortimo propuesto más adelante es una alternativa muy sencilla de programar y de aplicar dando resultados muy semejantes a los de otros algoritmos de clasificación, además de que son algoritmos que se ha demostrado su convergencia en un momento dado, tal como se mencionó en el capítulo anterior.[4, 5]

Para resolver el problema de la determinación de la arquitectura de una red neuronal perceptron multicapa se han realizado varios trabajos de investigación dedicados a aplicar ideas evolutivas que han producido buenos resultados. A continuación se mencionarán los trabajos más relevantes al respecto, sin dejar de mencionar que no son los únicos al respecto.

Antecedentes para la definición de la arquitectura de una red neuronal

Básicamente son cuatro los trabajos que se describirán a continuación, debido a su contenido e influencia en la propuesta que se hace más adelante. (Sin embargo se encuentra más información en [10-12])

El primero de estos trabajos es el publicado por David J. Montana 1995[6], quien usó algoritmos genéticos para escoger únicamente los pesos numéricos de una red neuronal. El problema a resolver con la red era un problema de clasificación, el cual consistía en proporcionarle a la red un conjunto de ejemplos para el entrenamiento de la red y después ésta debería ser capaz de seleccionar la clase adecuada para un ejemplo ajeno al conjunto de entrenamiento.

Trabajó dos tipos de redes neuronales, para este trabajo sólo se mencionará la arquitectura de nuestro interés, la cual fue una red neuronal perceptron multicapa, con una capa de entrada, una oculta y una de salida, las conexiones se hicieron hacia adelante de una capa hacia otra, con una función de transferencia sigmoidal y una función de fitness igual al error cuadrático medio.

Montana decidió usar un algoritmo genético, en donde principalmente se ocuparon los operadores genéticos de cruzamiento y mutación, además utilizó la estrategia de búsqueda "Hillclimbing" donde encuentra la zona donde se localiza el óptimo global; una vez localizada esta zona, aplicó retropropagación para encontrar el pico del óptimo global. Una ventaja de estos algoritmos es su generalidad ya que con pequeños cambios pueden ser usados para entrenar diferentes variedades de redes.

La red neuronal perceptron se representó en cada cromosoma, la longitud de cada cromosoma fue fijo, es decir, todos los cromosomas median lo mismo. El tamaño de la población fue de 50 individuos, la función de evaluación fue el error cuadrático y se inicializaron los pesos de forma aleatoria entre -1 y 1, los operadores genéticos utilizados fueron la mutación de los nodos, la mutación de los pesos, la mutación de los pesos de los sesgos y el cruzamiento de los pesos y de los nodos. El desempeño de los operadores genéticos en 800 generaciones fue el siguiente: se vieron mejoras con la mutación de pesos

(sin incluir la mutación de los pesos de los sesgos) con respecto al cruzamiento. Con relación a las dos formas de cruzamiento ligeramente mejora el cruzamiento de los pesos sobre el de los nodos. 2]

Otro trabajo que se consideró fue el de Angeline, Saunders y Pollack, 1994[7], que aunque no trabajaron con una red neuronal perceptron utilizaron un programa evolutivo llamado GNARL(GeNeralized Acquisition of Recurrent Links), para definir la topología de una red neuronal recurrente. Esta red era utilizada para tareas de control, no de clasificación, como lo es en nuestro caso. A diferencia de Montana, los autores de este trabajo aplicaron un programa evolutivo por las siguientes razones: El programa evolutivo da mayor importancia al operador genético mutación que al cruzamiento, lo cual no sucede con los algoritmos genéticos. La ventaja de usar mutación es evitar que por medio de cruzamientos se generaran varios individuos iguales, es decir, redes iguales tanto en topología como en pesos o idénticas topologías pero diferentes pesos. La población fue inicializada con *n* redes recurrentes generadas aleatoriamente, el número de nodos de entrada y los de la salida se definieron de acuerdo a la tarea a resolver, el número de nodos ocultos y sus conexiones variaron libremente. Como las conexiones se definieron aleatoriamente no era imposible encontrar una conexión al mismo nodo. En cada generación las redes fueron categorizadas por medio de una función de fitness f : N$\rightarrow\Re$, la función de fitness fue el error cuadrático medio, después de categorizar a cada individuo, se seleccionaron n/2 para reproducirse con mutaciones en cada generación. (n es el tamaño de la población). Para decidir si iban a realizar una mutación, primero se calculó la temperatura de cada individuo de la siguiente manera:

$$T(N) = U(0,1)(1 - \frac{f(N)}{f_{\text{máx}}})$$

donde $f_{máx}$ es el valor máximo de la función de fitness para el problema a resolver, es decir, cada individuo tiene un ECM y se toma el de mayor valor. U(0,1) es una variable uniforme aleatoria en el intervalo (0,1). Las redes con alta temperatura eran mutadas más severamente que aquellas con baja temperatura.

Los tipos de mutaciones que ocuparon fueron los siguientes: adicionar *k1* nodos ocultos con una probabilidad de adición, borrar *k2* nodos ocultos con una probabilidad de borrado, adicionar *k3* ligas con una probabilidad de adición de ligas y borrar *k4* ligas con su probabilidad correspondiente, los nodos borrados se eliminan con las ligas que le inciden y las nuevas ligas creadas se inicialzaron con 0. Estas mutaciones son más complejas que las analizadas en el capítulo anterior cuando se describieron los algoritmos genéticos y es debido a que se deben definir de acuerdo a la tarea a resolver.

GNARL se aplicó a diferentes problemas de control logrando mejores resultados que con un algoritmo genético. Uno de los experimentos fue maximizar el número de piezas de comida que recolecta una hormiga en un tiempo dado. Cada hormiga es controlada por una red neuronal recurrente con dos nodos de entrada y cuatro nodos de salida. Los nodos de

entrada significan la ausencia o presencia de comida directamente enfrente de la hormiga y los nodos de salida significan las acciones de la hormiga (hacia la derecha, izquierda, etc), el fitness es el número de piezas tomadas en 200 pasos de tiempo, una corrida fue considerada cuando la red tomo más de 80 de las 89 piezas de comida en el enrejado. Este experimento uso una población de tamaño 100, cada individuo es representado por una red neuronal recurrente.

El siguiente trabajo a mencionar es el realizado por Goddard, López y Ruffiner[8], donde aplicaron un programa genético para la determinación de una red perceptron para el diagnóstico médico, en específico, para desordenes cardiacos, la red una vez definida debía decidir si existía la presencia o ausencia de enfermedad cardiaca. La base de datos utilizada consistía de 13 atributos y 303 ejemplos, descartando aquellos ejemplos con problemas, quedando finalmente 270.

Aplicaron dos métodos con el programa genético, uno donde se utilizó el programa genético con tres diferentes funciones de aptitud, y el segundo donde se aplicó una forma de hibridación utilizando el algoritmo de retropropagación y el programa genétco.

La codificación de la red para el programa genético fue directa, es decir, cada cromosoma estaba constituido por una cadena de números reales que correspondían a los pesos y umbrales de la red, el tamaño de los cromosomas fue fijo, utilizaron el método de selección de individuos llamado de torneo (tournament, el cual compara la aptitud de N individuos y elige el mejor como progenitor de la siguiente generación), el operador de cruzamiento fue el de doble cruza con una probabilidad de 1, el operador de mutación empleado fue el de mutación con bias gaussiano, que consiste en sumar un valor aleatorio de distribución gaussiana a cada gen y se ocupó la estrategia de elitismo, la cual fue descrita en el capitulo anterior. Las funciones de aptitud utilizadas fueron las siguientes:

$$ECM = \frac{\sqrt{\sum error^2}}{Núm.\_ patrones}$$

$$EAM = \frac{\sum |error|}{Núm.\_ patrones}$$

$$MAL = Núm\_ mal\_ clasifcados$$

Obteniendo resultados para cada una de ellas por separado.

En este trabajo mostraron una tabla con resultados aplicando solo retropropagación para compararlos con los resultados aplicando solo el programa genético para cada una de las

funciones de fitness descritas, los mejores resultados se obtuvieron aplicando el algoritmo genetico con la función de fitness denominada MAL.

Además mostraron otra tabla con resultados aplicando el método de hibridación, en este método solo aplicaron las funciones de fitness ECM y MAL y consistió en las siguientes combinaciones:

1. Aplicaron cruza y selección varias épocas y luego retropropagación durante otro número determinado de épocas a cada individuo para finalmente calcular la aptitud de cada uno
2. Primero variaron la cantidad de épocas por individuo durante el entrenamiento de manera que al principio solo actuara el programa genético y al final retropropagación para obtener una solución
3. Se utilizó el programa genético sólo al inicio para encontrar un buen punto de partida para retropropagación.

Para los métodos mencionados se cambiaron algunos parámetros debido a que el objetivo del programa era hacer una búsqueda más extensa en la superficie a minimizar y que posteriormente el algoritmo de retropropagación bajara hasta el mínimo, los cambios fueron utilizar mutación con bias uniforme y cruza simple, debido a estos cambios la probabilidad de mutación y cruzamiento fueron por debajo de 1.

Definieron una población de tamaño 10, el número de generaciones fue de 150 y para el híbrido se aplicaron 495 épocas de retropropagación en cada individuo en cada generación, los mejores resultados reportados fueron la primera opción mencionada (en los tipos de combinaciones que hicieron) con el fitness ECM.

El último trabajo a mencionar es el de Xin Yao y Young Liu, 1997[9] quienes proponen un programa evolutivo (solo utilizan mutaciones) donde cada individuo representa la arquitectura de una red neuronal, generan una población de M redes neuronales de forma aleatoria, generando el número de nodos ocultos y la inicialización de las conexiones de forma aleatoria dentro de un cierto rango.

Este programa evolutivo consta de cinco operadores de mutación y un tipo de selección denominada de rango (*rank-based*).

Los operadores de mutación son los siguientes:

1. entrenamiento híbrido, que consta de la mutación de los pesos basándose en el algoritmo de retropropagación
2. eliminación de un nodo el cual se selecciona de forma aleatoria
3. eliminación de conexiones, (ver siguiente inciso)
4. adición de conexiones, tanto la eliminación como la adición se basa en una probabilidad calculada que mide la importancia de las conexiones dentro de la arquitectura de la red

5. adición de un nodo el cual se logra a través de la hendedura de un nodo existente.

Este programa evolutivo se aplicó en el problema del espiral el cual consta de 194 ejemplos y se consiguió la región formada lo más cercano posible al espiral con 14 nodos ocultos y 131 conexiones.

Bases para la propuesta de la definición de una red neuronal perceptron multicapa

Los trabajos mencionados en la sección anterior son una muestra representativa de lo que se ha hecho para definir la arquitectura de una red neuronal y es en estos trabajos donde surge la siguiente propuesta para la solución del problema.

Como se mencionó al principio de este capitulo se ha demostrado que con una capa oculta un perceptron multicapa tiene la capacidad de distinguir entre conjuntos de patrones complejos [1], es por esto que se decidió trabajar con un perceptron con esta característica.

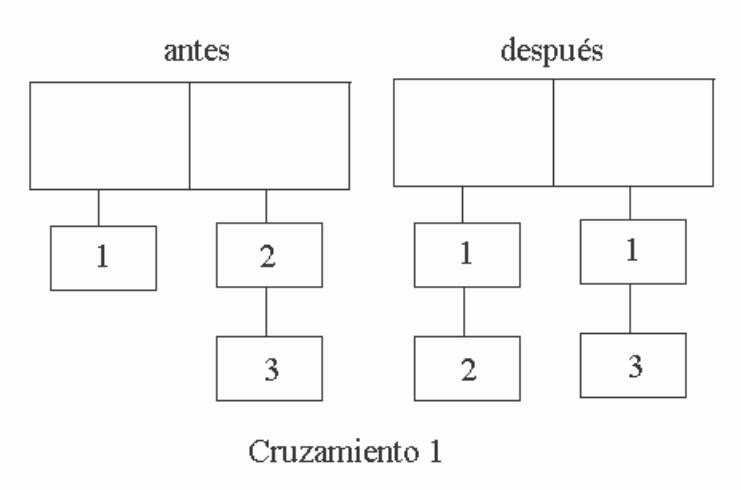
Los trabajos expuestos trabajan con una función de transferencia sigmoidal, es la más común para estos algoritmos además de que se busca usar funciones continuas, crecientes, diferenciables y no lineales. Por otro lado coinciden en usar el error cuadrático medio en la función de fitness, también aunque no es el único es uno de los más comunes. Ambos parámetros se toman en esta propuesta.

De acuerdo al trabajo de Angeline y colaboradores, el cual trabaja una red recurrente, se decidió trabajar con un programa evolutivo en lugar del algoritmo genético propuesto por Montana, debido a que primero se probó un algoritmo genético aplicando cruzamiento y mutaciones y los resultados no mejoraron así que se decidió optar por el programa evolutivo con los operadores de mutación que se explican en la propuesta final.

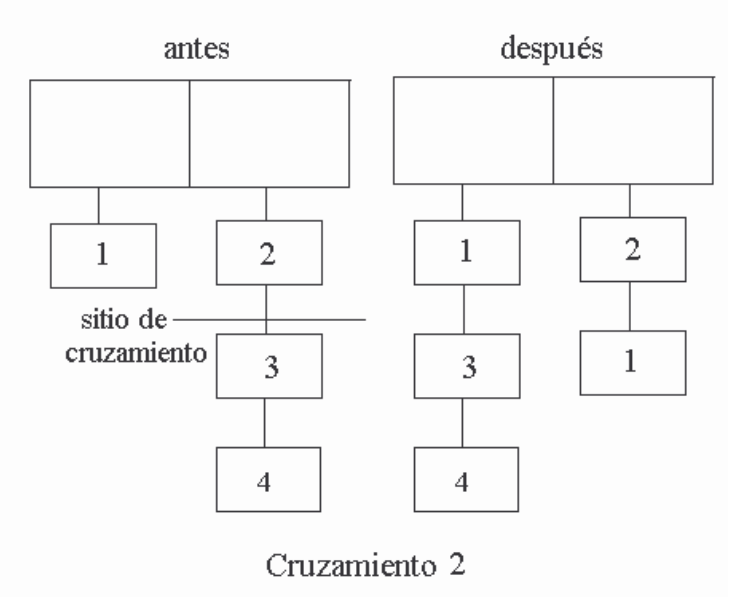
A continuación se explica el cruzamiento implementado y al final una tabla comparativa con resultados obtenidos aplicando este operador y los resultados obtenidos con la propuesta final.

Tipo de cruzamiento aplicado: se seleccionan dos individuos de forma aleatoria para ser cruzados

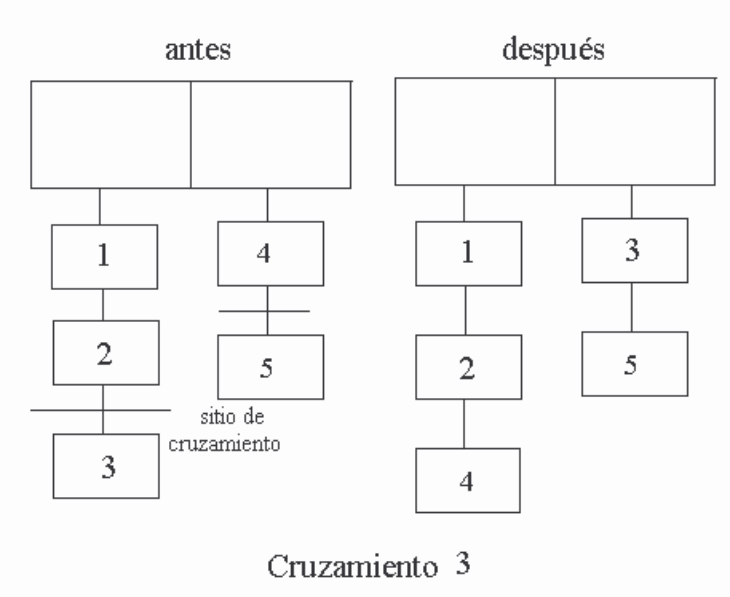
1. Si un individuo tiene un gen y el otro dos en sus cromosomas, el primer individuo queda con su gen y el primer gen del segundo individuo. El segundo individuo queda con el primer gen del primer individuo y el segundo de el mismo.



2. Si un individuo tiene un gen y el otro más de dos, se genera un número aleatorio que es el sitio de cruzamiento que esta entre 1 y el número máximo de genes del segundo individuo. Por ejemplo si el primer individuo tiene un gen y el segundo tiene 3 y el sitio de cruzamiento es 2, los genes a partir del segundo gen hacia abajo se unen con el gen del primer individuo y el segundo individuo se queda con su primer gen y el primero del primer individuo.



3. Si los dos individuos tienen más de un gen se generan dos números aleatorios que indican el sitio de cruzamiento para cada individuo, quedándose las primeros genes del corte de cruzamiento en un individuo y los últimos en el segundo individuo. En el ejemplo mostrado los sitios de cruzamiento son 3 y 2 respectivamente



Cruzamiento 3

En la tabla 1 se muestra los resultados (% de aciertos) aplicando este tipo de cruzamiento con la mutación de los pesos de los nodos, aplicando este cruzamiento con todos los tipos de mutación utilizados (mutar pesos, adicionar y eliminar nodos) y los resultados obtenidos con la propuesta final y con los clasificadores contra los que se compararon los resultados, (solo se muestran de dos bases de datos de las utilizadas en este trabajo):

Base de Datos	Cruzam. y mutar pesos	Cruzam. y mutar pesos, adic. y elimin. nodos	Propuesta final	% de aciertos reportados
Pima	73.7%	76.8%	74.25	76
Vino	42.4%	56.8%	98.11	Del 96-100

Tabla 1 Resultados incluyendo el operador de cruzamiento.

Se hicieron varias pruebas aunque solo se muestren dos y el utilizar el operador de cruzamiento no mejoró los resultados obtenidos con la versión final, estas bases de datos reportadas fueron representativas ya que aunque para algunas se quedaba casi igual el porcentaje de aciertos para otras bajaba mucho este porcentaje como es el caso de la base de datos de vino.

Es importante mencionar que todos los autores revisados en la sección anterior trabajaron con individuos de longitud fija, a excepción de Angeline y colaboradores, (mencionando nuevamente que la red con la que trabajaron fue recurrente.) y Xin Yao y Young Liu quienes trabajaron con redes neuronales con diferente número de nodos ocultos en la capa oculta.

Se coincide con dos operadores de mutación usados por Xin Yao y Young Liu, el de adición y eliminación de nodos ocultos.

Algunos autores trabajaron un entrenamiento híbrido con retropropagación, se intentó aplicar en programa evolutivo un número de épocas y luego retropropagación otro número de épocas para cada individuo y por otro lado se intento aplicar retropropagación a la solución del programa evolutivo propuesto para ver si los resultados mejoraban pero no fue así, no hubo un cambio significativo en los resultados. Así que se decidió no utilizar estos métodos.

Es importante mencionar que Yao y Liu por un lado y Goddard por el otro usaron entrenamiento híbrido pero sus operadores genéticos son diferentes a los usados en esta propuesta encontrándose ahí una diferencia

A continuación se describe la propuesta completa.

Programa evolutivo de cromosomas de longitud variable

Lo idóneo para resolver el problema planteado sería adoptar otro enfoque en la determinación de un perceptron multicapa el cual podría especificar todos los parámetros involucrados simultáneamente, tanto la arquitectura como los pesos numéricos. Es aquí donde entra la computación evolutiva para la solución a este problema, en la sección anterior se mencionó que primero se trato resolver el problema con un algortimo genético pero al cambiar los operadores de cruzamiento dándo más importancia a los de mutación se noto una mejoria en los resultandos optando por el programa evolutivo en lugar del genético.

Esta sección describe entonces un programa evolutivo con cromosomas de longitud variable y su aplicación a la determinación de los pesos, y la arquitectura de un perceptron multicapa para la solución de problemas de clasificación. Su definición está motivada por los artículos [6-10] los cuales fueron descritos anteriormente.

Descripción del programa evolutivo de cromosomas de longitud variable

A continuación se explicara la codificación del programa evolutivo.

Modelación de la red neuronal a un cromosoma

Para aplicar un programa evolutivo en un caso concreto, primero hay que decidir cómo se va a modelar el problema; esto implica la codificación de las soluciones potenciales del problema. En nuestro caso, la idea del programa evolutivo es definir una población de cromosomas en donde cada cromosoma represente un perceptron mutlicapa. Consideremos al perceptron multicapa con una sola capa oculta donde el número de nodos ocultos pueden variar. El número de nodos de entrada y de salida son constantes, dependiendo del

problema considerado. Un cromosoma deberá codificar el número de nodos ocultos de la red y los pesos que unen a los nodos de entrada con los de la capa oculta, y los de ésta hacia los nodos de salida, así como los pesos de los sesgos que se definieron en la capa de entrada y en la capa oculta. Esto nos lleva a la necesidad de considerar cromosomas con una longitud variable. Cada cromosoma, o perceptron multicapa, está implementado como una lista donde cada elemento de la lista representa conceptualmente un nodo oculto del perceptron multicapa.

La información que se guarda en un elemento de la lista son los valores de los pesos numéricos sobre las conexiones que llegan al nodo oculto de los nodos de entrada y los que salen de los nodos ocultos hacia los nodos de salida. Un cromosoma típico se muestra en la Fig.1 junto con el perceptron multicapa que representa.

