PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

ESCUELA DE SISTEMAS

TEMA:

EVOLUCIÓN DE REDES NEURONALES UTILIZANDO ALGORITMOS GENÉTICOS PARA UN PROBLEMA DE CLASIFICACIÓN DE SISMOS

AUTOR:

GORDILLO VERDEZOTO RAFAEL ALEJANDRO

TRABAJO PREVIO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS Y COMPUTACIÓN

QUITO, 2005

CONTENIDO

CAPITULO I: REDES NEURONALES Y EVOLUCIÓN EN REDES

El presente capítulo resume los puntos más importantes en el desarrollo de la tesis, detallando las definiciones de los temas necesarios en el que nos ayudarán a comprender de mejor manera el proceso de análisis, implementación y documentación de resultados del tema planteado.

Los conceptos presentados, están orientados hacia Redes Neuronales y Algoritmos Genéticos; estos conceptos nos ayudarán a comprender el proceso de evolución de redes neuronales con algoritmos genéticos en forma general.

1.1 Breve introducción a la Computación Evolutiva.

El término “computación evolutiva” nace en el año de 1993 y retoma conceptos de la evolución y la genética para resolver principalmente problemas de optimización. Esta rama de la inteligencia artificial tiene sus raíces en el desarrollo de tres disciplinas relacionadas pero independientes entre si: Algoritmos genéticos (AG), Programación evolutiva (PE), Estrategias Evolutivas (EE) que se originan en la década de los sesenta y setenta.

Estrategias Evolutivas:

Fue diseñada inicialmente con la meta de resolver problemas de optimización discretos y continuos, principalmente experimentales y considerados difíciles.

Utiliza *recombinación o cruce* y la operación de *selección***,** ya sea determinística o probabilística, elimina las peores soluciones de la población y no genera copia de aquellos individuos con una aptitud por debajo de la aptitud promedio.

Programación Evolutiva:

El procedimiento es muy similar a las estrategias evolutivas con la diferencia de que no emplea la *recombinación o cruce*, de tal forma que son denominadas en conjunto algoritmos evolutivos como una manera de diferenciarlas de los algoritmos genéticos.

Algoritmos Genéticos:

Modelan el proceso de evolución como una sucesión de frecuentes cambios en los *genes*, con soluciones análogas a *cromosomas***.** El espacio de

soluciones posibles es explorado aplicando transformaciones a éstas soluciones candidatas tal y como se observa en los organismos vivientes: *selección*, *cruce, mutación*.

1.2 Breve Introducción a Redes Neuronales.

Una Red Neuronal Artificial (RNA) consiste en un conjunto de elementos de procesamiento, también conocidos como neuronas o nodos, los cuales están interconectados. También puede ser descrito como un gráfico dirigido en el cual cada nodo *i* mejora una función de transformación.

Las RNAs se pueden dividir en unidireccionales y recurrentes según su conectividad.

Una RNA es unidireccional si existe un método que numere todos los nodos en la red de tal manera que no hay conexión de un nodo con un número grande a un nodo con un número más pequeño. Todas las conexiones son desde nodos con números pequeños a los nodos con números más grandes.

Una RNA es recurrente si no existe dicho método de la enumeración.

La arquitectura de una RNA es determinada por su estructura topológica, la conectividad total y la función de transferencia de cada nodo en la red.

Existen algunas otras formas de definir a las redes neuronales; desde las definiciones cortas y genéricas hasta las que intentan explicar más detalladamente qué son las redes neuronales. Por ejemplo:

- Una nueva forma de computación, inspirada en modelos biológicos.
- Un modelo matemático compuesto por un gran número de elementos procesales organizados en niveles.

- Un sistema de computación compuesto por un gran número de elementos simples, elementos de procesos interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas.
- Redes neuronales artificiales son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico.

Para una descripción den detalle de las redes neuronales referirse a Fremman y Skapura(2000).

Ventajas que ofrecen las redes neuronales.

Las redes neuronales artificiales presentan un gran número de características semejantes a las del cerebro humano (Damián Jorge Matich). Por ejemplo, son capaces de aprender de la experiencia, de generalizar de casos anteriores a nuevos casos, de abstraer características esenciales a partir de entradas que representan información irrelevante, etc. Esto hace que ofrezcan numerosas ventajas y que este tipo de tecnología se esté aplicando en múltiples áreas. Entre las ventajas se incluyen:

- Aprendizaje Adaptativo.- Las redes neuronales artificiales tienen la capacidad de aprender a realizar tareas basadas en un entrenamiento o experiencia inicial.

- Auto-organización.- Una red neuronal puede crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje.
- Tolerancia a fallos.- La destrucción parcial de una red conduce a una degradación de su estructura; sin embargo, algunas capacidades de la red se pueden retener, incluso sufriendo un gran daño.
- Operación en tiempo real.- Los cálculos neuronales pueden ser realizados en paralelo; para esto se diseñan y fabrican máquinas con hardware especial para obtener esta capacidad.

En esencia las redes neuronales buscan representar el funcionamiento del cerebro humano, a medida que el tiempo y la tecnología avanzan es posible acercarnos más hacia esta representación, la cual debe permitirnos de alguna manera que se puede conseguir que las computadoras piensen o realicen análisis por sí mismas.

1.3 Evolución en Redes Neuronales.

Aunque las redes neuronales son un campo bastante avanzado, el entrenamiento de una red neuronal, para un problema específico es un problema en el que hay que estar probando los diferentes parámetros que definen la red (neuronas de la capa oculta, funciones de activación, valores de entrenamiento, métodos de selección, cruce, mutación, etc.). El evolucionar una red sería una especie de automatización de esto ya que en el caso más general es el algoritmo que se utilice en el entrenamiento el que llegará a una buena estructura de red.

A diferencia de los algoritmos de gradiente, estaríamos ante un problema de búsqueda del mínimo global que un AG lo puede hacer mejor y para tratar este tema tomaremos un problema de evolucionar redes con algoritmos de gradiente, Perceptrón Multicapa (PMC).

1.4 Análisis y diseño de la Red Neuronal a utilizar - PMC.

1.4.1 Entrenamiento evolutivo versus entrenamiento de Basado en el Gradiente.

Como se menciona en el punto anterior, los algoritmos de gradiente no garantizan alcanzar un mínimo global de la superficie de error, ante este inconveniente, un algoritmo evolutivo para los pesos de conexión de una red neuronal, puede manejar el problema de búsqueda global, de mejor forma en una superficie extensa, compleja, multimodal, y no diferenciable. Por ejemplo, la aproximación evolutiva se ha utilizado para entrenar a RNAs recurrentes, RNAs de alto orden y RNAs difusas.

Por otro lado, un AG se puede utilizar para entrenar a muchas redes sin importar si son unidireccionales, recurrentes, o de alto orden. La aplicabilidad general de la aproximación evolutiva ahorra muchos esfuerzos humanos en desarrollar diversos algoritmos del entrenamiento para diversos tipos de RNAs.

Gracias al entrenamiento evolutivo es más fácil generar redes con características más simples y mejores.

El entrenamiento evolutivo puede ser lento para algunos problemas en comparación con variantes rápidas del algoritmo de retropropagación (RP) y

de los algoritmos basados en el gradiente. Sin embargo, los AGs son generalmente mucho menos sensibles a las condiciones iniciales del entrenamiento. Buscan siempre para una solución global óptima, mientras que un algoritmo de la pendiente del gradiente puede encontrar solamente un grado óptimo local en una vecindad de la solución inicial.

Para algunos problemas el entrenamiento evolutivo puede ser perceptiblemente más rápido y más confiable que el algoritmo de RP.

Los resultados obtenidos con AGs son completamente diversos. En estos resultados se encontró que un método AG-RP, es una técnica que primero utiliza un AG y después aplica el algoritmo de RP, "este método, es igualmente eficiente a ciertas variantes del algoritmo de RP en redes muy pequeñas, pero es menos eficiente en redes más grandes.

Los problemas de prueba incluyeron el problema de XOR, los problemas del tamaño de codificación / decodificación. Sin embargo, han habido muchos otros documentos que muestran resultados excelentes usando algoritmos evolutivos y del gradiente híbridos de la pendiente.

La discrepancia entre dos resultados aparentemente contradictorios se puede atribuir por lo menos en parte a los diversos AGs y RP comparados. Es decir, si la comparación está entre un AG binario clásico y un algoritmo rápido de RP, o entre un AG rápido y un algoritmo clásico de RP. La discrepancia nos demuestra que ninguno es mejor que el otro, siempre si se habla en términos del mejor algoritmo de entrenamiento. El mejor siempre depende del problema presentado. En algoritmos generales, los algoritmos

híbridos tienden a mejores resultados que otros para una gran cantidad de problemas.

La diferencia entre la aplicación de estos algoritmos nos permite empezar con un estudio de cada uno de ellos y poder documentar los resultados que se obtengan para un análisis.

1.4.2 El Perceptrón Multicapa[1].

En 1958 Rosenblatt publica sus primeros trabajos acerca de su modelo neuronal denominado *perceptrón* juntamente con el algoritmo de convergencia. Una red de capa simple que presentaba varias restricciones, las clases de tareas a realizar eran muy limitadas. Marvin Minsky y Seymour Papert en 1969 documentaron resultados negativos en perceptrones. Aunque técnicamente correctos, Minsky y Papert enfocaron sus trabajos sobre una clase restringida de técnicas de maquina de aprendizaje, mostrando que tales aproximaciones no podían resolver problemas naturales y simples (G. Cybenko, 2001)[2].

Una respuesta al problema de ajustar los pesos si se tiene capas ocultas fue presentada por Rumelhart, Hinton y Willians en 1986. La idea central es que el error para las unidades de la capa o capas ocultas se determina por una retropropagación del error de las unidades de la capa de salida. Por esta razón esta regla de aprendizaje se la conoce como retro-propagación.

[1] El presente es resumido de los Apuntes de Inteligencia Artificial II de Aguilar (2004).

[2] Tomado de Aguilar (2004)

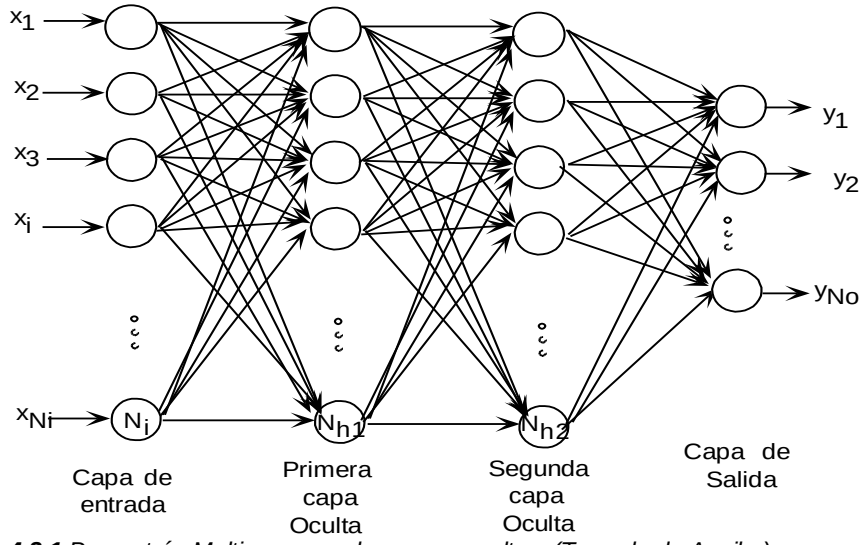


Figura 1.4.2.1 *Perceptrón Multicapa con dos capas ocultas. (Tomado de Aguilar)*

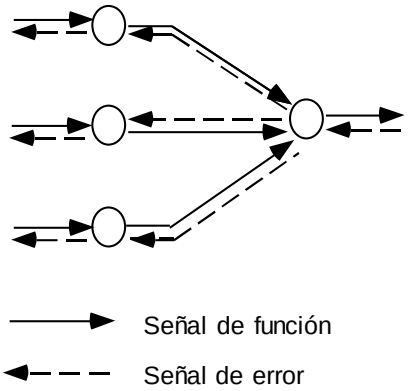


Figura 1.4.2.2 *Sentidos de propagación de las señales en un perceptrón multicapa. (Tomado de Aguilar)*

Este modelo, está formado por tres elementos o niveles de organización: las unidades sensoriales (entradas), las unidades asociativas (ocultas) y las unidades de respuesta (salidas). Las unidades asociativas actúan como pequeños procesadores diseñados para extraer información a partir de los ejemplos presentados.

Consideremos una RNA totalmente conectada como en la Fig. 1.4.2.1, en la que una neurona en una capa de la red está conectada a todas las neuronas de la capa siguiente y anterior. Esta es la estructura típica de un *perceptrón multicapa* donde identificamos dos tipos de señales (Fig. 1.4.2.2):

1. *Señal de Función.* Es la señal que se propaga desde la entrada (izquierda en la figura) hacia la capa de salida (derecha).
2. *Señal de Error.* Son las generadas por las neuronas de salida y que se retro propagaran en forma de ajuste de las conexiones sinápticas hacia la entrada, con el fin de ajustar la salida obtenida lo más fielmente a la salida deseada

1.4.2.1 Aprendizaje de la Red de Retropropagación (RBP).

Podemos definir un algoritmo de aprendizaje como sigue a continuación:

Iniciar los pesos de la red con valores aleatorios pequeños

Dar el patrón de entrada: $X(p) = (x_1(p), x_2(p), \ldots , x_n(p))$ [1]

Y especificar la salida deseada: $Y(p) = (y_{d1}(p), y_{d2}(p), \ldots , y_{dm}(p))$ [2]

Donde para cada ejemplo del conjunto de entrenamiento es necesario:

Calcular la salida actual de la red, para este fin es necesario en primer lugar determinar la salida en cada capa, en la dirección de propagación de la señal (hacia delante) hasta la capa final, este proceso se explica a continuación:

El primer cálculo se lo realiza para las neuronas de la(s) capa(s) oculta(s)

$$Sj(p) \quad y \quad Yj(p)$$

Luego para las neuronas de la capa de salida

$$Sk(p) \quad y \quad Yk(p)$$

Entonces se debe calcular los términos de error para todas las neuronas, en este caso, los pesos son ajustados por una suma proporcional al producto de la señal de error δ_k sobre la unidad k recibiendo la señal y sobre la unidad j enviando la señal.

$$\delta_k(p) = -\left(y_{dk}(p) - y_k(p)\right)\left(y_k(p)\right)' \quad [3]$$

Primero para las unidades de la capa de salida y luego para las unidades de las capas ocultas

$$\delta_j(p) = f'(s_j(p)) \sum_{k=1}^{N_o} \delta_k(p) w_{jk} \quad [4]$$

Finalmente se realiza la actualización de los pesos

Capa de salida

$$w_{jk}(t+1) = w_{jk}(t) + u\,\delta_k(p)\,y_j(p) \quad [5]$$

Capa(s) oculta(s):

$$w_{ij}(t+1) = w_{ij}(t) + u\,\delta j(p)\,y_i(p) \quad [6]$$

Una vez que la red ha aprendido, que ha definido bien el conjunto de pesos, entra en la etapa de funcionamiento, esto es, cada vez que se le de un ejemplo a la red, en este caso solo definido por los valores de entrada, X, esta debe determinar la salida Y que corresponda a ese X dado. Esto implica realizar solamente los cálculos hacia adelante, esto es:

$$y_k(p) = f(S_k(p)) \quad [7]$$

El nivel de activación y_k (p) producido en la entrada de la neurona k es función de S_k, por tanto:

$$S_k(p) = \sum_{j=0}^{N_h} w_{jk}(p) y_j(p) \quad [8] \text{ con}$$

$$y_j(p) = f(S_j(p)) \quad [9]$$

$$S_j(p) = \sum_{j=0}^{N_h} w_{ij}(p) x_i(p) \quad [10]$$

Cada neurona oculta o de salida de un perceptrón multicapa está diseñada para realizar dos cálculos: la propagación de la señal hacia adelante y el cálculo del vector gradiente necesario para la retro-propagación.

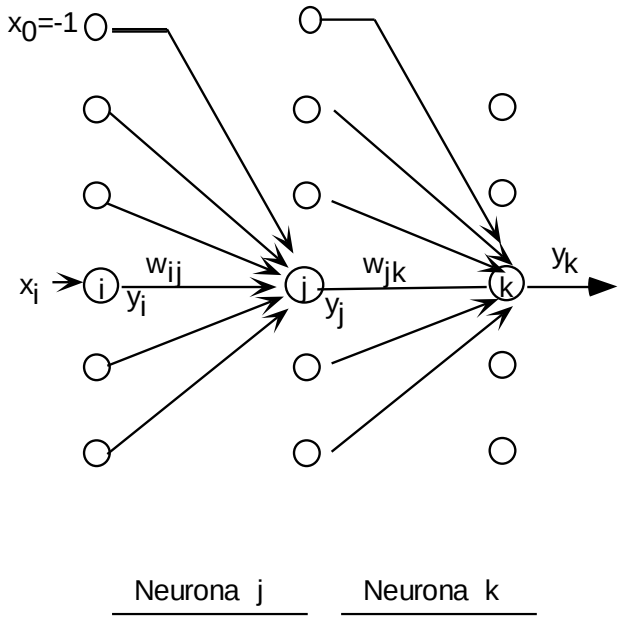


Figura 1.4.2.1.1 *Gráfico de señales en una neurona de salida k conectada a una neurona oculta j. (Tomado de Aguilar)*

La Fig. 1.4.2.1.1 donde se muestra la conexión entre las neuronas *i (entrada)*, *j (oculta)* y *k (salida),* la nomenclatura utilizada es la siguiente:

1. Los índices *i, j, k* denotan las neuronas ubicadas en las capas *i, j* y *k* respectivamente en el sentido de la propagación.
2. El símbolo $x_i(n)$ se refiere al elemento *i-ésimo* del vector o ejemplo número n de entrada.
3. El símbolo $y_{dk}(n)$ se refiere a la respuesta deseada o conocida del nodo k.
4. La salida de la red, para el nodo k se denota por $y_k(n)$.
5. La iteración p(n) se refiere al *n-ésimo* patrón presentado.

6. El símbolo e_k(n)se refiere al error a la salida de la neurona k, para el patrón n.
7. El símbolo *E(n)* se refiere a la suma cuadrática de los errores en la iteración *n-ésima*. La media sobre todas las muestras será Em.
8. El símbolo y_j(n) se refiere a la señal de salida de la neurona oculta *j.*
9. El símbolo w_{jk}(n) se refiere al peso de la conexión sináptica entre las neuronas j *y k* en la iteración *n.*

Se define un parámetro de aprendizaje u y uno de momento β.

De acuerdo a la explicación clara sobre el perceptrón multicapa a utilizar se puede recalcar el hecho que este tipo de red neuronal haya sido escogida para la realización de la tesis.

Debemos tomar en cuenta que la red no utilizará un algoritmo de retropropagación ya que lo que se pretende es demostrar el hecho que esta red puede ser evolucionada en su conjunto mediante el uso de un Algoritmo Genético, el cual será explicado y definido más adelante de igual manera que la parte correspondiente a redes neuronales.

1.5 Algoritmos Genéticos[3]

Los AGs son procedimiento de adaptación basado en la genética natural y la selección natural, son procesos interactivos que operan sobre un conjunto de individuos (población). Cada individuo representa una solución al problema a resolver. Esta solución se obtiene por medio de mecanismos de codificación y decodificación.

[3] El presente es resumido de los Apuntes de Inteligencia Artificial II de Aguilar (2004).

Primero la población se debe generar de forma aleatoria. Después de este proceso se evalúa a cada uno de los individuos, lo cual se hace con la función de aptitud, una medida de su error de acuerdo al problema presentado. Con este valor el algoritmo realizará la búsqueda.

Es necesario tomar en cuenta que el algoritmo realiza un balance de la búsqueda lo cual se denomina “estado de selección” y realiza una exploración de nuevas zonas del espacio de búsqueda lo que se denomina “estado de reproducción”.

El cálculo de un AG reside en las evaluaciones de las cadenas iniciales**.**

Normalmente la función objetivo es compleja, por ejemplo para una red neuronal que necesita inspeccionar docenas de patrones para calcular un valor objetivo simple, su desarrollo para saber si la respuesta que da la red se acerca a la realidad o salida conocida, puede ser complejo.

Los operadores como la selección, mutación, cruce son de complejidad lineal y trabajan a velocidades constantes para un problema dado.

Un AG tradicional trabaja con cadenas binarias de longitud fija y aplicando aptitudes proporcionales a la selección, La población inicial se genera siguiendo una distribución uniforme dada aleatoriamente, es decir cada posición en cada cadena contiene un 0 o un 1 con la misma probabilidad (0.5). El modelo de evolución es representado por generaciones sucesivas de poblaciones, así una nueva población reemplaza completamente a la anterior. El algoritmo se detiene cuando se ha alcanzado un número predefinido de generaciones.

Finalmente, la evaluación de cualquier cadena dada se hace mapeando el genotipo (Ej. la cadena de bits) al fenotipo (su valor real) por medio de una operación binaria a decimal para cada grupo de bits del genotipo representando un parámetro del problema. El modelo de un AG es el siguiente:

1.- Generar la población inicial G(0) aleatoriamente e inicializar i(0)

2.- REPETIR

a. Evaluar cada individuo de la población;

b. Seleccionar los padres desde G(0) basados en sus características en G(i);

c. Aplicar operadores de búsqueda a los padres y producir un descendiente desde G(i+1);

d. i = i+1;

3.- HASTA "criterio de terminación" está satisfecho.

Dentro de las características más predominantes se encuentran:

No dependen de la información del gradiente y no son así absolutamente convenientes para los problemas donde es inasequible o muy costosa la información a obtener o estimar. Pueden incluso ocuparse de problemas donde no hay disponible una función objetiva explícita y/o exacta. Estas características los hacen mucho más fuertes que otros algoritmos de búsqueda.

1.5.1 Representación de una solución.

Es necesario el conocer como se debe representar el espacio de búsqueda.

Codificación.

Como para muchos métodos de aprendizaje y búsqueda la codificación de las soluciones es un factor central. La mayoría de las aplicaciones de AG usan cadenas de bits de longitud fija y de orden fijo para codificar las soluciones candidatas, aunque existen otros tipos de codificación los cuales han sido utilizados en los últimos años.

Codificación Binaria.

Es la codificación más generalizada, además mucha de la teoría existente de los AG se basan en esta asunción codificación binaria de orden y longitud fijas.

Holland dio una justificación teórica para usar codificación binaria. El comparó dos codificaciones con aproximadamente la misma capacidad de conducción de la información, una con un número pequeño de aleles y una cadena larga (cadenas de bits de longitud 100) y la otra con un mayor número de aleles y una cadena corta (ej. Cadenas de decimales de longitud 30).

Holland argumentó que el primero sigue un grado más alto de paralelismo implícito que el segundo, ya que el primero contiene más esquenas que el segundo (11^{30} contra 3^{100}). El número de esquemas se calcula usando $(c+1)^4$ donde c es la cardinalidad del alfabeto y l la longitud de la cadena[4].

4 La razón por la que se suma uno a la cardinalidad es porque en los esquemas se usa un símbolo adicional (normalmente el asterisco o el símbolo de número) para indicar que “no nos importa” el valor de esa posición.

Entre las desventajas están que, la codificación binaria no es natural y es pesada para muchos problemas. (ej. Evolución de los pesos para una red neuronal).

En cuanto tiene que ver con la codificación binaria, la cual presenta problemas en el momento de su utilización conocido como el problema de los picos de Hamming, problema que será explicado más adelante (ver secciones 1.5.4 y 2.3).

Codificación de valores reales y de varios caracteres.

Para muchas aplicaciones, es más natural usar un alfabeto de muchos caracteres o números reales para formar el cromosoma.

El argumento del conteo del esquema de Holland parecería exhibir un peor rendimiento para codificaciones de múltiples caracteres que para codificación binaria.

Sin embargo varias comparaciones empíricas entre codificación binaria y codificación de varios caracteres o de valores reales han mostrado mejor rendimiento para este segundo caso. No obstante, el rendimiento depende mucho de la aplicación y de los detalles del AG usado (Mitchel, 19997) y no existen guías rigurosas para predecir que tipo de codificación trabaja mejor.

Lawrence Davis (1991), un investigador con mucha experiencia en problemas del mundo real recomienda usar siempre la codificación más natural para el problema y entonces divisar un algoritmo que pueda usar esa codificación.

1.5.2 Operadores

1.5.2.1 Operadores de Selección

El objetivo de este operador es transmitir y conservar la información que se considera valiosa a lo largo de las generaciones. La transmisión de la información se hace por medio de alguna estrategia de selección de los mejores individuos: el propósito es crear una población temporal *P´* de forma que los individuos con mayor evaluación estén representados un mayor número de veces que los demás, incrementándose la probabilidad de reproducir miembros que tengan una alta evaluación; es necesario sin embargo, el determinar un valor de aleatoriedad que permita la elección de individuos que aunque inicialmente no parezcan óptimos, puedan ser útiles en posteriores etapas. Pueden considerarse tres mecanismos fundamentales de selección:

- ***Ruleta:*** Cada componente de la población seleccionada para reproducirse es escogido de la población actual de manera independiente. Para ello se realiza una selección aleatoria en la que cada individuo tiene una probabilidad de ser seleccionado proporcional a su *aptitud*. Una descripción detallada del algoritmo que implementa este método se tiene en Goldberg D.E. (1989).
- ***Selección por ranking*****:** Ideado por Whitley L.D. (1987), consiste en construir la población seleccionada tomando en cuenta el orden de los valores de aptitud, es decir, el mejor individuo es seleccionado con probabilidad *p*1, el siguiente con *p*2, etc. Las formas de calcular

las probabilidades son diversas; la más extendida se denomina ranking lineal (Baker J.E. (1985)).

- ***Selección por torneo*:** Mecanismo de selección con menor componente de aleatoriedad que el de la Ruleta, ya que los individuos con bajo grado de aptitud pueden ser elegidos con menor probabilidad. Debido a su sencillez, el costo computacional es muy bajo: consiste en tomar un subgrupo de *t* individuos de la población actual, frecuentemente se toma *t*=2 (torneo binario); se genera un número aleatorio entre 0 y 1 de tal forma que si el número es menor que un cierto límite *K* (normalmente 0,75) el individuo seleccionado es el de mayor valor de bondad, en caso contrario el de menor. El parámetro *K* mide el nivel de aleatoriedad. Este algoritmo de selección comparte las mismas propiedades que la selección mediante ranking; de hecho, pueden parametrizarse de manera que su comportamiento sea globalmente equivalente (Bickle T. y Thiele L. (1995)).

1.5.2.2 Operadores de búsqueda.

Cruce.

Una de las formas de cruce más sencillas es el cruce basado en un punto, en el cual los dos individuos seleccionados pasan a jugar el papel de padres, son recombinados por medio de la selección de un punto de corte seleccionado aleatoreamente, para posteriormente intercambiar las secciones que se encuentran a la derecha de dicho punto.

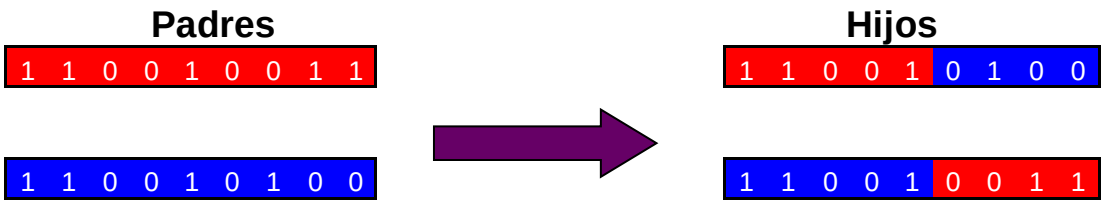


Figura 1.5.2.2.1 Método de un punto de cruce (Tomado de Aguilar).

Se han investigado otros operadores de cruce, teniendo en cuenta más de un punto de cruce. De Jong investigó el comportamiento del operador de cruce basado en múltiples puntos, concluyendo que el cruce basado en dos puntos, representaba una mejora mientras que añadir más puntos de cruce no beneficiaba el comportamiento del algoritmo. La ventaja de tener más de un punto de cruce radica en que el espacio de búsqueda puede ser explorado más fácilmente, siendo la principal desventaja el hecho de aumentar la probabilidad de ruptura de buenos esquemas.

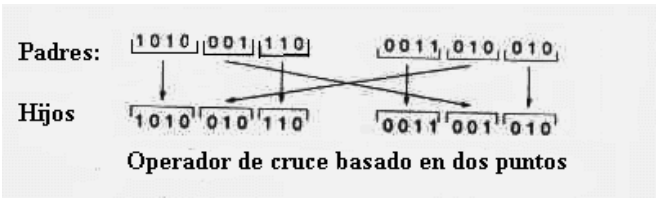


Figura 1.5.2.2.2 Método de dos puntos de cruce (Tomado de Aguilar).

En el operador de cruce basado en dos puntos, los cromosomas (individuos) pueden contemplarse como un circuito en el cual se efectúa la selección aleatoria de dos puntos.

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

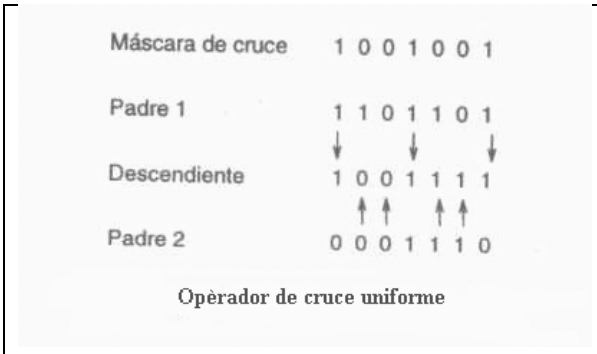


Figura 1.5.2.2.3 *Cruce uniforme generando una máscara de cruce. Cuando existe u 1 en la máscara de cruce, el bit del descendiente es copiado del un padre y cuando se tiene 0 se copia del segundo padre.*

En el denominado operador de cruce uniforme cada gen, en la descendencia se crea copiando el correspondiente gen de uno de los dos padres, escogido de acuerdo a una "máscara de cruce" generada aleatoriamente. Cuando existe un l en la "máscara de cruce", el gen es copiado del primer padre, mientras que cuando exista un 0 en la "máscara de cruce", el gen se copia del segundo padre. En la literatura, el término operador de cruce uniforme se relaciona con la obtención de la "máscara de cruce" uniforme, en el sentido que cualquiera de los elementos del alfabeto tenga asociada la misma probabilidad.

Mutación.

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

Existen muchas formas de efectuar la mutación:

- Con probabilidad pequeña p_m aleatoriamente se altera el contenido de las posiciones en la cadena (cromosoma).

Un seguro contra bits perdidos:

- En código binario, con p_m cambie 0 a 1 ó 1 a 0

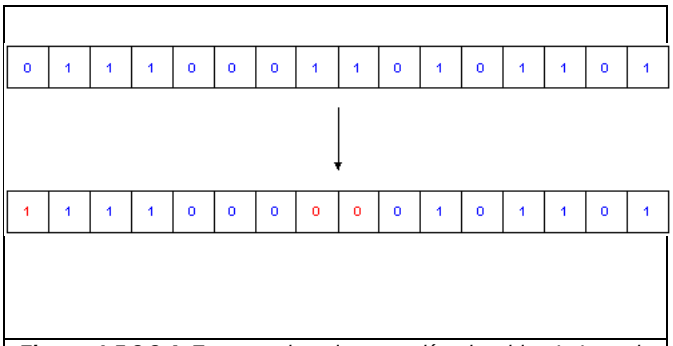


Figura 1.5.2.2.4 *En este tipo de mutación, los bits 1 1 en la posición determinada en forma aleatoria son cambiados a 0 0*

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

1.5.3 Resolviendo un problema: Genotipo y Aptitud.

En orden a aplicar los AG a un problema dado, la primera decisión a tomar se relaciona a la clase de genotipo que se adapte a las necesidades del problema. Esto significa que se debe decidir como se estructuran los parámetros del problema en la cadena de bits (cromosoma) (si se usara una cadena de bits). Esto implica:

Definir el número de bits por parámetro.

El rango de la decodificación binaria a parámetro decimal, si la cadena tiene

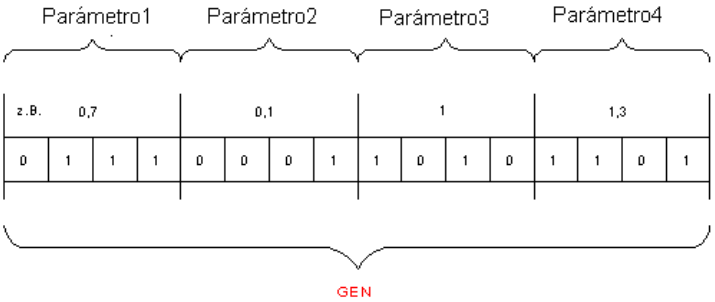


Figura 1.5.2.2.5 Un cromosoma formado por cuatro parámetros, que a su vez están formados por cuatro bits.

longitud constante o cambia durante las operaciones de los AG., etc. En aplicaciones tradicionales la cadena usa un alfabeto binario y su longitud es constante durante todo el entrenamiento.

Algunas aplicaciones de mayor complejidad necesitan cadenas de longitud dinámica o de un alfabeto diferente (no binario). Frecuentemente los genes de enteros o punto flotante son usados en muchos dominios, como en el entrenamiento de redes neuronales, funciones de optimización con un gran número de variables, problemas de reordemaniento como el del vendedor viajero, etc.

Un AG para poder determinar los pesos de una red neuronal codificará la información con un cromosoma que contenga el conjunto de pesos que definen todas las conexiones establecidas en un orden predefinido: por ejemplo para una red 3:2:1 (01 0.25 0.7 0.3 0.1 -.025 .95 –0.4).

La función de aptitud depende si se quiere maximizar o minimizar algún criterio.

El ambiente puede representar ruido en la evaluación (evaluación parcial por ejemplo).

La función de aptitud puede cambiar dinámicamente cuando el AG procede.

La función objetivo podría ser tan compleja que solo aproximaciones a su valor pueden calcularse.

La función objetivo podría localizar valores muy diferentes de la cadena en orden a facilitar el trabajo del operador de selección.

La función de aptitud debe considerar la limitación del problema. Normalmente soluciones no factibles pueden aparecer por lo que la función de aptitud debe destinar un valor pequeño de aptitud (si este un problema de maximización).

La función aptitud podría incorporar algunos sub-objetivos diferentes. Tales funciones multi-objetivo presentan problemas no convencionales cuando se usa en AG.

1.5.4 Genotipos no convencionales

Los AG han usado un genotipo binario, en el cual cada parámetro del problema es codificado en binario usando un número dado de bits. En número de bits para cada gen (parámetro) y el rango decimal en el cual ellos se codifican son usualmente los mismos pero no excluye la utilización de diferentes números de bits o rangos para cada gen.

Para evitar el llamado *hamming cliffs* el código de Gray es usado cuando se decodifica una cadena binaria. Con este código cualquier valor decimal adyacente es codificado con números binarios que solo difiere en un bit. Esto ayuda a los AG a hacer una aproximación agradable a un óptimo y normalmente reduce la complejidad de la búsqueda resultante.

Sin embargo en varias aplicaciones en los cuales AG han sido usados rinden muchos genotipos diferentes. Esto significa que la cadena puede ser construida sobre un alfabeto real o entero. Alfabetos enteros tienen propiedades interesantes para problemas de reordenamiento como el vendedor viajero y otros. Genes de punto flotante son muy útiles para optimización numérica (por ejemplo entrenamiento de redes neuronales).

Estos alfabetos de alta cardinalidad mejoran la búsqueda reduciendo el número de cadenas mal construidas resultantes de las aplicaciones de cruce y mutación. También cuando un número muy grande de parámetros ha sido optimizado es útil confiar en cadenas cortas (también porque la población y el número de pasos son menores).

Si el genotipo pude expresarse como una cadena, es necesario decidir para una aplicación dada si la longitud es constante o variable para cada cadena en la misma población. También en ambientes dinámicos (la función de aptitud cambia como la búsqueda procede) El uso de individuos diploides es una decisión interesante. En un individuo diploide, dos cromosomas homólogos describen el mismo conjunto de parámetros. Cuando se decodifica la cadena, alguna clase de operador dominante se necesita para decidir el valor del parámetro del problema (gene).

Ejemplos de genotipos sofisticados:

- Un individuo puede ser un árbol de símbolos (semejante a aplicaciones en programación genética)
- El individuo es una combinación de cadenas y árboles.

- Cadenas y un árbol.
- Algunas partes de la cadena pude evolucionar y otras no en diferentes momentos de la búsqueda.
- El individuo puede ser una matriz de símbolos.
- El genotipo podría ser una cadena de símbolos que indican operaciones a ser ejecutadas. Un simulador interpreta los símbolos y destina valores de aptitud dependiendo de los resultados de la interpretación de la cadena.

1.6 Algoritmo Genético para la evolución de redes neuronales.

Tomando en cuenta la teoría sobre AG antes planteado para el problema propuesto es necesario considerar el procedimiento de este tipo de algoritmos, tomando en cuenta que en cada uno de los pasos a seguir se deben realizar algunas consideraciones que permitirán un desarrollo óptimo del problema.

1.6.1 Determinación del Algoritmo Genético a utilizar.

Para el problema de evolución de redes, usaremos un Algoritmo Genético utilizando: Codificación Gray que permite solucionar el problema de los picos de Hamming que se presenta en el método de codificación binaria, método de selección Ruleta, cruce uniforme (máscara de cruce), mutación y la función de aptitud para el problema de aprendizaje supervisado en el que se tiene tanto datos de entrada y salidas conocidas. (ver sección 2.3).

CAPITULO II: EVOLUCION DE LA RED NEURONAL MEDIANTE ALGORITMOS GENETICOS.

En el presente capítulo se toma en cuenta los conceptos de los aspectos más importantes que intervienen en el problema planteado, es decir clasificación de sismos.

También se presenta la herramienta que será utilizada para la implementación del proyecto.

Otro punto importante en el este capítulo es el detalle completo de cómo fue implementado el proceso de evolución de redes neuronales con algoritmos genéticos para nuestro problema específico.

Para el problema planteado de evolución de redes neuronales, tomaremos un ejemplo de aprendizaje supervisado, evolucionar redes neuronales utilizando algoritmos genéticos.

2.1 Definición del problema de clasificación de sismos.

Lo importante en la identificación y discriminación del tipo de evento es que permite un mejor análisis del estado del volcán, la finalidad es mejorar la confianza en el nivel de alerta, ante la posibilidad de una erupción (Aguilar y Troncoso, 2005). Por ejemplo se ha observado una fuerte relación entre la ocurrencia de enjambres de eventos de largo período (LP) y la presencia muy cercana de erupciones volcánicas, como es el caso del volcán El Chichón en México. En el volcán Galeras los eventos LP quizás constituyen la actividad más característica asociada con emplazamiento de domos (Gil y Chouet, 1997) y de igual forma en el volcán Guagua Pichincha, los enjambres de LPs son considerados indicadores de crecimiento y extrusión de domos (Villagómez, 2000). Aunque en el volcán Cotopaxi (hasta previo a 2001) y el Cayambe, este tipo de actividad no se considera como precursora de actividad eruptiva (Villagómez, 2000), aunque tampoco ha formado enjambres.

Marco referencial de la clasificación de sismos.

Aguilar y Troncoso (2005) al trabajar en el problema de clasificación de sismos toman en cuenta tres tipos de sismos que para el caso del volcán Cotopaxi representan alrededor del 95% de toda la actividad del volcán. Como nuestro objetivo es la evolución de redes, tomaremos este mismo

problema de clasificación cuyos datos han sido proporcionados por Jorge Aguilar, con 50 datos por tipo de evento. En lo que sigue resumiremos las características de los eventos y la definición de variables del trabajo de Aguilar y Troncoso (2005).

CLASIFICACIÓN DE LOS SISMOS.

2.1.1 Eventos de Largo Período (LP).

Los eventos de largo período tienen una forma de onda que muestra un inicio muy emergente, muchas veces rico en alta frecuencias, coda monocromática carente de una onda S definida claramente, con duraciones de la señal que oscilan entre pocos segundos hasta algo más de un minuto. En tanto el contenido de frecuencias es restringido a una banda estrecha entre 1 – 3 Hz (Martín et al., 1993) o 1 - 5 Hz (McNutt, 1996), no poseen una única forma espectral definida pero son simples con picos espectrales centrados a una o varias frecuencias (Figura 2.1.1.1). Sin embargo, el contenido espectral de esta señal puede variar de volcán a volcán, e incluso dentro del mismo sistema volcánico (Ibáñez et al., 2000).

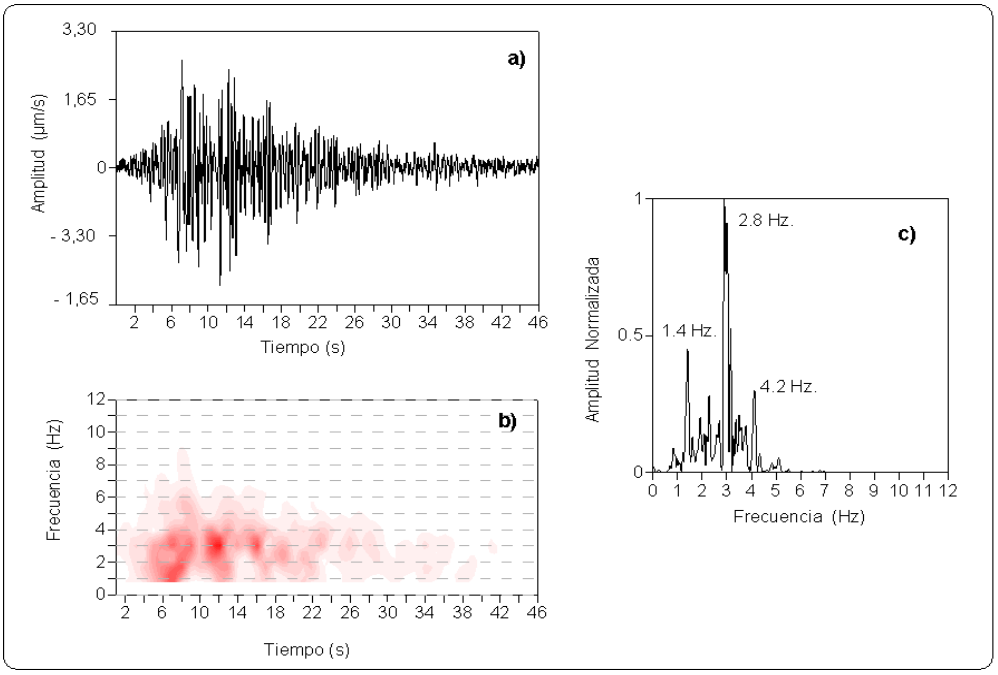


Figura 2.1.1.1 Sismo de largo período clásico.a) Forma de onda de un evento registrado el 18 de agosto de 2001 a las 21:40(GMT) en la estación VC1, b) espectrograma usando ventanas de 1s y c) espectro de la señal con el valor de los picos sobresalientes. (tomado de Troncoso, 2005)

2.1.2 Eventos híbridos

Los eventos híbridos muestran inicio emergente rico en altas frecuencias (6,0 – 8,9 Hz) con duraciones entre 4 y 6 s. La coda es característica de un evento de largo período, es decir, contiene bajas frecuencias (2,4 – 3,0 Hz). Generalmente el arribo de la onda P es emergente y el arribo de la onda S no es claro. En el espectrograma se muestra el predominio de dos picos espectrales, una de baja y otra de alta, pero el contenido de energía no sobrepasa los 10 Hz (Figura 2.1.2.1). Con respecto a las características espectrales se puede resaltar lo siguiente (Troncoso, 2005):

1. La frecuencia dominante varía, pero en la mayoría de los casos el pico dominante se encuentra en las bajas frecuencias (2,4 – 4,2 Hz).

2. El ancho de banda puede ser similar a un evento de largo período (1 – 2,4 Hz), a un volcano tectónico (6,8 – 6,5 Hz) o mostrar carácter intermedio (3 – 3,2 Hz).
3. El factor de calidad varía entre 0,62 y 2,66.
4. El coeficiente de atenuación de las ondas de coda varía entre 0,81 – 0,49.

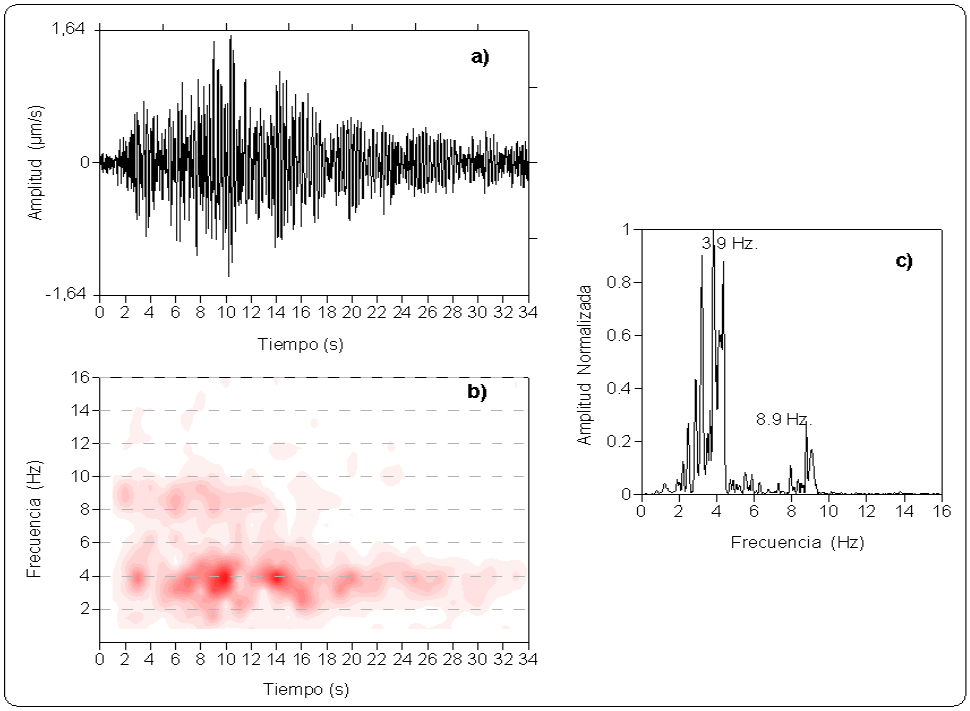


Figura 2.1.2.1 Evento híbrido del volcán Cotopaxi. a)Forma de onda de un evento registrado el 23 de marzo de 2002 a las 08:09(GMT) en la estación VC1, b) espectrograma usando ventanas de 1 s y c) espectro de la señal con el valor del pico principal. (tomado de Troncoso, 2005)

2.1.3 Eventos Volcano - Tectónicos

Los sismos volcano-tectónicos (Figura 2.1.3.1) son de duración variable (pocos segundos a minutos), su inicio suele ser más o menos impulsivo, es decir, se puede identificar el arribo de la onda P y es posible identificar el arribo de la onda S. Su forma de onda es característica, representada por las

llegadas claras de los paquetes de las ondas P y S, así como la contribución de ondas superficiales y un decaimiento final en forma de exponencial decreciente que es la coda (Ibáñez et al., 1997 y 2000).

El contenido espectral es amplio y su contenido de frecuencia es mayor a los 5 Hz (Wasserman, 2002), por lo que se llama también evento de alta frecuencia (HF-high frequency).

Estos sismos presentan magnitud menor a los sismos de origen puramente tectónico, por lo que no es común observar magnitudes mayores a 4 grados (Ibáñez y Carmona, 2000).