La población está guardada en un arreglo de estructuras de tamaño *n* que define *n-1* cromosomas. La estructura mantiene información sobre la aptitud de cada cromosoma, lo cual se explicará más adelante, además de un apuntador a la lista del cromosoma. La posición *n* del arreglo se utiliza para dejar el mejor perceptron multicapa encontrado de una generación a otra. (Véase Fig. 2)

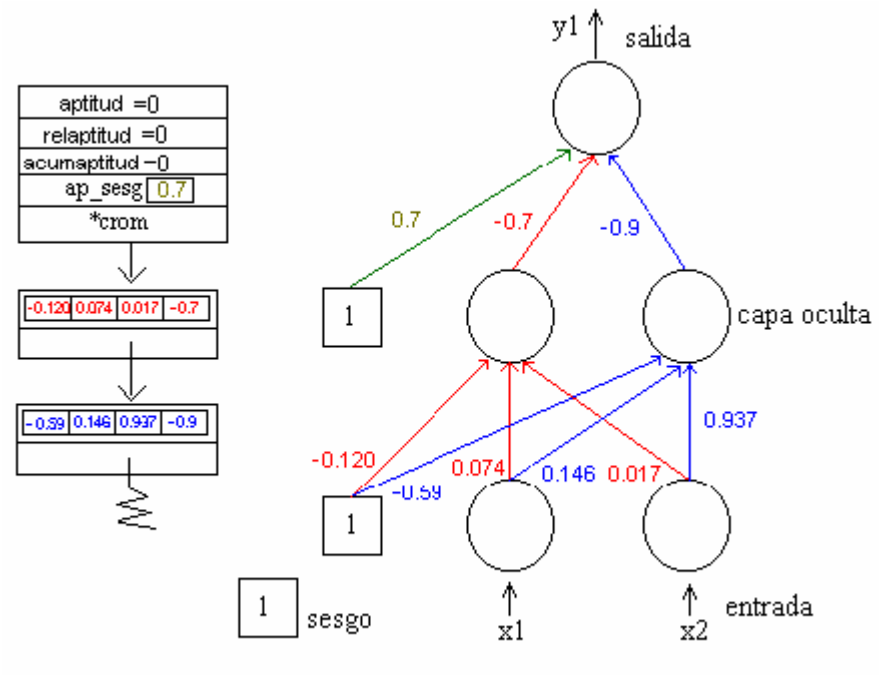


Fig. 1 Representación de un perceptron multicapa mediante un cromosoma en el programa evolutivo

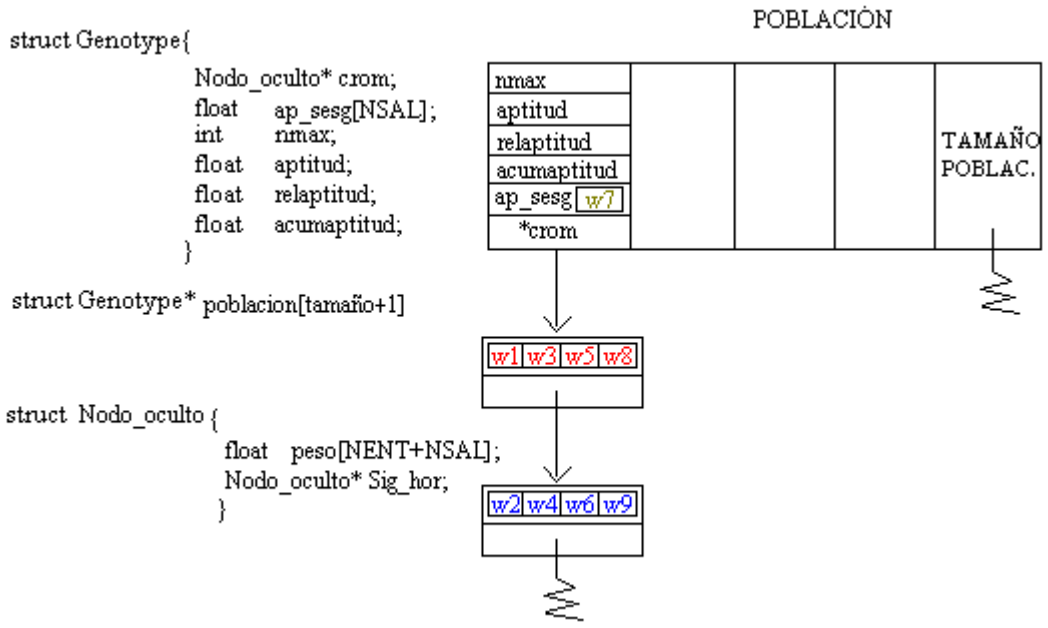


Fig. 2 Estructuras de datos del programa evolutivo

El Algoritmo Principal

El algoritmo principal del programa evolutivo se muestra en la Tabla 1.

```
Inicio
    Generación=0
    Inicializar la población
    Evaluar cada cromosoma
    Guardar el mejor cromosoma
    Mientras (generación<MAXGENS)
    Inicio
        Seleccionar la nueva población
        Aplicar Mutación Estructural
        Aplicar Mutación de los pesos
        Evaluar cada cromosoma
        Guardar el mejor cromosoma
        Generación=Generación+1
    Fin
Fin
```

Tabla 1. Algoritmo Principal del programa evolutivo

El algoritmo comienza con la inicialización de una población de cromosomas y la evaluación de cada uno de ellos. El tamaño de la población es fijo, aunque la longitud de

cada cromosoma puede variar. La población evoluciona varias generaciones aplicando los operadores de selección (para escoger la nueva población), mutación estructural (para modificar el número de nodos en un cromosoma), y mutación de los pesos.
A continuación se describen los operadores con mayor detalle.

Inicialización de la Población

Una función inicializa cada elemento de la población de la siguiente manera: el número de nodos ocultos representados en cada cromosoma se genera de manera aleatoria, cuidando que no exceda un número máximo definido por el usuario. De igual manera los pesos de cada nodo se inicializan de forma aleatoria con valores entre –1 y 1. (lo cual se discutió al principio en este mismo capítulo)

Función de Evaluación

La función de evaluación debe indicar la aptitud de cada cromosoma, que en nuestro contexto significa qué tan bien el perceptron multicapa correspondiente al cromosoma clasifica un conjunto de datos. Consideremos dos diferentes maneras de calcular ésta función, aunque hay otras posibilidades. Para explicarlas hay que definir la manera exacta en la cual un perceptron multicapa calcula su salida. En el caso del perceptron multicapa de la Fig.1, la entrada a cada uno de los nodos ocultos es el producto interno del vector de los pesos entrando en el nodo con el vector (1,x1,x2), dando:

$$-0.12+0.074*x1+0.017*x2$$
$$-0.59+0.146*x1+0.937*x2$$

La función de transferencia que se ocupa es la función sigmoide definida de la siguiente manera: $f(u) = 1/(1+\exp(-u))$. Se aplica esta función de transferencia para calcular la salida de cada uno de los nodos ocultos dando:

$$z1=f(-0.12+0.074*x1+0.017*x2)$$
$$z2=f(-0.59+0.146*x1+0.937*x2)$$

Finalmente, para calcular la salida del nodo en la capa de salida, se vuelve a aplicar la función de transferencia al producto interno del vector de los pesos que entran en el nodo con el vector (1,z1,z2), dando:

$$y1=f((0.7-0.7*z1-0.9*z2)$$

Al principio de este capítulo se mencionan las ventajas de usar esta función de transferencia, aunque existen otras, esta es una de las más usadas.

La función de evaluación se basa en el error cuadrático medio, ECM, que comúnmente se usa en el algoritmo de retropropagación para determinar el error. El ECM se define de la siguiente manera:

$$ECM=\sum(yi-di)^2$$

donde la suma es sobre todos los patrones de los datos, *yi* es la salida del perceptron multicapa para el patrón *i* calculado como se describió anteriormente, y *di* es la salida deseada, que seria el valor que queremos asociar con el patrón.

A la solución del programa evolutivo se le calcula cuántos errores tiene al clasificar.

Selección de la Nueva Población

Para escoger una nueva población se aplica el proceso conocido como la rueda de la ruleta, es importante mencionar que no es el único método que existe. La idea esencial es particionar un disco en sectores en proporción a la aptitud de cada cromosoma. Se gira el disco, y de acuerdo al sector donde éste para, se escoge el cromosoma correspondiente. Este operador fue descrito en el capítulo anterior.

Mutación Estructural

Es necesario tener al menos un operador que pueda cambiar el tamaño de los cromosomas; si no fuera así, el número de nodos ocultos en cada cromosoma quedaría igual al número con el cual fue inicializado. Esto es importante de acuerdo a nuestra meta de encontrar simultáneamente la arquitectura y los pesos más adecuados de un perceptron multicapa para resolver un problema de clasificación. El programa evolutivo utiliza dos operadores, Adicionar y Eliminar, definidos como sigue:

1. Adicionar, se encarga de aumentar un nodo oculto al cromosoma en cuestión inicializando sus pesos con 0. Se valida que al aumentar este nodo no sobrepase el número máximo de nodos permitidos por el usuario; si ya se tiene ese número máximo no se añade otro nodo. Este nodo se incorpora al final de la lista y se hace de acuerdo a un número generado de forma aleatoria: si este es menor a una probabilidad de adición definida por el usuario se adiciona el nodo, en otro caso no se hace ningún cambio al cromosoma. Con respecto al inicializar los pesos en ceros, se decidió así para que el programa evolutivo tratara de encontrar los pesos idóneos para ese nuevo nodo sin influir de alguna manera en esto, aunque bien se pudo haber aplicado algún algoritmo que acercará los pesos a algún valor idóneo de acuerdo a los pesos de los nodos ya existentes.

2. Para equilibrar el efecto que pudiera tener la función Adiciona, y para reducir el número de nodos ocultos en un cromosoma, se define otro operador de nombre Eliminar. De forma similar se genera un número aleatorio para cada individuo y se compara con la probabilidad de eliminación definida por el usuario, si dicho número es menor se genera otro número

aleatorio entre uno y el máximo número de nodos existente en ese individuo y se elimina dicho nodo de la lista del cromosoma en cuestión.

En los dos casos se reduce linealmente el parámetro de probabilidad de su valor inicial a un valor pequeño definido por el usuario, conforme evoluciona el algoritmo; con esto, al final de la corrida no es necesario modificar la arquitectura del perceptron multicapa.

Mutación Paramétrica

Para modificar los pesos de un cromosoma, se aplica un operador a los pesos de un nodo oculto del cromosoma. Esto quiere decir que para cada nodo oculto de un cromosoma se genera un número aleatorio y si éste es menor a la probabilidad de mutación definida por el usuario, se modifican los pesos agrupados en el nodo; en caso contrario se conserva el mismo valor de los pesos. Si se decide cambiar el valor de un peso se le agrega el valor generado por una gaussiana, con media 0 y varianza definida por el usuario. La varianza está reducida linealmente conforme evoluciona el algoritmo; esto se hace para evitar perturbaciones "grandes" a los pesos lo cual no es benéfico al final de la corrida.

Es importante mencionar que se probó el programa evolutivo reduciendo la probabilidad de mutación de igual manera para la mutación estructural y paramétrica y por el otro lado se probó solo reduciendo la probabilidad de mutación estructural, dejando la probabilidad de mutación paramétrica constante, ocasionando que al principio se definiera la estructura de la red y se acercarán los pesos y después sólo se definieran los pesos. Reportándose mejores resultados para el segundo método, el cual es el que se aplica. Por ejemplo, para la base de datos de Pima se encontró un 72.85% de aciertos primer método contra 74.25% del segundo método y para la base de vinos un 95.2% del primer método contra un 98.11% del segundo método.

Guardar el mejor cromosoma

En la literatura se han reportado buenos resultados usando un operador llamado elitismo. El cometido de éste es asegurar que el mejor cromosoma encontrado sobreviva de una generación a otra. En el programa evolutivo, si el cromosoma con mejor aptitud de la población actual supera al seleccionado en la población anterior, éste es reemplazado, conservando siempre al mejor cromosoma de una generación a otra. En otro caso reemplaza el peor cromosoma de la población actual con el mejor de la generación anterior.

Referencias

[1] Mohamad H. Hassoun, “Fundamentals of artificial Neural Networks”, The MIT Press 1995.
[2] Richard O. Duda, Peter E. Hart, David G. Stork, J. Wiley. “Pattern Classification” , 2nd ed., 2000.
[3] http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-6.html.
[4] Günter Rudolph, “Convergence Analysis of Canonical Genetic Algorithms”, 1992.
[5] A. E. Eiben, E.H.L. aarts and K.M. Van Hee, “Global convergence of genetic algorithms: A Markov chain analysis”, in Parallel Problem Solving from nature”, H.P. Schwefel and R. Männer (Eds.), Berlin and Heidelberg: Springer, 1991, pp4-12.
[6] David J. Montana, "Neural Network Weight Selection Using Genetic Algorithms", capitulo 5 de Intelligent Hybrid Systems, Goonatilake and Khebbal (Eds.), John Wiley & Sons, 1995.
[7] Peter J. Angeline, Gregory M. Saunders y Jordan B. Pollack, "An Evolutionary Algorithm that Constructs Recurrent Neural Networks", IEEE Transactions on Neural Networks, 5 (1), pp. 54-65, 1994.
[8] John Goddard, Irineo López, Leonardo Rufiner, “Diagnóstico de cardiopatías mediante redes neuronales y algoritmos genéticos”, Revista Argentina de Bioingeniería, Vol.2, No.2, 11-19, 1996.
[9] Xin Yao, Young Liu. Towards Designing Artificial Neural Networks by Evolution, 1999.
[10] Xin Yao “Evolving artificial Neural Networks” Proc. of the IEEE, 87(9); pp 1423-1447, Sept. 1999.
[11] Fréderic Gruau “Genetic Programing of Neural Networks: Theory and Practice”. Capítulo 13 de Intelligent Hybrid Systems, Goonatilake and Khebbal, John Willey & Sons, 1995.
[12] David B. Fogel. “Evolutionary computation. Toward a New Philosophy of Machine Intelligence” The Institute of electrical and electronic engineers, New york, 1995.

RESULTADOS Y CONCLUSIONES

En el capítulo anterior se explicaron varias opciones que se probaron para el algoritmo las cuales se retoman en las conclusiones. Además en este capítulo se hace un análisis de los resultados obtenidos al aplicar el programa evolutivo de cromosomas de longitud variable propuesto en el capítulo anterior, a las bases de datos descritas en el apéndice B, haciendo una comparación con los resultados reportados obtenidos por otros algoritmos en la clasificación de cada base de datos y con los obtenidos aplicando retropropagación.

Resultados

En el apéndice B se describieron detalladamente las bases de datos que se utilizaron para probar el desempeño del programa evolutivo propuesto, por tal motivo, en la tabla 1 y 2 solo se presentan las características más relevantes de cada base de datos.

Nombre de la BD	**Clasific. De:**	**Atribu - tos**	**Clases**	**Ejemplos Totales**	**Ejemp. entren.**	**Ejemp. Prueba**	**% de aciert. Reportado**
Iris	Plantas	4	3	150	105	45	100
Pima	Diabetes(si/no)	8	2	336	235	101	76
Desord. cardiac.	Corazón (si/no)	13	2	270	189	81	77.5 y 81
Peterson Barney	Voz	3	2	275	192	83	80
Proteinas	Localiz. de prot.	7	8	336	235	101	81
Vino	Vino	13	3	178	125	53	Del 96 al 100%

Tabla 1

El número de ejemplos de entrenamiento corresponde al 70% de los ejemplos totales y el 30% restante corresponde a los ejemplos de prueba.

La tabla 2 muestra la distribución de los ejemplos en las clases

Base de datos	**Distribución de los ejemplos en las clases**
Iris	50 ejemplos para cada clase
Pima	111 clase diabetes negativa y 225 clase diabetes positiva
Desord. card.	150 clase sanos y 120 clase enfermos
Peterson Barney	132 clase UH y 143 clase UW
Proteínas	Mínimo 2 ejemplos, máximo 143 son 8 clases
Vino	59 ejemp. Clase 1, 71 ejemp. Clase 2 y 48 ejemp. Clase 3

Tabla 2

En la tabla 3 se reportan los parámetros obtenidos por medio de experimentación para el programa evolutivo propuesto, el tamaño de la población para cada base de datos varió en 10, 15 y 20 y el número de generaciones para cada base de datos fueron de 100, 150, 200, 250, 500 y 1000 obteniéndose los mejores resultados en el tamaño de la población y número de generaciones reportadas, cabe mencionar que no era necesario hacer todas las

combinaciones de número de generaciones y tamaño de la población ya que con la experimentación el programa evolutivo en muy pocas corridas determinaba el número de nodos ocultos y cuáles eran los parámetros idóneos.

El número máximo de tamaño de los cromosomas se obtuvo también a base de experimentación, basándonos en la complejidad de clasificación de cada base de datos se hicieron pruebas con diferentes tamaños obteniéndose mejores resultados en los reportados.

La última columna reporta el tiempo que tardo el programa evolutivo en ejecutarse con los parámetros indicados, cabe mencionarse que el programa se ejecutó en una PC pentium, con 32 megas de RAM bajo ambiente windows, el programa esta escrito en lenguaje C.

BD	**Población**	**Generaciones**	**Max. tam cromosom.**	**Tiempo ejecuc.**
Iris	10	150	15	25 seg
Pima	15	200	15	60 seg
Desord. Card	10	150	50	10 seg
Peterson Barney	10	500	100	40 seg
Proteínas	10	500	50	190 seg
Vino	10	350	50	50 seg.

Tabla 3

En la tabla 4 se reportan los porcentajes de aciertos de los ejemplos de entrenamiento y de los de prueba, además del fitness final calculado y el número de nodos ocultos encontrado por el programa evolutivo en cuestión, incluyendo nuevamente los datos de los porcentajes de aciertos reportados en otros trabajos utilizando otros algoritmos para clasificación y el porcentaje de aciertos obtenido utilizando retropropagación. Para obtener los resultados que se muestran en la tabla 4 con retropropagación se utilizó la caja de herramientas de redes neuronales de Matlab, los parámetros utilizados fueron los encontrados por el programa evolutivo, es decir, se entrenó la red neuronal con retropropagación con el número de nodos ocultos encontrado y con el número de generaciones con que trabajo el programa evolutivo para cada base de datos.

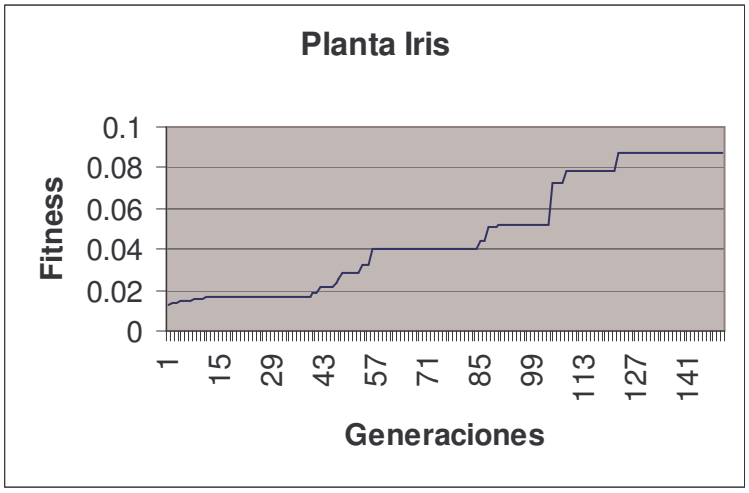
BD	**Entrenam. % aciertos**	**Prueba % aciert.**	**Fitness**	**Nodos Ocult.**	**% aciertos retroprop.**	**% aciert reportados**
Iris	90.4	**100**	0.0869	6	**86.6**	**100**
Pima	80.85	**74.25**	0.0315	5	**75.71**	**76**
Desord. card	83	**77.7**	0.0493	9	**79**	**77.5 y 81**
Peterson Barney	89.07	**78.26**	0.0650	20	**77.17**	**80**
Proteínas	68.9%	**63.6%**	0.0125	31	**63.36**	**81**
Vino	92.8	**98.11**	0.1215	18	**96.22**	**Del 96 a l 100**

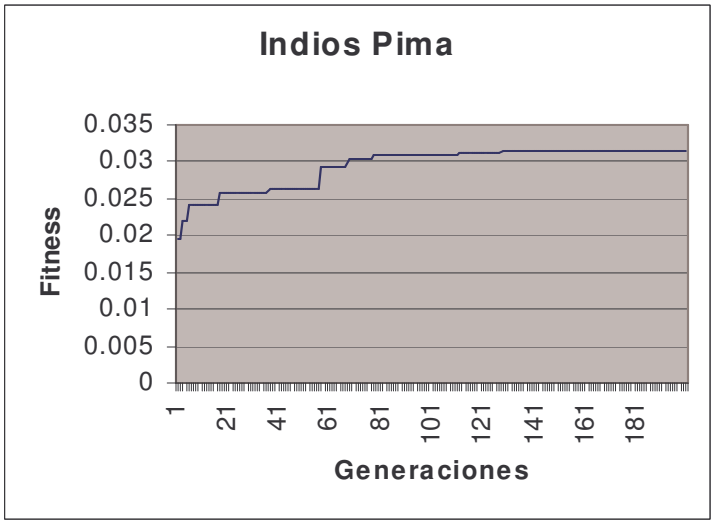
Tabla 4

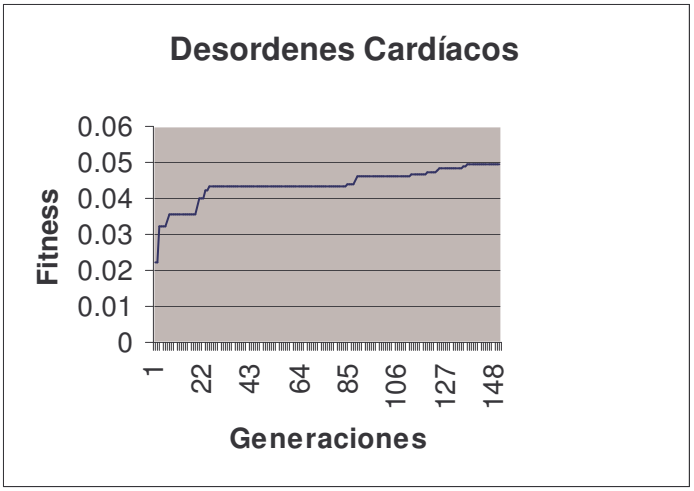
Es importante notar que en los resultados reportados por otros autores para la base de datos de los desordenes cardíacos siempre se trabajo un algoritmo genético con longitud fija

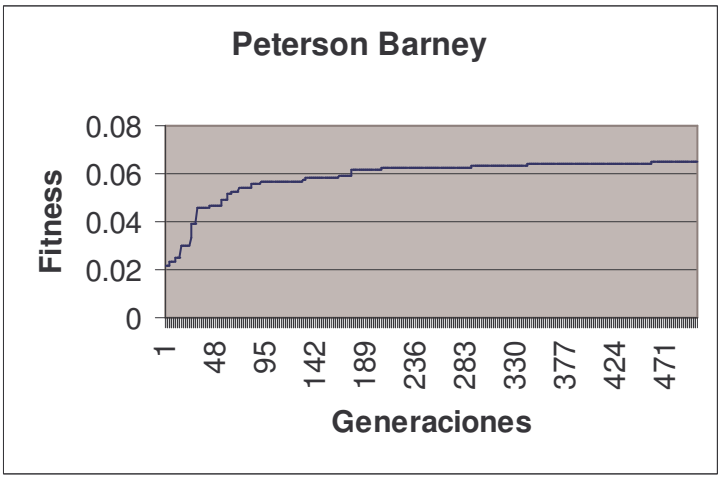
equivalente a 13 nodos ocultos y también recordar que el programa evolutivo nos proporciona el número de nodos ocultos y los pesos de las conexiones del perceptron multicapa.

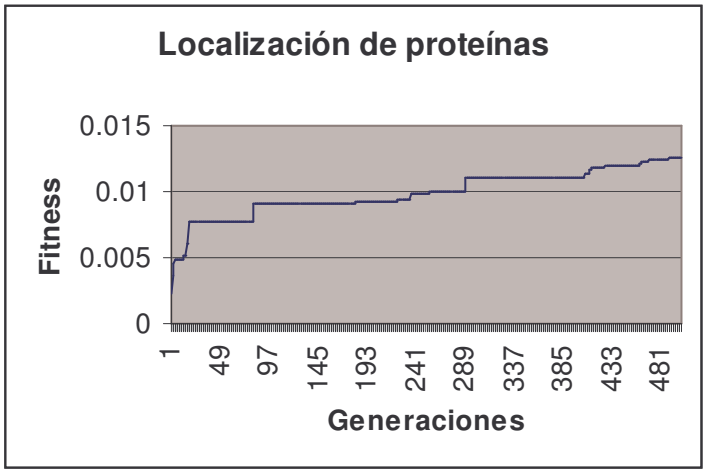
A continuación se presentan las gráficas del fitness contra número de generaciones para cada base de datos, esto nos ayuda a visualizar como el programa evolutivo se aproxima a las soluciones presentadas.