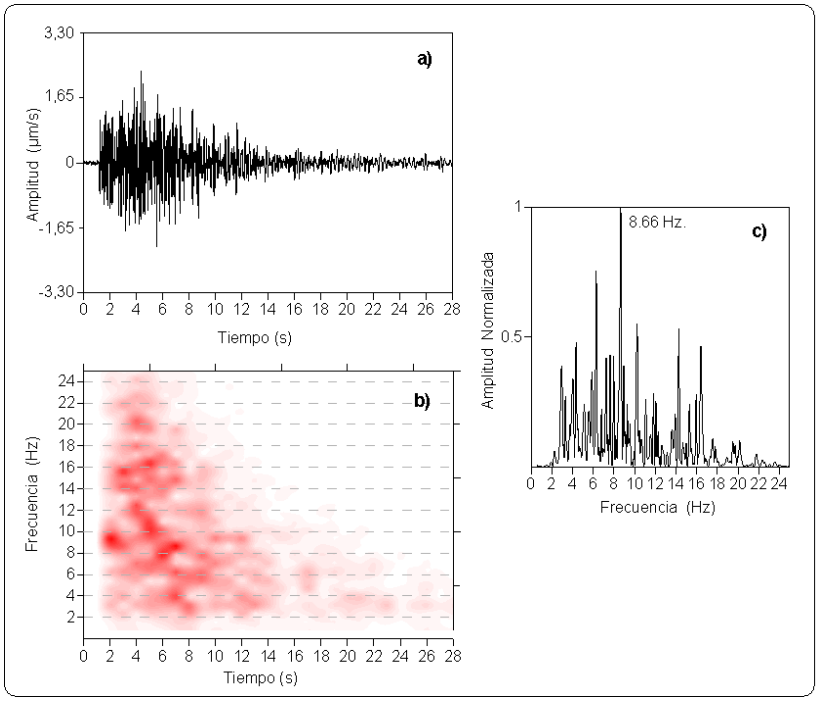


Figura 2.1.3.1 Sismo volcano-tectónico del volcán Cotopaxi. a) Forma de onda de un evento registrado el 28 de octubre de 2001 a las 10:35(GMT) en la estación VC1, b) espectrograma usando ventanas de 1s y c) espectro de la señal con el valor del pico principal. (tomado de Troncoso, 2005).

2.1.4 Definición de variables.

De esta forma, Aguilar y Troncoso (2005), definen cuatro variables que identifican a estos tipos de sismos: Todas son extraídas del espectro del sismo: Las dos primeras variables están definidas por el ancho de la señal al 25% y 50% de su amplitud máxima y las dos restantes corresponden a la frecuencia a la que se dan los dos picos más importantes. Al trabajar con redes neuronales, con un PMC llegan a un 95% de aciertos en los datos de validación, este porcentaje se mejora a 98% para una red probabilística, para estos resultados trabajan con 150 datos en los que el 50% son para entrenamiento y el 50% para datos de prueba.

2.2 Herramienta a utilizar e implementación del algoritmo.

Matlab

MATLAB es un entorno flexible que permite realizar análisis avanzados, visualizar datos y desarrollar algoritmos empleando menos tiempo que con otros lenguajes de programación. El lenguaje abierto de MATLAB permite compartir rápidamente ideas y desarrollar soluciones. Las ventajas que ofrece MATLAB frente a otros lenguajes de programación tradicionales son su potencia, flexibilidad, la capacidad de generar código fuente o ejecutables.

Además, MATLAB destaca por su capacidad para realizar componentes aislados para su uso externo en Office, Visual Basic, desarrollos C, Web, etc.

Asimismo, ofrece funcionalidades tales como la programación basada en

matrices, funciones matemáticas avanzadas y lenguajes flexibles simplificados, que permiten centrarse en resolver problemas más que en programar.

También podemos destacar de este sistema que implementa un toolbox de redes neuronales, el cual nos permitirá realizar las pruebas antes mencionadas con el modelo de Perceptrón Multicapa, con el que realizaremos las comparaciones de los resultados obtenidos luego del desarrollo del Algoritmo Genético.

Neural Network Toolbox 4.0.4

Diseño y simulación de redes neuronales.

El toolbox de Matlab proporciona un sistema completo de funciones y una interfaz gráfica para el diseño, la puesta en práctica, la visualización, y la simulación de redes neuronales. Además soporta las arquitecturas más comunes como son de aprendizaje supervisado y no supervisado, cuenta también con un sistema comprensivo de entrenamiento y funciones de aprendizaje.

2.3 Análisis del problema.

El tema propuesto de tesis tiene como objetivo el proceso de evolución de redes neuronales aplicando algoritmos genéticos, de esta forma y con la teoría antes expuesta el proceso mismo de desarrollo de la aplicación que nos permitirá determinar y documentar los resultados sobre el problema específico a solucionar.

Evolución en Redes Neuronales Artificiales.

Introducción.

A nivel de evolución de redes neuronales artificiales (RNA), (Xin Yao, 1998) describe tres diferentes niveles: pesos de conexión, arquitecturas, y reglas de aprendizaje.

La evolución de los pesos introduce un acercamiento adaptativo y global al entrenamiento, especialmente en el aprendizaje reforzado y en el paradigma recurrente del aprendizaje donde los algoritmos basados en el gradiente experimentan a menudo grandes dificultades.

La evolución de arquitecturas permite a las RNAs adaptar sus topologías a diversas tareas sin la intervención humana, y proporcionan así un acercamiento al diseño automático de redes neuronales, mientras que las estructuras y los pesos de conexión de estas pueden ser desarrollados.

La evolución de las reglas de aprendizaje puede ser considerada como un proceso de “aprendiendo a aprender”, en donde la adaptación de las reglas de aprendizaje se alcanza con la evolución. Puede también considerarse como un proceso adaptativo del descubrimiento automático de reglas de aprendizaje.

Ya que la evolución de las redes neuronales está basada en los tres niveles descritos, detallamos un poco más estos aspectos, basados en el trabajo de Xin Yao (1998).

Formas o niveles de la evolución de RNA.

1) Evolución de los pesos de conexión.

El entrenamiento de los pesos en una RNA se establece generalmente como minimización de una función de error, este error se determina entre la salida de la red calculada con estos pesos que definen la red y los valores objetivos o salidas deseadas que son parte de los datos de entrada. La mayoría de los algoritmos de entrenamiento, tales como retropropagación y algoritmos de gradiente, se basan en la pendiente del gradiente. Se han tenido algunos usos acertados del algoritmo de retropropagación en varias áreas, pero este tiene desventajas justamente por el uso de la pendiente del gradiente. Este algoritmo regularmente converge a un mínimo local de la función de error, lo que implica que por lo general no se alcanza el mínimo global. Si la función de error es multimodal y/o no diferenciable no es posible aplicar el algoritmo ya que se requiere justamente de estimación de derivadas.

Una forma para superar los defectos de los algoritmos basados en la pendiente del gradiente es utilizar RNA Evolutivas para establecer un proceso de entrenamiento como la evolución de los pesos de conexión en el ambiente determinado por la arquitectura teniendo como objetivo el aprendizaje.

Los AGs pueden utilizarse con eficacia en la evolución para encontrar un sistema óptimo cercano de pesos de conexión global sin el cálculo de la información de la pendiente del gradiente. La aptitud de una RNA se puede definir según diversas necesidades. Dos factores importantes que aparecen a menudo en la función de aptitud son el error calculado entre los valores objetivos y las salidas reales, y el otro factor es la complejidad de la RNA.

Por ejemplo de los resultados, para una red más compleja (mayor número de neuronas), se tiene mayores errores; para el problema planteado existe una relación entre a mayor número de neuronas en la capa oculta el valor de error aumenta, es decir que se tiene un menor número de aciertos, además también después del entrenamiento realizado en la red se logró determinar que con un número de neuronas de capa oculta los resultados obtenidos no son los óptimos.

Dentro del caso de entrenamiento con algoritmos basados en el gradiente, la función de la aptitud (o error) debe ser diferenciable, pero para un AG no tiene que ser diferenciable ni siquiera continua puesto que los AGs no dependen de la información del gradiente. Ya que el tratamiento de los AGs pueden ser largo, complejo, no diferenciable y puesto que los espacios multimodales, son el caso típico en el mundo real, la investigación y el uso de estos algoritmos han permitido la evolución de los pesos de la conexión.

El acercamiento evolutivo para el entrenamiento de los pesos en RNAs consiste en dos fases importantes.

- La primera fase es decidir la representación de los pesos de conexión, si es en la forma de secuencias binarias o no.
- Segundo es el proceso evolutivo simulado por un AG, en el cual los operadores de la búsqueda tales como cruce y mutación tienen que ser decididos conjuntamente con el esquema de la representación.

Existen diferentes representaciones y operadores de búsqueda los cuales permiten un funcionamiento completamente diverso dentro del entrenamiento.

2) Evolución de las arquitecturas.

Dentro del desarrollo de la evolución de las redes neuronales artificiales se asume que la arquitectura de una RNA está predefinida y fijada durante la evolución de los pesos de conexión.

La arquitectura de una RNA incluye su estructura topológica, conectividad, y la función de transferencia de cada capa en la RNA. El diseño de la arquitectura es muy importante en el uso acertado de RNAs porque la arquitectura tiene impacto significativo en las capacidades de tratamiento de la información de una red.

Ya que la tarea es aprender, una RNA con solamente pocas conexiones y nodos lineales posiblemente no puedan realizar la tarea definida, debido a su capacidad limitada, mientras que una RNA con una gran cantidad de conexiones y de nodos no lineales podría disminuir el error en los datos del entrenamiento y tener una gran capacidad de generalización.

Hasta ahora, el diseño de la arquitectura para un problema dado, sigue siendo un trabajo de expertos. Depende esencialmente de la experiencia y de un proceso del ensayo y error. No hay manera sistemática de diseñar una arquitectura óptima, en forma automática.

En este sentido, se habla de algoritmos constructivos y destructivos. La investigación sobre estos representa un esfuerzo hacia el diseño automático de arquitecturas. De manera general, un algoritmo constructivo comienza con una red mínima (red con el número mínimo de capas, de nodos, y de conexiones ocultas) y agrega nuevos capas, nodos, y conexiones cuando es necesario durante el entrenamiento mientras que un algoritmo destructivo

hace lo contrario, comienza con la red máxima y suprime capas, nodos, y conexiones innecesarios durante el entrenamiento.

El diseño de arquitecturas óptimas para una RNA se puede definir como un problema de búsqueda en el espacio de topologías donde cada punto representa una arquitectura. Dado algunos criterios de funcionamiento, el error más bajo del entrenamiento, la complejidad más baja de la red, etc., sobre arquitecturas, el nivel de funcionamiento de todas las arquitecturas forman una superficie discreta en el espacio.

El diseño óptimo de la arquitectura es equivalente a encontrar el punto más alto en esta superficie. Hay varias características de una superficie que hacen que los AGs sean mejores en la búsqueda de la superficie que los algoritmos constructivos y destructivos.

Esas características son:

- La superficie es infinitamente grande ya que el número de nodos y de conexiones posibles es ilimitado.
- La superficie es no diferenciable ya que los cambios en el número de nodos o de conexiones son discretos y pueden tener un efecto discontinuo en el funcionamiento de las RNAEs.
- La superficie es compleja ya que el diseño de una arquitectura a su funcionamiento es indirecto, y dependiente en el método de evaluación usado.
- La superficie es engañosa ya que las arquitecturas similares pueden tener funcionamiento diferente.

- La superficie es multimodal ya que diversas arquitecturas pueden tener funcionamiento similar.

Al igual que la evolución de los pesos de conexión, dos fases importantes implicadas en la evolución de arquitecturas son: El esquema de representación del genotipo de arquitectura y el AG utilizado para desarrollar la arquitectura de una RNA.

Una de los puntos claves en la codificación de las arquitecturas de una RNA es que se debe decidir cuánta información sobre una arquitectura se debe codificar en el cromosoma. En un caso extremo, todos los detalles, cada conexión y el nodo de una arquitectura se pueden especificar por el cromosoma.

Esta clase de esquema de la representación se llama codificación directa. En el otro extremo, solamente los parámetros más importantes de una arquitectura, tales como el número de capas ocultas y de nodos ocultos en cada capa se codifican. Otros detalles sobre la arquitectura se dejan al proceso del entrenamiento para decidir. Esta clase de esquema de la representación se llama codificación indirecta.

Una investigación considerable sobre el desarrollo de las arquitecturas de una RNA se ha realizado en los últimos años. La mayoría de las investigaciones se han concentrado en la evolución de las estructuras topológicas de una RNA. Se ha hecho relativamente poco en la evolución de las funciones de la transferencia de cada nodo, dejando solamente la evolución simultánea de ambas estructuras y funciones topológicas de la transferencia del nodo.

3) Evolución de reglas de aprendizaje.

Un algoritmo de entrenamiento de una RNA puede tener un funcionamiento diferente cuando se está aplicando a varias arquitecturas. En el diseño de los algoritmos de entrenamiento, lo más fundamental son las reglas de aprendizaje usadas para ajustar los pesos de conexión, lo cual depende del tipo de arquitecturas usadas en la investigación. Algunas variantes de la regla de aprendizaje de Hebbian se han propuesto para ocuparse de diversas arquitecturas.

Sin embargo, diseñar una regla óptima de aprendizaje llega a ser muy difícil cuando hay poco conocimiento anterior sobre la arquitectura de una RNA. Es necesario desarrollar una manera automática y sistemática para adaptar la regla de aprendizaje a una arquitectura y a la tarea a ser realizada.

Diseñar una regla de aprendizaje implica manualmente a menudo que algunas asunciones, que no son necesariamente verdades en la práctica, tienen que ser hechas.

De hecho, cuán necesario es en una RNA su capacidad de ajustar su regla de aprendizaje según su arquitectura y la tarea a ser realizada. Es decir una RNA debe aprender que su regla de aprendizaje es dinámica, antes que tener que diseñarla y fijada manualmente.

La relación entre la evolución y el aprendizaje es compleja. Se han propuesto varios modelos solamente la mayoría de ellos con la aplicación cómo el aprender puede dirigir la evolución y la relación entre la evolución de arquitecturas y de la de los pesos de la conexión. La investigación en la evolución de las reglas de aprendizaje todavía está en sus primeros tiempos.

Esta investigación es importante no solamente en el abastecimiento de una manera automática de optimizar reglas de aprendizaje y en modelar la relación entre aprender y la evolución, es también importante para modelar el proceso creativo desde aprender nuevamente desarrollado las reglas lo cual puede ocuparse de un ambiente complejo y dinámico.

Técnicas de Evolución de Redes NA.

1) Evolución con Algoritmos Genéticos.

Los algoritmos genéticos parten de una población inicial donde cada individuo se representa con un código genético (normalmente una secuencia de bits) en la que se encuentra codificada su información. Sobre esta población se realiza una serie de operaciones, en primer lugar se seleccionan parejas de soluciones para que se reproduzcan (a este proceso se le llama cruce), siendo los hijos una mezcla del código genético de los padres. A continuación se producen una serie de mutaciones que alteran los genes de los “recién nacidos” y por último de entre toda la población se eligen aquellos que van a sobrevivir desechándose el resto (la población en un algoritmo genético típico permanece constante en todas las iteraciones). Tanto a la hora de la reproducción, como en el momento de elegir las soluciones supervivientes en cada iteración, se favorece a aquellos individuos que según la función de evaluación sean más fuertes. El algoritmo terminará cuando se llegue a un número de iteraciones seleccionado previamente, o cuando se observe que tras una serie de iteraciones no se ha detectado ninguna mejora en la población.

Para desarrollar el algoritmo genético en base el problema planteado, clasificación de señales de dos tipos de eventos sísmicos, es necesario seguir los siguientes pasos:

a) Codificación de las soluciones.

El primer paso a realizar es codificar las soluciones de forma que queden representadas por una secuencia constante de bits. La elección de esta codificación es la más acertada especialmente si estamos tratando un problema de optimización. Por ejemplo una codificación directa de los números enteros puede traernos problemas a la hora de que el algoritmo converja, ya que números consecutivos, como por ejemplo el 15 y el 16, al pasarlos a binario son "demasiado" diferentes (10000, 01111) con lo que una solución con el valor 15 en un campo, difícilmente llegará a evolucionar a una solución con el valor 16 (ya que debería cambiar simultáneamente todos los bits). A este problema se le conoce como "Picos de Hamming". Para disminuir su efecto podemos utilizar otras codificaciones. Por ejemplo codificando según los "Códigos de Gray" aseguramos que enteros consecutivos solo se diferenciarán en un bit.

La codificación Gray es un método para transformar una función de forma que las representaciones binarias de valores numéricos consecutivos diferencian por un solo bit. La codificación Gray ha sido utilizada en la codificación del número de neuronas ocultadas para conseguir abarcar los resultados del plano evolutivo *(Harri Niska 2004).*

Para ilustrar este concepto, aclaremos con el siguiente ejemplo. Consideremos el caso de una red 4:1:2, en este caso la codificación será:

Generando los pesos iniciales aleatoriamente, estos son generados de acuerdo al número de neuronas en capa de entrada, capa oculta y capa de salida; para el ejemplo propuesto se deben generar inicialmente, 6 pesos.

Estos pesos se generan aleatoriamente en el sistema desarrollado, estos valores de acuerdo a la función que los genera en Matlab (rand), nos determina valores entre 0 y 1.

Una vez generado este conjunto de pesos para una estructura de red inicial, como nuestro ejemplo planteado, estos pesos son primeramente transformados a código binario, y para su representación se ha determinado 5 bits para cada uno de los valores de los pesos.

Luego de que el conjunto de pesos ha sido codificado en forma binaria, y ya que de acuerdo a la teoría explicada anteriormente, existe un tipo de codificación propuesto para evolución de pesos, el cual permite solucionar los problemas que presenta la codificación binaria, este conjunto es transformado o codificado en la forma Gray.

La codificación Gray partiendo del código binario se realiza de la siguiente manera:

i) Binario a Gray.

Para pasar un número binario al código binario Gray, hay una regla fácil de implementar en un lenguaje de programación:

1. Un número en binario siempre empieza en 1, los ceros a la izquierda no cuentan; Pues en Gray también. Ej: 1000011110000 en binario se escribe 1xxxxXXXXxxxx.

2. Ahora nos fijamos en el segundo dígito. Si es igual al dígito anterior se pone un 0 (no cambia); Si es diferente como es el caso, pues el dígito anterior era un 1 y el que observamos un 0 se pondrá un 1 (cambia). Ej: El número del ejemplo anterior será: 11xxxXXXXxxxx.
3. En los casos sucesivos se repite el paso anterior, observando en el número binario 'natural' el dígito anterior al que se evalúa. Ej: El número del ejemplo anterior, pasado a código Gray será: 1100010001000.

Ejemplos.

1010 - 1111
111000 - 100100
011001 - 010101
110101010001 - 101111111001

ii) Gray a Binario.

Hacer el cambio contrario es simplemente invertir lo que hace el anterior, de forma que si se encuentra un cero (siempre que no sea al principio) se debe poner la cifra anterior; En cambio si pone un 1 es porque la cifra ha cambiado así que si había un 0 ahora se pone un 1 y viceversa.

De esta manera se obtiene la codificación Gray, la cual ha sido utilizada para el desarrollo de la tesis. Por otro lado el algoritmo que se debe implementar para esta codificación es bastante sencillo, este algoritmo es el que se encuentra implementado en la tesis:

Variables

n Número de bits de la cadena de caracteres.

i Contador.

G[n] Vector que contiene la cadena de caracteres gray.

b[n] Vector que contiene la cadena de caracteres binario.

XOR(bit,bit).

1 1 →0

0 0 →0

1 0 →1

0 1 →1

```
Función de binario a gray {
        i=1;
        G[i]=b[i];
        While (i≠n) {
            i=i+1;
            G[i]=xor(b[i-1],b[i]);
        }
}
Función de gray a binario {
        i=1;
        b[i]=G[i];
        While (i≠n) {
            i=i+1;
            b[i]=xor(b[i-1],G[i]);
        }
}
```

Para la estructura de red 4:1:2, como se explicó anteriormente se generan 6 pesos, y así mismo para la codificación se estableció que cada uno de los pesos estarían representados por 5 bits, es decir que cada cadena de bits generada para cada conjunto de pesos en el ejemplo planteado sería de 30 bits.

Código binario:

0 0 0 1 1 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 1 0 1 0 0 0 1 0 0 0

Código Gray:

0 0 0 1 0 1 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 1 0 0

Elección de la población inicial:

En un algoritmo genético clásico se trabaja con una población constante. Cuanto mayor sea la población más exhaustivo será el estudio del problema, pero como esto afecta directamente, también será mayor el tiempo necesario para el cálculo, y para la convergencia del algoritmo.

Existe una variante de los algoritmos genéticos (algoritmos generacionales) en la que la población tiene un margen de elasticidad que le permite cambiar durante la ejecución. Lamentablemente no hay una teoría clara sobre como debe variar esta población.

En el caso específico, tomaremos poblaciones aleatoriamente generadas con valores de pesos entre 0 y 1, y con un conjunto inicial de población el cual puede ser variable por el usuario de acuerdo al problema planteado, estos valores para las pruebas realizadas han sido tomados entre (10 - 100)

individuos, soluciones generadas en forma aleatoria tal como se indicaron en el punto referente a la codificación.

b) Elección de la función de evaluación.

La función de evaluación debe reflejar qué características queremos que tenga nuestra población, para que nuestro algoritmo sea capaz de distinguir cuáles son los mejores individuos. Un algoritmo genético toma todas sus decisiones según esta función por lo que elegirla convenientemente es un factor clave. *Lamentablemente hay pocas indicaciones sobre cómo elegirla, ya que depende por completo del problema a tratar.*

Por lo que un buen conocimiento del problema ayudará a definirla de mejor forma.

Un caso específico en el que se está investigando actualmente son las funciones de coste colectivas, estas funciones evalúan como se complementan los individuos unos con otros, y no solo como es cada uno de ellos por separado. Usar este tipo de funciones requiere un tratamiento especial para detectar y premiar a los grupos de individuos que mejor se complementen entre sí. De nuevo la idea viene inspirada por la naturaleza. Una hormiga (termita, o abeja) puede parecernos un organismo muy limitado, sin embargo, hay pocas cosas tan eficientes como un hormiguero. De la misma forma podemos tratar de diseñar algoritmos genéticos cuyo objetivo no sea encontrar al mejor individuo posible, sino una población que cooperando obtenga todavía mejores resultados. Lamentablemente los algoritmos genéticos con este tipo de funciones son actualmente más un objetivo, que una realidad.

Generalmente para problemas de optimización se elige una función constante, mientras que en Inteligencia Artificial se pueden utilizar funciones que dependan del entorno para que de esta forma el sistema pueda adaptarse. Para el problema de evolución de redes, ya que estamos en aprendizaje supervisado, la función de aptitud debe considerar la salida del sistema y la salida conocida. Entre más parecida sea la salida de la red a la salida deseada, mayor aptitud tendrá la red y menor en caso contrario. De esta forma, para el problema de clasificación de sismos, la función de aptitud debe considerar la salida conocida, que representa la clase de evento y la salida que da el sistema, comparando estos datos, para algunos eventos, se tiene una medida del error para cada red neuronal y las redes con menores errores serán mejores que aquellas que den errores mayores.

Concretamente, la evaluación de la red neuronal se la realiza, tomando en cuenta los datos de entrada y los pesos de las redes generadas de manera aleatoria, se calcula la sumatoria del producto entre los datos de entrada o variables y los pesos de cada conexión en entre la capa de entrada y la capa oculta, a este resultado se aplica a función de transferencia la cual de acuerdo a la teoría indicada, normalmente se aplica una función tangencial; luego de realizado este primer cálculo en la capa oculta obtenemos resultados provisionales los cuales servirán para el último cálculo en la capa de salida, en donde, se calcula de igual forma la sumatoria del producto de los valores obtenidos en la capa oculta por los pesos de conexión entre la capa oculta y la capa de salida, en la cual se aplica nuevamente una función que de acuerdo a la teoría se debe aplicar una función lineal, a este

resultado final aplicamos una función de Matlab (round), el cual nos permite obtener valores enteros y de esta forma poder comparar estos resultados con el conjunto de *Ydeseado* y así determinar si existe acierto o no en el cálculo de la función de aptitud.La salida deseada está dada por el tipo de evento: LP, VT y HB. El cálculo de la respuesta de la red evolucionada, PMC, lo hacemos con la ecuación $S_j(p)=\sum_{j=0}^{N_h} w_{ij}(p)x_i(p)$ presentada en la teoría del PMC (ver sección 1.4.2.), de tal forma que vamos a tener estos tres tipos de eventos. Si el evento determinado por la red es el mismo que el valor conocido, se tiene un acierto, de esta manera se cuenta el número de aciertos para todos los ejemplos considerados, y este valor representa la aptitud de la red, mayores aciertos, mayor aptitud.

Reproducción o cruce.

En este campo hay dos decisiones a tomar. En primer lugar debemos decidir cuantos descendientes queremos, y qué individuos van a reproducirse. Generalmente se establece un factor de cruce constante, que representa la probabilidad de que dos individuos se crucen entre sí (normalmente este valor está entre 0.6 y 0.9), pero también se puede trabajar con probabilidades que dependan de la función de evaluación, favoreciendo que se reproduzcan los mejores individuos.