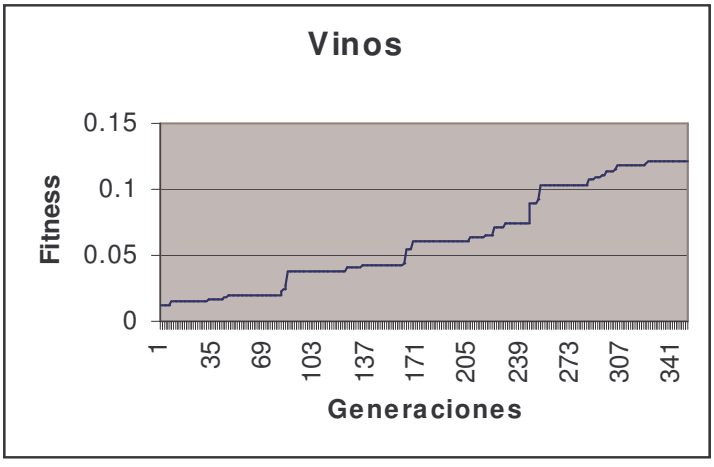





Desordenes Cardíacos
Fitness
0.06
0.05
0.04
0.03
0.02
0.01
0
1
22
43
64
85
106
127
148
Generaciones


Peterson Barney
Fitness
0.08
0.06
0.04
0.02
0
1
48
95
142
189
236
283
330
377
424
471
Generaciones


Localización de proteínas
Fitness
0.015
0.01
0.005
0
1
49
97
145
193
241
289
337
385
433
481
Generaciones


Vinos
Fitness
0.15
0.1
0.05
0
1
35
69
103
137
171
205
239
273
307
341
Generaciones

Conclusiones

Los resultados presentados son satisfactorios, el algoritmo cumple con su objetivo que es definir una arquitectura perceptron multicapa (con una capa oculta) para la solución de problemas de clasificación, proporcionando el número de nodos ocultos y los pesos de sus conexiones. Los porcentajes de aciertos para cada base de datos presentados son muy similares a los reportados por otros autores utilizando otros algoritmos de aprendizaje. Se probó que es mejor un programa evolutivo para este tipo de problema con respecto a un algoritmo genético (véase el capítulo anterior), ya que se probó cruzamiento y los resultados no mejoraron a los presentados. Se intentó aplicar un algoritmo híbrido con retropropagación (véase capítulo anterior) sin embargo no mejoraron los resultados, quitando esta alternativa. El algoritmo propuesto es muy sencillo y tiene una estructura muy flexible.

Con respecto a los porcentajes de aciertos en las bases de datos usadas, en la base de datos Iris se iguala el porcentaje de aciertos al reportado con la ganancia de proporcionar la arquitectura de la red neuronal perceptron idónea a este problema. En la base de datos de los indios Pima, reportan un 76% de aciertos utilizando un algoritmo genético cuya población es representada por una red neuronal perceptron multicapa con 13 nodos ocultos en cada individuo, el programa evolutivo propuesto reporta un 74.25% de aciertos y define una arquitectura con 9 nodos ocultos mejorando así, la arquitectura reportada; con respecto a el porcentaje de aciertos, es mínima la diferencia. El resto de las bases de datos se acercaron a los porcentajes reportados, salvo el resultados de la base de datos de la localización de proteínas cuyo porcentaje obtenido fue muy bajo comparado con el reportado, sin embargo, igual que en Iris, se gana la definición de la red neuronal idónea para las otras bases de datos.

Al comparar los resultados del programa evolutivo con los obtenidos ocupando solo el algoritmo de retropropagación se ven que están muy semejantes, la diferencia de los porcentajes de aciertos es mínima, esto implica que el programa evolutivo esta encontrando pesos muy similares a los obtenidos con retropropagación. Recordemos que los parámetros para el entrenamiento de la red con retropropagación fueron los nodos ocultos obtenidos con el programa evolutivo y el número de generaciones que ocupó el programa evolutivo para encontrarlos.

Con respecto al tiempo de ejecución, el más tardado fue para la base de datos de proteínas, debido al número de ejemplos(336) y número de generaciones (500). Si se observa la tabla 3 se ve que los tiempos de ejecución son muy cortos y la PC donde se ejecutaron no es muy grande en su capacidad (Pentium de 32 megas de RAM y un disco duro de 2GB)

Finalmente en las gráficas de fitness contra generaciones, se observa que el fitness va aumentando, esto es porque se manejo la maximización del fitness.

Perspectivas

El algoritmo propuesto se puede probar combinando varios parámetrso, por ejemplo existen varias funciones para evaluar el fitness, así como para la selección de la población y para la mutación tanto paramétrica como estructural, dando oportunidad para seguir trabajando en el y de ser posible llegar a un algoritmo que mejore cada vez la propuesta hecha hasta este momento.

Actualmente se están aplicando técnicas para la paralelización del algoritmo propuesto y aunque los tiempos no son muy grandes se pueden probar bases de datos más robustas donde la paralelización de algoritmos genéticos y programas evolutivos ayuden a una ejecución más rápida.

Apéndice A

REDES NEURONALES ARTIFICIALES

Las redes neuronales artificiales son una rama importante dentro del área de la Inteligencia Artificial. Consisten en modelos simplificados de las redes de neuronas biológicas con la finalidad de tener un comportamiento similar. Se ha demostrado que las redes neuronales artificiales cuya arquitectura es un perceptron multicapa funcionan satisfactoriamente como clasificador automático[1] tratando de dar una solución conveniente al problema de clasificación que se explicó en el capítulo 1. De aquí la importancia de este apéndice donde se desarrolla la teoría de este tipo de redes neuronales artificiales, la cual se hace de forma breve omitiendo varios detalles, para mayor información consultar la bibliografía al final de este apéndice.

Redes neuronales artificiales

Las redes neuronales artificiales pueden verse como modelos simplificados de las redes biológicas de neuronas. De manera similar a éstas, intentan "aprender" a partir de los datos que les son suministrados.

Una neurona biológica consta de múltiples prolongaciones cortas muy ramificadas llamadas dendritas y otra, única y larga, que sale del soma en dirección contraria llamada axón. Las dendritas son prolongaciones especializadas en recibir estímulos provenientes de otras células, mientras que el axón está especializado para conducir impulsos. En el soma converge un número considerable de sinapsis, o contactos, que provienen de otras neuronas; los efectos de estos contactos se suman para obtener el potencial integrado. Si este potencial excede un cierto umbral, la célula dispara y genera un potencial de acción que inicia su viaje a lo largo de su axón. Se calcula que en el cerebro humano existen del orden de 10^3 conexiones y 10^{10} neuronas (Figura 1) [2, 3].

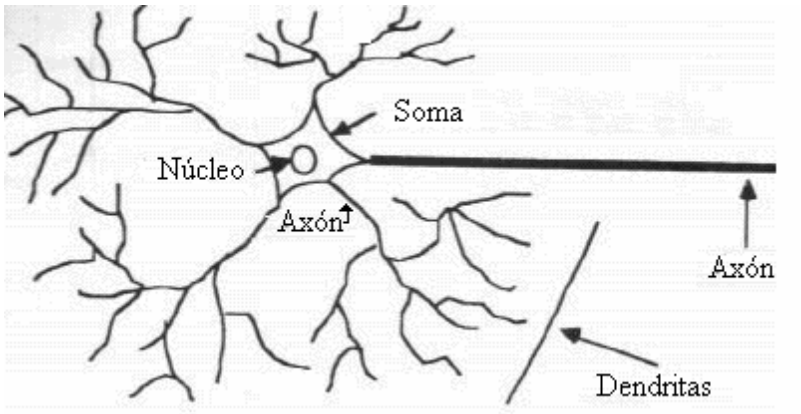


Fig. 1 Red neuronal biológica

Las redes neuronales artificiales tienen un comportamiento similar. Las señales que llegan a las sinapsis son las entradas a la neurona; éstas se ponderan, mediante atenuaciones o amplificaciones, a través de un parámetro denominado peso, asociado a la sinapsis correspondiente. Estas señales de entrada pueden excitar a la neurona, funcionando como

sinapsis con peso positivo, o inhibirla, sinapsis con peso negativo. El efecto es la suma de las entradas ponderadas; si la suma es mayor o igual que el umbral de la neurona esta se activa, de lo contrario permanece inactiva[4].

Panorama histórico

A continuación se dará una breve semblanza de la historia de las redes neuronales artificiales y los tipos de arquitectura existentes. La cual esta basada de la cita [4].

En 1936 Alan Turing fue el primero en estudiar el cerebro como una forma para ser modelada por medio de la computación; sin embargo los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch, un neurofisiólogo y Walter Pitts, un matemático, quienes en 1943 lanzaron su teoría acerca del funcionamiento de una neurona. Ellos modelaron una red neuronal simple mediante circuitos eléctricos. En 1957 Frank Rosenblatt comenzó el desarrollo del perceptron, este tipo de red es el más antiguo y se usa hoy en día de varias formas para aplicación de reconocedor de patrones, este modelo era capaz de generalizar, es decir, después de haber aprendido una serie de patrones era capaz de reconocer otros similares, aunque no se le hubieran presentado anteriormente. Sin embargo tenía sus limitaciones, tal vez la más conocida era su incapacidad para resolver el problema de la función OR-exclusiva (XOR), en general no era capaz de clasificar clases no separables linealmente.

En 1959 Bernard Widrow y Marcial Hoff de Stanford, desarrollaron el modelo ADALINE (ADAptive LINear Elements), esta fue la primera red neuronal artificial aplicada a un problema real, (filtros adaptivos para eliminar eco de las líneas telefónicas).

Otro investigador importante desde los años 60 es Stephen Grossberg quien ha escrito numerosos libros y desarrollado modelos de redes neuronales, él realizó en 1967 una red de nombre Avalancha que resolvía problemas como el reconocimiento continuo del habla y aprendizaje del movimiento de los brazos de un robot.

En 1962 surgieron numerosas criticas que frenaron, hasta 1982, el crecimiento que estaban experimentando los investigadores sobre redes neuronales. Marvin Minsky y Seymour Papert, del Instituto Tecnológico de Massachussets publicaron el libro *Perceptrons*, que además de contener un análisis matemático detallado del perceptron consideraban que la extensión a perceptrones multinivel (el perceptron original sólo poseía dos capas, la de entrada y la de salida) era completamente estéril.

A pesar del libro *Perceptrons* algunos investigadores continuaron su trabajo, tal fue el caso de James Anderson, quien desarrolló un modelo lineal, llamado Asociador Lineal que consistía en elementos integradores lineales (neurona) que sumaban sus entradas, este modelo se basa en el principio de que las conexiones entre neuronas son reforzadas cada vez que están activadas.

Otros investigadores importantes lo son Teuvo Kohonen, ingeniero electrónico de la Universidad de Helsinki quien desarrolló un modelo similar al de Anderson, la red de

Kohonen es aplicada a la solución de problemas de optimización y John Hopfield quien presentó su trabajo sobre redes neuronales en la Academia Nacional de las Ciencias, en él describe con claridad y rigor matemático una red neuronal que lleva su nombre cuya aplicación es la reconstrucción de patrones y optimización (1982).

Funcionamiento básico de una red neuronal artificial

Como se mencionó anteriormente existen diferentes enfoques de redes neuronales artificiales de acuerdo a su aplicación. La arquitectura que nos interesa corresponde a un perceptron multicapa, pero para abordarla es necesario comprender la metodología de un perceptron simple, que es la que desarrollo Rosenblatt en 1958.

La arquitectura de perceptron simple se presenta en la figura 3 y es aquella cuya función discriminante es lineal. La descripción de esta sección esta basada en la cita [4,5,7,8,9].

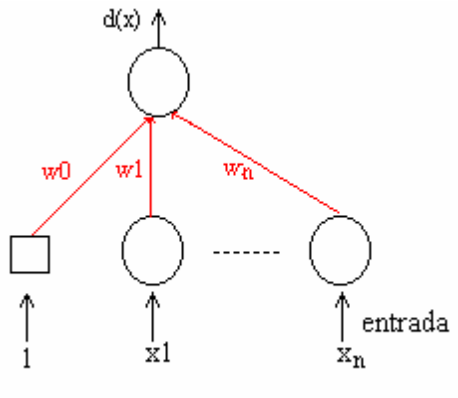


Fig. 3 Arquitectura de una red neuronal artificial con dos capas

Este tipo de arquitectura tiene sus limitaciones, como lo es el no establecer regiones de decisión más complejas que la de dos semiplanos.

Con esta arquitectura se pretende resolver un problema de clasificación donde el número de clases son 2: C_1 y C_2. Esta arquitectura consta de dos capas. La de entrada con $n+1$ nodos y la de salida con un nodo. Los nodos de la capa de entrada están conectados con el nodo de salida; cada conexión tiene un valor numérico llamado peso, el primer nodo de la capa de entrada tiene un valor numérico constante llamado sesgo y es igual a 1, los demás nodos tienen el valor de los atributos del objeto a clasificar. El sesgo ayuda a encontrar un hiperplano dando oportunidad de ubicarlo de mejor manera en el espacio, no necesariamente pasando por el origen. En la primer capa de la red no existe ningún tipo de procesamiento. En la segunda se hace el cálculo de la función discriminante, la cual se define de la siguiente manera:

$$d(x) = < w, x >$$

donde $w=(w_0,w_1,...,w_n)$ y $x=(1,x_1,...,x_n)$.

Para asignar el objeto a una clase se analiza la salida: si $d(x)>0$ el objeto a clasificar pertenece a la clase C_1 en otro caso, si $d(x)<0$ el objeto se asigna a la clase C_2. En este caso $d(x)=1w_0+x_1w_1+...+x_nw_n$.

Como ya se mencionó, este tipo de arquitectura tiene sus limitaciones, tal vez la más conocida, es el no poder resolver el problema de la función XOR que consiste en separar las siguientes clases: para los valores de entrada 00 y 11 se devuelve la clase 0 y para las entradas 01 y 10 la clase 1. sin embargo no existe un hiperplano que separe los patrones de una clase con respecto a la otra. (fig. 4)

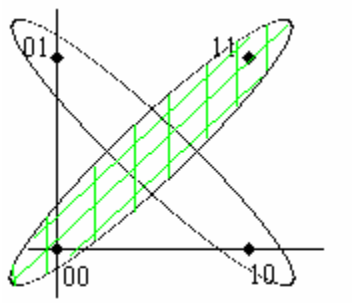


Fig. 4 Función XOR. No es posible obtener una recta que separe las dos clases

La arquitectura descrita es la más sencilla. Se dejó un solo nodo de salida pensando que son dos clases las que se van a separar; en caso de haber más clases, se aumenta el número de nodos de salida. En el caso de tener un problema donde las clases no estén bien separadas en el espacio (es decir, que no son linealmente separables) se necesitará modificar la red creando una o más capas ocultas. Las capas ocultas se encuentran entre la capa de entrada y la de salida.

A continuación se describe el tipo de regiones de decisión que se pueden obtener de acuerdo a la arquitectura de un perceptron. Información basada en la cita [4].

El perceptron básico de dos capas (la de entrada y la de salida, como la arquitectura descrita anteriormente) solo puede establecer dos regiones separadas por una frontera lineal en el espacio de patrones de entrada. Un perceptron con tres capas de neuronas (la capa de entrada, la capa oculta y la de salida) puede formar cualquier región convexa en este espacio. Las regiones convexas se forman mediante la intersección entre las regiones formadas por cada neurona de la segunda capa. Cada uno de estos elementos se comporta como un perceptron simple (de dos capas), el problema se reduce a definir el número de nodos que componen a la capa oculta, en general este número deberá ser lo suficientemente grande para que se forme una región lo suficientemente compleja para la resolución del problema, sin embargo tampoco es conveniente que el número de nodos sea tan grande que la estimación de los pesos no sea fiable para el conjunto de patrones de entrada disponibles. Un perceptron con cuatro capas puede formar regiones de decisión arbitrariamente complejas. El proceso de separación en clases que se lleva a cabo consiste en la partición de la región deseada en pequeños hipercubos (cuadrados para dos entradas de la red).

No es necesario una arquitectura con más de cuatro capas debido al tipo de regiones de decisión que pueden formar, la tendencia es no complicar la arquitectura de la red sino estudiar el tipo de función de activación que se puede aplicar.

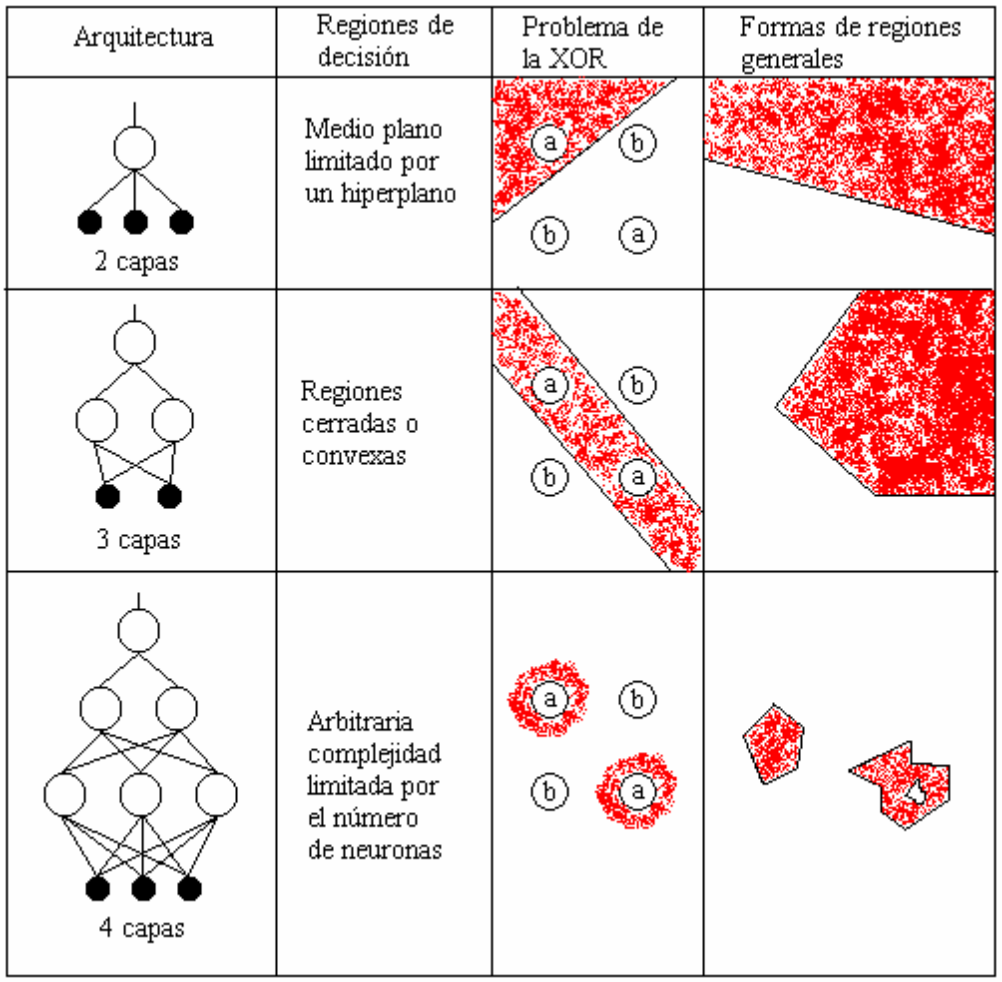


Figura 6 distintas formas de regiones generadas por un perceptron multinivel[4]

En la figura 6 se ilustra lo explicado y se añade la solución al problema XOR (visto anteriormente), como se muestra en la figura 6, en la sección que corresponde a la arquitectura de dos capas, no existe alguna recta que separe los patrones de una clase con respecto a la otra. Por eso es necesario experimentar con otras arquitecturas para darle una solución a este problema.

En la figura 7 se muestra una arquitectura con una capa oculta, denominada también red neuronal multicapa o red neuronal perceptron de 3 capas. En caso de existir capas ocultas se añade un nodo denominado sesgo (el cual se explicó anteriormente) en cada una de ellas con un valor constante igual a 1.

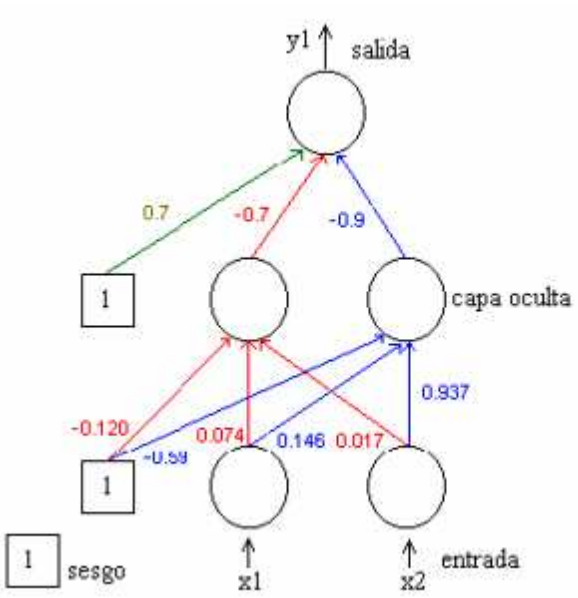


Fig. 7 Red neuronal perceptron multicapa

Para mayor claridad, con base a la figura 7 se realizaran las operaciones necesarias para calcular la salida de la red.

En el caso de la arquitectura del ejemplo, la entrada a cada uno de los nodos ocultos es el producto interno del vector de los pesos entrando en el nodo con el vector (1,x1,x2), dando:

$$-0.12+0.074*x1+0.017*x2$$
$$-0.59+0.146*x1+0.937*x2$$

La función de transferencia puede ser una función sigmoide definida de la siguiente manera: $f(u) = 1/(1+\exp(-u))$ o una función escalón (ambas se discutirán más adelante dentro del mismo capítulo). Se aplica esta función de transferencia para calcular la salida de cada uno de los nodos ocultos dando:

$$z1=f(-0.12+0.074*x1+0.017*x2)$$
$$z2=f(-0.59+0.146*x1+0.937*x2)$$

Finalmente, para calcular la salida del nodo en la capa de salida, se vuelve a aplicar la función de transferencia al producto interno del vector de los pesos que entran en el nodo con el vector (1,z1,z2), dando:

$$y1=f((0.7-0.7*z1-0.9*z2)$$

Si hubieran existido más capas ocultas el procedimiento sería el mismo. El número de capas intermedias y el número de nodos de cada capa dependerá del tipo de aplicación al que se vaya a destinar la red neuronal. Los pesos se inicializan aleatoriamente con valores reales entre 0 y 1.