La segunda decisión es como vamos a cruzar los individuos. La figuras presentan las tres posibilidades más comunes. Con el cruce simple se elige un bit a partir del cual se intercambian los genes de los "padres". En el cruce doble se eligen dos bits y se intercambia todo el material genético que haya

entre ellos. Por último con el cruce con máscara se genera aleatoriamente un vector de máscara, de forma que si en una determinada posición el vector generado tiene un 1 se intercambian los bits, y si tiene un cero se mantienen (ver sección 1.5.2.2).

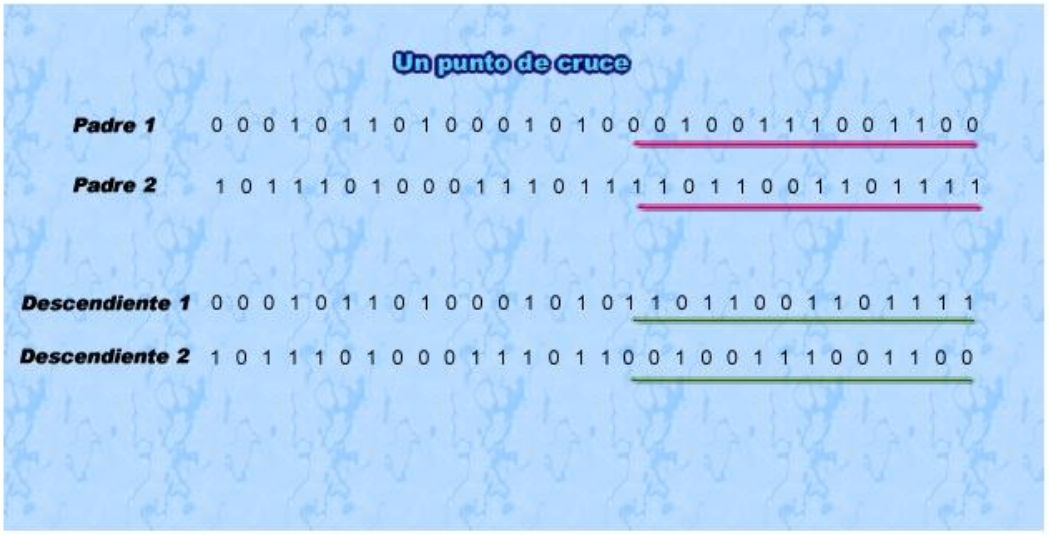


Figura 2.3.1 Método de cruce basado en un punto elaborado por Rafael Gordillo



Figura 2.3.2 Método de cruce basado en dos puntos elaborado por Rafael Gordillo

Para nuestro ejemplo planteado anteriormente, el cruce por máscara sería de la siguiente forma:



Figura 2.3.2 Método de cruce uniforme elaborado por Rafael Gordillo

Mutación:

Una vez cruzadas las soluciones sometemos a los descendientes a un proceso de mutación. Para ello elegimos un factor de mutación que representara la probabilidad de mutar cada uno de sus bits. El factor de mutación suele ser bastante más pequeño que el factor de cruce (entre 0.01 y 0.001), ya que un factor grande limita la convergencia de los algoritmos. Para aprovechar las posibilidades que ofrece la mutación en mayor medida, sin afectar a la convergencia, se puede utilizar un factor de mutación variable.

En ese caso al principio asignaríamos un valor alto para favorecer la exploración del espacio de soluciones, y cada cierto intervalo lo

decrementaríamos para evitar que con el paso del tiempo, las buenas soluciones puedan perderse debido a la mutación.

Para nuestro ejemplo planteado en la codificación Gray, siguiendo el desarrollo del algoritmo evolutivo, el proceso de mutación está determinado de la siguiente forma:

Luego de haber sido realizado el cruce con el método planteado en la sección correspondiente, tenemos un conjunto de la nueva generación y a cada uno de estos se realiza el proceso de mutación, en el caso de nuestro problema el método de cruce ha sido desarrollado en base a un punto de cruce, tomando como base un número aleatorio el cual de acuerdo al valor de probabilidad establecida por el usuario, el proceso de cruce se toma en el punto establecido y se efectúa de la siguiente forma: $1 \rightarrow 0$ o $0 \rightarrow 1$.

En nuestro ejemplo planteado tomando en cuenta uno de los descendientes obtenidos mediante el método de cruce con máscara el resultado sería el siguiente:

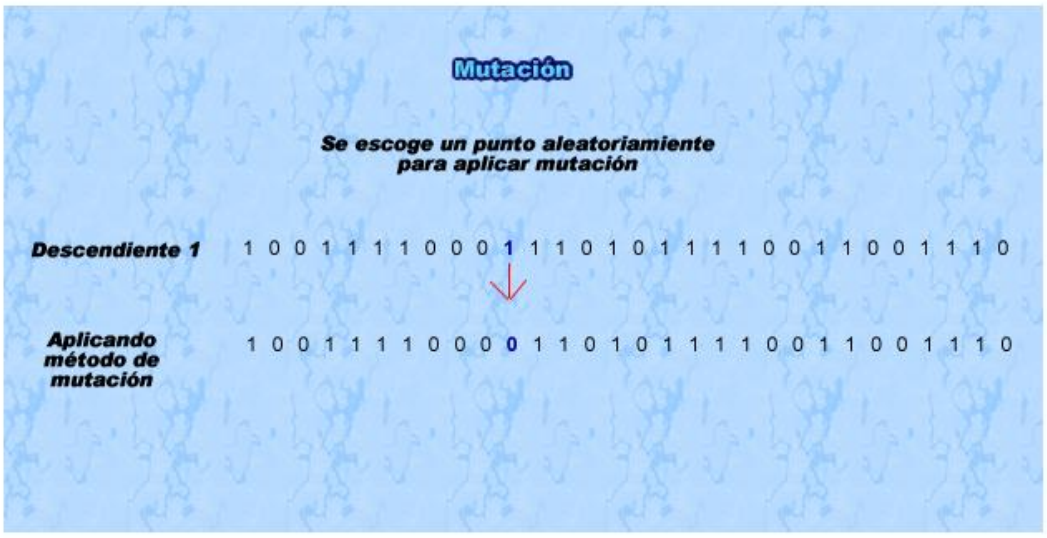


Figura 2.3.2 Método de mutación elaborado por Rafael Gordillo

Elección de los supervivientes.

Para acabar debemos decidir que individuos constituirán la población de la próxima generación. Esta decisión depende mucho del objetivo del algoritmo. Si necesitamos una solución rápida podemos elegir directamente a los mejores individuos, de esta forma el algoritmo converge antes, pero explora menos el espacio de soluciones. Una opción para efectuar una exploración más amplia es realizar torneos entre la población.

En esta variante creamos grupos de individuos aleatoriamente y los hacemos competir de forma que sobrevivan los mejores. La diferencia está en que cualquier solución puede sobrevivir si tiene la suerte de que sus contrincantes sean peores, de esta forma se favorece la diversidad genética.

La velocidad de convergencia es un arma de doble filo, queremos que nuestro algoritmo sea rápido, pero si nos enfrentamos a un problema de optimización y encontramos muy rápidamente una solución que parece la mejor, seguramente nos habremos dejado engañar por un óptimo local. Sin embargo si favorecemos que siempre haya una gran variedad genética muchas de las soluciones no nos resultarán útiles y estaremos desperdiciando nuestra capacidad de búsqueda. De nuevo, al igual que al elegir el factor de mutación, puede resultar interesante decidir dinámicamente, fomentando al principio la variedad genética y favoreciendo la convergencia en la fase final.

Si queremos utilizar el algoritmo genético para realizar una aplicación que se adapte a los cambios de su entorno, favorecer la convergencia hacia los mejores individuos es especialmente peligroso, ya que las soluciones que

parecían buenas en un determinado instante, pueden dejar de serlo cuando se produzca un cambio. En estos casos es fundamental disponer en todo momento de una amplia diversidad genética que permita afrontar los cambios.

El método de selección planteado para nuestro problema es el método de Ruleta, el cual al tomar en cuenta los valores de aptitud de cada uno de los individuos de las generaciones, serán seleccionados los padres que pasarán por el proceso de evolución los cuales al tener un mayor valor de aptitud tienen mayor probabilidad de ser seleccionados sin dejar de lado a aquellos individuos que su valor de aptitud es pequeño, esto es para no limitar el espacio de búsqueda.

Son varias las formas de entrenar a una red neuronal mediante técnicas evolutivas, de estas técnicas, tomaremos los algoritmos genéticos, los cuales nos permitirán una implementación comprensible.

La aplicación específica será con el problema de clasificación de sismos.

De las tres aproximaciones discutidas: Evolución de pesos, evolución de arquitecturas y evolución de reglas de aprendizaje, tomaremos en la evolución de pesos que es la más simple de todas. Para esto debemos plantear inicialmente la red neuronal como tal, la cual evolucionará desde en conjunto de pesos creados en forma aleatoria a una red final mediante el proceso de evolución, el objetivo como tal es evolucionar pesos. También evolucionaremos arquitecturas pero en este caso nos centraremos en la evolución de los pesos en cada una de las estructuras que se generan, es decir para determinar que estructura de red es la más óptima en el proceso

de evolución primero se genera una estructura inicial de red, esta estructura es generada de la siguiente forma: Inicialmente se plantea una estructura 4:n:2, donde n se genera aleatoriamente en el desarrollo del programa, este n puede ser establecido con un valor máximo y este representa el número de neuronas de capa oculta.

Entonces para el proceso de evolución de pesos, generamos una estructura a partir de la cual se genera un conjunto de pesos para dicha estructura, que está determinado por el tamaño máximo de población que se desee.

A este conjunto de pesos inicialmente generado se aplica el proceso de evolución mediante el Algoritmo Genético desarrollado, obteniendo un resultado de acuerdo al valor de aptitud que este conjunto obtiene, y se determina una solución o conjunto de soluciones en caso que el valor de aptitud sea el mismo para más de una de las generaciones.

Una vez obtenido este resultado de la primera estructura de red, se genera una nueva estructura aleatoriamente como se indicó antes, y se sigue con el proceso de evolución detallado en el párrafo anterior. Este proceso puede terminar de dos formas: la primera es cuando el algoritmo cumpla con el número máximo de redes preestablecidas, este número lo determina el usuario; y la segunda forma es cuando en cada uno de los procesos de evolución de los pesos la estructura de red que está siendo evaluada en ese instante en su valor de aptitud sobrepasa con el valor de error establecido.

Como se discutió anteriormente, el problema de clasificación de sismos tiene cuatro variables de entrada y son tres clases de eventos o señales sísmicas, estas clases en el desarrollo del algoritmo están determinados en

la función de aptitud, esta función realiza un proceso de evaluación de acuerdo a PMC y al resultado obtenido se aplica una aproximación del valor, obteniéndose como resultado, un valor entero, el cual es comparado con cada uno de los valores del conjunto del *Ydeseado* y si obtenemos un acierto el valor de aptitud incrementa, caso contrario el valor de aptitud se mantiene.

Tomaremos el problema de evolucionar solamente los pesos, de acuerdo al problema planteado.

La representación inicial de la estructura de la red neuronal explicado anteriormente se ha determinado así 4:n:2, lo que quiere decir cuatro neuronas en la capa de entrada, el número de neuronas en la capa oculta se inicializará aleatoriamente determinando un número máximo de neuronas para el entrenamiento, y dos en la capa de salida. Se ha determinado esta estructura inicial de la red debido que a se tienen 4 variables de ingreso y como salida tomaremos para el acierto un bit, si obtenemos un acierto sería un valor de 1 caso contrario sería 0, por lo tanto se determina las dos neuronas en la capa de salida, (como se explicará más delante de acuerdo a la función de aptitud que se escoja lo que llegaremos a determinar es si un resultado es bueno o no basado en experiencias o resultados ya obtenidos mediante otros métodos aplicados actualmente).

Si inicialmente se va a plantear esta estructura de red entonces se entiende que lo importante es encontrar la forma de evolucionar la red para obtener resultados óptimos aplicando los AGs.

Analizando la teoría de AGs. lo primero a determinar es el tipo de codificación a usar.

Para el problema presentado se ha determinado que la codificación más adecuada sería la codificación Gray, como fue explicado anteriormente nos permite resolver el problema que presenta el tipo de codificación binaria, y además abarcar un conjunto de soluciones más amplio *(Harri Niska, 2004),* la cual por el proceso de cruce y mutación permitiría obtener mejores resultados. En la teoría donde se indicaba la codificación más adecuada para un problema de evolución de redes neuronales se indicaban los beneficios tanto de la codificación binaria como de la codificación con valores reales, pero también se explica que no existe una regla específica para escoger una u otra opción, se dice también que esto se determina de acuerdo al problema planteado; por eso y ya que se desea optimizar el obtener resultados adecuados se ha escogido la codificación Gray, ya que el la codificación binaria presenta problemas en cuanto a lo que tiene que ver con evolución de pesos de una red neuronal (ver secciones 1.5.1, 1.5.4).

Para esto establecimos un número constante de bits el cual nos permitirá en proceso de selección, cruce y mutación, este número de bits básicamente fue obtenido de acuerdo al número o cantidad de pesos a ser representados por la red neuronal, además de acuerdo a esto podemos nosotros también determinar el orden en el cual se han establecido los resultados, en el caso de representar los valores de los pesos con sus respectivas conexiones, lo cual depende del número de neuronas en la capa oculta.

Una vez que se ha decido una forma de codificación, es necesario determinar la función de aptitud para el problema de clasificación de sismos, para esto de acuerdo a los datos proporcionados por el Instituto Geofísico podemos tener una forma de solución adecuada, ya que estos datos son reales y aplicados actualmente en el proceso de clasificar sismos; es decir en términos de A.G. el *yd* (y-deseado) es óptimo para poder evaluar los resultados presentados por la evolución.

Para el siguiente punto en cuanto a la evolución de la red neuronal se debe determinar el método de selección de los padres, en un problema de evolución de redes neuronales de acuerdo a las conclusiones obtenidas por *(Y.K. Wong 2000),* donde se explica de acuerdo a 3 métodos de selección utilizados para las pruebas (ruleta, torneo, ranking), que el método de selección de Ruleta es el que ha dado una mejor optimización en los resultados tanto para la evolución de pesos como de topología de una red neuronal, por lo que para nuestro problema se aplicará este método.

En cuanto se refiere al método de cruce que se utilizará en nuestro problema se ha decido por el método uniforme de cruce *(Y.K. Wong 2000),* este método ha sido probado de tal forma que permite que cada bit de la cadena codificada tenga la misma oportunidad de ser seleccionado para el cruce entre los padres, lo cual es más eficiente que los métodos de 1 o 2 puntos de cruce.

CAPITULO III: OBTENCIÓN, EVALUACIÓN Y COMPARACIÓN DE RESULTADOS.

En este capítulo se presentan los resultados sobre las pruebas realizadas en el proceso de evolución de redes, aplicando algoritmos genéticos, además también resultados obtenidos con el modelo de red escogido (PMC), y con estos resultados se realiza la comparación de los dos modelos, y así determinar cual de los dos es el modelo más eficiente en la solución del problema planteado.

3.1 Clasificación de resultados.

De acuerdo al AG desarrollado y aplicado en el proceso de evolución presentado anteriormente se han realizado varias pruebas, variando, valores de probabilidad de cruce, de mutación, tamaño de población y valores de error.

En el proceso de evolución se explicó que se genera primero aleatoriamente conjuntos de estructuras o redes para poder generalizar el espacio de solución y buscar el número de neuronas de capa oculta que permita obtener una estructura de red más adecuada luego del entrenamiento y de esta forma resolver más eficientemente el problema planteado.

Después de determinado este espacio de solución (conjunto de estructuras de redes a entrenar), se realizarán pruebas en este conjunto de entrenamiento para así obtener el resultado más eficiente.

Una vez establecidos estos criterios para determinar el mejor resultado, se obtuvieron los siguientes resultados:

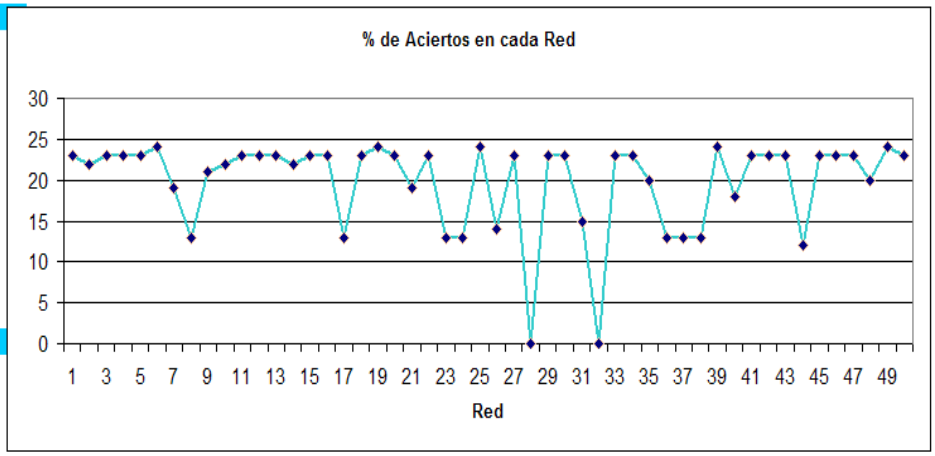
Cada uno de los ejemplos presentados a continuación han sido evaluados con conjuntos de datos de entrenamiento y de prueba de diferente tamaño en cada unos de los cuales se han variado los valores de las variables antes mencionadas para evaluar cada una de las estructuras de redes neuronales.

En las figuras 3.1.1 a 3.1.8 se muestran como resultados el valor de aptitud (% de Aciertos) de cada estructura de red con el número de neuronas de capa oculta con el que fue entrenada.

El primer gráfico (Fig. 3.1.1) nos muestra los resultados obtenidos en la evolución de redes con un conjunto de 40 datos de entrenamiento y 40 datos de prueba.

Red	% de Aciertos	Neuronas de Capa Oculta
1	23	10
2	22	8
3	23	7
4	23	9
5	23	3
6	**24**	**6**
7	19	9
8	13	2
9	21	3
10	22	4
11	23	8
12	23	9
13	23	4
14	22	3
15	23	7
16	23	4
17	13	2
18	23	5
19	**24**	**7**
20	23	8
21	19	3
22	23	7
23	13	3
24	13	2
25	**24**	**6**
26	14	2
27	23	8
28	0	1
29	23	8
30	23	8
31	15	3
32	0	1
33	23	6
34	23	8
35	20	9
36	13	2
37	13	2
38	13	2
39	**24**	**5**
40	18	3
41	23	8
42	23	8
43	23	6
44	12	9
45	23	7
46	23	4
47	23	7
48	20	3
49	**24**	**6**
50	23	5

Probabilidad de cruce	**0.9**
Probabilidad de mutación	**0.6**
Tamaño de población por red	**100**
Máximo de redes	**50**
Máximo de generaciones	**90**
Máximo de neuronas ocultas	**10**
Error	**0.01 %**



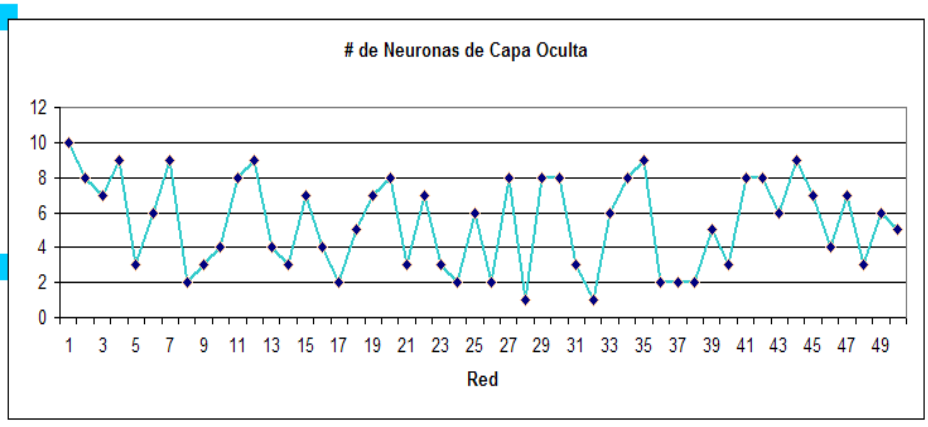
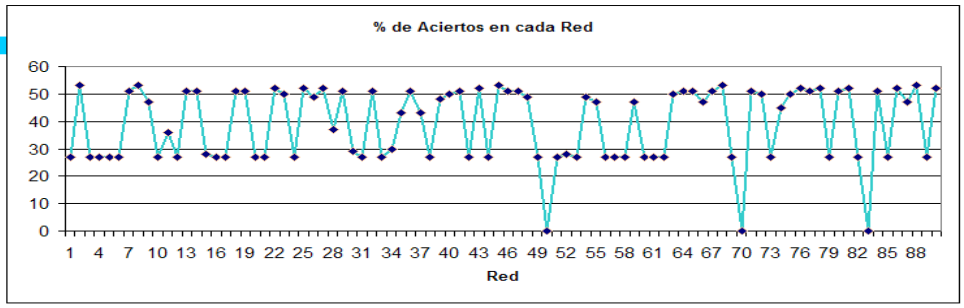


Figura 3.1.1 El gráfico muestra los resultados de la evolución de de diferentes estructuras de redes neuronales. En este se aprecia que las redes con 5, 6 y 7 neuronas en la capa oculta representan las mejores arquitecturas.

El siguiente gráfico (Fig. 3.1.2) nos muestra los resultados obtenidos para un conjunto de datos de entrenamiento de 80 datos, y 40 datos para el conjunto de datos de prueba de las redes obtenidas.

Red	% de Aciertos	Neuronas de Capa Oculta
1	27	2
2	**53**	**7**
3	27	13
4	27	9
5	27	12
6	27	11
7	51	4
8	**53**	**5**
9	47	8
10	27	11
11	36	2
12	27	12
13	51	4
14	51	8
15	28	10
16	27	11
17	27	15
18	51	8
19	51	6
20	27	14
21	27	12
22	52	6
23	50	5
24	27	12
25	52	6
26	49	5
27	52	4
28	37	2
29	51	7
30	29	10
31	27	12
32	51	5
33	27	14
34	30	10
35	43	2
36	51	5
37	43	3
38	27	12
39	48	3
40	50	3
41	51	3
42	27	14
43	52	5
44	27	2
45	**53**	**4**
46	51	7
47	51	8
48	49	5
49	27	15
50	0	1
51	27	12
52	28	9
53	27	13
54	49	4
55	47	3
56	27	15
57	27	13
58	27	15
59	47	3
60	27	12
61	27	12
62	27	11
63	50	3
64	51	9
65	51	5
66	47	3
67	51	5
68	**53**	**4**
69	27	13
70	0	1
71	51	5
72	50	9
73	27	14
74	45	9
75	50	8
76	52	8
77	51	5
78	52	7
79	27	13
80	51	4
81	52	6
82	27	11
83	0	1
84	51	4
85	27	12
86	52	6
87	47	3
88	**53**	**4**
89	27	9
90	52	4
	53	

Probabilidad de cruce	**0.6**
Probabilidad de mutación	**0.4**
Tamaño de población por red	**80**
Máximo de redes	**90**
Máximo de generaciones	**60**
Máximo de neuronas ocultas	**15**
Error	**0.1 %**



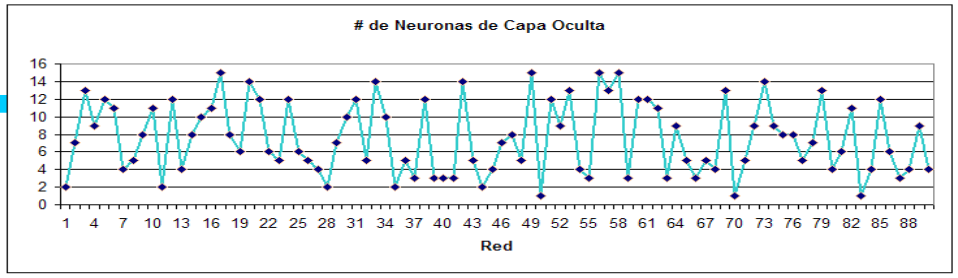
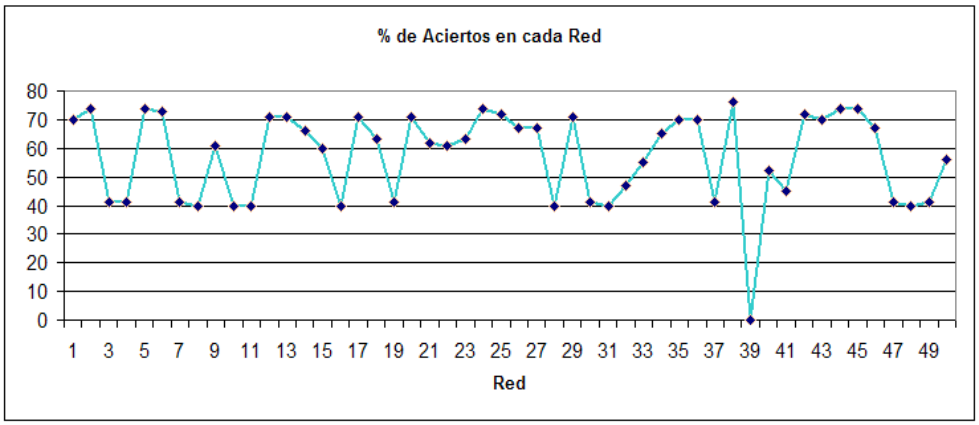


Figura 3.1.2 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución.

El tercer gráfico (Fig. 3.1.3) nos muestra los resultados que se han obtenido en el proceso de evolución de redes para un conjunto de 120 datos de entrenamiento y 30 datos de prueba.

Red	% de Aciertos	Neuronas de Capa Oculta
1	70	3
2	74	7
3	41	2
4	41	2
5	74	7
6	73	4
7	41	8
8	40	2
9	61	5
10	40	9
11	40	10
12	71	3
13	71	7
14	66	5
15	60	3
16	40	8
17	71	3
18	63	3
19	41	2
20	71	3
21	62	3
22	61	6
23	63	4
24	74	5
25	72	6
26	67	6
27	67	3
28	40	9
29	71	5
30	41	2
31	40	10
32	47	7
33	55	6
34	65	5
35	70	3
36	70	4
37	41	2
38	**76**	**7**
39	0	1
40	52	6
41	45	9
42	72	4
43	70	5
44	74	5
45	74	6
46	67	8
47	41	2
48	40	9
49	41	2
50	56	3

Probabilidad de cruce	**0.7**
Probabilidad de mutación	**0.001**
Tamaño de población por red	**100**
Máximo de redes	**50**
Máximo de generaciones	**50**
Máximo de neuronas ocultas	**10**
Error	**3%**



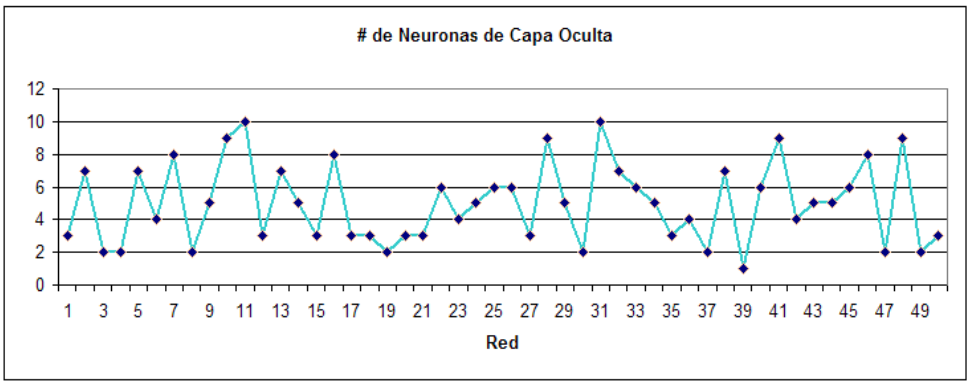
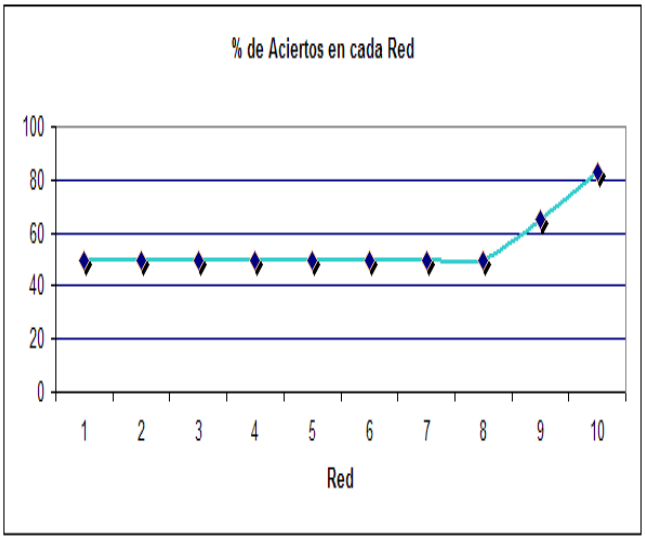


Figura 3.1.3 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución.

En los siguientes resultados (Figs. 3.1.1 – 3.1.8), se han realizado varias pruebas para el entrenamiento de las redes tomando para el mismo, todo el conjunto de datos proporcionados para el entrenamiento, es decir que en los resultados que se presentan a continuación se toman 150 datos para el entrenamiento y prueba siendo el mismo conjunto de datos para ambos casos.

Red	% de Aciertos	Neuronas de Capa Oculta
1	50	14
2	50	14
3	50	9
4	50	8
5	50	11
6	50	8
7	50	14
8	50	10
9	65	3
10	83	5

Probabilidad de cruce	0.7
Probabilidad de mutación	0.02
Tamaño de población por red	30
Máximo de redes	10
Máximo de generaciones	15
Máximo de neuronas ocultas	15
Error	1%



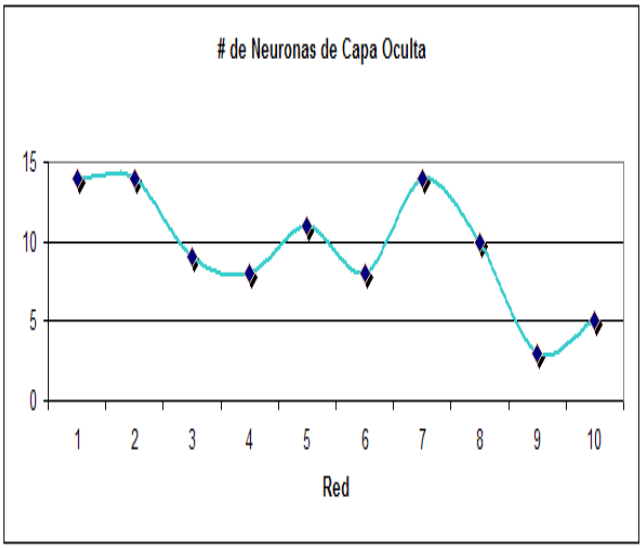
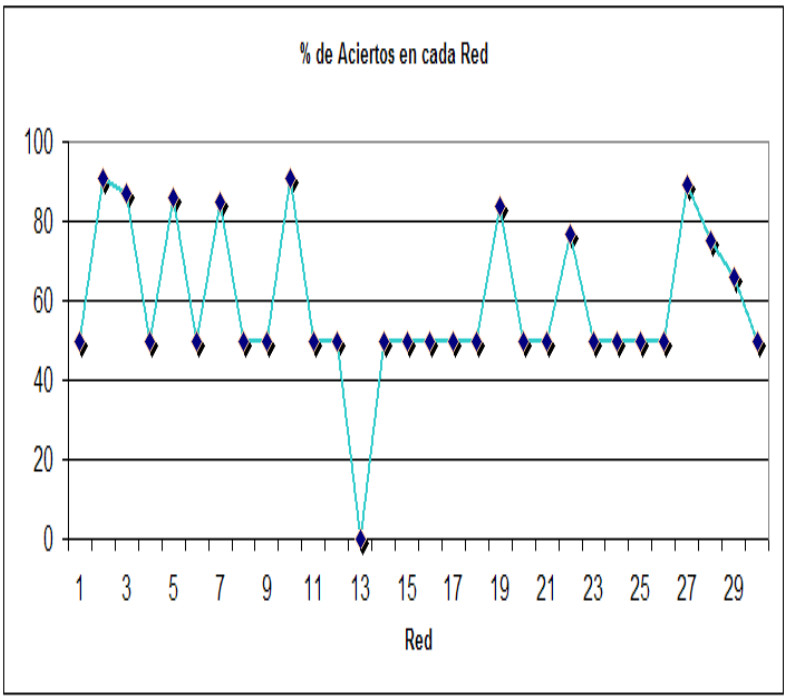


Figura 3.1.4 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución.

Red	% de Aciertos	Neuronas de Capa Oculta
1	50	12
2	91	6
3	87	4
4	50	10
5	86	4
6	50	10
7	85	7
8	50	13
9	50	2
10	91	4
11	50	9
12	50	14
13	0	1
14	50	14
15	50	12
16	50	11
17	50	13
18	50	8
19	84	4
20	50	14
21	50	7
22	77	5
23	50	5
24	50	2
25	50	11
26	50	8
27	89	7
28	75	3
29	66	4
30	50	2

Probabilidad de cruce	0.8
Probabilidad de mutación	0.01
Tamaño de población por red	50
Máximo de redes	30
Máximo de generaciones	30
Máximo de neuronas ocultas	15
Error	1%



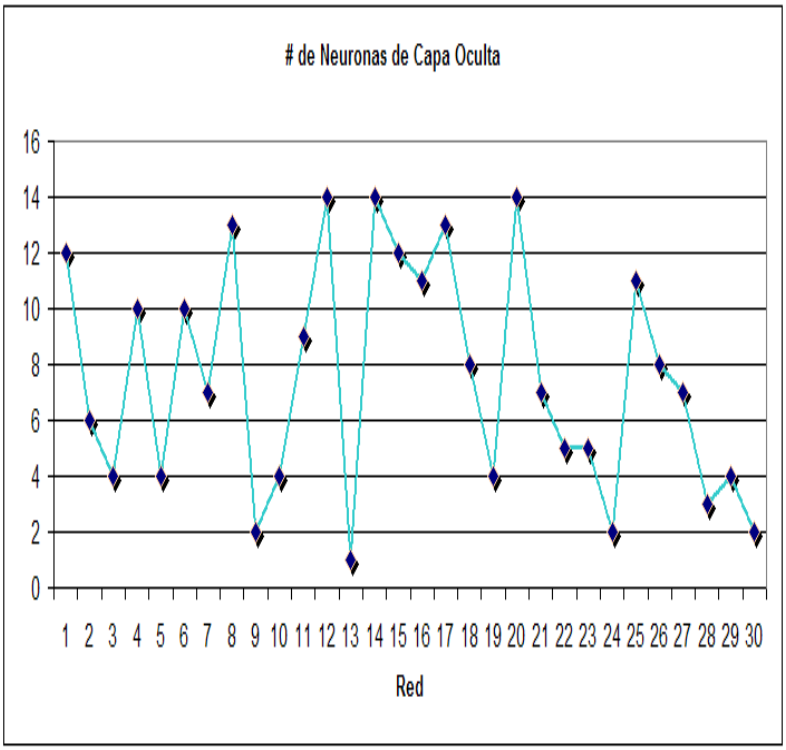
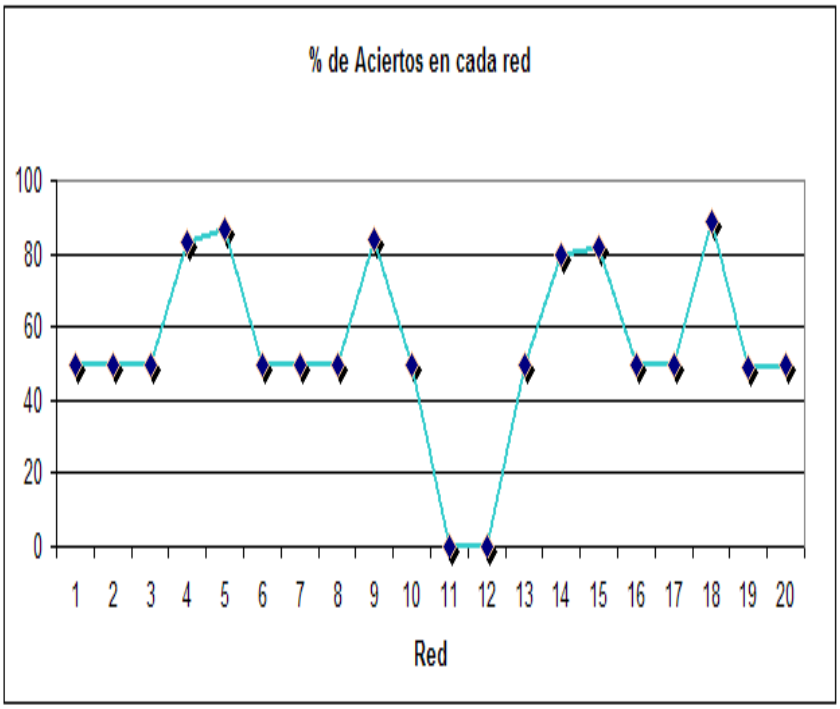


Figura 3.1.5 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución.

Red	% de Aciertos	Neuronas de Capa Oculta
1	50	0
2	50	2
3	50	10
4	83	3
5	87	4
6	50	13
7	50	10
8	50	9
9	84	6
10	50	14
11	0	1
12	0	1
13	50	10
14	80	5
15	82	4
16	50	2
17	50	2
18	89	3
19	49	2
20	50	10

Probabilidad de cruce	0.6
Probabilidad de mutación	0.01
Tamaño de población por red	100
Máximo de redes	20
Máximo de generaciones	30
Máximo de neuronas ocultas	15
Error	1%



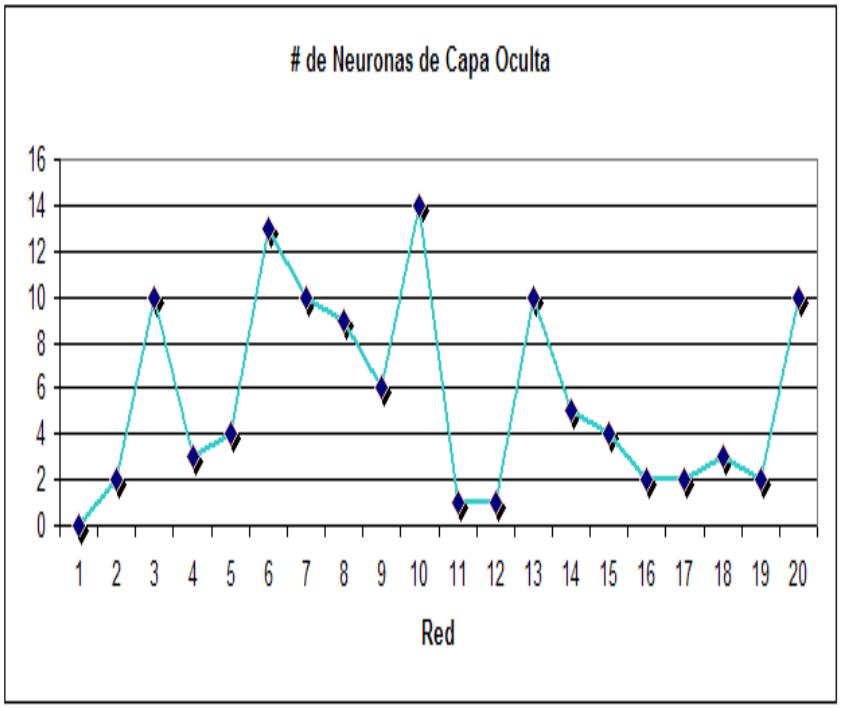
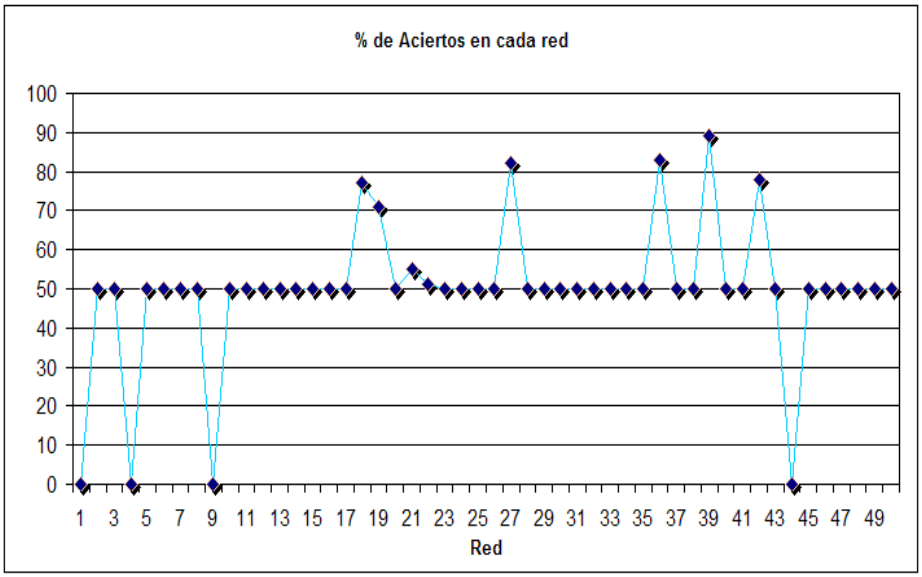


Figura 3.1.6 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución.

Red	% de Aciertos	Neuronas de Capa Oculta
1	0	1
2	50	10
3	50	10
4	0	1
5	50	18
6	50	15
7	50	9
8	50	17
9	0	1
10	50	13
11	50	11
12	50	10
13	50	8
14	50	11
15	50	13
16	50	16
17	50	20
18	77	7
19	71	3
20	50	10
21	55	8
22	51	8
23	50	11
24	50	18
25	50	10
26	50	11
27	82	4
28	50	13
29	50	13
30	50	9
31	50	17
32	50	15
33	50	20
34	50	12
35	50	9
36	83	3
37	50	19
38	50	14
39	89	6
40	50	18
41	50	10
42	78	5
43	50	19
44	0	1
45	50	11
46	50	9
47	50	12
48	50	9
49	50	18
50	50	12

Probabilidad de cruce	0.8
Probabilidad de mutación	0.01
Tamaño de población por red	100
Máximo de redes	50
Máximo de generaciones	50
Máximo de neuronas ocultas	20
Error	3%



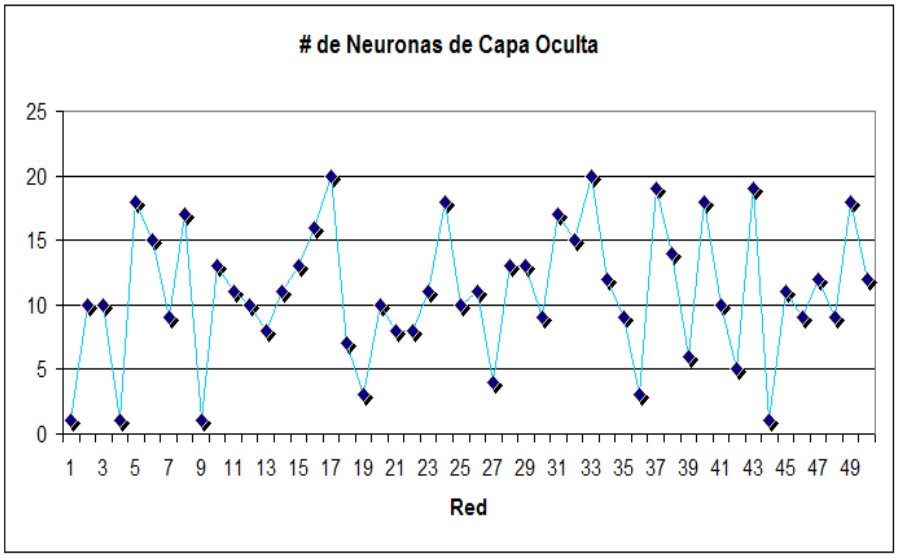
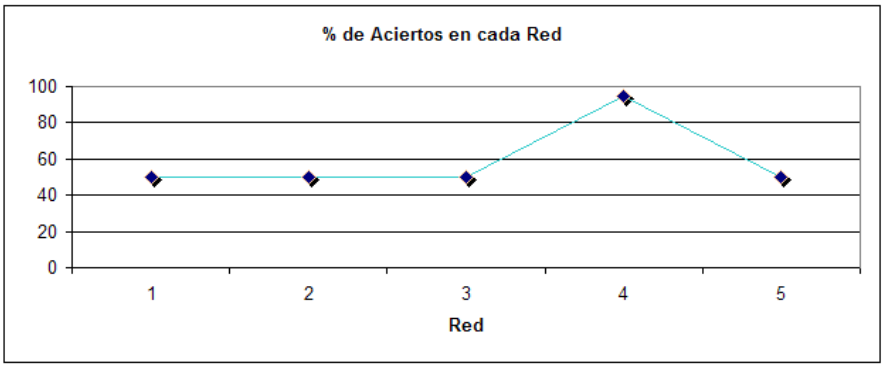


Figura 3.1.7 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución.

Red	% de Aciertos	Neuronas de Capa Oculta
1	50	7
2	50	8
3	50	7
4	**95**	**6**
5	50	7

Probabilidad de cruce	**0.8**
Probabilidad de mutación	**0.01**
Tamaño de población por red	**30**
Máximo de redes	**5**
Máximo de generaciones	**10**
Máximo de neuronas ocultas	**10**
Error	**2%**



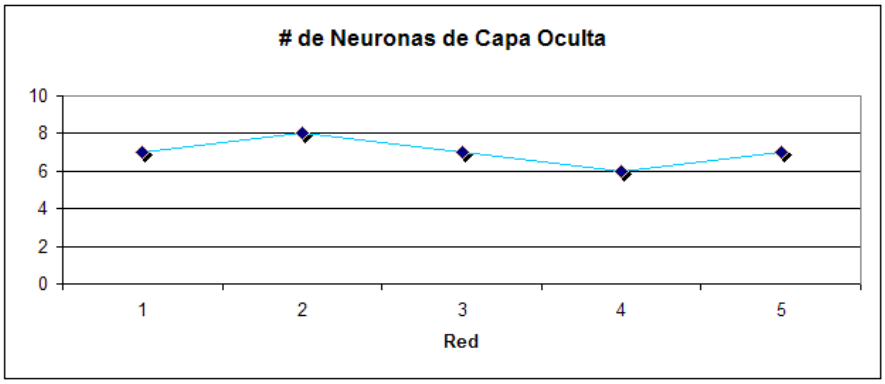


Figura 3.1.8 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución.

Estos resultados obtenidos muestran en forma clara el valor óptimo para el número de neuronas de la capa oculta está entre 3 y 7. Se aprecia además muy claramente que para un número grande de datos de entrenamiento los resultados mejoran significativamente (entre 80% a 90% de aciertos), estos datos disminuyen en forma gradual con un conjunto de datos de entrenamiento menor.

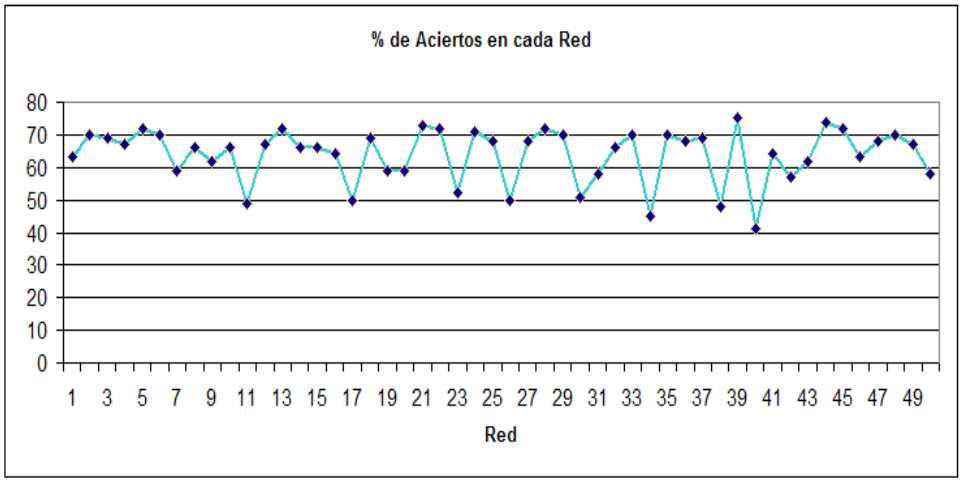
Por lo tanto, nos centraremos en el estudio de estas redes con tres, cuatro, cinco, seis y siete neuronas de la capa oculta y así determinar la mejor estructura de red para el problema planteado.