A continuación se presenta de modo de síntesis los pasos y fórmulas para calcular la salida de una red neuronal multicapa (véase fig. 8, tomado de la cita 4 y 7):

1. Se inicializan los pesos de la red con valores pequeños aleatorios entre 0 y 1
2. Se proporciona un patrón de entrada en forma de vector X_{p1}, X_{p2}, ...,X_{pn}
3. Se calculan las entradas netas para las neuronas ocultas procedentes de la neurona de entrada.
   Para una neurona j oculta:

$$net_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h} x_{pi}$$

   donde el índice h se refiere a magnitudes de la capa oculta, el subíndice p, al p-ésimo vector de entrada, j a la j-ésima neurona oculta y N es el número de patrones de entrada

4. Se calculan las salidas de las neuronas ocultas:

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

5. Se realizan los mismos cálculos para obtener las salidas de las neuronas de salida, (capa 0 es la de salida y L es el número de neuronas ocultas):

$$net_{pk}^{0} = \sum_{j=1}^{L} w_{kj}^{0} y_{pj}$$

$$y_{pk} = f_{k}^{0}(net_{pk}^{0})$$

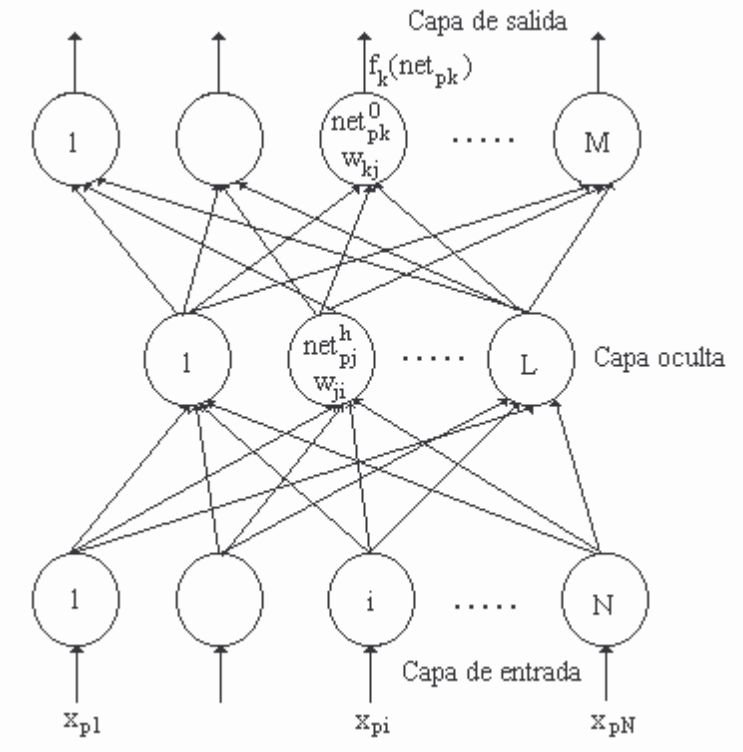


Fig. 8 Modelo de una arquitectura perceptron multicapa

Ejemplo de clases linealmente separables y no linealmente separables dentro de la misma base de datos

Un ejemplo de un conjunto de datos donde se aprecia los dos casos mencionados, uno de clases linealmente separables y el otro donde no lo son, es el mencionado en el capítulo anterior, y es el de la base de datos de la planta Iris. En la figura 9 se muestra la distribución en dos dimensiones de dos de los atributos de los ejemplos que corresponden a este conjunto de datos. Este consta de tres clases de planta iris: Iris Setosa, Iris Versicolor e Iris Virginica; cada clase consta de 50 ejemplos y cada ejemplo tiene 4 atributos para ser diferenciados: longitud de sépalo, ancho de sépalo, longitud de pétalo, y ancho de pétalo, todos dados en cms. Se puede observar que la clase Iris Setosa es linealmente separable a cualquiera de las otras dos clases, es decir, se puede encontrar un hiperplano que las logre separar, sin embargo la clase Iris Versicolor no es linealmente separable de la clase Iris Virginica ya que hay ejemplos traslapados. Por tanto sería difícil tratar de distinguir estas dos clases con una red como la que se describió al inicio de la sección [5,6].

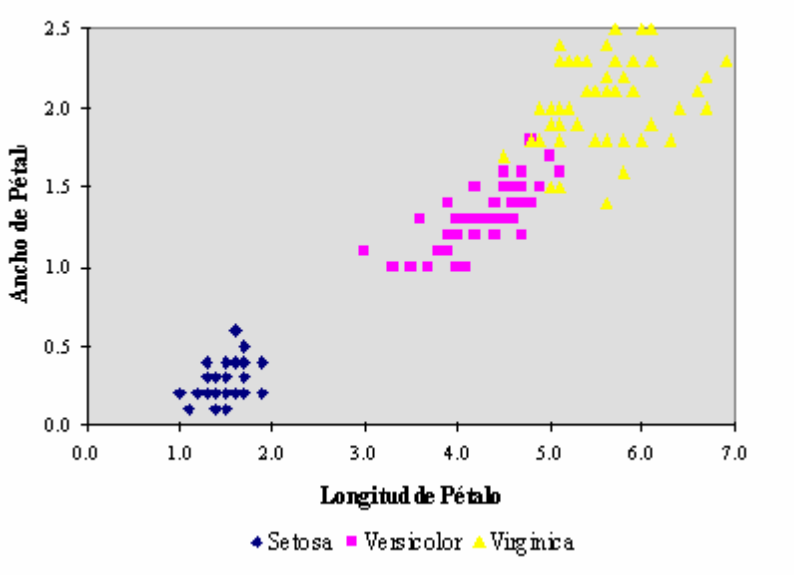


Fig. 9 Distribución de las tres clases de la planta Iris

Función de transferencia

Existen varias funciones de transferencia, o funciones de salida que determinan el estado final de la red con respecto al objeto a estudiar; estas funciones de transferencia cambian el estado actual de activación de la neurona en una señal de salida. A continuación se explican solo dos funciones de transferencia: la función escalón y la función sigmoidal, debido a que son las que utilizamos en la descripción del funcionamiento de una red neuronal perceptron en este trabajo, además de que la función sigmoide es la utilizada en la propuesta que se hace en el presente. [4,7,8]

Función de transferencia escalón

En la descripción del funcionamiento de un perceptron se explicó cómo se calcula cada *d(x)*; al resultado de cada una de éstas se le aplica una función de transferencia. En caso de la función escalón se define un umbral θ que depende de la aplicación del perceptron

$$y=1 \text{ si } d(x) >= \theta \quad \text{o} \quad y=0 \text{ si } d(x) < \theta$$

donde y es la salida y el umbral θ con frecuencia será 0.

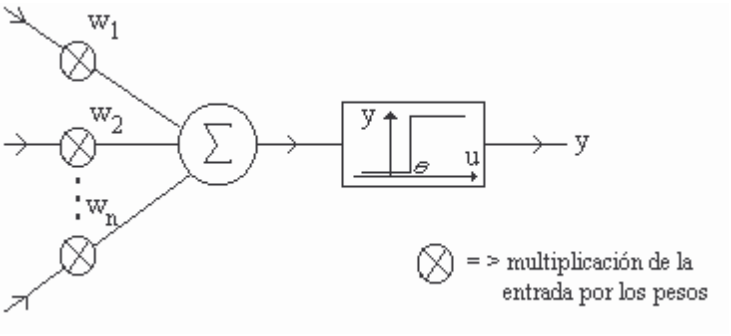


Fig. 10 Función de transferencia escalón

La función escalón o umbral únicamente se utiliza cuando las salidas de la red son binarias. (Fig. 10)

Función de transferencia sigmoidal

En las redes artificiales la función escalón limita mucho la salida a dos valores sin embargo esto puede superarse suavizando la función escalón mediante una función continua sigmoidea como se muestra en la figura 11.

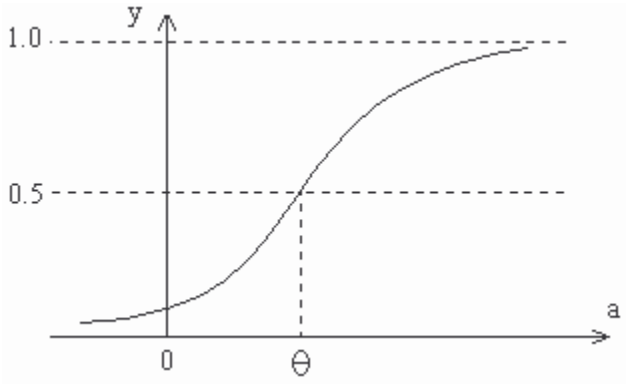


Fig. 11 Función de transferencia sigmoide

cuya definición matemática es:

$$y = \frac{1}{1 + e^{-a/\rho}} \qquad (1)$$

donde ρ determina la forma de la sigmoide; entre más grande sea el valor de ρ, más llana se hace la curva. En muchos textos este parámetro se omite quedando como 1. Las unidades que cuentan con esta función reciben el nombre de unidades semilineales. En la ecuación 1 el umbral es cero, si se requiere el parámetro diferente de cero entonces se debe incluir un desfasamiento dado por la ecuación 2

$$y = \frac{1}{1 + e^{-(a-\theta)/\rho}} \qquad (2)$$

Tipos de aprendizaje básicos

Una de las formas de definir el concepto general de aprendizaje es "La modificación del comportamiento inducido por la interacción con el entorno y como resultado de experiencias conduciendo al establecimiento de nuevos modelos de respuesta a estímulos externos", esta definición fue dada antes de la existencia de las redes neuronales, sin embargo es aplicada a ellas.

Biológicamente se suele aceptar que la información memorizada en el cerebro esta más relacionada con los valores sinápticos de las conexiones entre las neuronas que con ellas mismas, es decir, el conocimiento se encuentra en las sinapsis. En el caso de las redes neuronales artificiales, se puede considerar que el conocimiento se encuentra representado en los pesos de las conexiones entre las neuronas. Se puede decir, que el proceso de aprendizaje se lleva a cabo modificando los valores de los pesos de la red. Cada modelo de red neuronal cuenta con sus propias técnicas de aprendizaje.

Durante el proceso de aprendizaje, los pesos de las conexiones de la red sufren modificaciones, se puede decir que este proceso se ha terminado (la red ha aprendido) cuando los valores de los pesos dan una salida aceptable o permanecen estables. Es importante aclarar que tal vez para un problema dado la red neuronal no encuentre pesos que resuelvan el problema adecuadamente.

Un aspecto importante en este proceso es determinar los criterios que se siguen para cambiar el valor de los pesos de una red cuando se pretende que la red aprenda. Estos criterios determinan lo que se conoce como la regla de aprendizaje de la red. De forma general se consideran dos tipos de reglas conocidas con el nombre de aprendizaje supervisado y no supervisado[4].

En este trabajo el más importante es el aprendizaje supervisado, sin embargo se da la definición del segundo para tener la referencia.

El aprendizaje supervisado que es aquel donde se conoce la salida que se desea de la red. Así una vez que se le proporcionan a la red los datos de entrada se compara su salida con la salida esperada. Si hay diferencias se van ajustando los parámetros de la red.

El aprendizaje no supervisado es aquel donde no existe ningún tipo de guía. Lo único que puede hacer la red es reconocer patrones en los datos de entrada y crear categorías a partir de estos patrones. Así cuando se le proporcione algún dato, después del entrenamiento, la red será capaz de clasificarlo e indicar a qué categoría lo ha asignado.

Una forma de llevar a cabo el aprendizaje supervisado es por corrección de error, el cual consiste en ajustar los pesos de las conexiones de la red en función de la diferencia entre los valores deseados y los obtenidos en la salida de la red, es decir, la función de error cometido en la salida. Un algoritmo simple de aprendizaje por corrección de error es el siguiente:

$$\Delta W_{ji} = \alpha y_i (d_j - y_j)$$

donde W_{ji} es la variación en el peso de la conexión entre las neuronas i y j
α es el factor de aprendizaje, entre 0 y 1, que regula la velocidad del aprendizaje
y_i es el valor de entrada de la neurona i
d_j es el valor deseado para la neurona j, y
y_j es el valor obtenido en la salida de la neurona j

Un ejemplo de este tipo de algoritmo lo constituye la regla de aprendizaje del perceptron, sin embargo este algoritmo tiene algunas limitaciones como lo es el no considerar la magnitud del error global cometido durante el proceso completo de aprendizaje de la red, considerando únicamente los errores individuales correspondientes al aprendizaje de cada información por separado. Este algoritmo se aplica a redes sin capas ocultas donde se puede conocer el valor de la salida directamente[4].

Un algoritmo que permite un aprendizaje más rápido es el propuesto por Widrow y Hoff. En 1960 denominado regla delta o regla del mínimo error cuadrado. (Aplicable a arquitecturas sin capas ocultas)

Widrow y Hoff Definieron una función que permitía cuantificar el error global cometido en cualquier momento durante el proceso de entrenamiento de la red, lo cual es importante ya que entre más conocimiento se tenga del error cometido más rápido se puede aprender.

Este error se define como sigue:

$$Error_{global} = \frac{1}{2P}\sum_{k=1}^{P}\sum_{j=1}^{N}(y_j^{(k)} - d_j^{(k)})^2$$

donde

N es el número de neuronas de salida
P el número de datos que debe de aprender la red

$$\frac{1}{2}\sum_{j=1}^{N}(y_j^k - d_j^k)^2$$

error aprendido en el aprendizaje de la información k-ésima

Aquí de lo que se trata es encontrar los pesos para las conexiones de la red que minimicen la función de error. Para lograr esto, el ajuste de los pesos de las conexiones de la red se pueden hacer de forma proporcional a la variación relativa del error que se obtiene al variar el peso correspondiente:

$$\Delta W_{ij} = k(\frac{\partial Error_{global}}{\partial w_{ji}})$$

Mediante este proceso se llegan a obtener un conjunto de pesos con los que se consigue minimizar el error medio.

Otro algoritmo de aprendizaje mediante corrección de error lo constituye el denominado regla delta generalizada o algoritmo de retropropagación (error backpropagation). Este algoritmo se abordará en la siguiente sección.

Algoritmo de retropropagación

En 1986, Rumelhart, Hinton y Williams basándose en otros trabajos formalizaron un método para que una red neuronal aprendiera la asociación que existe entre los patrones de entrada a la misma y las clases correspondientes, utilizando más niveles de neuronas que las que utilizo Rosenblatt para desarrollar el perceptron. Este método es conocido como retropropagación (backpropagation, propagación del error hacia atrás) esta basado en la generalización de la regla delta.

El algoritmo de retropropagación (backpropagation) es una regla de aprendizaje que se puede aplicar en modelos de redes con más de dos capas de neuronas. Una característica importante de este algoritmo es la representación interna del conocimiento que es capaz de organizar en la capa intermedia de las neuronas para conseguir cualquier correspondencia entre la entrada y la salida de la red.

De forma simplificada el funcionamiento del algoritmo es el siguiente: primero se aplica un patrón de entrada como estímulo para la primera capa de las neuronas de la red, se va propagando a través de todas las capas superiores hasta generar una salida, se compara el resultado obtenido en las neuronas de salida con la salida que se desea obtener y se calcula un valor del error para cada neurona de salida. A continuación estos errores se transmiten hacia atrás partiendo de la capa de salida hacia todas las neuronas de la capa intermedia que contribuyan directamente a la salida, recibiendo el porcentaje de error aproximado a la partición de la neurona intermedia en la salida original. Este proceso se repite capa por capa, hasta que todas las neuronas de la red hayan recibido un error que describa su aportación relativa al error total. Basándose en el valor del error recibido, se reajustan los pesos de conexión de cada neurona, de manera que en la siguiente vez que se presente el mismo patrón, la salida este más cercana a la deseada, es decir, el error disminuya.

La importancia de este algoritmo consiste en su capacidad de autoadaptar los pesos de las neuronas de las capas intermedias para aprender la relación que existe entre un conjunto de patrones dados como ejemplos sus salidas correspondientes. Para poder aplicar esa misma relación, después del entrenamiento, a nuevos vectores de entrada con ruido o incompletas dando una salida activa si la nueva entrada es parecida a las presentadas durante el aprendizaje. Esta característica importante que se exige a los sistemas de aprendizaje, es la capacidad de generalización, entendida como la facilidad de dar salidas satisfactorias a entradas que el sistema no ha visto nunca en su fase de entrenamiento.

Regla delta generalizada

La regla propuesta por Widrow en 1960 (regla delta) ha sido extendida a redes con capas intermedias (regla delta generalizada) con conexiones hacia delante y cuyas neuronas tiene función de activación continuas (lineales o sigmoidales) dando lugar al algoritmo de retropropagación. Estas funciones continuas son no decrecientes y derivables, un ejemplo de esto son las funciones sigmoidales a diferencia de la función escalón que se utiliza en el perceptron ya que esta última no es derivable en el punto donde se encuentra la discontinuidad.

Este algoritmo utiliza también una función o superficie de error asociada a la red, buscando el estado estable de la mínima energía o de mínimo error a través del camino descendente de la superficie del error. Por ello, realimenta el error del sistema para modificar los pesos en un valor proporcional al gradiente decreciente de dicha función de error.

Funcionamiento del algoritmo de retropropagación

El método consiste en actualizar los pesos de forma proporcional a la delta, es decir, a la diferencia entre la salida deseada y la obtenida

Dada una neurona (U_i) y la salida que produce y_i, el cambio que se produce en el peso de la conexión que une la salida de dicha neurona con la unidad U_j (w_{ji}) para un patrón de aprendizaje p determinado es:

$$\Delta w_{ji} = \alpha \delta_{pj} y_{pj}$$

en donde el subíndice p se refiere al patrón de aprendizaje concreto y α es la constante o tasa de aprendizaje.

La diferencia entre la regla delta y la regla delta generalizada es en el valor concreto de $\delta_{pj.}$ Además en las redes multinivel, a diferencia de las redes sin neurona ocultas, en principio no se puede conocer la salida deseada de las neuronas de las capas ocultas para poder determinar los pesos en función del error cometido, sin embargo, inicialmente si se puede conocer la salida deseada, según esto si se considera la unidad U_j de salida, definiendo

$$\delta_{pj} = (d_{pj} - y_{pj}) f'(net_j)$$

donde δ_{pj} es la salida deseada de la neurona j para el patrón p y net_j es la entrada neta que recibe la neurona j. Esta fórmula es como la de la regla delta, excepto a la derivada de la función de transferencia. En caso de que U_j no sea una neurona de salida, el error que se produce esta en función del error que se comete en las neuronas que reciben como entrada la salida de U_j.

$$\delta_{pj} = (\sum_k \delta_{pk} w_{kj}) f'(net_j)$$

donde el rango de k cubre todas aquellas neuronas a las que esta conectada la salida de U_j, así el error que se produce en una neurona oculta es la suma de los errores que se producen en las neuronas a las que está conectada la salida de ésta, multiplicando cada uno de ellos por el peso de la conexión.

El nombre de éste es el método del gradiente descendente. Este método requiere un número importante de cálculos para lograr el ajuste de los pesos de la red. En la implementación se habló de la tasa de aprendizaje α, a mayor tasa mayo es la modificación de los pesos en cada iteración con lo que el aprendizaje será más rápido pero por otro lado puede dar lugar a oscilaciones. Rumelhart, Hinton y Williams (1986) sugirieron filtrar estas oscilaciones añadiendo un término β (momento) quedando la expresión como:

$$\Delta w_{ji}(t+1) = \alpha\delta_{pj} y_{pi} + \beta\Delta w_{ji}(t)$$

donde β es una constante (momento) que determina el efecto en t+1 del cambio de los pesos en el instante t.

Retomando la síntesis que se hizo anteriormente sobre el calculo de la salida de la red se hace el calculo de retropropagación:

1. Se inicializan los pesos de la red con valores pequeños aleatorios entre 0 y 1
2. Se proporciona un patrón de entrada en forma de vector $X_{p1}, X_{p2}, \ldots, X_{pn}$
3. Se calculan las entradas netas para las neuronas ocultas procedentes de la neurona de entrada.
   Para una neurona j oculta:

$$net_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h} x_{pi}$$

   donde el índice h se refiere a magnitudes de la capa oculta, el subíndice p, al p-ésimo vector de entrada, j a la j-ésima neurona oculta y N es el número de patrones de entrada.
4. Se calculan las salidas de las neuronas ocultas:

$$y_{pj} = f_j^{h}(net_{pj}^{h})$$

5. Se realizan los mismos cálculos para obtener las salidas de las neuronas de salida, (capa 0 es la de salida y L es el número de neuronas ocultas):

$$net_{pk}^{0} = \sum_{j=1}^{L} w_{kkj}^{0} y_{pj}$$

$$y_{pk} = f_k^{0}(net_{pk}^{0})$$

6. Cálculo del error para todas las neuronas
   Si la neurona k pertenece a la capa de salida el valor de la delta es:

$$\delta_{pk}^{0} = (d_{pk} - y_{pk}) f_k^{0'}(net_{pk}^{0})$$

   f debe ser derivable, por lo tanto no se puede utilizarla función escalón. Por lo general se aplica la función sigmoidal definida como sigue:

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

y su derivada queda definida como:

$$f_k^{0'} = f_k^0(1 - f_k^0) = y_{pk}(1 - y_{pk})$$

por lo que el término de error queda como sigue:

$$\delta_{pk}^0 = (d_{pk} - y_{pk})\, y_{pk}(1 - y_{pk})$$

en caso de que la neurona j no es de salida, entonces la derivada parcial del error no puede ser evaluada directamente, por lo tanto, se obtiene el desarrollo a partir de valores que son conocidos y otros que pueden ser evaluados. La expresión obtenida en este caso es:

$$\delta_{pj}^h = f_j^{h'}(net_{pj}^h) \sum \delta_{pk}^0 w_{kj}^0$$

donde el error en las capas ocultas depende de todos los términos de error de la capa de salida. De aquí el término de propagación hacia atrás. En particular para la función sigmoidal

$$\delta_{pj}^h = x_{pi}(1 - x_{pi}) \sum \delta_{pk}^0 w_{kj}^0$$

donde k se refiere a todas las neuronas de la capa superior a la de la neurona j.

7. Actualización de los pesos

El algoritmo va de las neuronas de salida hacia atrás hasta llegar a la capa de entrada.

Para los pesos de las neuronas de la capa de salida:

$$w_{kj}^0(t+1) = w_{kj}^0(t) + \Delta w_{kj}^0(t+1)$$

$$\Delta w_{kj}^0(t+1) = \alpha \delta_{pk}^0 y_{pj}$$

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

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \Delta w_{ji}^h(t+1)$$

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

8. El proceso se repite hasta que el término de error

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

Resulta aceptablemente pequeño para cada uno de los patrones aprendidos

Consideraciones sobre el algoritmo de retropropagación

Este algoritmo encuentra un valor mínimo de error (local o global) mediante la aplicación de pasos descendentes (gradiente descendente). Cada punto de la superficie de la función de error corresponde a un conjunto de valores de los pesos de la red. Con el gradiente descendente siempre que se realiza un cambio en todos los pesos de la red, se asegura el descenso por la superficie del error hasta encontrar el valle más cercano, lo que puede hacer que el proceso de aprendizaje se detenga en un mínimo local de error. Este es uno de los problemas de este algoritmo, ya que si cae en un mínimo local o en algún punto estacionario, no se llega a encontrar el mínimo global de la función de error, aunque a veces basta con alcanzar un error mínimo preestablecido.

Control de la convergencia

En las técnicas de gradiente descendente es conveniente avanzar por la superficie de error con incrementos pequeños de los pesos. Esto se debe a que existe una información local de la superficie y no se sabe lo lejos o lo cerca que se está del punto mínimo. Con incrementos grandes, se corre el riesgo de pasar por encima del punto mínimo sin conseguir estacionarse en el, con incrementos pequeños aunque se tarde más en llegar, se evita que ocurra esto. (Fig. 12)

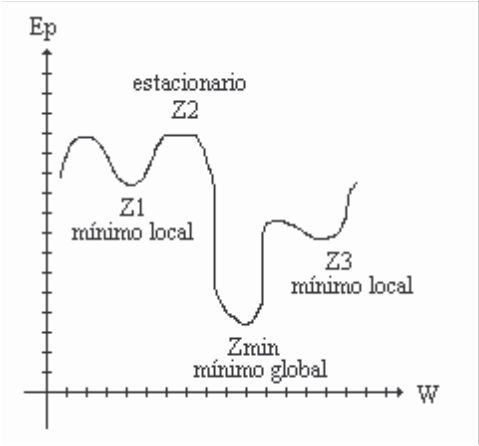


Fig. 12 Representación de una superficie de error donde w representa los posibles valores de los pesos de la red

La velocidad con que converge el algoritmo se controla a través de la constante de proporcionalidad o tasa de aprendizaje α, normalmente debe ser un número pequeño (del orden de 0.05 a 0.25) para asegurar que la red llegue a asentarse en una solución. Esto implica que la red debe hacer un gran número de iteraciones. Si esta constante es muy grande, los cambios de pesos son muy grandes, avanzando muy rápidamente por la superficie de error, con el riesgo de saltar el mínimo y estar oscilando alrededor de él pero sin poderlo alcanzar. Lo habitual es aumentar el valor de α a medida que disminuye el error

de la red durante la fase de aprendizaje, así se acelera la convergencia aunque sin llegar nunca a valores de α demasiado grandes, que hicieran que la red oscilase alejándose demasiado del valor mínimo.