En los siguientes gráficos (3.1.9 – 3.1.13), los resultados que se muestran son de cada una de las estructuras de red que se determinó anteriormente

para el entrenamiento, y poder determinar el mejor valor de aptitud en dichas estructuras.

EL primer gráfico nos muestra los resultados obtenidos de la evolución de redes para la estructura de red con tres neuronas de la capa oculta: Red 4:3:2

Red	% de Aciertos	Neuronas de Capa Oculta
1	63	3
2	70	3
3	69	3
4	67	3
5	72	3
6	70	3
7	59	3
8	66	3
9	62	3
10	66	3
11	49	3
12	67	3
13	72	3
14	66	3
15	66	3
16	64	3
17	50	3
18	69	3
19	59	3
20	59	3
21	73	3
22	72	3
23	52	3
24	71	3
25	68	3
26	50	3
27	68	3
28	72	3
29	70	3
30	51	3
31	58	3
32	66	3
33	70	3
34	45	3
35	70	3
36	68	3
37	69	3
38	48	3
39	75	3
40	41	3
41	64	3
42	57	3
43	62	3
44	74	3
45	72	3
46	63	3
47	68	3
48	70	3
49	67	3
50	58	3

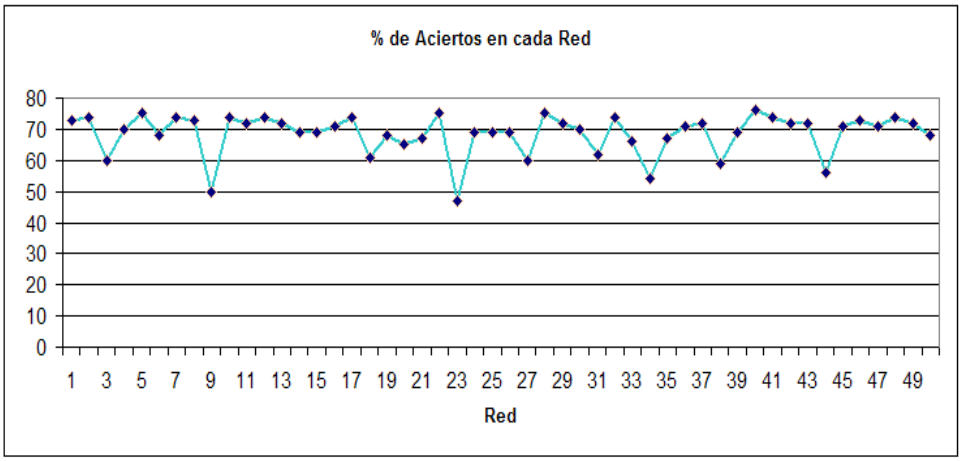


Probabilidad de cruce	0.9
Probabilidad de mutación	0.001
Tamaño de población por red	100
Máximo de redes	50
Máximo de generaciones	60
Número de neuronas ocultas	3
Error	1%

Figura 3.1.9 Conjunto de entrenamiento 120 datos y 30 datos de prueba. Primera estructura seleccionada para el entrenamiento. Número de neuronas de capa oculta: 3.

La siguiente estructura a ser evolucionada es 4:4:2.

Red	% de Aciertos	Neuronas de Capa Oculta
1	73	4
2	74	4
3	60	4
4	70	4
5	75	4
6	68	4
7	74	4
8	73	4
9	50	4
10	74	4
11	72	4
12	74	4
13	72	4
14	69	4
15	69	4
16	71	4
17	74	4
18	61	4
19	68	4
20	65	4
21	67	4
22	75	4
23	47	4
24	69	4
25	69	4
26	69	4
27	60	4
28	75	4
29	72	4
30	70	4
31	62	4
32	74	4
33	66	4
34	54	4
35	67	4
36	71	4
37	72	4
38	59	4
39	69	4
40	76	4
41	74	4
42	72	4
43	72	4
44	56	4
45	71	4
46	73	4
47	71	4
48	74	4
49	72	4
50	68	4

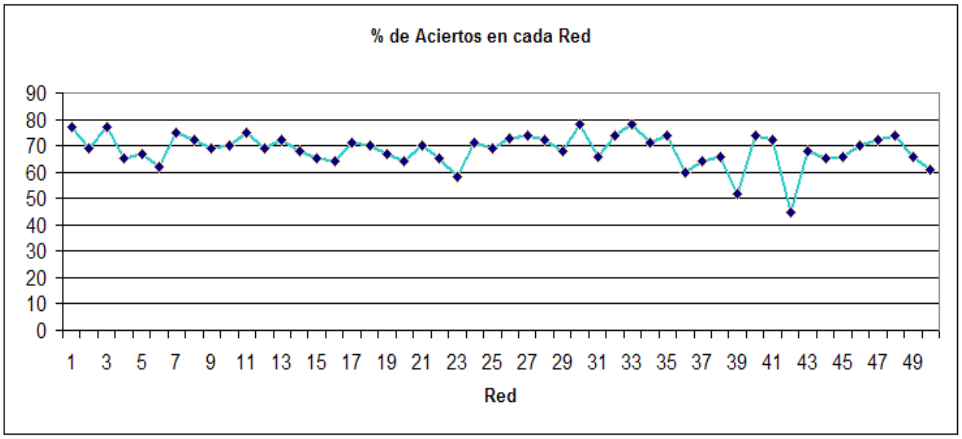


Probabilidad de cruce	0.8
Probabilidad de mutación	0.01
Tamaño de población por red	100
Máximo de redes	50
Máximo de generaciones	50
Número de neuronas ocultas	4
Error	2%

Figura 3.1.10 Conjunto de entrenamiento 120 datos y 30 datos de prueba. Segunda estructura seleccionada para el entrenamiento. Número de neuronas de capa oculta: 4.

La siguiente estructura a ser evolucionada es 4:5:2.

Red	% de Aciertos	Neuronas de Capa Oculta
1	77	5
2	69	5
3	77	5
4	65	5
5	67	5
6	62	5
7	75	5
8	72	5
9	69	5
10	70	5
11	75	5
12	69	5
13	72	5
14	68	5
15	65	5
16	64	5
17	71	5
18	70	5
19	67	5
20	64	5
21	70	5
22	65	5
23	58	5
24	71	5
25	69	5
26	73	5
27	74	5
28	72	5
29	68	5
30	**78**	**5**
31	66	5
32	74	5
33	**78**	**5**
34	71	5
35	74	5
36	60	5
37	64	5
38	66	5
39	52	5
40	74	5
41	72	5
42	45	5
43	68	5
44	65	5
45	66	5
46	70	5
47	72	5
48	74	5
49	66	5
50	61	5

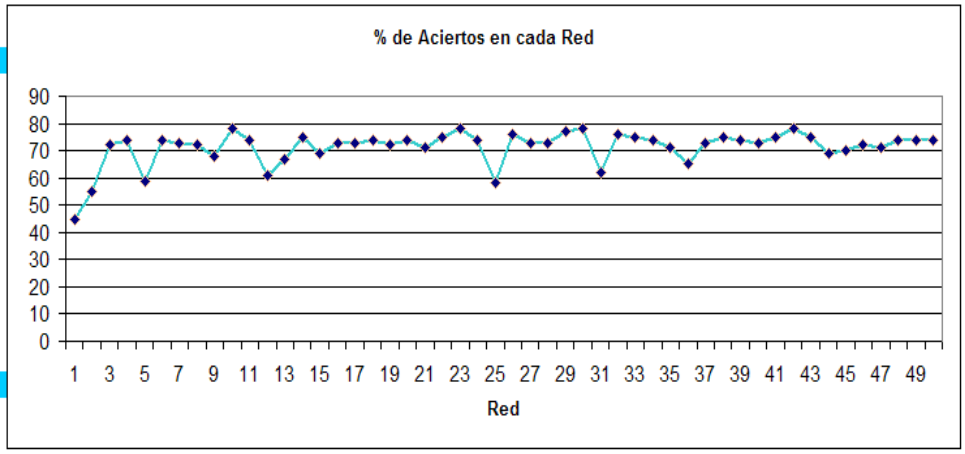


Probabilidad de cruce	**0.7**
Probabilidad de mutación	**2**
Tamaño de población por red	**100**
Máximo de redes	**50**
Máximo de generaciones	**60**
Número de neuronas ocultas	**5**
Error	**3%**

Figura 3.1.11 Conjunto de entrenamiento 120 datos y 30 datos de prueba. Tercera estructura seleccionada para el entrenamiento. Número de neuronas de capa oculta: 5.

La siguiente estructura a ser evolucionada es 4:6:2.

Red	% de Aciertos	Neuronas de Capa Oculta
1	45	6
2	55	6
3	72	6
4	74	6
5	59	6
6	74	6
7	73	6
8	72	6
9	68	6
10	**78**	**6**
11	74	6
12	61	6
13	67	6
14	75	6
15	69	6
16	73	6
17	73	6
18	74	6
19	72	6
20	74	6
21	71	6
22	75	6
23	**78**	**6**
24	74	6
25	58	6
26	76	6
27	73	6
28	73	6
29	77	6
30	78	6
31	62	6
32	76	6
33	75	6
34	74	6
35	71	6
36	65	6
37	73	6
38	75	6
39	74	6
40	73	6
41	75	6
42	**78**	**6**
43	75	6
44	69	6
45	70	6
46	72	6
47	71	6
48	74	6
49	74	6
50	74	6



Probabilidad de cruce	**0.7**
Probabilidad de mutación	**2**
Tamaño de población por red	**110**
Máximo de redes	**50**
Máximo de generaciones	**70**
Número de neuronas ocultas	**6**
Error	**1%**

Figura 3.1.12 Conjunto de entrenamiento 120 datos y 30 datos de prueba. Cuarta estructura seleccionada para el entrenamiento. Número de neuronas de capa oculta: 6.

La siguiente estructura a ser evolucionada es 4:7:2.

Red	% de Aciertos	Neuronas de Capa Oculta
1	74	7
2	74	7
3	71	7
4	74	7
5	73	7
6	50	7
7	72	7
8	75	7
9	74	7
10	64	7
11	75	7
12	71	7
13	73	7
14	75	7
15	74	7
16	74	7
17	61	7
18	73	7
19	75	7
20	56	7
21	**78**	**7**
22	74	7
23	54	7
24	54	7
25	74	7
26	77	7
27	74	7
28	73	7
29	72	7
30	73	7
31	57	7
32	64	7
33	72	7
34	74	7
35	75	7
36	64	7
37	75	7
38	70	7
39	45	7
40	74	7
41	75	7
42	74	7
43	74	7
44	74	7
45	73	7
46	73	7
47	75	7
48	75	7
49	71	7
50	70	7

Probabilidad de cruce	**0.6**
Probabilidad de mutación	**5**
Tamaño de población por red	**120**
Máximo de redes	**50**
Máximo de generaciones	**80**
Número de neuronas ocultas	**7**
Error	**2%**

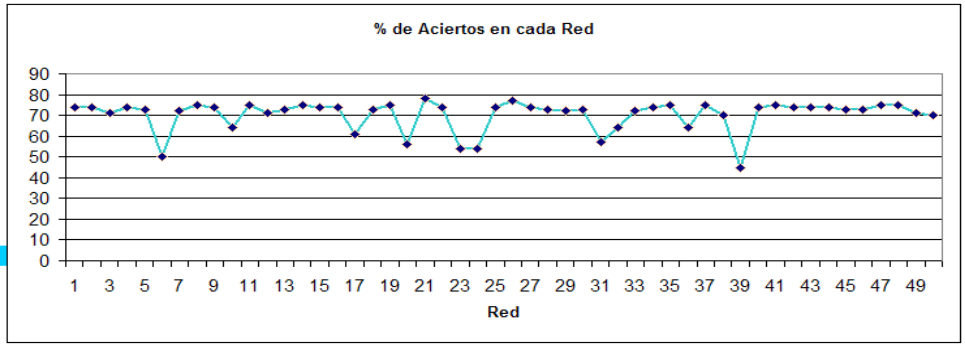


Figura 3.1.13 Conjunto de entrenamiento 120 datos y 30 datos de prueba. Quinta estructura de red seleccionada para el entrenamiento. Número de neuronas de capa oculta: 7.

En cada uno de los gráficos anteriores se evolucionaron 50 redes para cada estructura, y en cada uno de los entrenamientos se cambiaron solamente los valores de probabilidad de cruce y mutación, tamaño de población, error estimado.

3.2 Evaluación de los resultados obtenidos.

De los resultados obtenidos en el proceso de evolución los mejores han estado determinados por el tamaño del conjunto de datos de entrenamiento y de prueba.

	# Máximo de Redes	# Máximo de Neuronas de Capa Oculta	Probabilidad de cruce	Probabilidad de mutación	# Máximo de generaciones	Tamaño de población	% de Error	% de Aptitud Máxima	# Óptimo de Neuronas de Capa Oculta	Cantidad de datos de entrenamiento	Cantidad de datos de prueba
Figura 3.1.1	50	10	0.9	0.6	90	100	0.01	24	6	40	40
Figura 3.1.2	90	15	0.6	0.4	60	80	0.1	53	4	80	40
Figura 3.1.3	50	10	0.7	0.001	50	100	3	76	7	120	30
Figura 3.1.4	10	15	0.7	0.02	15	30	1	83	5	150	150
Figura 3.1.5	30	15	0.8	0.01	30	50	1	91	4 y 6	150	150
Figura 3.1.6	20	15	0.6	0.01	30	100	1	89	3	150	150
Figura 3.1.7	50	20	0.8	0.01	50	100	3	89	6	150	150
Figura 3.1.8	5	10	0.8	0.01	10	30	2	95	6	150	150
Figura 3.1.9	50	3	0.9	0.001	60	100	1	75	3	120	30
Figura 3.1.10	50	4	0.8	0.01	50	100	2	76	4	120	30
Figura 3.1.11	50	5	0.7	2	60	100	3	78	5	120	30
Figura 3.1.12	50	6	0.7	2	70	110	1	78	6	120	30
Figura 3.1.13	50	7	0.6	5	80	120	2	78	7	120	30

Tabla 3.2.1 Cuadro comparativo de los resultados obtenidos en el entrenamiento con Algoritmos Genéticos

Hemos visto a lo largo del proceso de entrenamiento que para el problema de tesis, a medida que los valores del conjunto de datos de entrenamiento aumentan (entre 40 y 120 datos de entrenamiento), los valores de aptitud

obtenidos para cada estructura de red mejoran, y en todos los casos los mejores valores de aptitud para determinar las mejores estructuras siempre estuvieron para valores de número de neuronas de capa oculta entre 3 y 7. Luego de haber determinado el conjunto de soluciones de estructuras de red sobre los cuales se centró el proceso de entrenamiento se obtuvieron los mejores resultados para las estructuras de redes 4:5:2, 4:6:2, 4:7:2.

3.3 Entrenamiento de un Perceptron Multicapa para la clasificación de sismos.

La evolución de redes neuronales, mediante AGs, para el problema de clasificación de sismos planteado, nos sugieren que las mejores arquitecturas están entre 4 a 7 neuronas el la capa oculta. Realizaremos a algunas pruebas con los algoritmos del PMC (ver sección 1.4.2).

Para el entrenamiento de PMC se tomaron en cuenta en primer lugar los mismos datos y las mismas proporciones de los datos (120 datos de entrenamiento y 30 datos de prueba) de los resultados obtenidos en el proceso de evolución en el espacio de solución seleccionado.

Los resultados obtenidos en el entrenamiento de una red PMC son los siguientes (estos resultados fueron obtenidos realizando pruebas en Matlab utilizando el algoritmo que por defecto Matlab implementa):

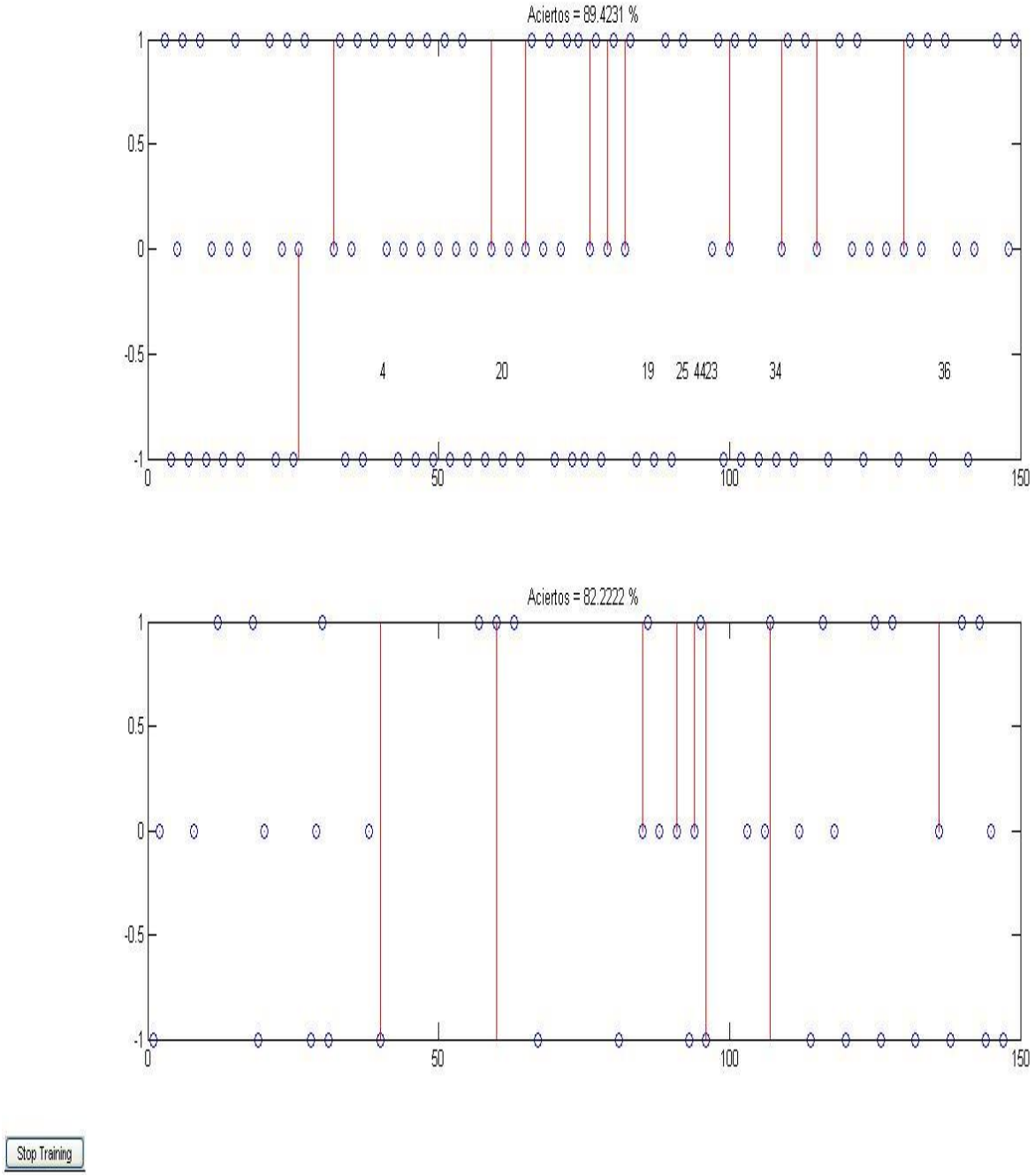


Figura 3.3.1 Conjunto de entrenamiento 120 datos y 30 datos de prueba.

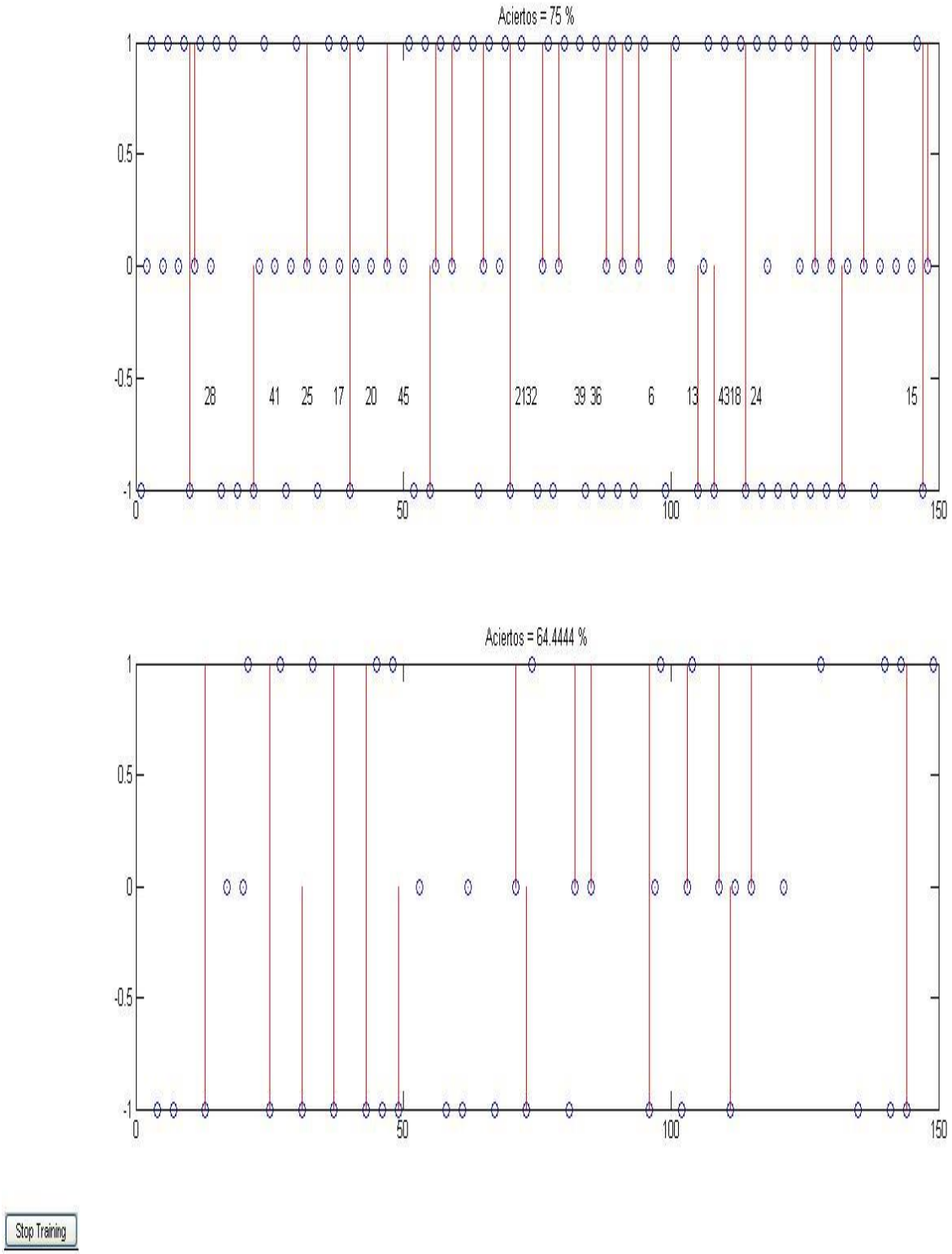


Figura 3.3.2 Conjunto de entrenamiento 120 datos y 30 datos de prueba.