Otro aspecto a tener en cuenta es que en el desarrollo matemático que se ha desarrollado para llegar al algoritmo de retropropagación no se asegura que el mínimo que se encuentre sea global. Una vez que la red se asienta en un mínimo, sea local o global, cesa el aprendizaje, aunque el error siga siendo demasiado alto. En caso de que se haya alcanzado un mínimo local, si la solución es admisible no importa que sea un mínimo local pero en caso contrario se realiza un cambio en el conjunto de pesos iniciales o en el número de nodos ocultos, otra opción es cambiar los parámetros de aprendizaje[4].

Aplicaciones reales de un perceptron (con retropropagación)

Las siguientes aplicaciones son tomadas de la cita [4].

1. Codificación de información

   La idea principal consiste en que la información de entrada se recupere en la salida a través de un código interno. Si esto se consigue, en la fase de funcionamiento sólo habrá que proporcionar a la red la información codificada, compuesta por los estados de activación de las neuronas ocultas y los pesos correspondientes obtenidos durante la etapa de aprendizaje para generar correctamente la salida deseada.

2. Traducción de texto en lenguaje hablado

   Aplicación desarrollada por Sejnowski y Rosenberg. Consiste en codificar en la capa de entrada cada una de las posibles letras como el contexto local, que consiste en las tres letras procedentes y posteriores a través de una ventana que se va moviendo por el texto secuencialmente. En la capa de salida se codifica la información correspondiente a la pronunciación de una serie de fonemas, más ciertas características que dan idea de sus vecinos. Con un número adecuado de unidades ocultas, la red establece la correspondencia texto-fonemas que permitirá, durante la fase de funcionamiento, generar automáticamente los fonemas correspondientes a un determinado texto.

3. Reconocimiento del lenguaje hablado

   Esta es una aplicación difícil de atacar, la dificultad radica en saber tratar correctamente la información temporal. Una de las propuestas es introducir retardos que muestren explícitamente la existencia de una ventana temporal en la captura de la información.

   Cada unidad oculta se encuentra conectada a cada unidad de la capa de entrada a través de diferentes conexiones, que dan cada una de ellas la idea de diferentes retardos. De esta forma se pretende generar respuestas similares a patrones de entrada similares, pero que están desplazados en el tiempo.

4. Aplicaciones en cardiología
   Clasificación de señales electrocardiográficas (ECG)

   Clasificación de electrocardiogramas con 12 derivaciones. En este problema las señales ECG se obtenían de una base de datos de arritmias creadas en la Universidad de Leuven. Las 3266 señales utilizadas correspondían a 7 tipos diferentes: normales, con hipertrofia ventricular izquierda (LVH), hipertrofia ventricular derecha (RVH), hipertrofia bi-ventricular(BVH), infarto de miocardio anterior (AMI), infarto de miocardio inferior (IMI) e infarto de miocardio combinado(MIX), de este conjunto 2446 señales seleccionadas aleatoriamente se utilizaron para el aprendizaje de la red y el resto (820) para comprobar posteriormente su funcionamiento. En este problema la base de datos utilizada por la red neuronal es preprocesada, parametrizando cada señal mediante un conjunto de 39 valores obtenidos por un programa comercial de análisis de ECG, 37 de ellos se obtienen de los complejos QRS y de las ondas T en diferentes derivaciones. A este conjunto de 37 valores se le añadían valores como el sexo y la edad del paciente. Número de salidas de la red, 7. En cuanto a resultados, expresados en porcentaje de aciertos de cada tipo de ECG, estuvieron entre el 87.9% para el caso de ECG normales y el 96.1% para ECG con infarto de miocardio anterior (AMI)

   Como la aplicación anterior se hicieron trabajos para la detección de ventriculares y supraventriculares, detección de complejos QRS anómalos, reconocimiento de formas anormales en señales ECG, etc.[4].

Referencias

[1] Mohamad H. Hassoun, “Fundamentals of artificial Neural Networks”, The MIT Press 1995
[2] Donald B. Stratton.Neurofisiología, Edit. Limusa,1984, pág. 25-28
[3] http://www-dse.doc.ic.ac.uk/~nd/surprice_96/journal/vol4/cs11/report.html
[4] José R. Hilera, Víctor J. Martínez. “Redes Neuronales Artificiales. Fundamentos, Modelos y Aplicaciones”. Edit. Addison-Weslwy Iberoamerica. 1995
[5] Notas no publicada de reconocimiento de patrones autor Dr. John Goddard
[6] R.A. Fisher, “The use of multiple measurements in taxonomic problems”, Annual Eugenics, 7, Part II, 1936, pp. 179-188
[7] Artturo Hernández Aguirre “Introducción a las Redes Neuronales Artificiales” Soluciones Avanzadas, Año 7. No. 63, Nov. 1998, pp 25-34
[8] Xin Yao “Evolving artificial Neural Networks” Proc. of the IEEE, 87(9); 1999, pp 1423-1447,
[9] Simon Haykin “Neural Networks” Macmillan, cap. 4, 1994, pp 106-113, 1994

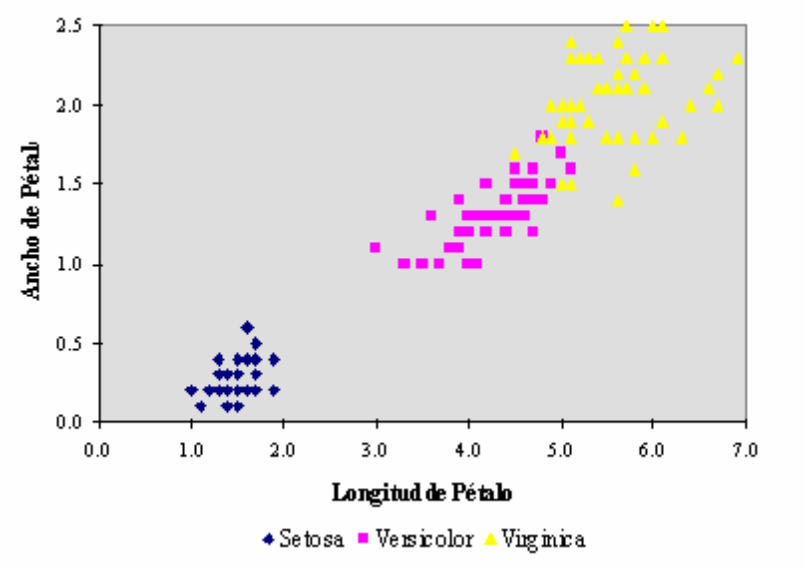
Apéndice B

DESCRIPCIÓN DE LAS BASES DE DATOS

En este capítulo se presenta la descripción de cinco bases de datos seleccionadas para observar el desempeño del programa evolutivo descrito en el capítulo anterior, estas bases de datos tienen diferentes grados de complejidad en la clasificación de sus datos además de tener diferente número de ejemplos, de atributos y clases. Dos de estas bases de datos fueron usadas como ejemplo en los problemas que se pueden presentar en las bases de datos seleccionados para un clasificador automático, descritos en el capítulo 1 del presente trabajo. En su descripción existen algunos atributos que no se tradujeron, tal es el caso de la base de datos de la clasificación de vinos.

1. Base de datos IRIS

Base de datos creada por R. A. Fisher y donada por Michael Marshall en julio de 1988[1]. Esta base de datos es quizá, la más conocida en la literatura de reconocimiento de patrones. Consta de 150 ejemplos, 3 clases y 4 atributos, la distribución de los ejemplos es, 50 ejemplos para cada clase. No se cuentan con datos perdidos. Cada clase se refiere a un tipo de planta iris, las cuales son las siguientes: iris setosa, iris versicolor e iris virginica. Una clase (iris setosa) es linealmente separable de las otras dos, lo cual se puede observar en la gráfica 1, sin embargo la clase iris versicolor e iris virginica cuentan con ejemplos traslapados evitando ser linealmente separables entre sí. Los atributos son el largo y ancho del sépalo y el largo y ancho del pétalo.



Gráfica 1

En [2] se reporta el 100% de clasificación de la clase iris setosa y muy pocos ejemplos mal clasificados para el resto de las clases.

2. Base de datos de los indios PIMA

Base de datos original del Instituto Nacional de Diabetes y Enfermedades del Riñón y Digestivas su donante: Vincent Sigillito, Applied Physics Laboratory, The Johns Hopkins University, su fecha de recepción es del 9 de mayo de 1990.[1]

En esta base de datos se determinaron algunas restricciones las cuales se mencionan a continuación, de la base de datos original se seleccionaron todos los pacientes del sexo femenino de al menos 21 años de edad y descendentes de la tribu de indios pima. Cuenta con 768 ejemplos y aunque no se reportan datos perdidos al examinar la base de datos se encuentran una gran cantidad de ejemplos con atributos sin valor, reduciendo la base de datos a 336 ejemplos, cuya distribución es la siguiente: 225 ejemplos corresponden a la clase 1 (prueba de diabetes positiva) y 111 a la clase 0 (prueba de diabetes negativa). El número de atributos es de 8 y cuenta con dos clases , como ya se había mencionado.

Los atributos son los siguientes:

1. Número de embarazos
2. Concentración de glucosa en plasma a dos horas en una prueba de tolerancia oral a la glucosa.
3. Presión sanguínea diastólica (mm Hg)
4. Medición del pliegue en el tríceps (mm)
5. Insulina en Serum 2 horas (mu U/ml)
6. Índice de masa corpórea (peso en Kg/(estatura en m)^2)
7. Función de pedigree de diabetes
8. Edad (años)

En [3] reportan, utilizando un algoritmo de aprendizaje denominado ADAP, el 76% de aciertos sobre los datos de prueba.

3. Desordenes Cardiacos

Esta base de datos es compilada originalmente en la Cleveland Clinic Foundation y suministrada por Robert Detran M.D., Ph. D. Del V. A. Medical Center, Long Beach CA, es parte de una colección de datos de la Universidad de California recolectada por David Aha[1]

El propósito de esta base de datos es predecir la presencia o ausencia de enfermedad cardiaca dados los resultados de varias pruebas médicas llevadas a cabo sobre el paciente. La base de datos posee 13 atributos que fueron extraídos de un conjunto mayor de 75. Los 13 atributos son los siguientes:

1. Edad
2. Sexo
3. Dolor de pecho (4 valores)
4. Presión sanguínea en reposos
5. Colesterol (mg/dl)
6. Nivel de azúcar en sangre > 120mg/dl (en ayunas)

7. Resultados electrocardiográficos en reposo (valores 0,1,2)
8. Máxima frecuencia cardiaca alcanzada
9. Angina inducida por ejercicio
10. Depresión del segmento ST inducida por ejercicio con respecto al reposo
11. Pendiente del segmento ST (pico del ejercicio)
12. No. de vasos mayores coloreados por fluoroscopia (0-3)
13. Thal.:3=normal, 6=defecto fijo, 7=defecto reversible

El conjunto de datos original contenía 303 ejemplos pero algunos fueron descartados quedando tan solo 270, existen dos clases: presencia o ausencia de enfermedad cardiaca. La distribución de ejemplos de cada clase es la siguiente: 150 ejemplos son sanos lo cual corresponde al 55.56% del total y 120 ejemplos corresponden a la clase de enfermos lo cual equivale a un 44.44% del total.

Los mejores resultados publicados en [4] sobre esta base de datos es del 81% de aciertos sobre los datos de prueba utilizando un algoritmo hibrido (algoritmo genético y retropropagación) y del 77.5% utilizando únicamente un algoritmo genético.

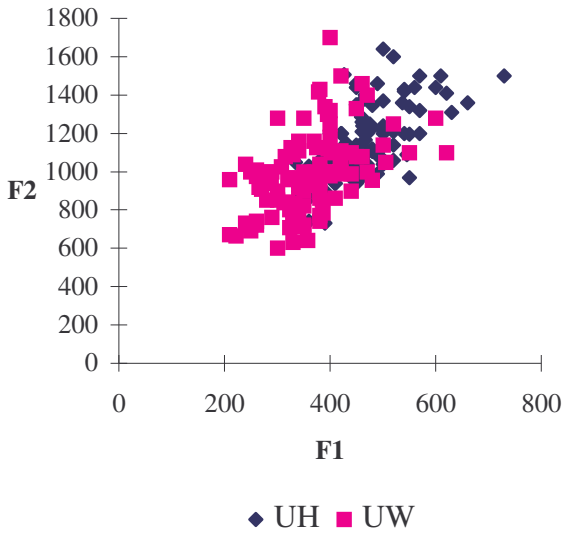
4. Base de datos Peterson Barney

Peterson Barney es una base de datos que reporta las cuatro primeras formantes generadas por la señal de la voz de 33 hombres, 28 mujeres y 15 niños, teniendo un total de 76 personas produciendo dos repeticiones de 10 vocales obteniendo así 1520 muestras de voz en el idioma inglés[5].

Las vocales generadas son las siguientes:

1	IY	[i]
2	IH	[I]
3	EH	[e]
4	AE	[ae]
5	AH	[^]
6	AA	[a]
7	AO	[o]
8	UH	[U]
9	UW	[u]
10	ER	[3]

Existen ejemplos en la base de datos donde los expertos no logran identificar a que vocal pertenecen por lo tanto estos han sido eliminados, para probar el algoritmo propuesto se escogieron los ejemplos de las dos vocales más difíciles de clasificar: UH y UW debido al gran traslape de sus datos, en la gráfica 2 se muestra su distribución y se decidió eliminar el primer atributo ya que se considera F1, F2 y F3 las formantes con mayor información.



Del total de ejemplos de la base de datos original se tomaron 275, de los cuales 132 corresponden a la clase UH y 143 a la clase UW

5. Localización de proteínas

Esta base de datos es creada por Kenta Nakai del Institue of Molecular and Cellular Biology Osaka, University y donada por Paul Horton en 1996[1]. Consta de 336 ejemplos y ocho atributos, los cuales se mencionan a continuación:

1. Sequence Name: Accession number for the SWISS-PROT database
2. mcg: McGeoch's method for signal sequence recognition.
3. gvh: von Heijne's method for signal sequence recognition.
4. lip: von Heijne's Signal Peptidase II consensus sequence score. Binary attribute.
5. chg: Presence of charge on N-terminus of predicted lipoproteins. Binary attribute.
6. aac: score of discriminant analysis of the amino acid content of outer membrane and periplasmic proteins.
7. alm1: score of the ALOM membrane spanning region prediction program.
8. alm2: score of ALOM program after excluding putative cleavable signal regions from the sequence.

No existen valores perdidos. Las clases son el sitio de localización de proteínas, son ocho que a continuación se mencionan con la distribución de los ejemplos:

cp (cytoplasm)	143
im (inner membrane without signal sequence)	77
pp (perisplasm)	52
imU (inner membrane, uncleavable signal sequence)	35
om (outer membrane)	20
omL (outer membrane lipoprotein)	5

imL (inner membrane lipoprotein)	2
imS (inner membrane, cleavable signal sequence)	2

En [6] se reporta un 81% de aciertos con un modelo probabilístico y un porcentaje similar con un árbol de desición binario.

6. Reconocimiento de vino

Creada por el Instituto de Farmaceutica y Análisis y Tecnología de la Comida en Italia y dada de alta en 1998 por C. Blake[1]. Estos datos son los resultados de un análisis químico de vinos de la misma región de Italia pero derivado de diferentes cultivos de árboles, el análisis determina 13 componentes encontrados en cada uno de los tipos de árboles de vinos.

Los atributos son estos 13 componentes y se enlistan a continuación:

1) Alcohol
2) Malic acid
3) ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10)Color intensity
11)Hue
12)OD280/OD315 of diluted wines
13)Proline

Existen 3 clases cuyo nombre no es reportado y su distribución es la siguiente:

Clase 1	59 ejemplos
Clase 2	71 ejemplos
Clase 3	48 ejemplos

En total 178 ejemplos con ningún valor perdido

Aplicando varios algoritmos para su clasificación se reportan los siguientes resultados[7]:

Algoritmo	Porcentaje de aciertos
RDA	100%
QDA	99.4%
LDA	98.9%
INN	96.1%

Referencias

[1] UCI Repository of Machine Learning Databases and Domian Theories, http://www.ics.uci.edu.pub/machine-learning-databases

[2] Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No.1,67-71.

[3] Smith, J.W., Everhart J.E., Dickson W.C., Knowler W.C., Johannes R.S., Usan el algoritmo de aprendizaje ADAP para diagnósticar la diabtes mellitus, proceedings of the Symposium of computer applications and medical care, pp. 261-265, 1988.

[4] John Goddard, Irineo López, Leonardo Rufiner, "Diagnóstico de cardiopatías mediante redes neuronales y algoritmos genéticos", Revista Argentina de Bioingeniería, Vol.2, No.2, 11-19, 1996. (trabajo donde se reportan los resultados para clasificación de la base de datos de los desordenes cardiacos)

[5] http://www.svr-ftp.eng.cam.ac.uk/pub/comp.speech/data/

[6] Paul Horton & Kenta Nakai,"A Probablistic Classification System for Predicting the Cellular Localization Sites of Proteins", Intelligent Systems in Molecular Biology, 109-115. St. Louis, USA 1996.
(trabajo donde se reportan los resultados para clasificación de la base de datos de localizac. de proteínas)

[7] S. Aeberhard, D. Coomans and O. de Vel, Comparison of Classifiers in High Dimensional Settings, Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland.
(trabajo donde se reportan los resultados para clasificación de la base de datos de vinos)

Apéndice C

En este apéndice se reportan los pesos encontrados por el programa evolutivo con cromosomas de longitud variable para cada una de las bases de datos trabajadas.

Los pesos se interpretan de la siguiente manera, para cada nodo oculto los primeros n pesos corresponden a las conexiones de las entradas hacia ese nodo oculto y los siguientes m pesos corresponden a las conexiones del nodo oculto en cuestión hacia la salida, siempre de izquierda a derecha. Esto corresponde a la forma de codificación explicada en el capítulo 5. El primer peso siempre corresponde al del sesgo de la capa de entrada, a parte se reportan los pesos del sesgo de la capa oculta hacia la salida. En la figura 1 se muestra la decodificación únicamente del primer nodo de la base iris, y se representa la arquitectura general encontrada como solución para este problema, de forma similar se decodifican el resto de las arquitecturas.

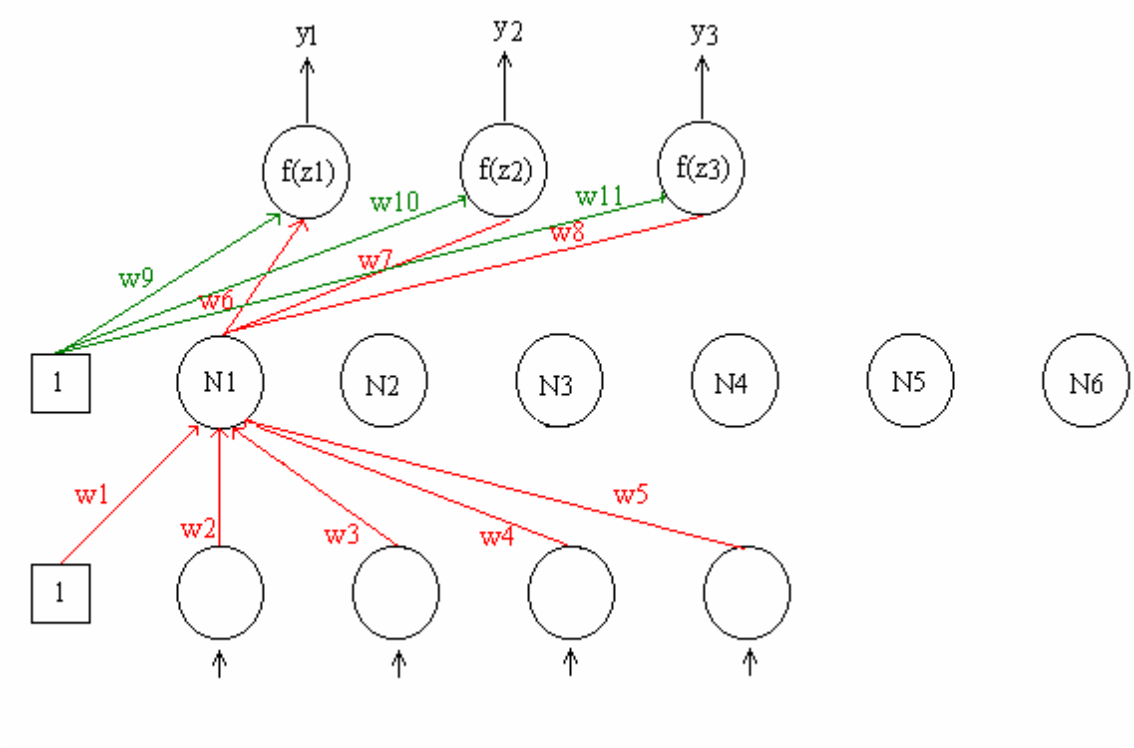


Fig. 1 Arquitectura para la base de datos iris, solo se decodifica el primer nodo, la decodificación de los demás es análogo

	Nodo 1		
W1	-2.803472	W9	321400
W2	-4.142658	W10	0.2058
W3	-0.002114	W11	0.27038
W4	4.574587		
W5	3.871881		
W6	-4.72618		
W7	-3.773201		
W8	3.619206		

Base de datos Iris

Nodos de entrada:5 (incluyendo el sesgo)
Nodos ocultos: 6
Nodos de salida:3
Para cada nodo oculto:
Número de pesos de las conexiones de la capa de entrada a la capa oculta: 5
Número de pesos de las conexiones de la capa oculta a los nodos de salida: 3

Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5	Nodo 6
-2.803472	-3.759826	-3.898866	-0.727554	2.329354	0.153693
-4.142658	-2.037033	1.824679	-0.754161	0.79104	0.569986
-0.002114	-5.765961	1.602906	0.5695	-0.457153	0.068214
4.574587	-2.786708	-2.415697	1.560327	0.410847	-0.640247
3.871881	2.358953	-4.454161	-1.02298	0.523307	0.016966
-4.72618	-7.063549	2.237718	-2.381833	-0.9519	1.0051
-3.773201	1.26332	-6.852508	0.73182	1.135246	0.701587
3.619206	-3.9044	-5.21966	-0.92388	-0.47304	-0.71438

Pesos de la conexión del sesgo de la capa oculta hacia los nodos de salida

0.321400
0.2058
0.27038

Base de datos de los indios Pima

Nodos de entrada:9 (incluyendo el sesgo)
Nodos ocultos: 5
Nodos de salida:1
Para cada nodo oculto:
Número de pesos de las conexiones de la capa de entrada a la capa oculta: 9
Número de pesos de las conexiones de la capa oculta a los nodos de salida: 1

Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
7.132045	-1.88682	-2.1594	-1.11525	-0.010255
0.267658	-1.933374	-0.550284	-0.573509	0.261635
-7.306608	0.70092	1.398715	-1.3873	-0.10942
-1.250681	-6.347713	-1.077776	-0.542195	-0.19345
-1.517729	-2.690041	2.14395	-0.087905	-0.226505
-2.090544	-0.78393	1.72117	-0.11556	0.162275
-3.063506	0.29928	-1.924669	-1.04499	0.171135
-2.575536	-6.530955	-3.278049	0.754225	0.01476
-4.265496	-4.348335	1.815847	-0.05576	0.051005
4.481837	0.029544	-1.24877	0.44877	0.10485

El peso del sesgo de la capa oculta a la capa de salida es 0.712405

Base de datos Desordenes Cardiacos

Nodos de entrada:14 (incluyendo el sesgo)
Nodos ocultos: 9
Nodos de salida:1
Para cada nodo oculto:
Número de pesos de las conexiones de la capa de entrada a la capa oculta: 14
Número de pesos de las conexiones de la capa oculta a los nodos de salida: 1

Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5	Nodo 6	Nodo 7
1.462346	1.100286	-2.833372	-0.3391	0.416353	0.473593	-0.736867
3.829008	-1.637532	-0.477413	0.9703	-0.204833	-1.115233	-0.457427
1.948386	-3.962861	-0.233754	1.51062	-0.25286	-1.078094	0.007127
2.176633	-2.668633	-0.6142	0.102206	0.232173	1.037627	0.411207
2.275071	-2.744172	3.167238	-1.038881	-0.971753	0.065473	-0.70088
4.150582	-0.348614	-3.658695	-0.99776	1.0479	0.061947	0.7688
-3.674741	0.364698	2.095432	-0.001213	1.179474	0.264247	-0.122754
3.904228	2.364187	0.860195	2.085766	0.832267	-0.117706	0.064647
-3.417788	-0.777706	0.852298	0.055486	0.07426	-0.318953	-0.0595
1.9633	-5.34878	-1.51828	1.046328	-0.09728	0.185353	-1.448487
5.494993	3.474267	-0.204727	0.865273	0.098474	1.985914	0.258547
-1.769113	-6.028668	-0.787114	1.07078	0.755467	1.236553	-0.460594
1.188173	-3.885943	-1.062387	0.375901	-0.149526	0.30072	-0.665927
-0.027027	-2.676354	-4.777301	-1.071914	-0.58126	0.77554	-0.90894
2.940207	-1.31968	-1.244434	-0.606979	0.038773	1.109521	0.02868

Nodo 8	Nodo 9
0.31518	0.051367
-0.377146	0.595307
0.695907	0.209773
-0.68342	-0.090553
-0.681187	0.159707
0.25406	-0.65974
-0.34436	-0.45004
-0.106687	-0.082073
-0.1342	-0.394713
0.165367	-0.193033
-0.039933	-0.3992
-0.54568	-0.593787
-0.332927	-0.058267
0.383107	0.41882
-0.602767	0.324593

El peso del sesgo de la capa oculta a la capa de salida es 0.011600

Base de datos Peterson Barney

Nodos de entrada:4 (incluyendo el sesgo)
Nodos ocultos: 20
Nodos de salida:1
Para cada nodo oculto:
Número de pesos de las conexiones de la capa de entrada a la capa oculta: 4
Número de pesos de las conexiones de la capa oculta a los nodos de salida: 1

Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
-2.954983	-8.249058	5.295592	2.760254	2.817629
13.094953	5.17693	-7.797122	-10.34163	1.639577
0.69833	-2.763871	-1.083762	-0.610403	-5.132427
-4.137764	-9.173223	-4.042999	0.65113	-10.37666
7.007998	-6.967889	-5.041055	-10.85319	5.693135

Nodo 6	Nodo 7	Nodo 8	Nodo 9	Nodo 10
0.046805	2.801786	-0.893359	0.06828	-0.707582
1.669753	-1.719796	-0.540309	-1.406446	0.370873
-1.12407	-0.740026	3.265029	0.274756	-2.025125
2.239771	1.986386	2.898828	-3.278168	-1.911682
6.095649	3.926741	-1.484195	-0.13939	-0.440401

Nodo 11	Nodo 12	Nodo 13	Nodo 14	Nodo 15
-2.777118	1.114661	0.400386	1.025658	-0.709866
-0.103773	1.083836	-0.043962	-0.33805	-0.161114
0.644222	-1.675326	-0.075262	0.472574	0.776176
1.287166	0.654956	-0.565684	-0.810664	-0.116416
-1.306878	0.797862	0.571794	0.420694	1.271714

Nodo 16	Nodo 17	Nodo 18	Nodo 19	Nodo 20
0.30691	-0.442784	0.024294	-0.06833	-0.031002
0.095402	-0.185754	-0.156912	0.063974	0.01877
-0.024826	-0.06213	-0.082132	-0.13486	-0.03002
0.317146	0.382924	-0.336688	0.10047	0.062328
-0.868742	0.720464	0.084126	0.18457	0.00591

El peso del sesgo de la capa oculta a la capa de salida es 0.7268

Base de datos de localización de proteínas

Nodos de entrada: 8 (incluyendo el sesgo)
Nodos ocultos: 31
Nodos de salida: 8
Para cada nodo oculto:
Número de pesos de las conexiones de la capa de entrada a la capa oculta: 9
Número de pesos de las conexiones de la capa oculta a los nodos de salida: 8

Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
-4.045177	-6.367445	1.641274	-3.433531	4.178276
10.671314	-0.163036	-1.378017	-0.945695	7.930146
-1.794437	-1.019344	-5.261369	-2.625035	-3.094093
-3.450366	8.179777	1.049792	-9.607882	0.351904
-2.581147	-3.438052	-0.657587	1.409904	-0.874794
-5.036123	0.35259	-4.682941	-4.727633	3.825422
-4.091053	-5.64064	-12.421911	0.035756	5.257163
-5.546011	-10.389609	1.019131	-4.337109	-7.351027
-1.540256	-4.085409	-2.404171	-1.145891	4.815328

-1.881886	-3.001482	-5.897619	2.859957	-6.836909
-5.938709	-3.203846	-3.295431	-4.603006	0.495177
-7.065669	2.288863	-1.405517	-9.500401	-4.339987
2.252645	2.507818	-3.857962	1.770101	-10.331827
1.183308	-4.362706	-5.029772	-5.227487	3.214341
1.114228	-11.176922	-1.553609	-3.846415	-8.49804
-3.611133	-6.642243	-1.323242	2.631822	3.551621

Nodo 6	**Nodo 7**	**Nodo 8**	**Nodo 9**	**Nodo 10**
2.512238	-6.249693	-2.322999	-3.890921	-2.791505
-0.433657	4.713761	-1.669668	-4.015954	-3.802136
-2.496623	-2.711807	-3.458484	1.135676	-4.315002
-0.201904	-3.499269	-0.015201	-0.086786	3.843618
-2.10815	0.828302	-0.417167	1.484145	5.080608
2.279803	-1.332875	0.402068	-5.339017	-2.083951
-3.954026	-1.236199	-2.03844	0.00655	4.648998
1.553974	-2.322091	6.118194	7.31258	3.018189
1.525437	-1.79095	-1.212579	-7.88263	0.885418
-2.464098	6.874563	-0.233773	2.18492	4.896364
-5.049983	1.8636	-1.935008	-4.199197	5.495436
3.666361	-1.877372	-2.775074	-7.662992	-1.945238
-1.164563	4.966095	3.223929	-2.140441	-3.016746
0.863387	4.63904	-1.206134	3.421865	-11.217728
0.641225	-8.020555	-3.112647	4.592627	1.465257
-3.414071	-0.514308	2.864413	0.146637	-7.358183

Nodo 11	**Nodo 12**	**Nodo 13**	**Nodo 14**	**Nodo 15**
5.840256	-6.778232	2.226393	0.171159	-2.223198
-0.819992	-3.046351	-1.92069	-1.382378	6.653583
0.316114	3.784401	-4.837162	-7.224502	-2.943504
-1.678834	1.826968	-5.009257	2.581071	6.414998
-2.482988	-1.785758	-2.14501	-5.937152	-0.061621
-1.135954	3.009918	1.885827	4.196453	5.893936
-6.186137	0.449911	2.149702	-2.606296	2.075333
-1.188253	-1.88541	-3.172658	-2.224495	-1.533311
5.513786	1.375055	0.635027	3.550178	-6.558541
-2.784051	-2.634446	-2.775178	-3.506593	2.964402
-4.944868	-1.516105	-9.003262	7.25738	-7.830214
-3.163521	-0.41648	-10.218345	-3.025928	5.636934
0.993281	-2.195451	-1.502168	1.390383	-5.366062
3.232028	2.428855	-0.243235	-2.432107	-2.192477
-0.509984	-12.839687	0.080221	-2.337	-3.581681
-5.675581	5.417643	-3.958115	7.635646	2.661289

Nodo 16	Nodo 17	Nodo 18	Nodo 19	Nodo 20
-8.86807	5.045478	-2.685512	1.244315	-2.015331
1.614479	0.050548	5.589473	0.303405	-2.009769
2.135399	-2.041337	0.919944	-0.51143	-1.746739
-3.400466	-2.947779	-0.114164	6.180372	2.371116
0.24641	5.017127	1.883586	-4.738473	1.841311
2.818425	5.393669	-0.135417	-1.188964	-1.763199
-3.576425	-1.230783	-0.099422	-1.25736	-2.565209
-5.716401	3.249066	-5.34985	-0.652637	2.220121
-2.373177	-0.3599	-3.844966	1.73434	-1.797246
4.182332	1.981825	-1.329861	-1.224541	0.490752
-0.590246	0.520633	-3.110537	-3.355958	-0.767298
-5.643219	-2.662274	-7.690903	3.22749	-2.22076
2.671107	0.279053	-4.177223	3.269645	4.098105
-4.756637	-0.590823	-1.36236	-3.583861	-0.206322
2.181775	1.601687	0.352658	1.265664	-0.978893
1.034556	-0.687041	1.963703	-1.938904	-1.479712

Nodo 21	Nodo 22	Nodo 23	Nodo 24	Nodo 25
-5.508865	-1.535376	-2.522858	-2.322616	-1.130224
2.195896	-1.140209	0.956515	-1.032562	-1.057204
-1.107161	2.226825	-2.325681	-2.449237	-0.916574
0.027154	-2.568878	0.220281	0.491129	0.32203
1.274833	3.401049	0.982132	-1.518437	-0.39427
-1.343512	-1.003309	0.288174	-1.43474	0.53371
-0.985029	-2.318475	1.410056	-0.426302	-0.163273
0.657069	1.183155	-3.187943	-1.200688	-0.948288
0.062627	-0.084063	-3.237225	-0.439498	-0.47432
1.128036	-0.811778	0.597692	-0.333699	-1.623264
0.454904	-0.482424	2.141457	1.633067	0.003965
-3.358061	-5.296992	3.506518	-1.138666	0.751692
-4.234402	2.047487	-2.034631	-0.242816	-0.058967
1.169044	-5.017359	-2.021988	-3.097697	0.755388
1.537784	-0.639838	0.649102	-1.550351	-0.696155
0.638613	-0.951984	-2.203158	0.76339	0.254461

Nodo 26	Nodo 27	Nodo 28	Nodo 29	Nodo 30
0.477546	-1.594928	-0.184466	-0.315252	0.008132
-0.321394	-0.438484	-0.219942	-0.876728	-0.089032
-0.412146	-1.025818	0.304652	-0.099686	-0.041354
-0.123228	-0.572932	-0.020154	0.043884	-0.10561
0.483552	-1.004192	-0.217232	0.250496	0.033736
0.10984	-0.03372	-0.0191	-0.1831	0.245226
0.189802	-0.38145	-0.219794	-0.172714	0.022788

-0.33262	-0.990685	-0.194344	0.014992	-0.116118
-1.153764	0.207448	-0.163546	0.131174	-0.187594
0.068178	-1.304386	0.36832	0.16474	-0.060482
-1.437648	-0.580684	0.021844	-0.303906	0.164436
-1.197514	-0.391512	-0.564064	-0.237194	-0.140272
-0.339738	1.10686	-0.518408	-0.026268	0.22585
0.860904	0.31959	-0.15598	-0.225086	-0.185836
-1.156112	0.938312	-0.188622	0.274154	0.040552
0.396996	-0.93579	-0.859758	-0.07668	0.091186

Nodo 31
0.017972
-0.029958
-0.060234
-0.053068
0.117322
0.04494
-0.092242
0.067498
0.020152
-0.065846
0.076546
-0.127954
0.06275
-0.011532
0.064046
0.036554

Los pesos del sesgo de la capa oculta a los nodos de salida son:

0.00588
0.5444
0.006528
0.00166
0.00042
0.001716
0.006068
0.000612

Base de datos de vinos

Nodos de entrada: 14 (incluyendo el sesgo)
Nodos ocultos: 18
Nodos de salida: 3
Para cada nodo oculto:
Número de pesos de las conexiones de la capa de entrada a la capa oculta: 14

Número de pesos de las conexiones de la capa oculta a los nodos de salida: 3

Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5	Nodo 6
-1.833903	-4.129988	-4.454063	2.71279	4.087116	1.604218
2.459468	0.57424	-7.677393	-6.573143	-6.019753	-0.722756
4.770983	5.457274	-0.828553	-3.812144	5.302668	-3.687256
-4.890325	-3.310554	1.420853	-3.268992	-4.856759	0.322215
3.153761	0.985095	8.611248	0.089168	4.024314	1.677554
3.225711	3.414874	1.263798	-4.518054	-8.779606	3.336666
0.569813	-5.560907	3.977304	-4.703702	0.421614	1.017527
-12.086696	-4.212113	-7.278929	-3.411095	2.378889	-0.37743
2.25753	2.17555	-4.240839	1.110022	2.89876	-2.421866
-2.137065	8.775921	0.390978	4.620545	-7.248014	1.860099
5.957213	-5.19061	-3.886651	-10.517247	-3.029212	-0.290877
-2.205524	1.255491	0.367229	-7.43789	6.2462	0.095712
-3.736031	-1.891706	2.59369	-1.990524	-5.667535	-2.021303
-3.08501	-6.759221	2.288139	-1.0748	2.664367	2.141391
-3.86734	-6.647645	0.368124	1.533911	-4.95198	0.374095
-9.376386	-0.142118	4.801907	1.624421	4.017991	-2.339068
8.061195	0.153311	-6.397236	-0.91579	-4.479234	2.190206

Nodo 7	Nodo 8	Nodo 9	Nodo 10	Nodo 11	Nodo 12
-2.402919	-1.550895	2.277928	-0.247297	3.974254	0.626263
2.896895	2.750978	0.256505	-1.99102	-2.459532	1.637475
-0.003926	0.612896	-1.998136	-0.004072	-0.778948	-2.305583
-4.857866	2.132132	0.766564	0.37986	-1.102654	2.683395
-0.03555	-4.77517	0.608582	1.781508	-1.065277	0.862514
-3.098255	-2.311124	-2.616757	0.230425	0.258596	3.586489
-2.164919	5.982512	-1.039496	-2.84265	0.75322	0.195049
-1.075211	2.218063	1.986836	-1.438192	-0.383889	-1.552188
-2.768298	-1.521558	-3.062192	-1.733629	0.605931	0.822217
-0.604734	-0.435213	-1.351784	-1.802683	0.98526	0.572303
-3.068629	0.012712	-2.408409	-1.217022	-2.114582	2.367108
-0.97721	3.917924	0.97196	2.928789	-0.905855	-0.318589
1.319789	-1.352869	2.397195	2.178206	-3.013656	-1.662075
-7.120017	-2.878069	-7.16025	-1.634891	-0.820945	-0.713117
-3.304145	3.877412	-4.322334	0.484116	0.990019	-1.574248
6.90557	-2.44008	1.556806	1.705011	3.400909	1.350299
4.507361	-3.477863	-0.246157	-2.755772	2.320228	-1.612011

Nodo 13	Nodo 14	Nodo 15	Nodo 16	Nodo 17	Nodo 18
-0.24218	-1.402421	0.284046	0.078628	-0.165871	-0.190583
-0.002138	0.532683	-0.694985	-0.018406	-0.061814	-0.204889
-0.242545	-0.404937	0.296174	-0.224032	0.004766	0.251814
-0.193885	-0.983052	-1.054458	0.690897	-0.358734	-0.158471
-1.859563	-1.202757	-0.804712	0.654095	0.037426	0.099903
-1.290455	-0.325108	0.108908	0.422557	-0.02186	-0.000274
-1.23588	1.258537	-0.54786	-0.130717	-0.063866	0.262189
0.783292	-1.391237	-0.687449	-0.144206	0.035691	0.260631
-1.893263	1.395583	0.118946	-0.16738	0.39612	-0.036431
-2.799823	0.015817	-0.036438	0.16	0.162951	0.279809
-0.121577	-0.958331	0.138155	0.080754	0.062031	0.156228
0.388377	1.064008	-1.399252	-0.210465	0.227311	-0.021729
1.21354	-0.528355	-0.059352	-0.309214	0.0098	-0.125543
0.968216	0.104046	-0.336306	0.392383	0.11898	-0.230017
-0.015203	-0.984186	-0.540954	0.139617	-0.156251	-0.118769
-2.160321	-0.515972	0.985312	0.635251	-0.250406	-0.414769
-1.351846	-1.109266	-1.54918	-0.419686	-0.060654	0.15134

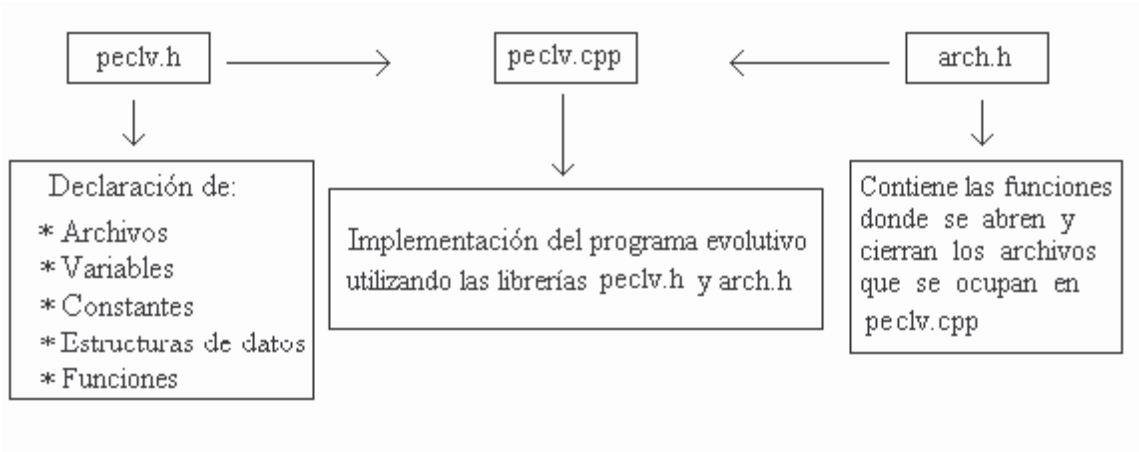
Los pesos del sesgo de la capa oculta a los nodos de salida son:

0.21189
0.16480
0.33531

APENDICE D

Este apéndice contiene la descripción general del programa evolutivo así como el listado del mismo

Estructura general del Programa Evolutivo con Cromosomas de Longitud Variable (peclv):



Archivo peclv.h :

En este archivo se encuentra la siguiente información:

1. Declaración de los archivos que se usan durante todo el programa evolutivo, así como sus respectivos apuntadores.

Dentro de estos archivos están, los que contienen información necesaria para la ejecución del programa y los que se generan durante dicha ejecución. A continuación se describen estos archivos.

Archivos de entrada:

Son dos: **InputFile** que contiene los archivos de entrenamiento y **Datprue** que contiene los archivos de prueba.

Archivos que se generan durante la ejecución del programa:

reporte.dat, archivo que almacena un encabezado con información general: tamaño de la población, número máximo de generaciones, tamaño máximo de cromosoma, etc e información más detallada como lo es: número de generación, fitness, número de nodos ocultos del mejor individuo de la generación en cuestión, los pesos encontrados del mejor individuo, etc.

sal.dat archivo que contiene las salidas del mejor individuo con los datos de entrenamiento

salp.dat archivo que contiene las salidas del mejor individuo con los datos de prueba

2. Declaración de variables

Las variables que se declaran en esta sección son las siguientes:

```
int    fitness_aciertos=0;      Variables que contabilizan aciertos y errores
int    fitness_error=0;
int    sesgo=1;                 Variable que representa al sesgo
float  sigma=0.5;               Variable que se utiliza al hacer la mutación en la función randval1
int conta=1,cont=0;             Variables que se utilizan solo para escribir las salidas del úlltimo
                                individuo
float err,errp;                       Variables donde se almacena el error encontrado, err es para
                                los datos de entrenamiento y errp almacena el error para los datos
                                de prueba
float  PELIMIN= 0.5;            Probabilidad de eliminacion de un nodo 1
float  PADIC=0.8;               Probabilidad de adicionar un nodo  0.4
int       k;                    Contador que indica que individuo de la población se esta
                                evaluando
float     generacion;           Contador de las generaciones
int       el_mejor;             Indice del mejor individuo de la población
```

Definición de vectores:

```
float     Vector[NENT+1];                   Tiene la información de los atributos
float     vector_pesos[(NSAL+NENT)*MaxSize]; Guarda los pesos para calcular cada net
float     SalidaDeseada[NSAL];              Tiene la clase de cada ejemplo en cuetión
float     f_transf[5*MaxSize];                    Función de transferencia del perceptron

float     Salida[NSAL];                     Guarda la salida rela de la red
float     Dif[NSAL];                        Diferencia entre la salida deseada y la real
```

3. Definición de constantes

```
#define  NENT         8      Nodos de entrada, depende de la base de datos que se este
                              utilizando para el entrenamiento y prueba del programa, no
                             incluye el sesgo
#define  NSAL         1      Nodos de salida, depende de la base de datos que se este utilizando
#define  MaxTam       20     Máximo tamaño de un cromosoma, número de nodos ocultos
#define  TAMPOB       20     Tamaño de la población
#define  MAXGENS      250    Número de generaciones
#define  PMUTNODO 0.8        Probabilidad de mutación de un nodo
```

4. Definición de estructuras de datos

```
struct Nodo{
            float   peso[NENT+1+NSAL];
            Nodo*   Sig_vert;
           };

struct Genotype{
                float   sesgo[NSAL];
                int     nmax;                   Dimensión máxima de los cromosomas
                float   fitness;                Fitness de cada individuo
                float   rfitness;               Fitness relativo
                float   cfitness;               Fitness acumulado
                Nodo*   crom;                   Apuntador al cromosoma
               };
struct Genotype población[TAMPOB+1];
struct Genotype nuevapoblación[TAMPOB+1];  Se utiliza cuando se aplica selección para
                                           guardar a los individuos que sobreviven de una
                                           generación a otra y luego copiarlos al vector
                                           población para seguir con el programa
```

5. Declaración de funciones

Son 33 funciones declaradas en peclv.h usadas en peclv.cpp, las cuales se describirán en la sección *archivo peclv.cpp* y se encuentran al final del apendice, en el listado del programa.

Archivo arch.h

Contiene dos funciones, una que abre los archivos declarados en peclv.h y otra función que los cierra al terminar el programa.