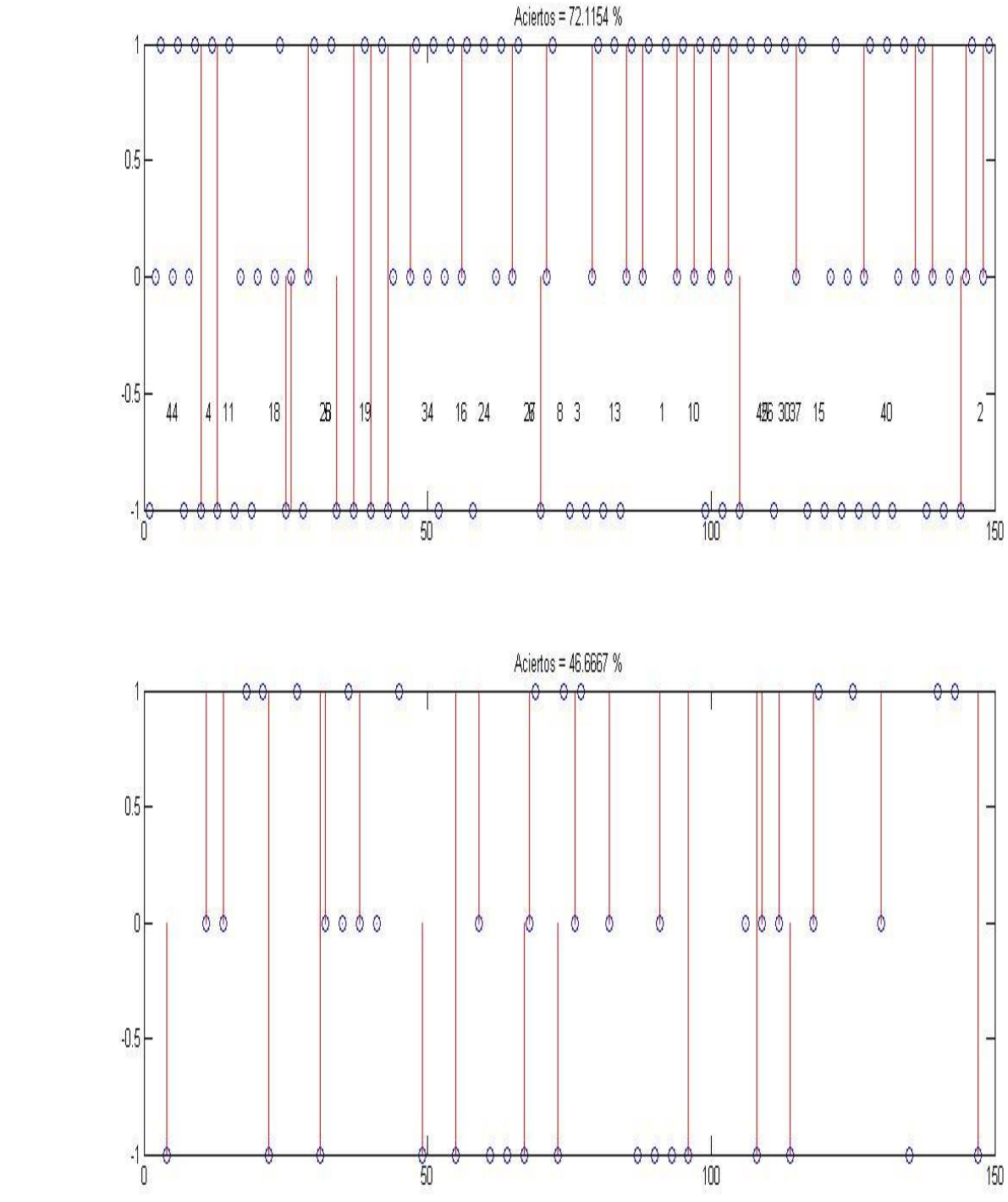


Figura 3.3.3 Conjunto de entrenamiento 120 datos y 30 datos de prueba.

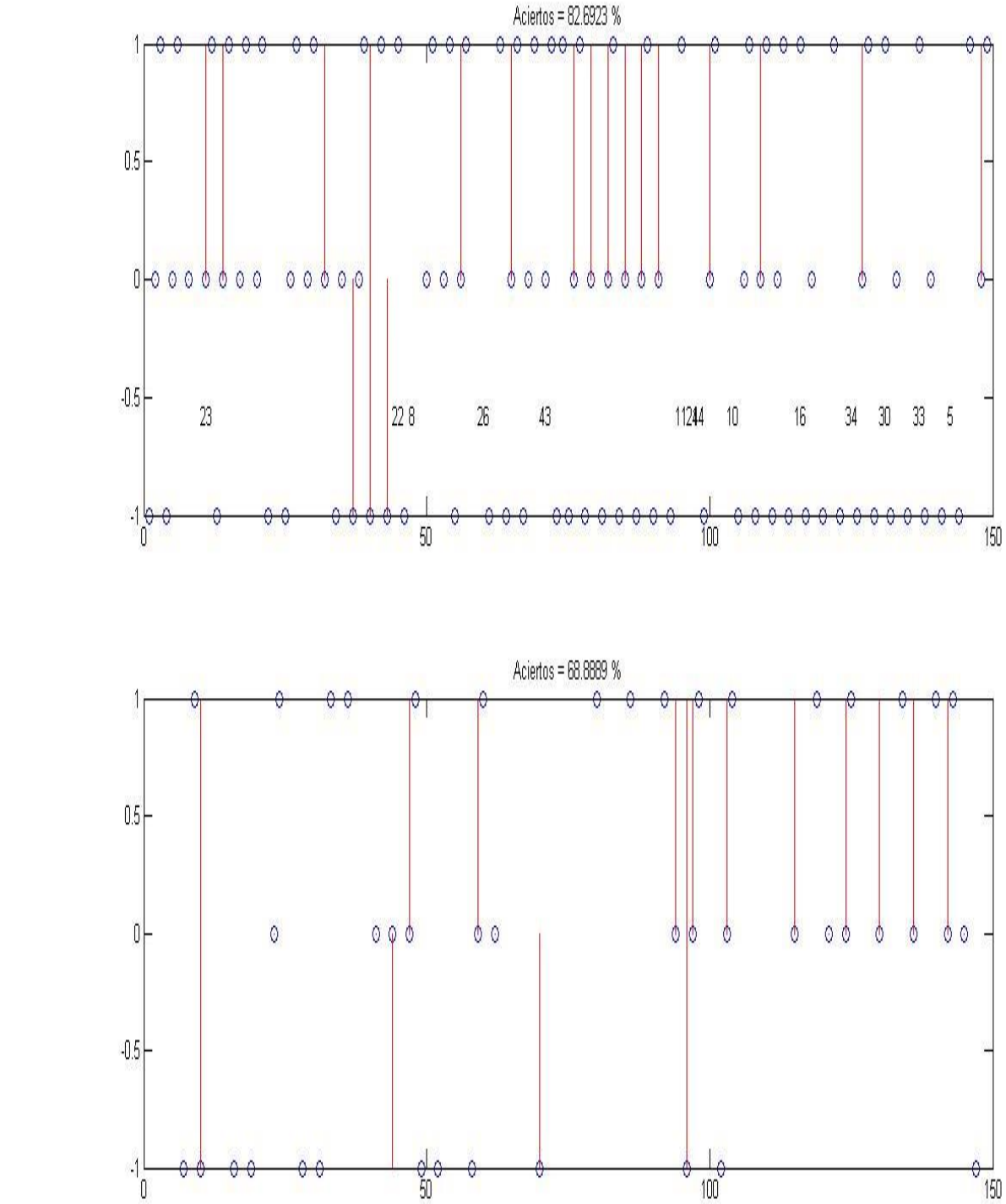


Figura 3.3.4 Conjunto de entrenamiento 120 datos y 30 datos de prueba.

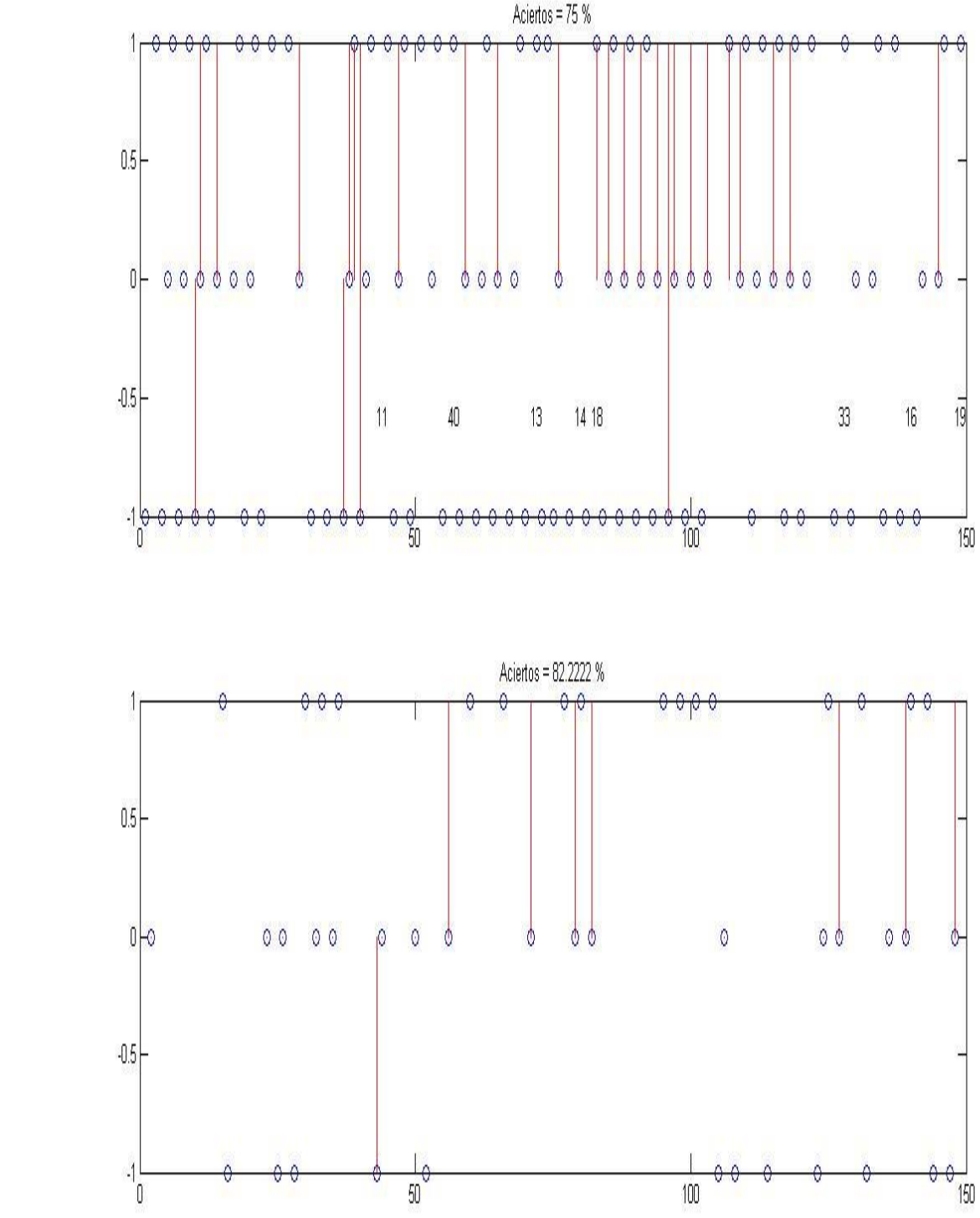


Figura 3.3.5 Conjunto de entrenamiento 120 datos y 30 datos de prueba.

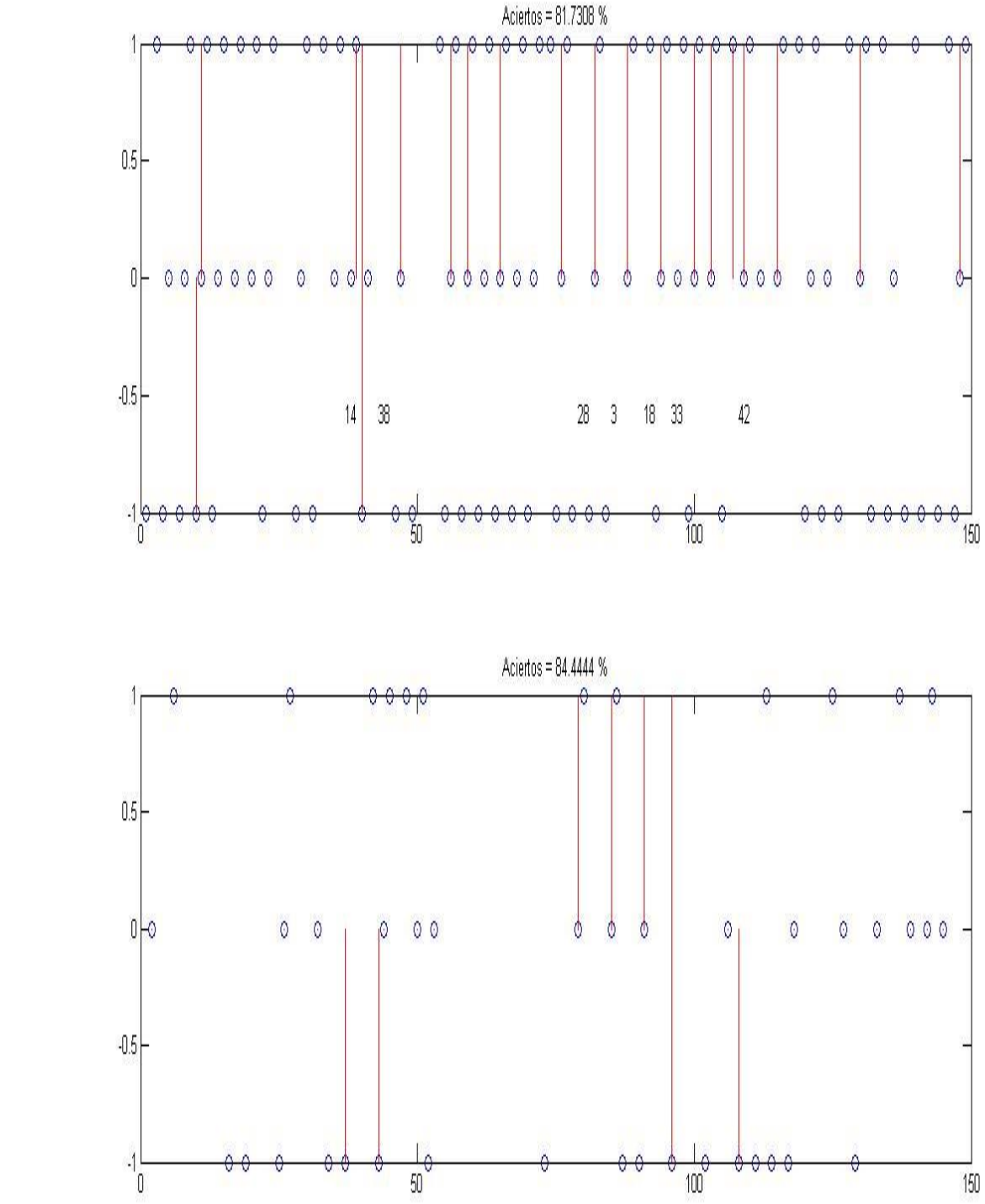


Figura 3.3.6 Conjunto de entrenamiento 120 datos y 30 datos de prueba.

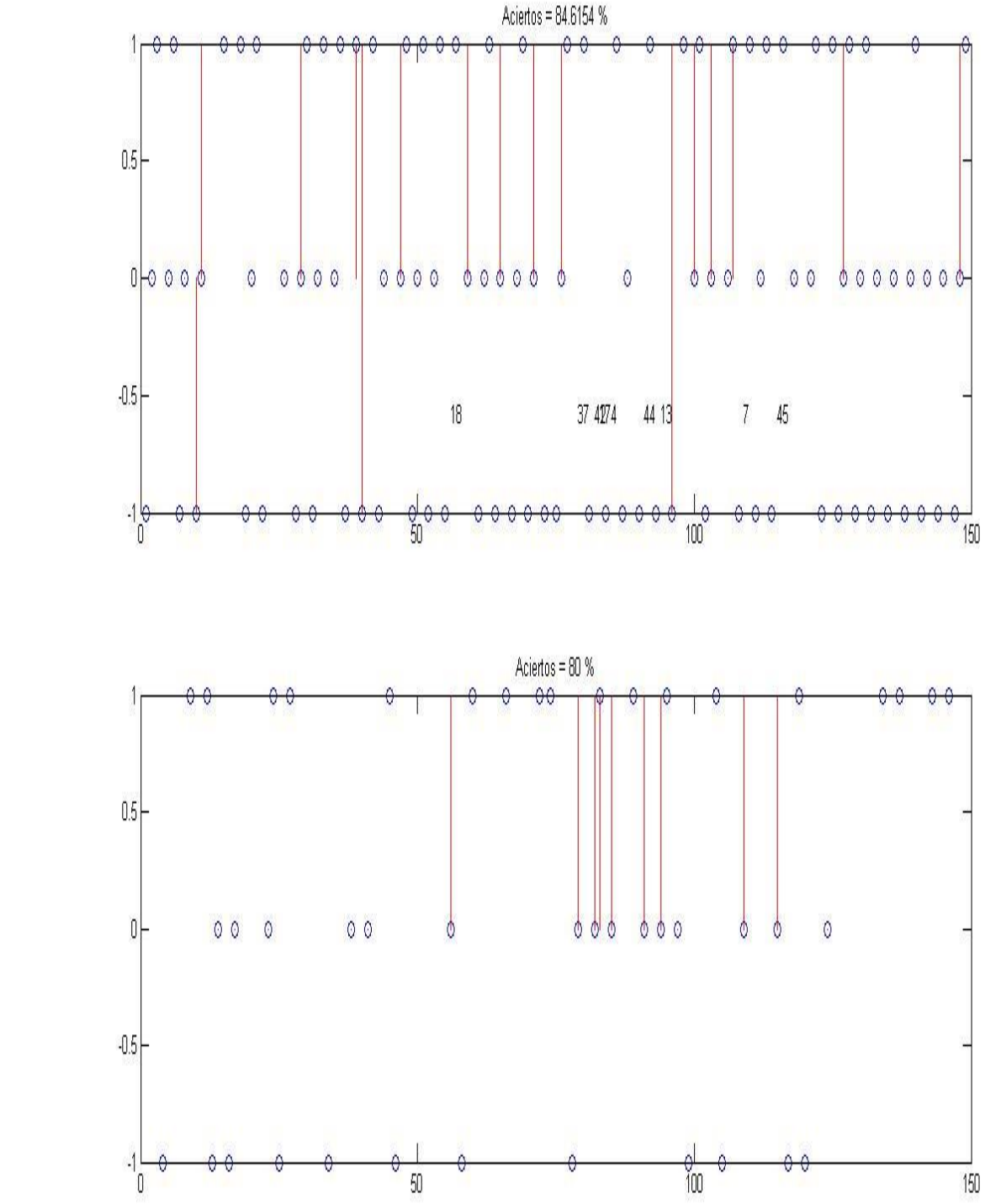


Figura 3.3.7 Conjunto de entrenamiento 120 datos y 30 datos de prueba.

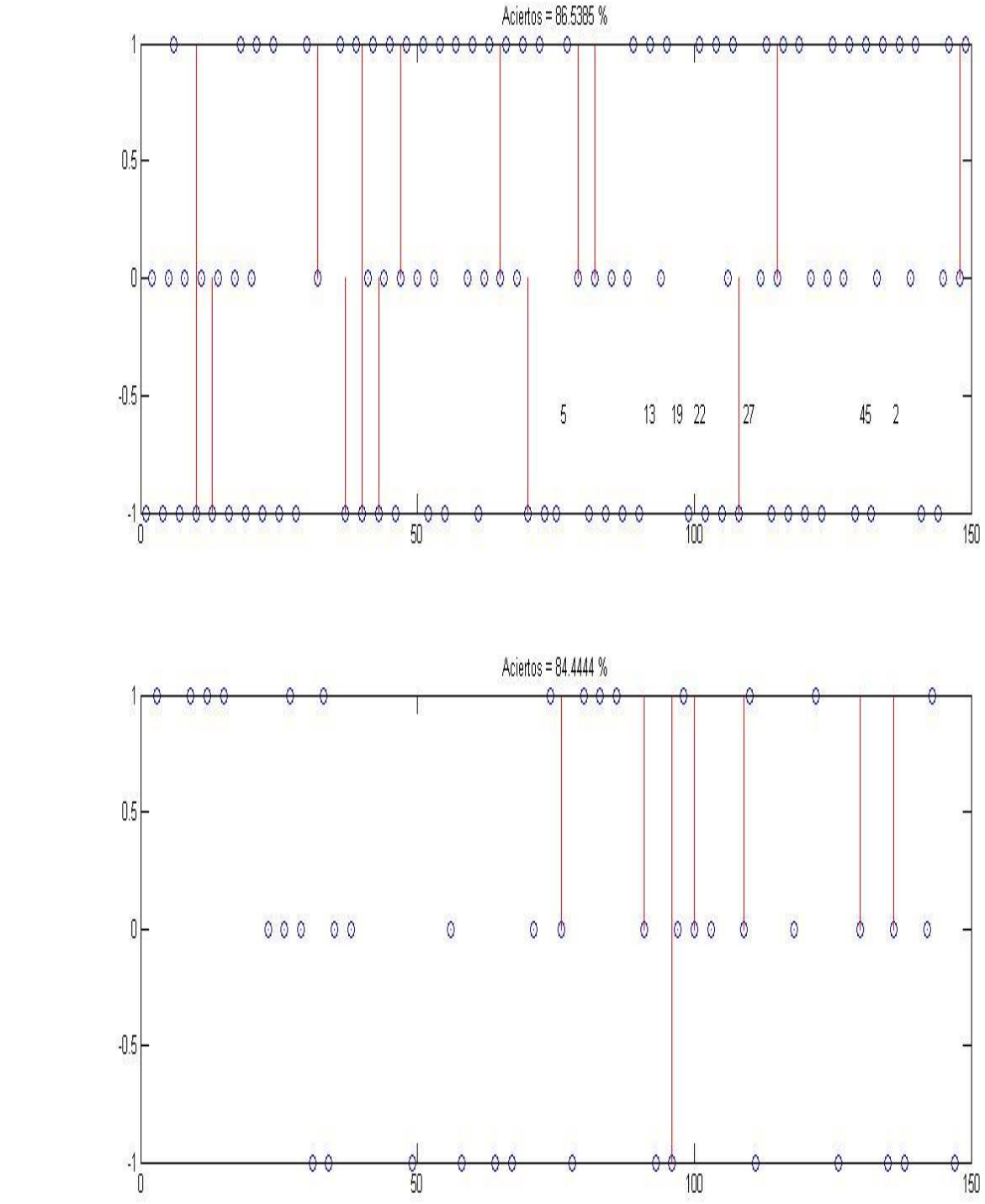


Figura 3.3.8 Conjunto de entrenamiento 120 datos y 30 datos de prueba.

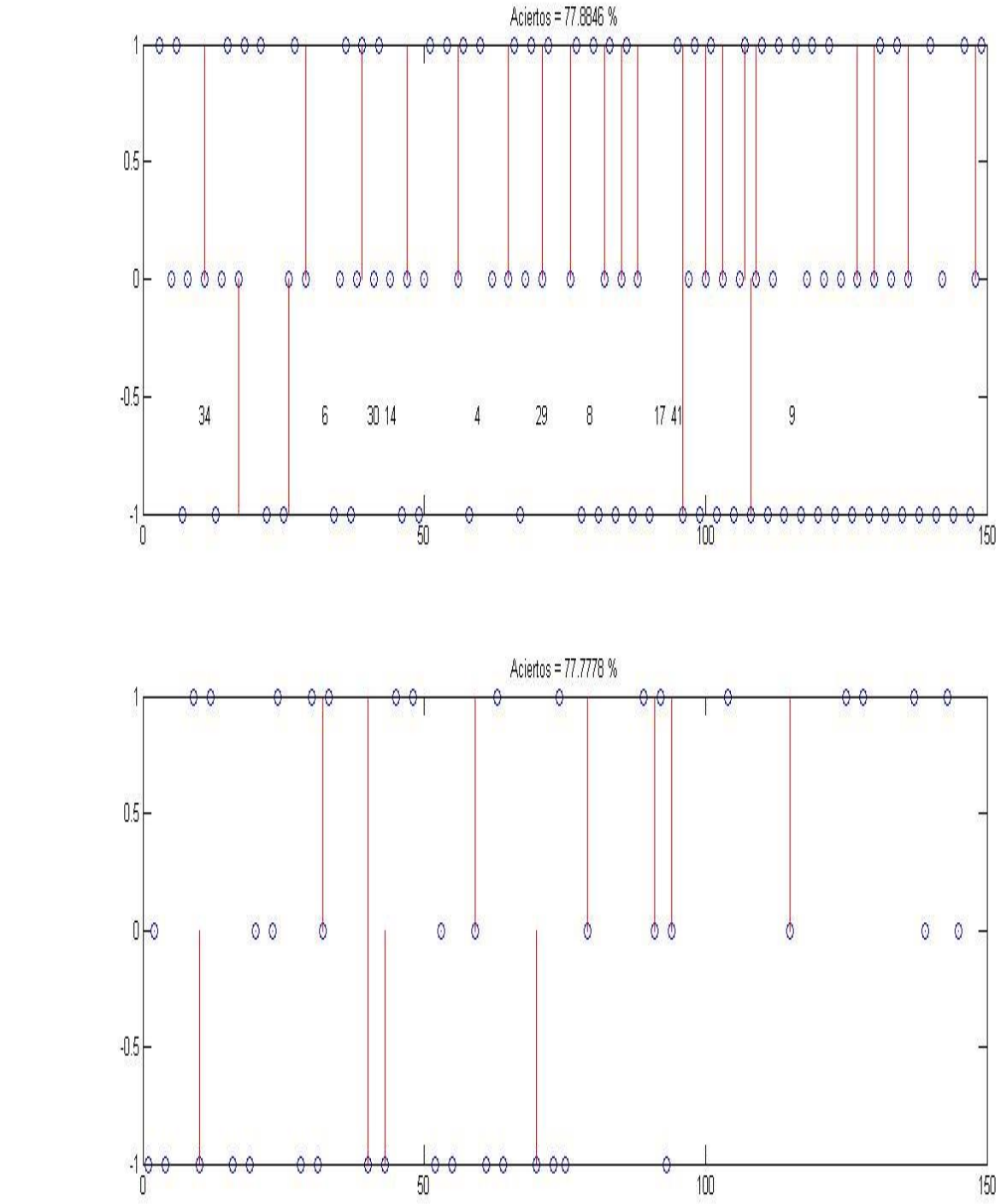


Figura 3.3.9 Conjunto de entrenamiento 120 datos y 30 datos de prueba.

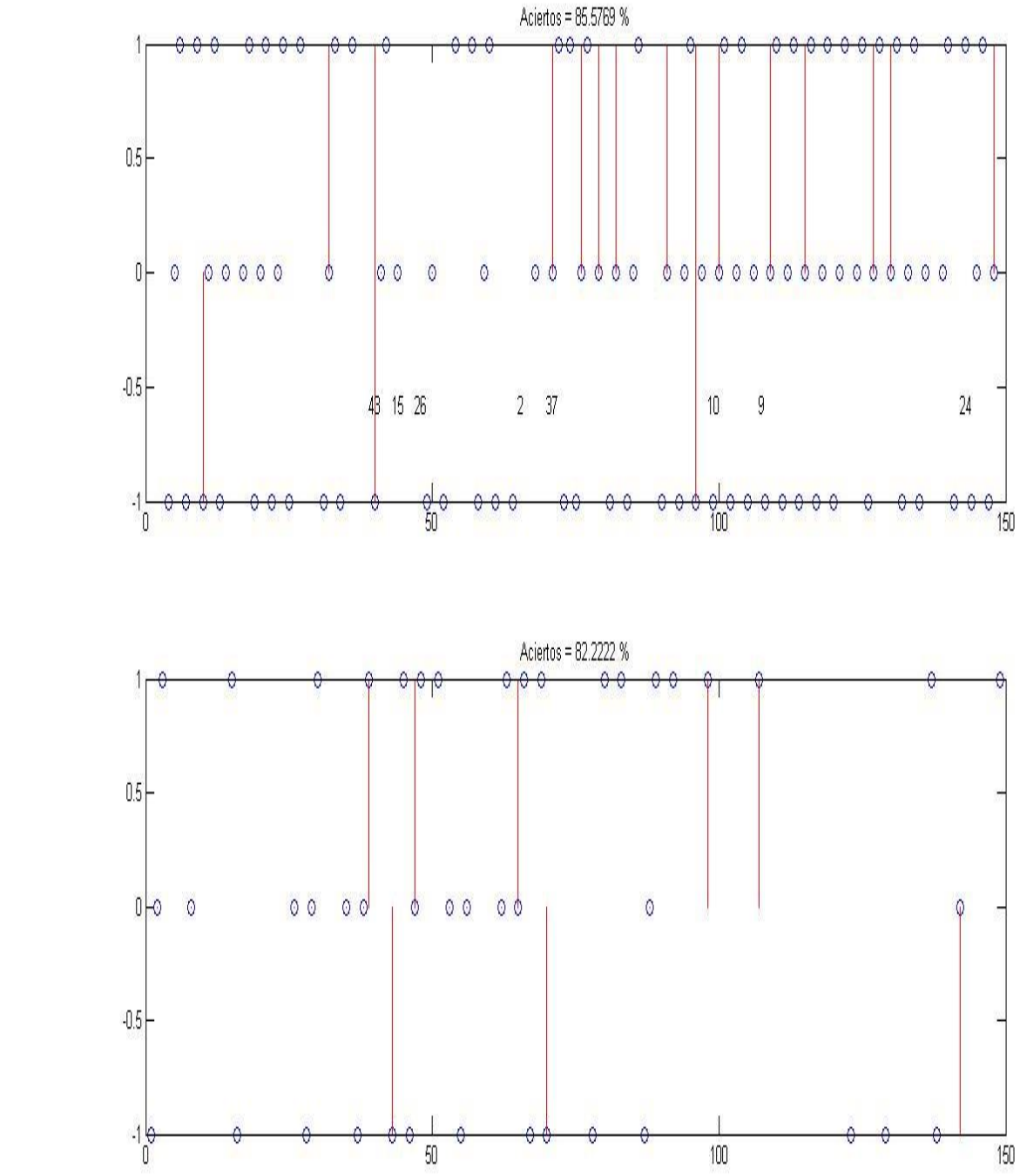


Figura 3.3.10 Conjunto de entrenamiento 120 datos y 30 datos de prueba.

3.4 Comparación de los resultados obtenidos con Perceptrón.

Los resultados que se obtuvieron en la evolución de redes aplicando AGs fueron muy buenos.

Estos resultados al ser comparados con los obtenidos con PMC son similares, pero es necesario tomar en cuenta que para el proceso de evolución de redes con el algoritmos genéticos desarrollado, el tiempo de procesamiento es más elevado que al aplicar PMC.

Para ambos casos en los resultados obtenidos el conjunto de datos de entrenamiento y de prueba es elevado.

En PMC se consideró un conjunto de datos de entrenamiento de 120 datos, y si comparamos con los obtenidos en el proceso de evolución los resultados más óptimos se empiezan a ver cuando el conjunto de datos de entrenamiento aumenta.

CAPITULO IV: CONCLUSIONES Y RECOMENDACIONES

En el siguiente capítulo presentamos las conclusiones que se han determinados después del proceso de obtención de resultados planteados en el tercer capítulo, se incluye también algunas recomendaciones que pueden servir de ayuda para futuras investigaciones sobre el mismo tema o aplicaciones de evolución de redes neuronales aplicando algoritmos genéticos para otros problemas que puedan plantearse; además se incluye un detalle del código fuente del programa desarrollado para el proceso de evolución con algoritmos genéticos, sobre las funciones desarrolladas y las variables principales que intervienen.

4.1 Conclusiones.

Dentro del proceso de evolución de redes neuronales hemos podido determinar que para el problema planteado el tiempo de procesamiento, tomando en cuenta el algoritmo genético desarrollado es muy alto.

También se han podido identificar que los mejores valores de aptitud se desarrollaron con conjuntos grandes de datos de entrenamiento. En el proceso de evolución los valores de aptitud óptimos de acuerdo a los resultados obtenidos se fueron generando a medida que el tamaño de conjunto de datos de entrenamiento aumentaba.

En comparación con los resultados obtenidos con PMC, utilizando el algoritmo que por defecto implementa Matlab, se puede determinar que para este problema es bastante óptimo aplicar algoritmos genéticos, siempre y cuando se tenga un conjunto de datos de entrenamiento alto, tomando en cuenta que con PMC el tiempo de entrenamiento de una red neuronal es mucho menor.

De acuerdo a los resultados obtenidos con las estructuras finales determinadas después del entrenamiento podemos concluir que sus resultados tienen similares resultados en cuanto a la aptitud, lo que nos indica que es factible un entrenamiento de estas estructuras de red para el problema planteado, debiendo tomar en cuenta la complejidad de estructura de la red.

El entrenamiento con algoritmos evolutivos para problema especificado puede ser aplicado para obtener resultados más óptimos y competitivos comparados con otros modelos; considerando que la búsqueda es sobre un

modelo de entrenamiento exhaustivo, para el cual se requiere un conjunto grande de datos de entrenamiento, y el tiempo de procesamiento es bastante alto en comparación a los otros modelos.

El entrenamiento planteado dentro de la evolución de redes neuronales para este problema específico es aplicable dentro de otros parámetros, ya sea dentro del mismo programa donde se pueden realizar cambios ya sea en proceso de cruce, mutación, selección, así como en la función de aptitud, esto siempre teniendo como objetivo fundamental el mejorar el rendimiento del proceso de evolución.

Por otro lado ya que el problema de clasificación de sismos es un problema específico, es necesario determinar la complejidad del mismo, así entonces en el proceso de análisis, definición e implementación del problema se puede determinar que los valores obtenidos han estado determinados de la manera más óptima.

4.2 Recomendaciones.

El proceso de evolución de red desarrollado en la tesis puede ser aplicado a cualquier problema en general en un proceso de aprendizaje supervisado donde se tienen *X* y *Ydeseado*, debiendo considerar un punto importante que en el proceso de evolución de redes para cada problema planteado la función de aptitud debe desarrollarse en base al mismo debiendo tomar en cuenta cual es el objetivo principal de la función de aptitud, en el problema planteado tomando en cuenta aprendizaje supervisado el objetivo es minimizar el error.

Así mismo el proceso de evolución de redes desarrollado en la tesis, puede ser evaluado con otro tipo de métodos en cada uno de los procesos que intervienen y poder así mismo de ser posible obtener resultados favorables en el entrenamiento.

REFERENCIAS y BIBLIOGRAFIA.

1. Niskaa Harri, Hiltunena Teri, Karppinenb Ari, Ruuskanena Juhani, Kolehmainena Mikko (2004). “Evolving the neural network model for forecasting air pollution time series”.

2. Wong M.H., Chung T.S., Wong Y.K. “An evolving neural network approach in unit commitment solution”.

3. Holland, J. (1992). "Adaptation in Natural and Artificial System". MIT Press.

4. Koza, J. (1992). "Genetic Programming: On the programming of computers by means of natural selection". Bradford Books, Cambridge.

5. Hitch Hiker. "The Hitch Hiker's Guide to Evolutionary Computation".

6. Holland, J. "Festschrift Conference in Honor".

7. Yao Xin. “Evolving Artificial Neural Networks”. School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, United Kingdom.

8. Matich Jorge Damián. “Redes Neuronales: Conceptos Básicos y Aplicaciones.”

9. Aguilar Jorge (200-2004), Apuntes de Inteligencia Artificial II, Perceptron Multicapa, Cap 3, PUCE

10. Aguilar Jorge (200-2004), Apuntes de Inteligencia Artificial II, Algoritmos Genéticos, Cap 8, PUCE

11. Aguilar Y Troncoso (2005), Clasificación de sismos volcánicos mediante redes neuronales, Reporte Interno, Instituto Geofísico,

12. Aragón Alberto (2002), “Tesis Métodos Evolutivos para el Aprendizaje de Redes Neuronales”.

ANEXO A

DESCRIPCIÓN CÓDIGO FUENTE

Para la implementación del Algoritmo Genético descrito en el proceso de evolución de redes para el problema planteado, utilizando Matlab, se desarrollaron varios archivos, tomando como base un archivo principal, en el cual se encuentran las variables principales que intervienen en el proceso de ejecución del programa, también donde se obtienen los datos necesarios para el entrenamiento y prueba del proceso de evolución; y además en este archivo está el proceso principal del Algoritmo Genético desde donde se llaman a las funciones del mismo.

Estas funciones fueron elaboradas en diferentes archivos para poder identificar el proceso de cada una de las funciones como son, la función de aptitud, función de cruce, función de mutación, etc.

A continuación se hará un breve detalle del cada unos de los archivos creados así como de las variables principales.

Archivos:

Archivo	Descripción
Pesos.m	Archivo principal que contiene las variables principales para configuración del usuario
Generar.m	Función que permite de generar la población inicial en el proceso de evolución en código binario
Generargray.m	Función que permite realizar la codificación Gray para el proceso de evolución implementado
Generarbinario.m	Función que permite trasformar el código Gray a binario, para poder determinar la aptitud de la población.
Aptitud.m	Función para evaluar la aptitud de los individuos en cada iteración, para el conjunto de datos de entrenamiento.
Aptitudredfinal.m	Función para evaluar la aptitud de las estructuras de red determinadas en el proceso de evolución, para el conjunto de datos de prueba.
Seleccion.m	Función que implementa el método de selección de ruleta.
Cruceuniforme.m	Función que realizar el proceso de cruce uniforme o máscara de cruce de los individuos seleccionados.
Mutar.m	Función para realizar el proceso de mutación.
Evaluar.m	La función evaluar nos permite determinar el mejor valor tomando en cuenta los individuos resultantes del proceso de evolución.
Evaluarredfinal.m	La función evaluarredfinal nos permite determinar la mejor red tomando en cuenta los individuos resultantes del proceso de evolución, recibe como parámetros los datos de la estructura de red óptima.

En el archivo Pesos.m como se indica en la descripción las variables principales las cuales son establecidas por el usuario son:

1. La variable datos nos permite obtener los valores del conjunto de datos de entrenamiento y prueba.

   datos=xlsread('c:\MATLAB7\work\Tesis\TipoCotopaxi.xls');

2. En el inicio del proceso de evolución de redes se estableció la variable maximo_ocultas determinaría el número máximo de neuronas en la capa oculta para establecer las mejores estructuras de red.

   maximo_ocultas=10;

3. Esta variable permite configurar el valor de la probabilidad de cruce.

   probabilidad_cruce=0.8;

4. Esta variable permite configurar el valor de la probabilidad de mutación.

   probabilidad_mutacion=0.01;

5. Esta variable permite configurar el valor del porcentaje de error.

   error=3;

6. Esta variable permite configurar el valor máximo de generaciones para el proceso de evolución.

   maxima_generacion=10;

7. Esta variable permite configurar el valor máximo del tamaño de población para el proceso de evolución, en cada estructura de red.

   tamano_poblacion=30;

8. Esta variable permite configurar el valor máximo del conjunto de redes que se desea evolucionar

```
maximo_redes=5;
```

9. En el inicio del proceso de evolución de redes se estableció la variable neuronas_capa_oculta la que determinaría el número de neuronas en la capa oculta para el proceso de evalución, este valor cambiaría en cada iteración donde que esta determinado por el conjunto de redes a ser evolucionas.

   ```
   neuronas_capa_oculta=round(rand*(maximo_ocultas-1))+1;
   ```

   El valor de esta variable luego de haber seleccionado el conjunto solución será constante en el desarrollo es decir como se explicó en la teoría el conjunto solución establecido luego de un proceso de evolución de estructuras generado aleatoriamente es: 4:3:2, 4:4:2, 4:5:2, 4:6:2, 4:7:2

   Por lo que en el proceso de evolución de cada estructura el valor de esta variable tomará los valores de 3,4,5,6,7.

AGRADECIMIENTO

En primer lugar agradezco a Dios quien me ha permitido alcanzar mis metas con el apoyo de mi familia.

A mi Padre le agradezco su paciencia, su perseverancia y apoyo constantes no solo en el desarrollo de mi tesis sino a lo largo de mi vida personal y profesional, su guía ha sido muy valiosa en mi vida y estoy seguro de haber tenido un gran ejemplo a seguir y me siento muy orgulloso.

A mi Madre un ser muy especial en mi vida, con su cariño, amor, dedicación y esmero quien a pesar de todos los errores que como hijos cometemos, ha estado conmigo y siempre con sus palabras de aliento me han ayudado a seguir con orgullo cada día y lograr mis metas.

A mis hermanos, gracias por todo, por estar siempre conmigo a lo largo de mi vida y mi desarrollo profesional, su apoyo siempre ha sido muy valioso para mi y además al ser yo el menor de los hermanos siempre he tenido a dos grandes ejemplos de vida y dedicación.

A mi Director de Tesis un agradecimiento muy especial ya que gracias a su guía me permitió cumplir con los objetivos planteados y mucho más ya que siempre uno tiene cosas que aprender y mucho más de quien desinteresadamente está dispuesto a brindar su apoyo constante.

A mis Correctores de Tesis, que sin duda sin su apoyo no hubiera sido posible tampoco la culminación de este proyecto.

DEDICATORIA

Mi tesis está dedicada a mis padres y hermanos, este logro profesional es un orgullo individual del cual me siento muy orgulloso de compartir con mi familia.

A mis padres no solo les dedico este logro como un logro profesional ya que no solo lo considero así, les dedico como un éxito en mi vida.

INDICE

Contenido.

CAPITULO I: REDES NEURONALES Y EVOLUCIÓN EN REDES……1

1.1 Breve Introducción a la Computación Evolutiva……..……2

1.2 Breve Introducción a Redes Neuronales……..……3

1.3 Evolución en Redes Neuronales……..……..5

1.4 Análisis y diseño de la Red Neuronal a utilizar - PMC.……..6

1.4.1 Entrenamiento evolutivo versus entrenamiento de Basado en el Gradiente…….....6

1.4.2 El Perceptrón Multicapa……..……8

1.4.2.1Aprendizaje de la Red de Retropropagación (RBP)……10

1.5 Algoritmos Genéticos……13

1.5.1 Representación de la solución…..……16

1.5.2 Operadores.……18

1.5.2.1 Operadores de Selección……...…18

1.5.2.2 Operadores de Búsqueda……..19

1.5.3 Resolviendo un problema: Genotipo y Aptitud……...22

1.5.4 Genotipos no convencionales……...24

1.6 Algoritmo Genético para la evolución de redes neuronales……26

1.6.1 Determinación del Algoritmo Genético a utilizar.……...26

CAPITULO II: EVOLUCIÓN DE LA RED NEURONAL A TRAVÉS DE ALGORITMOS GENÉTICOS……..27

2.1 Definición del problema de clasificación de sismos……..28

2.1.1 Eventos de largo Período (LP)……..29

2.1.2 Eventos híbridos……..30

2.1.3 Eventos Volcano – Tectónicos……..31

2.1.4 Definición de variables……...33

2.2 Herramienta a utilizar e implementación del algoritmo……....33

2.3 Análisis del problema……34

CAPITULO III: OBTENCIÓN, EVALUACIÓN Y COMPARACIÓN DE RESULTADOS……..60

3.1 Clasificación de resultados……...61

3.2 Evaluación de los resultados obtenidos……78

3.3 Entrenamiento de un Perceptrón Multicapa para la clasificación de sismos……79

3.4 Comparación de los resultados obtenidos con Perceptrón……90

CAPITULO IV: CONCLUSIONES Y RECOMENDACIONES…..……91

4.1 Conclusiones……...92

4.2 Recomendaciones……..93

Referencias y Bibliografía……..……95

Anexos…..……97

Introducción.

En la actualidad el estudio de la Inteligencia Artificial ha abarcado un amplio campo de aplicación.

Dentro de este campo de estudio el proceso de evolución de redes neuronales es uno de ellos con la aplicación de computación evolutiva.

El proceso de evolucionar redes neuronales aplicando algoritmos genéticos tiene como objetivo el optimizar los resultados que se obtienen aplicando otros modelos, basados en la aplicación de otros algoritmos.

En este proceso de evolución es necesario poder determinar los mejores métodos de cada uno de los procesos que intervienen en la evolución de redes neuronales con la aplicación de algoritmos genéticos.

Para el problema específico que es el de clasificación de sismos existen otros métodos de optimización de resultados, entre ellos está la aplicación de redes neuronales con algoritmos de retropropagación; tomando en cuenta esto nos centraremos en el estudio de este tipo de aplicación para poder realizar la comparación de los resultados obtenidos y de esta forma identificar si el proceso de evolución planteado nos permite optimizar los resultados que actualmente se obtienen.

A lo largo del desarrollo de la tesis encontraremos la información necesaria de sobre el proceso de evolución de redes con todos los aspectos más relevantes y que nos permitirán comprender de mejor forma el proceso.

INDICE DE FIGURAS Y TABLAS

FIGURAS.

Figura 1.4.2.1 Perceptrón Multicapa con dos capas ocultas……..8

Figura 1.4.2.2 Sentidos de propagación de las señales en un perceptrón multicapa……..8

Figura 1.4.2.1.1 Gráfico de señales en una neurona de salida *k* conectada a una neurona oculta *j*……....11

Figura 1.5.2.2.1 Método de un punto de cruce……19

Figura 1.5.2.2.2 Método de dos puntos de cruce……..19

Figura 1.5.2.2.3 Cruce uniforme generando una máscara de cruce…..…….20

Figura 1.5.2.2.4 Método de cruce……..21

Figura 1.5.2.2.5 Un cromosoma formado por cuatro parámetros……...22

Figura 2.1.1.1 Sismo de largo período clásico……28

Figura 2.1.2.1 Evento híbrido del volcán Cotopaxi……29

Figura 2.1.3.1 Sismo volcano-tectónico del volcán Cotopaxi……30

Figura 2.3.1 Método de cruce basado en un punto elaborado por Rafael Gordillo……49

Figura 2.3.2 Método de cruce basado en dos puntos elaborado por Rafael Gordillo……50

Figura 2.3.2 Método de cruce uniforme elaborado por Rafael Gordillo...……...51

Figura 2.3.2 Método de mutación elaborado por Rafael Gordillo……53

Figura 3.1.1 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución……..62

Figura 3.1.2 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución……..64

Figura 3.1.3 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución……..65

Figura 3.1.4 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución……..66

Figura 3.1.5 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución……..67

Figura 3.1.6 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución……..68

Figura 3.1.7 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución……..69

Figura 3.1.8 Entrenamiento de diferentes estructuras para determinar el mejor conjunto solución……..70

Figura 3.1.9 Conjunto de entrenamiento 120 datos y 30 datos de prueba. Número de neuronas de capa oculta: 3……72

Figura 3.1.10 Conjunto de entrenamiento 120 datos y 30 datos de prueba. Número de neuronas de capa oculta: 4……73

Figura 3.1.11 Conjunto de entrenamiento 120 datos y 30 datos de prueba. Número de neuronas de capa oculta: 5……74

Figura 3.1.12 Conjunto de entrenamiento 120 datos y 30 datos de prueba. Número de neuronas de capa oculta: 6……75

Figura 3.1.13 Conjunto de entrenamiento 120 datos y 30 datos de prueba. Número de neuronas de capa oculta: 7……76

Figura 3.3.1 Conjunto de entrenamiento 120 datos y 30 datos de prueba (PMC)……..78

Figura 3.3.2 Conjunto de entrenamiento 120 datos y 30 datos de prueba (PMC)……..79

Figura 3.3.3 Conjunto de entrenamiento 120 datos y 30 datos de prueba (PMC)……..80

Figura 3.3.4 Conjunto de entrenamiento 120 datos y 30 datos de prueba (PMC)……..81

Figura 3.3.5 Conjunto de entrenamiento 120 datos y 30 datos de prueba (PMC)……..82

Figura 3.3.6 Conjunto de entrenamiento 120 datos y 30 datos de prueba (PMC)……..83

Figura 3.3.7 Conjunto de entrenamiento 120 datos y 30 datos de prueba (PMC)……..84

Figura 3.3.8 Conjunto de entrenamiento 120 datos y 30 datos de prueba (PMC)……..85

Figura 3.3.9 Conjunto de entrenamiento 120 datos y 30 datos de prueba (PMC)……..86

Figura 3.3.10 Conjunto de entrenamiento 120 datos y 30 datos de prueba (PMC)……..87

TABLAS.

Tabla 3.2.1 Cuadro comparativo de los resultados obtenidos en el entrenamiento con Algoritmos Genéticos…….77

INDICE DE FORMULAS

Fórmula 1 Conjunto de datos de entrada (X)……9

Fórmula 2 Conjunto de datos de *Y deseado*……9

Fórmula 3 Cálculo de valor de error capa de salida……..10

Fórmula 4 Cálculo de valor de error capa de oculta……..10

Fórmula 5 Actualización de pesos capa de salida……10

Fórmula 6 Actualización de pesos capa de oculta……10

Fórmula 7 Cálculo de Y hacia delante (capa oculta)……10

Fórmula 8 Cálculo de Y hacia delante aplicando nivel de activación Y_k (nivel de capa oculta)……10

Fórmula 9 Cálculo de Y hacia delante (capa salida)……..11

Fórmula 10 Cálculo de Y hacia delante aplicando nivel de activación Y_j (nivel de capa de salida)……11