Archivo peclv.cpp

Descripción del programa principal:

1. Se inicializa generacion = 0, el cual es el contador que lleva el control del número de veces que se van a aplicar los operadores genéticos a la población.
2. Se abren los archivos descritos en la sección peclv.h
3. Se llama a la función Inicializa, quien a su vez manda llamar a la función Crea_lista para crear la estructura descrita en la figura 1.En este proceso se inicializan los pesos entre –1 y 1 y los nodos ocultos entre 0 y máximo tamaño de cromosoma de forma aleatoria. La información que guarda cada gen o nodo del cromosoma son los pesos de entrada y los de salida del nodo culto que representan.
4. Se llama a la función Evaluared, esta función obtiene una evaluación de cada individuo por medio de una función de fitness. Esta función fitness indica que tan eficiente es el individuo para la solución del problema a clasificar que se le esta presentando.

5. Se llama a la función Guarda_el_mejor. La población esta formada por un conjunto de individuos, cada individuo representa una red neuronal perceptron multicapa y es evaluada por medio de una función llamada fitness, el individuo con mejor fitness, es considerado el mejor y es guardado en la última posición del vector población por medio de la función Guarda_el_mejor.
6. Se empieza a generar el archivo reporte.dat con la función reporte, esta función guarda los datos del mejor individuo de la generación 0.
7. Se define un cilo while de la siguiente manera:
   While (generacion<MAXGENS-1)
   {
   generacion=generacion +1
   8. Se manda llamar a la función Seleccion, quien simula el proceso de una ruleta, la forma como lo hace es la siguiente:
      a. calcula el fitness total de la población, sumando el fitness de cada individuo menos el de la última casilla
      b. calcula el fitness relativo de cada individuo dividiendo el fitness de cada individuo entre el fitness total
      c. calcula el fitness acumulado inicializando el fitness acumulado del primer individuo con su fitness relativo y calculando el fitness del siguinete individuo sumando el fitness acumulado del anterior más el fitness relativo del individuo en cuestión. Con este fitness acumulado se obtiene la proporción de cada cromosoma dentro de la población finalmente se genera un número aleatorio el número de veces según el número de individuos, si este es mayor o igual al fitness acumulado de individuo j y es menor al del individuo j+1, sobrevive el individuo j+1.
   9. Se manda llamar a la función Adiciona, esta función adiciona dependiendo de un número generado aleatoriamente y una probabilidad de adición un nodo al final de la lista que representa un cromosoma, los pesos de este nodo se inicializan con 0
   10. Se llama a la función Elimina, esta función elimina (dependiendo de un número generado aleatoriamente y una probabilidad de eliminación) un nodo de forma aleatoria de la lista que forma el cromosoma, modificando la red quitándole un nodo oculto (Mutación estructural)
   11. Se llama a la función Mutanodo, esta función selecciona que pesos van a ser modificados por medio de una aproximación gaussiana, se genera un número aleatorio, si es menor a la probabilidad de mutación del nodo (PMUTNODO, definida en peclv.h) modifica el peso (Mutación paramétrica)
   12. Nuevamente se manda llamar a Evaluared, quien evalua a cada individuo por medio de una función fitness para esta nueva generación
   13. Se llama a la función Elitismo, esta función encuentra al mejor individuo de la generación en cuestión de acuerdo al valor de su fitness, si es mejor al que se guardo en la última casilla de la población en la generación anterior, este se reemplaza , en otro caso se reemplaza el peor individuo de la población de la generación actual con el mejor de la generación anterior.
   14. Nuevamente se llama a la función reporte para guardar los resultados de la generación en cuestión.

   } Del paso 8 al 14 se van a repetir el número de generaciones definidas por el usuario.
15. Se evalúa al mejor individuo de todas las generaciones
16. Se salvan los pesos de la arquitectura encontrada

17. Se cierran los archivos que se abrieron al principio del programa

Fin del programa

Para correr el programa sólo hay que entrar al archivo peclv.h e indicar el nombre de los archivos de entrenamiento y de prueba, así como modificar el número de nodos de entrada y de salida con los que se va a trabajar de acuerdo a la base de datos con la que se va a entrenar y probar el algoritmo. Si se desea modificar el valor de alguna constante es en este archivo donde se deben modificar.

Finalmente se manda llamar al archivo peclv.cpp para ser compilado con los nuevos datos y ejecutarlo.

A continuación se encuentra el listado del programa evolutivo

```
/*****************************************************************************/
/* 10/febrero/1999                                                           */
/* Implementación de un programa evolutivo basado en  cromosomas de longitud*/
/* variable, donde cada individuo de la población representa la arquitectura*/
/* de una red neuronal perceptron multicapa.                                 */
/* Cada individuo tiene como información a la de un conjunto de nodos        */
/* ocultos, los cuales forman la arquitectura de una red, dicha información */
/* son sus respectivos pesos de entrada y salida de cada nodo oculto         */
/*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <io.h>
#include <alloc.h>
#include <conio.h>
/*CLVbp-Cromosomas de Longitud Variable*/
#include "c:\almatesi\messy\CLV\CLVbp.h"
#include "c:\almatesi\messy\CLV\Arch.h"

//Programa principal

main()
{
   Nodo *ap, *apunta;
   int ocult;
   float e;

   clrscr();
   generacion=0;
   k=0;
   randomize();

   printf("%s%d\n","Tama¤o de la poblaci¢n ", TAMPOB);
   Ab_arch_esc();
   //encabezados del reporte que se genera en el archivo reporte.dat
   fprintf(out, "\n Poblacion  %d, Max_gener %2.0f, Max_tam_crom %d,   \n",
   TAMPOB, MAXGENS,MaxTam );
   fprintf(out, "\n generac., mejor_val,  promedio , ");
   fprintf(out, " fit_desv_stand, nmax");
   Inicializa();
   Evaluared(sal);
   Guarda_el_mejor();
   Reporte();
   printf("generacion:  %2.0f\n", generacion);
   while (generacion<(MAXGENS-1))
   {
      generacion=generacion+1;
      Seleccion();  /*ruleta*/
      Adiciona();
      Elimina();
      Mutanodo();
      Evaluared(sal);
      printf("generacion: %2.0f, mejor fitness: %f, tama¤o: %d\n",
      generacion,poblacion[TAMPOB].fitness,poblacion[TAMPOB].nmax);
      Elitismo();
      Reporte();
```

```
      sigma=1-(generacion/MAXGENS); /*se utiliza en Randval*/
   }
   conta=0;
   apunta=poblacion[TAMPOB].crom;
   ocult=poblacion[TAMPOB].nmax;
   err=Calcula(apunta,ocult,TAMPOB,sal);/*datos de entrenamiento mejor indiv.*/
   fprintf(sal,"%f",err);
   err=Calculap(apunta,ocult,TAMPOB,salp);/*datos de prueba mejor indiv.*/
   fprintf(salp,"%f",err);
   Salvapesos();
   Cierrarch();
   printf("ya");
   getchar();

   return(1);
}

/*************************************************************************/
/* Nombre de la función: Inicializa                                      */
/* Descripción de la función:                                            */
/* inicializa es una función que manda llamar a las funciones necesarias */
/* para ir creando la estructura que se va a manejar y también para irla */
/* inicializ., en este proceso se van inicializ. los pesos de forma aleat.*/
/* entre -1 y 1, al igual que los nodos ocultos entre 0 y MaxTam         */
/* Funciones que llama: Crea_lista                                       */
/* Funciones que la llaman:main                                          */
/*************************************************************************/

void Inicializa(void)
{
   int i,j,ocultos;

   for(j=0; j<TAMPOB; j++)
   {
     /*los nodos ocultos se inicializan aleatoriamente para cada individuo*/
     ocultos=rand()%MaxTam+1;  /*el rango va de 0 a MaxTam=10 solo para
     inicializar*/
     /*inicializac. aleatoria de los sesgos, el rango va de -1 a 1*/
     for (i=0; i<NSAL;i++)
       poblacion[j].sesgo[i]=((2*((rand()%1000)/1000.0))-0.99);
      poblacion[j].fitness=0.0;
      poblacion[j].rfitness=0.0;
      poblacion[j].cfitness=0.0;
      poblacion[j].nmax=ocultos;
      poblacion[j].crom=NULL;
      poblacion[j].crom=Crea_lista(ocultos);
   }
   /*inicializaci¢n del ultimo individuo*/
   poblacion[TAMPOB].crom=NULL;
   for (i=0; i<NSAL;i++)
      poblacion[TAMPOB].sesgo[i]=0;
   poblacion[TAMPOB].fitness=0.0;
   poblacion[TAMPOB].rfitness=0.0;
   poblacion[TAMPOB].cfitness=0.0;
   poblacion[TAMPOB].nmax=0;
}
```

```
/*****************************************************************************/
/* Nombre de la función: Crea_lista                                          */
/* Descripción de la función:                                                */
/* Se crea una lista ligada que representa cada elemento de la lista un nodo*/
/* oculto de la red, a su vez cada nodo tiene un arreglo del tamaño de nodos*/
/* de entrada + 1 sesgo + nodos de salida y se inicializan los pesos de     */
/* estos nodos con respecto al nodo oculto, ie, los pesos que le llegan y   */
/* los que salen del nodo oculto se guardan en este arreglo, la inicializac.*/
/* es aleatoria                                                              */
/* Funciones que llama:                                                      */
/* Función que la llama: Inicializa                                          */
/*****************************************************************************/

Nodo* Crea_lista(int ocultos)
{
   Nodo *Nodos_ocultos, *ap_lista, *Aux;
   int i,j;

   Nodos_ocultos=new Nodo;
   Nodos_ocultos->Sig_vert=NULL;
   for (i=0;i<(NENT+1+NSAL); i++)
      Nodos_ocultos->peso[i]=((2*((rand()%1000)/1000.0))-0.99);
   ap_lista=Nodos_ocultos;
   for (i=1;i<ocultos;i++)

   {
      Aux=new Nodo;
      Aux->Sig_vert=NULL;
      for (j=0;j<NENT+1+NSAL; j++)
       Aux->peso[j]=((2*((rand()%1000)/1000.0))-0.99);
      ap_lista->Sig_vert=Aux;
      ap_lista=Aux;
   }
   return(Nodos_ocultos);
}

/*****************************************************************************/
/* Nombre de la función. Evaluared                                           */
/* Descripción de la función:                                                */
/* Se manda llamar a la función Calcula quien va a calcular el error de la   */
/* red y lo guarda como el fitness                                           */
/* ap_nodo es un apuntador al nodo que apunta el inicio de la lista que      */
/* representa a los nodos ocultos y ocultos me dice el tamaño de esa lista   */
/* Funciones que llama: Calcula                                              */
/* Funciones que la llaman: main()                                           */
/*****************************************************************************/

void Evaluared(FILE *salida)
{
   Nodo* ap_nodo;
   int i,ocultos;

   for(i=0; i<TAMPOB; i++)
   {
      ap_nodo=poblacion[i].crom;
      ocultos=poblacion[i].nmax;
      poblacion[i].fitness=Calcula(ap_nodo,ocultos,i,salida);
```

```
   }
}

/*****************************************************************************/
/* Nombre de la función: Calcula                                             */
/* Descripción de la función:                                                */
/* Calcula abre el archivo de datos manda llamar a LeeArch que saca los      */
/* datos a un vector y manda llamar a las funciones necesarias para          */
/* calcular el error de la red                                               */
/* Funciones que llama: LeeArch, Calculaf_net,Calcula_salida,Calculaerror    */
/* Funciones que la llaman: Evaluared                                        */
/*****************************************************************************/

float Calcula(Nodo* ap_nodo,int ocultos, int k, FILE *salida)
{
   float errort=0.0, e1=0.0, ECM=0.0;
   int npats=0;  /*num. de patrones leidos*/

   if ((in=fopen(InputFile,"r"))==NULL)
   {
      printf("Error al abrir el archivo de datos\n");
      exit(1);
   }

   while( LeeArch(in))
   {
      Calculaf_net(ap_nodo, ocultos);
      Calcula_salida(ap_nodo,ocultos,k);
      //Las siguientes tres lineas calculan sumatoria(y-d)^2  error b
      e1=Calculaerror1(salida);
      ECM=float(ECM+e1);
      npats++;
   }
     err=ECM;
     if( ECM<0.01)
      errort=100;
     else
      errort=float(1/ECM);
   fclose(in);
   return(errort);
}

/*****************************************************************************/
/* Nombre de la función:LeeArch                                              */
/* Descripción de la función:                                                */
/* Saca la información del archivo de datos y la coloca en dos vectores      */
/* un vector de nombre Vectors tiene la info. de los atributos, el otro      */
/* se llama SalidaDeseada y tiene la clase de los atributos seleccionados    */
/* Funciones que llama:                                                      */
/* Funciones que la llaman:Calcula                                           */
/*****************************************************************************/

int LeeArch( FILE *f )
{
   int i;
   //Lee entradas
```

```
   Vector[0]=1;    //sesgo de la entrada
   for( i=1; i<NENT+1; i++ )
   {
      if( feof( f ) )
       return 0;
      fscanf( f, " %f ", &(Vector[i]) );
   }
   // Lee salidas deseadas
   for( i=0; i<NSAL; i++ )
   {
      if( feof( f ) )
       return 0;
      fscanf( f, " %f ", &(SalidaDeseada[i]) );
   }
   return 1;
}

/*****************************************************************************/
/* Nombre de la función: Calculaf_net                                        */
/* Descripción de la función:                                                */
/* Calcula la función de transferencia de la red, primero saca los pesos     */
/* del nodo oculto correspondiente en un vector, luego los multiplica con el*/
/* vector de los datos, para sacar la sumatoria del producto de pesos por    */
/* los datos de entrada y finalmente se le aplica la funci¢n de transferen- */
/* cia, que en este caso es la sigmoide para ir llenando el vector con estos*/
/* resultados                                                                */
/* Los parámetros que recibe son: ap_nodo que apunta a la lista ligada de    */
/* nodos ocultos y ocultos que dice el tamaño de la lista                    */
/* Funciones que llama:Saca_pesos, Multiplica, Sigmoide                      */
/* Funciones que la llaman: Calcula                                          */
/*****************************************************************************/

void Calculaf_net(Nodo* ap_nodo,int ocultos)
{
   Nodo *apunta_nodo;
   float net;
   int i;

   f_transf[0]=1; /*se guarda el sesgo de la segunda capa*/
   f_t[0]=1;

   apunta_nodo=ap_nodo;
   for (i=0; i<ocultos;i++)
   {
      Saca_pesos(apunta_nodo,0,NENT+1);
      net=Multiplica(vector_pesos, Vector,NENT+1);
      f_transf[i+1]=Sigmoide(net);
      f_t[i+1]=f_transf[i+1];
      apunta_nodo=apunta_nodo->Sig_vert;
   }
}

/*****************************************************************************/
/* Nombre de la función: Saca_pesos                                          */
/* Descripción de la función:                                                */
/* Saca los pesos de entrada de un nodo oculto en específico en un vector    */
/* Los parámetros que recibe son: ap_nodo que dice de que nodo se van a      */
```

```
/* sacar los pesos para guardarlos en el arreglo vector_pesos                */
/* y los limites que indican de que posición del arreglo a cual posición se */
/* van  sacar                                                                */
/* Función que llama:                                                        */
/* Función que la llama:Calculaf_net                                         */
/*****************************************************************************/

void Saca_pesos(Nodo* ap_nodo,int lim_inf, int lim_sup)
{
   Nodo* apunta_nodo=ap_nodo;
   int i,k;
   k=0;

   for (i=lim_inf; i<lim_sup; i++)
   {
      vector_pesos[k]=apunta_nodo->peso[i];
      k++;
   }
}

/*****************************************************************************/
/* Nombre de la función: Multiplica                                          */
/* Descripción de la función:                                                */
/* Multiplica los pesos por los datos de entrada y realiza la sumatoria de  */
/* estos productos                                                           */
/* Funciones que llama:                                                      */
/* Funciones que la llaman: Calculaf_net, Calcula_salida                     */
/*****************************************************************************/

float Multiplica(float *A,float *B, int limite)
{
   int i;
   float sumatoria=0.0;

   for(i=0; i<limite;i++)
      sumatoria=sumatoria+(A[i]*B[i]);
   return(sumatoria);
}

/*****************************************************************************/
/* Nombre de la función: Sigmoide                                            */
/* Descripción de la función:                                                */
/* Evalua la función de transferencia para evaluar la red neuronal           */
/* Función que llama:                                                        */
/* Función que la llama:Calcula                                              */
/*****************************************************************************/

float Sigmoide(float u)
{
   float sol;
   int a=-1;

   sol=1/(1+exp(a*u));
   return(sol);
}

/*****************************************************************************/
```

```
/* Nombre de la función: Calcula_salida                                    */
/* Descripción de la función:                                              */
/* ap_nodo es un apuntador a la lista ligada de nodos ocultos, ocultos     */
/* dice el tamaño de la lista, k, dice que individuo estoy evaluando       */
/* Saca_pesos2 saca los pesos que salen de los nodos ocultos a los de      */
/* salida, multiplica hace el calculo de la sumatoria del producto de los  */
/* pesos por la entrada, de la capa oculta hacia la salida y Salida        */
/* es un vector que guarda el resultado de evaluar toda la red para cada   */
/* nodo de salida                                                          */
/* Funciones que llama: Saca_pesos2, Multiplica                            */
/* Funciones que la llama: Calcula                                         */
/*************************************************************************/

void Calcula_salida( Nodo* ap_nodo,int ocultos, int k)
{
   Nodo* apunta_nodo;
   int i;
   float net;

   for (i=0; i<NSAL; i++)
   {
      apunta_nodo=ap_nodo;
      Saca_pesos2(apunta_nodo, i+1, ocultos,k);
      net=Multiplica(vector_pesos, f_transf,ocultos+1);
      Salida[i]=Sigmoide(net);
   }
}

/*************************************************************************/
/* Nombre de la función: Saca_pesos2                                       */
/* Descripción de la función:                                              */
/* k nos dice que individuo estoy evaluando y por lo tanto por medio de k  */
/* saco los sesgos que le corresponden de la capa intermedia a las salidas */
/* en vector_pesos guardo los pesos de entrada para cada salida, ie, los de */
/* salida de cada nodo oculto, cont me dice a partir de cual posición dentro*/
/* del arreglo estan los pesos de salida de cada nodo oculto               */
/* Funciones que llama:                                                    */
/* Funciones que la llaman: Calcula_salida                                 */
/*************************************************************************/

void Saca_pesos2( Nodo* apunta_nodo,int cont, int ocultos,int k)
{
   Nodo* ap_nodo=apunta_nodo;
   int i;

   vector_pesos[0]=poblacion[k].sesgo[cont-1];  /*peso del sesgo*/
   for (i=0; i<ocultos; i++)
   {
      vector_pesos[i+1]=ap_nodo->peso[NENT+cont];
      ap_nodo=ap_nodo->Sig_vert;
   }
}

/*************************************************************************/
/* Nombre de la función: Calculaerror                                      */
/* Descripción de la función:                                              */
/* Calcula el error de la siguiente manera: la sumatoria                   */
```

```
/* la diferencia de la salida deseada menos la salida de la red              */
/* esta función regresa un real que al final se divide entre el número de    */
/* patrones totales que existen                                              */
/* Funciones que llama:                                                      */
/* Funciones que la llaman:Calcula                                           */
/*****************************************************************************/

float Calculaerror(FILE *salida)
{
   float e=0.0;
   int i,aux=0.0;

   for (i=0; i<NSAL; i++)
   {
      e=e+fabs(SalidaDeseada[i]-Salida[i]);
      Dif[i]=fabs(SalidaDeseada[i]-Salida[i]);
      if (conta==0)
      {
       if (cont<NSAL)
       {
          fprintf(salida,"%f  ",Salida[i]);
          cont++;
       }
       else
       if (cont>NSAL-1)
       {
          cont=1;
          fprintf(salida,"\n");
          fprintf(salida, "%f  ", Salida[i]);
       }
      }
      if (Dif[i]<=0.45)
      aux=aux+1;
   }
   if (aux==NSAL)
      fitness_aciertos=fitness_aciertos+1;
   totaldepruebas=totaldepruebas+1;
   return(e);
}

/*****************************************************************************/
/* Nombre de la función: Calculaerror1                                       */
/* Descripción de la función:                                                */
/* Calcula el error de la siguiente manera: la sumatoria del cuadrado de     */
/* la diferencia de la salida deseada menos la salida de la red              */
/* esta función regresa un real que al final se divide entre el número de    */
/* patrones totales que existen                                              */
/* Funciones que llama:                                                      */
/* Funciones que la llaman:Calcula                                           */
/*****************************************************************************/

float Calculaerror1(FILE *salida)
{
   float e1=0.0;
   int i,aux=0.0;
```

```
   for (i=0; i<NSAL; i++)
   {
      e1=e1+fabs ((SalidaDeseada[i]-Salida[i])*(SalidaDeseada[i]-Salida[i]) );
      Dif[i]=fabs(SalidaDeseada[i]-Salida[i]);
      if (conta==0)
      {
       if (cont<NSAL)
       {
          fprintf(salida,"%f  ",Salida[i]);
          cont++;
       }
       else
       if (cont>NSAL-1)
       {
          cont=1;
          fprintf(salida,"\n");
          fprintf(salida, "%f  ", Salida[i]);
       }
      }
      if (Dif[i]<=0.49)
      aux=aux+1;
   }
   if (aux==NSAL)
      fitness_aciertos=fitness_aciertos+1;
   totaldepruebas=totaldepruebas+1;
   return(e1);
}

/*****************************************************************************/
/* Nombre de la función: Guarda_el_mejor                                     */
/* Descripción de la función:                                                */
/* Encuentra el mejor individuo de la población buscando el de menor         */
/* fitness, el fitness es el error cuadrático y manda llamar CopiaGen para   */
/* copiar el mejor individuo en la ultima casilla de la población            */
/* Funcines que llama: CopiaGen                                              */
/* Funciones que la llaman: main                                             */
/*****************************************************************************/

void Guarda_el_mejor(void)
{
   int mem;
   float min = poblacion[0].fitness;
   el_mejor=0;

   for (mem = 1; mem < TAMPOB; mem++)
   {
      if (poblacion[mem].fitness > min)
      {
       el_mejor = mem;
       min = poblacion[mem].fitness;
      }
   }
   /* una vez encontrado el mejor individuo copia los genes*/
   CopiaGen(poblacion,el_mejor,poblacion,TAMPOB);
}

/*****************************************************************************/
```

```
/* Nombre de la función: CopiaGen                                          */
/* Descripción de la función:                                              */
/* copia un  individuo de una casilla del vector a otra o de un vector     */
/* población a otro vector nueva población, A y B son vectores donde A es  */
/* el vector fuente y B el destino y fuente dice la posición del individuo */
/* que se va a copiar del vector A al vector B en la posisción destino     */
/* Funciones que llama: Copia_lista                                        */
/* Funciones que la llaman: Guarda_el_mejor, Seleccion, Elitismo           */
/*************************************************************************/

void CopiaGen(Genotipo *A, int fuente, Genotipo *B, int destino)
{
   int i,ocultos;

   for (i=0; i<NSAL;i++)
       B[destino].sesgo[i]=A[fuente].sesgo[i];
   B[destino].fitness=A[fuente].fitness;
   B[destino].rfitness=A[fuente].rfitness;
   B[destino].cfitness=A[fuente].cfitness;
   B[destino].nmax=A[fuente].nmax;
   ocultos=B[destino].nmax;
   B[destino].crom=Copia_lista(A,fuente,ocultos);
}

/*************************************************************************/
/* Nombre de la función: Copia_lista                                       */
/* Descripción de la función:                                              */
/* Los nodos ocultos se van ligando en una lista y cada nodo tiene un      */
/* vector de pesos que le correponde, copia_lista copia esta lista, del    */
/* vector población A, copia el individuo el_mejor y ocultos dice de que   */
/* tamaño es la lista, para cada individuo el tamaño de la lista cambia    */
/* Funciones que llama:                                                    */
/* Funciones que la llaman:CopiaGen                                        */
/*************************************************************************/

Nodo* Copia_lista(Genotipo *A,int el_mejor, int ocultos)
{
   Nodo *apunta_nodo,*Nodos_ocultos,*ap_lista,*Aux;
   int i,j;

   apunta_nodo=A[el_mejor].crom;
   Nodos_ocultos=new Nodo;
   Nodos_ocultos->Sig_vert=NULL;
   for (i=0;i<(NENT+1+NSAL); i++)
      Nodos_ocultos->peso[i]=apunta_nodo->peso[i];
   ap_lista=Nodos_ocultos;
   apunta_nodo=apunta_nodo->Sig_vert;

   for (i=1;i<ocultos;i++)
   {
      Aux=new Nodo;
      Aux->Sig_vert=NULL;
      for (j=0;j<NENT+1+NSAL; j++)
       Aux->peso[j]=apunta_nodo->peso[j];
      ap_lista->Sig_vert=Aux;
      ap_lista=Aux;
      apunta_nodo=apunta_nodo->Sig_vert;
```

```
   }
   return(Nodos_ocultos);
}

/*************************************************************************/
/* Nombre de la función: Reporte                                         */
/* Descripción de la función:                                            */
/* Reporta el desarrollo de la simulación, por cada generación guarda en el */
/* archivo de salida la siguiente información:                           */
/* generacion,mejor_val=mejor fitness,avg=promedio del fitness,stddev=desviac*/
/* standar del fitness,poblacion[TAMPOB]->nmax=num. m x. de nodos del ult.   */
/* individuo, es decir, del mejor individuo                              */
/* Funciones que llama:                                                  */
/* Función que la llama: main                                    */
/*************************************************************************/

void Reporte(void)
{
   int i;
   float mejor_val;     /* mejor fitness */
   float prom;          /* promedio de fitness de la pob.*/
   float desv_std;       /* desviacion standar de fitness */
   float sum_cuad;   /* suma de cuadrados para el calc. std*/
   float cuad_sum;   /* el cuadtrado de la suma para el calc. std */
   float sum;          /* total pop fitness */
   float a;

   sum = 0.0;
   sum_cuad = 0.0;

   for (i = 0; i < TAMPOB; i++)
   {
      sum += (float)poblacion[i].fitness;
      sum_cuad += pow((float)poblacion[i].fitness,2) ;
   }
   prom = sum/(float)TAMPOB;
   cuad_sum = prom * prom * (float)TAMPOB;
   a=(sum_cuad-cuad_sum)/(TAMPOB - 1);
   desv_std = sqrt(a);
   mejor_val = (float)poblacion[TAMPOB].fitness;
   fprintf(out,"\n%2.0f          %6.5f      %6.5f        %6.5f          %d\n
",generacion,mejor_val,prom,desv_std,poblacion[TAMPOB].nmax);
}

/*************************************************************************/
/* Nombre de la función: Seleccion (M,todo de la ruleta)                 */
/* Descripción de la función:                                            */
/* Encuentra el fitness total de la población sumando todos los fitness  */
/* menos el ultimo, calcula el fitness relativo de cada individuo dividiendo*/
/* su fitness entre el fitness total, con el fitness relativo calcula el */
/* fitness acumulado y con el fitness acumulado se obtiene la proporción de */
/* cada cromosoma y se trata de seleccionar por lo general al mejor individ.*/
/* sin descartar la posibilidad de guardar uno que otro peor individuo   */
/* Funciones que llama:Ini_nuevapob,fin, CopiaGen                        */
/* Función que la llama: main                                            */
/*************************************************************************/
```

```
void Seleccion(void)
{
   int mem, i, j;
   float sum = 0;
   float p;

   Ini_nuevapob();
   /* Encuentra el fitness total de la poblacion*/
   for (mem = 0; mem < TAMPOB; mem++)
      sum += poblacion[mem].fitness;
   /* calcula el fitness relativo */
   for (mem = 0; mem < TAMPOB; mem++)
      poblacion[mem].rfitness = (poblacion[mem].fitness)/sum;
   poblacion[0].cfitness = poblacion[0].rfitness;
   /* calcula fitness  acumulado*/
   for (mem = 1; mem < TAMPOB; mem++)
      poblacion[mem].cfitness = poblacion[mem-1].cfitness +
      poblacion[mem].rfitness;
   /* selecciona al individ. sobreviviente usando el fitness acumulado*/
   for (i = 0; i < TAMPOB; i++)
   {
      p = rand()%1000/1000.0;
      if (p < poblacion[0].cfitness)
      {
       Fin(nuevapoblacion,i);
       CopiaGen(poblacion,0,nuevapoblacion,i);
      }
      else
      {
       for (j = 0; j < TAMPOB; j++)
          if (p >= poblacion[j].cfitness && p < poblacion[j+1].cfitness)
          {
             Fin(nuevapoblacion,i);
             CopiaGen(poblacion,j+1,nuevapoblacion,i);
          }
      }
   }
   /* una vez creada la nueva poblaci¢n la regresa a su original arreglo */
   for (i = 0; i < TAMPOB; i++)
   {
      Fin(poblacion,i);
      CopiaGen(nuevapoblacion,i,poblacion,i);
   }
   for(i=0;i<TAMPOB;i++)
      Fin(nuevapoblacion,i);
}

/*************************************************************************/
/* Nombre de la función: Ini_nuevapob                                    */
/* Descripción de la función:                                            */
/* Inicializa un vector como el de población llamado nuevapoblacion y sirve */
/* como auxiliar para hacer copias de los individuos según los procesos  */
/* donde sea llamado                                                     */
/* Función que llama:                                                    */
/* Funciones que la llaman: Seleccion                                    */
/*************************************************************************/
```

```
void Ini_nuevapob(void)
{
   int i,j;

   for (i=0;i<=TAMPOB;i++)
   {
     nuevapoblacion[i].crom=NULL;
     for (j=0; j<NSAL;j++)
       nuevapoblacion[i].sesgo[j]=0;
     nuevapoblacion[i].fitness=0.0;
     nuevapoblacion[i].rfitness=0.0;
     nuevapoblacion[i].cfitness=0.0;
     nuevapoblacion[i].nmax=0;
   }
}

/*****************************************************************************/
/* Nombre de la función: Fin                                                 */
/* Descripción de la funci¢n:                                                */
/* Elimina todo el individuo de la posici¢n indice del vector A, esto es,    */
/* pone en ceros los valores reales y borra la lista de los nodos ocultos    */
/* dejando este campo en NULL                                                */
/* Función que llama:                                                        */
/* Funciones que la llaman: Seleccion, Elitismo                              */
/*****************************************************************************/

void Fin (Genotipo *A, int indice)
{
   Nodo* ap_nodo=A[indice].crom;
   int j;

   for (j=0; j<NSAL;j++)
       A[indice].sesgo[j]=0;
   A[indice].fitness=0.0;
   A[indice].rfitness=0.0;
   A[indice].cfitness=0.0;
   A[indice].nmax=0;
   /*ap_nodo=A[indice].crom;*/
   while (ap_nodo!=NULL)
   {
      A[indice].crom=ap_nodo->Sig_vert;
      free (ap_nodo);
      ap_nodo=A[indice].crom;
   }
   A[indice].crom=NULL;
}

/*****************************************************************************/
/* Nombre de la función: Mutanodo                                            */
/* Descripción de la función:                                                */
/* Se selecciona cuales nodos tendra sus pesos mutados                       */
/* usando una "Random   gaussian mutation"                                   */
/* Funciones que llama: Randval                                              */
/* Funciones que la llaman: main                                             */
/*****************************************************************************/

void Mutanodo(void)
```

```
{
   int i,j;
   float x,peso;
   Nodo* ap_nodo;


   for (i = 0; i < TAMPOB; i++) /*recorre a todos los individuos*/
   {
      for (j=0; j<NSAL;j++)
      {
       x=rand()%1000/1000.0;
       if (x < PMUTNODO)     /*recorre los sesgos de cada individuo*/
       {
          sesgo=poblacion[i].sesgo[j];
          poblacion[i].sesgo[j]=Randval(sesgo);
       }
      }
      ap_nodo = poblacion[i].crom;
      /*recorre a todos los nodos ocultos de un individ. */
      while (ap_nodo != NULL)
      {
       x = rand()%1000/1000.0;
       if (x < PMUTNODO)
       {
          for (j=0;j<NENT+1+NSAL; j++)
          {
             peso=ap_nodo->peso[j];
             ap_nodo->peso[j] = Randval(peso);
          }
       }
       ap_nodo=ap_nodo->Sig_vert;
      }
   }
  // PMUTNODO=1-(generacion/MAXGENS);
}

/*****************************************************************************/
/* Nombre de la función: Randval                                             */
/* Descripción de la función:                                                */
/* Se calcula una aproximación Gaussiana, este es el valor de la mutaci$n    */
/* que se aplica, es decir, el peso seleccionado para mutarlo, se cambia por*/
/* este valor, el par metro values es el peso que se va a mutar (o cambiar) */
/* Funciones que llama:                                                      */
/* Funciones que la llaman: Mutanodo                                         */
/*****************************************************************************/

float Randval(float value)
{
   int i;
   float val=0;

   for (i = 0; i < 12; i++)
      val += (float)((rand()%1000)/1000.0);
   val= value+sigma*(val-6);
   return (val);
}
```

```
/*****************************************************************************/
/* Nombre de la función: Elitismo                                            */
/* Descripción de la función:                                                */
/* Encuentra el mejor y el peor individuo de cada generación si el mejor     */
/* individuo de la nueva población es mejor que el mejor individuo de la     */
/* población anterior (el cual esta guardado en el último lugar del arreglo */
/* poblacion) se copia el individuo de la nueva población, en otro caso      */
/* reemplaza el peor individuo de la población actual con el mejor de la     */
/* generación anterior                                                       */
/* Funciones que llama:Fin, CopiaGen                                         */
/* Función que la llama: main                                                */
/*****************************************************************************/

void Elitismo(void)
{
   int i;
   float mejor, peor;        /* mejor y peor valor del fitness */
   int mejor_mem, peor_mem;   /* indices para el mejor y el peor miembro*/

   mejor = poblacion[0].fitness;
   peor = poblacion[0].fitness;
   for (i = 0; i < TAMPOB - 1; ++i)
   {
      if (poblacion[i].fitness > poblacion[i+1].fitness)
      {
       if (poblacion[i].fitness >= mejor)
       {
          mejor = poblacion[i].fitness;
          mejor_mem = i;
       }
       if (poblacion[i+1].fitness <= peor)
       {
          peor = poblacion[i+1].fitness;
          peor_mem = i + 1;
       }
      }
      else
      {
       if (poblacion[i].fitness <= peor)
       {
          peor = poblacion[i].fitness;
          peor_mem = i;
       }
       if (poblacion[i + 1].fitness >= mejor)
       {
          mejor = poblacion[i + 1].fitness;
          mejor_mem = i + 1;
       }
      }
   }
   /*si el mejor individuo de la nueva poblaci¢n es mejor que el mejor   */
   /*individuo de la poblaci¢n anterior se copia el individuo de la nueva*/
   /*poblaci¢n, en otro caso reemplaza el peor individuo de la poblaci¢n */
   /*actual con el mejor de la generaci¢n anterior                       */

   if (mejor > poblacion[TAMPOB].fitness)
   {
```

```
      Fin(poblacion,TAMPOB);
      CopiaGen(poblacion,mejor_mem,poblacion,TAMPOB);
   }
   else
   {
      Fin(poblacion,peor_mem);
      CopiaGen(poblacion,TAMPOB,poblacion,peor_mem);
   }
}

/*****************************************************************************/
/* Nombre de la función: Salvapesos                                          */
/* Descripción de la función:                                                */
/* Copia los pesos del mejor individuo de todas las generaciones asi como el*/
/* resto de sus datos                                                        */
/* Funciones que llama:                                                      */
/* Funciones que la llaman: main                                             */
/*****************************************************************************/

void Salvapesos()
{
     int i,j;
     Nodo* Aux;

     fprintf(out,"\nindividuo final\n");
     for (i=0; i<NSAL;i++)
      fprintf(out, "sesgo   %f\n",poblacion[TAMPOB].sesgo[i]);
     fprintf(out, "fitness   %f\n",  poblacion[TAMPOB].fitness);
     fprintf(out, "rfitness  %f\n",  poblacion[TAMPOB].rfitness);
     fprintf(out, "cfitness  %f\n",  poblacion[TAMPOB].cfitness);
     fprintf(out, "nmax      %d\n",  poblacion[TAMPOB].nmax);

     fprintf(out, "pesos\n");
     Aux=poblacion[TAMPOB].crom;

     for (i=0;i<poblacion[TAMPOB].nmax;i++)
     {
      for (j=0;j<NENT+1+NSAL; j++)
         fprintf(out,"%f\n", Aux->peso[j]);
      fprintf(out,"\n");

      Aux=Aux->Sig_vert;
     }
}
/*****************************************************************************/
/*Nombre de la función: Adiciona                                             */
/*Descripción de la función: Adiciono un nodo al final de la lista con       */
/*pesos inicializados a cero                                                 */
/*Función que la llama:main()                                                */
/*****************************************************************************/

void Adiciona(void)
{
   Nodo *apuntador, *ap_aux;
   int i,j;

   float x;
```

```
   for(i=0; i<TAMPOB; i++)
   {
      x=rand()%1000/1000.0;
      if (x<PADIC)
      {
       if (poblacion[i].nmax+1<=MaxTam)
       {
          apuntador=poblacion[i].crom;
          while(apuntador->Sig_vert!=NULL)
            apuntador=apuntador->Sig_vert;
          ap_aux=new Nodo;
          ap_aux->Sig_vert=NULL;
          for(j=0; j<NENT+NSAL+1; j++)
             ap_aux->peso[j]=0;
          apuntador->Sig_vert=ap_aux;
          poblacion[i].nmax=poblacion[i].nmax+1;
       }
      }
   }
   PADIC=1-(generacion/MAXGENS); //+0.005//
}

/*************************************************************************/
/* Nombre de la función:Elimina                                          */
/* Descripción de la función: Elimina el primer miembro de la lista de   */
/* nodos ocultos                                                         */
/* Función que la llama: main()                                          */
/*************************************************************************/

void Elimina(void)
{
   Nodo *aux, *auxi;
   int i,j,ocultos,cont;
   float x;

   for(i=0; i<TAMPOB; i++)
   {
      x=rand()%1000/1000.0;
      cont=2;
       /*elige si va a eliminar un gen de este indiv. o no*/
      if ((x<PELIMIN) && (poblacion[i].nmax>1))
      {
       ocultos=poblacion[i].nmax;
       aux=poblacion[i].crom;
       auxi=aux->Sig_vert;
       /*j es el gen a eliminar*/
       j=random(ocultos);
       j++;
       if (j==1)
       {
          poblacion[i].crom=aux->Sig_vert;
          delete aux;
       }
       else
       {
          while (cont!=j)
```

```
            {
               aux=auxi;
               auxi=auxi->Sig_vert;
               cont++;
            }
            aux->Sig_vert=auxi->Sig_vert;
            delete auxi;
         }
         poblacion[i].nmax=poblacion[i].nmax-1;
        }
    }
    PELIMIN=1-(generacion/MAXGENS);
}


void Elimina1(void)
{
    Nodo *aux;
    int i,j,ocultos,num;
    float x;

    for(i=0; i<TAMPOB; i++)
    {
       x=rand()%1000/1000.0;
        /*elige si va a eliminar el primer miembro de este indiv. o no*/
       if ((x<PELIMIN) && (poblacion[i].nmax>1))
       {
        aux=poblacion[i].crom;
        poblacion[i].crom=aux->Sig_vert;
        delete aux;
        poblacion[i].nmax=poblacion[i].nmax-1;
       }
    }
    PELIMIN=1-(generacion/MAXGENS);
}

/*****************************************************************************/
/* Nombre de la función: Calculap                                            */
/* Descripción de la función:                                                */
/* Calcula abre el archivo de datos de prueba manda llamar a LeeArch que     */
/* saca los datos a un vector y manda llamar a las funciones necesarias para*/
/* calcular el error de la red del último individuo                          */
/* Funciones que llama: LeeArch, Calculaf_net,Calcula_salida,Calculaerror    */
/* Funciones que la llaman: Evaluared                                        */
/*****************************************************************************/

float Calculap(Nodo* ap_nodo,int ocultos, int k, FILE *salida)
{
    float errort=0.0, el=0.0, ECM=0.0; /*ECM=Error Cuadratico Medio*/
    int npats=0;  /*num. de patrones leidos*/

    if ((datp=fopen(Datprue,"r"))==NULL)
    {
       printf("Error al abrir el archivo de datos\n");
       exit(1);
    }
```

```
   while( LeeArch(datp))
   {
      Calculaf_net(ap_nodo, ocultos);
      Calcula_salida(ap_nodo,ocultos,k);
      //Las siguientes tres lineas calculan sumatoria(y-d)^2  error b
      e1=Calculaerror1(salida);
      ECM=float(ECM+e1);
      npats++;
   }
      err=ECM;
      if( ECM<0.01)
       errort=100;
      else
       errort=float(1/ECM);
   fclose(datp);
   return(errort);
}
```

```
/*************************************************************************/
/*Declaración de las estructuras de datos para el programa evolutivo    */
/*y de las funciones que se usan durante todo el programa               */
/*************************************************************************/

// Archivos que se usan:
      // Archivo donde se genera el reporte
char  OutFile[50]= "c:\\almatesi\\apizaco\\reporte.dat",
      // Archivo donde estan los datos de entrenamiento
      InputFile[50]= "c:\\almatesi\\basedat\\chela\\dnaent.txt",
      // Archivo donde estan los datos de prueba
      Datprue[50]="c:\\almatesi\\basedat\\chela\\dnaprue.txt",
//Archivo donde se guarda la salida de la red con los datos de entrenamiento
      SalFile[50]= "c:\\almatesi\\apizaco\\Sal.dat" ,
//Archivo donde se guarda la salida de la red con los datos de entrenamiento
// y aplicando backpropagation
      SalFilebp[50]= "c:\\almatesi\\apizaco\\Salbp.dat" ,
//Archivo donde se guarda la salida de la red con los datos de prueba
      SalFilep[50]= "c:\\almatesi\\apizaco\\Salp.dat" ,
      PesosFile[50]="c:\\almatesi\\apizaco\\pesosfin.dat",
//Archivo donde se guarda la salida de la red con los datos de prueba y
// aplicando backpropagation
      Salprue[50]="c:\\almatesi\\apizaco\\salprue.dat",
      Salpruead[50]="c:\\almatesi\\apizaco\\salpruead.dat",
      // archivo que guarda los aciertos y errores
      Aciertos[50]= "c:\\almatesi\\apizaco\\aciertos.dat" ;

int    fitness_aciertos=0;  //var. que contabilizan aciertos y errores
int    fitness_error=0;
long   totaldepruebas=0;
int    sesgo=1;
float  sigma=0.5;  //se utiliza al hacer la mutaci¢n en randval1
int conta=1,cont=0; //son var. para escribir solo las salidas del £ltimo
individuo
//Se escribe el error en pantalla, en el main se llama a Calcula y ahi se
//ocupa esta variable
float err,errp;

float  PELIMIN= 0.5;     // probabilidad de eliminacion de un nodo 1
float  PADIC=0.8;        // probabilidad de adicionar un nodo  0.4

// Apuntadores a los archivos

FILE *out, *in, *sal, *salbp,*ac, *datp, *salprue, *salpruead, *salp, *pesos;

//Definici¢n de constantes

#define  NENT     180         // Nodos de entrada, no cuenta el sesgo
#define  NSAL     3        // Nodos de salida
#define  MaxSize 20        // max. tama¤o de un cromosoma, minimo es 3, nodos
ocultos
#define  POPSIZE 20        // tama¤o de la poblacion
#define  MAXGENS 250       // numero de generaciones
#define  PXOVER 0.15       // probabilidad de cruzamiento 0.9
#define  PMUTATION 0.8     // probabilidad de mutacion
#define  PMUTNODO 0.8      // probabilidad de mutacion de un nodo
#define  n          0.1;   // raz¢n de aprendizaje (se ocupa en el backprop.)
```

```
#define  M          0.99  // momentum (se ocupa en el backprop)
#define  c          0.5   // suaviza el punto de eliminaci¢n (se ocupa en el
backprop)
#define  alfa       0.1   //ctes. para el error tipo d
#define  beta       0.2

float     Vectors[NENT+1];
float     vector_pesos[(NSAL+NENT)*MaxSize];
float     SalidaDeseada[NSAL];
float     f_transf[5*MaxSize]; /* funci¢n de transferencia*/
float     f_t[5*MaxSize];
float     Salida[NSAL];
float     Dif[NSAL];   //diferencia entre La salida deseada y la real
float     d[NSAL];

int       k;
float     generacion;     //contador de las generaciones
int       el_mejor;       //indice del mejor individuo de la poblacion

struct Nodo{
           float   peso[NENT+1+NSAL];
           Nodo*   Sig_vert;
         };

struct Genotype{
             float   sesgo[NSAL];
             int     nmax;          // dimension of the chromosome
             float   fitness;          /* GT's fitness */
             float   rfitness;    // relative fitness
             float   cfitness;    // cumulative fitness
             Nodo*   chrom;     //ap. al cromosoma
           };

struct Genotype population[POPSIZE+1];
struct Genotype newpopulation[POPSIZE+1];

// Declaracion de funciones

void      Inicializa(void);
Nodo*     Crea_lista(int ocultos);
void      Evaluared(FILE*);
float     Calcula(Nodo*, int, int, FILE*);
float     Calculap(Nodo*, int, int, FILE*);
int       ReadFact( FILE* );
void      Calculaf_net(Nodo*,int);
void      Saca_pesos(Nodo*,int,int);
float     Multiplica(float *A,float *B, int);
float     Sigmoide(float);
void      Calcula_salida(Nodo*, int,int);
float     Calculaerror(FILE*);
float     Calculaerror1(FILE*);
void      Saca_pesos2(Nodo*, int,int,int);
float     Calcula_error(void);
void      Guarda_el_mejor(void);
void      CopyGene(Genotype *A, int,Genotype *B,int);
Nodo*     Copia_lista(Genotype *A,int,int);
void      Reporte(void);
```

```
void      Select(void);
void      Ini_newpopul(void);
void      Fin(Genotype*, int);
void      Mutanodo(void);
float     Randval1(float);
void      Elitist(void);
void      Salvapesos(void);
void      Salvapesos(void);
void      Salvapesosad(void);
void      Backprop(Nodo*, int, int);
float     Calculad_salida();
void      Modificapesos_sesgo(int);
void      Modificapesos(Nodo*,int);
void      Calculaf_netB(Nodo*, int);
void      Calculad_interm(Nodo*, int);
float     Calculasumat(Nodo*);
void      Modificapesos1capa(Nodo*, int);
void      Adiciona(void);
void      Elimina(void);
void      Ab_arch_esc(void);
void      Cierrarch(void);
void      Temperatura(void);
```