UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

REDES NEURONALES EVOLUCIONADAS CON ALGORITMO GENETICO

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

DANIEL RAMIREZ AVILA

DIRECTOR DE TESIS: M.I. JUAN MANUEL GOMEZ GONZALEZ

CIUDAD UNIVERSITARIA, MAYO 2004

Dedicatorias:

A mi Madre:
Por su apoyo incondicional y la educación que me brindo.

Araceli:
Por ser la inspiración para ser mejor y seguir adelante.

Agradecimiento:

Al Grupo LINDA, por una formación verdadera.

Redes Neuronales Evolucionadas con Algoritmo Genético

Índice General

Capítulo 1.

Introducción:

El uso de cómputo suave en problemas de difícil acercamiento, es cada vez mas común en diversas áreas del conocimiento. Al igual que la mezcla de estas técnicas para aumentar el desempeño es cada vez más frecuentemente, y esto ha sido un área de la investigación desde los primeros desarrollos en el área del cómputo suave.

Dado esto se ha descubierto que la combinación de habilidades que pudieran denominarse como una característica humana o que de una u otra manera se encuentran intrínsecas en la naturaleza, a innovado desarrollos en técnicas para atacar problemas reales por medio de estas herramientas de computo.

Siendo ejemplos de estas herramientas las siguientes:

- Redes Neuronales
- Lógica Difusa
- Algoritmos Genéticos
- Programación Genética

Y cualquier otra variación de estas, ya que a través de los años, y con el estudio de las técnicas se han creado una cantidad enorme de variaciones y adaptaciones dependiendo del autor o del programador, y alteraciones en la forma en que obtuvo sus resultados. De igual manera parte también afecta la forma en que se ataca el problema con sus características internas. Por otra parte haciendo una analogía, ya como también nos sucede a nosotros como humanos, al buscar la solución a un problema, comúnmente se tiene una metodología o camino para llegar a la misma por cada individuo involucrado, dependiendo la información que tenga cada uno, de la experiencia personal y la forma de apreciar el conocimiento. Por lo que podríamos ejemplificar esto con una cita de una definición de

Inteligencia Artificial: "La programación de Inteligencia Artificial es cuando programas operan datos, siguiendo reglas en búsqueda de metas" (W.A. Taylor).

En esta Tesis abordaremos el uso de dos de estas técnicas complementándose entre si, las redes neuronales y los algoritmos genéticos, para generar una herramienta que permite atacar una amplia gama de problemas de reconocimiento bajo el esquema de asociar una entrada contra una salida, en problemas no lineales. Ya que prácticamente cualquier problema de reconocimiento lo podríamos definir bajo este ámbito.
Incrustando una técnica dentro de la otra, usando una herramienta de aprendizaje como son las redes neuronales, las cuales dependen de una arquitectura y sus entradas se ven asociadas a una salida, buscamos la modificación de estos parámetros de diseño de las redes neuronales, por medio de una segunda herramienta como son los algoritmos genéticos, los cuales se utilizan para adaptar la arquitectura y las entradas de las redes neuronales en función del desempeño después de su entrenamiento y de la complejidad de su arquitectura. El primer concepto del desarrollo de esta herramienta fue siguiendo una filosofía muy común entre los programadores y gente que trabaja en las áreas de la inteligencia artificial y cómputo suave, tendiendo a programar herramientas que libren tarde o temprano al programador de seguir programando. Esto puede sonar muy extraño, pero evidentemente las herramientas mejores por excelencia son aquellas que ahorran tiempo a la persona que las desarrolla o que las usa, para lograr mejores resultados en un periodo menor de tiempo, o requiriendo una inversión parcialmente menor.

En los capítulos siguientes describiremos brevemente estas herramientas y mas tarde como se realizo la mezcla de estas técnicas, para generar una herramienta nueva, la cual fue programada bajo el ambiente de Matlab, y también explicaremos las pruebas que se realizaron generando redes con su respectiva arquitectura para ajustar un modelo financiero.

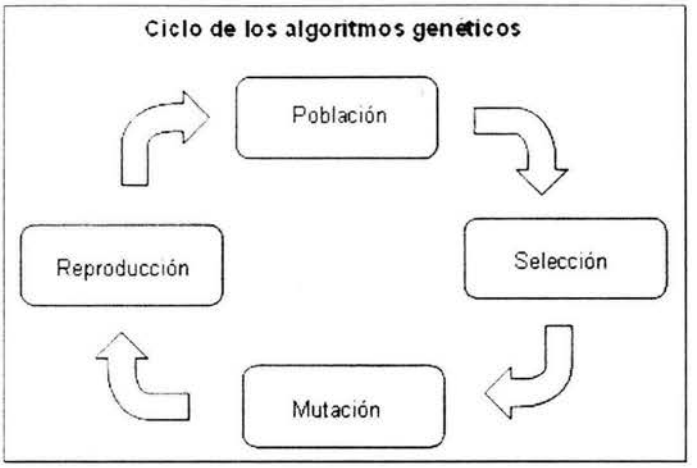
Capítulo 2.

2.1 Algoritmos Genéticos

Descritos inicialmente por John Holland en 1957, como una aproximación biológica de resolver problemas en diversas áreas. Todos los métodos y teoría necesarios fueron después analizados por Goldberg en 1989 [1]. Los algoritmos genéticos tienen varios usos que pueden ir desde búsquedas, hasta aprendizaje pasando por optimización y generación de ecuaciones y diversos modelos matemáticos [2].

En esta técnica las poblaciones que barren el espacio de búsqueda, simulando individuos representados por vectores codificados, que son afectados por medio de operadores genéticos, generación tras generación, realizando interacciones entre si, en base a la aptitud de cada individuo, que es la fuerza básica de la Selección Natural Darwinista [3]. Donde la aptitud de un individuo nos da su probabilidad de sobrevivir y de lograr crear descendencia hacia la siguiente generación.

Los algoritmos genéticos operan en un ciclo muy típico alrededor de una población de individuos declarados de manera aleatoria, denominada población inicial, los cuales representan puntos dentro del espacio de búsqueda, esta población es trasformada a lo largo de las generaciones en la búsqueda de un objetivo. Podemos definir la estructura general de un ciclo de evolución de la siguiente manera:



Este ciclo puede tener variaciones en sus parámetros, modos de evaluación y demás formas de correr el algoritmo.

Estas son las características principales que diferencian a un algoritmo genético de los demás métodos de búsqueda:

1. Los algoritmos genéticos trabajan con la codificación de los parámetros, no con los parámetros por si mismos.
2. Los algoritmos genéticos buscan con una población de puntos en el espacio, no con punto único.
3. Los algoritmos genéticos hacen uso de la función objetivo, no de información adicional.
4. Las reglas de transición son probabilísticas, no determinísticas.

El uso de otra técnicas de optimización usualmente requieren mas parámetros, información adicional, como pueden ser derivadas como en las técnicas de gradientes. El uso de algoritmos genéticos no requieren otro tipo de información adicional para lograr optimizar la función ni de recorrer el espacio de búsqueda de manera exhaustiva.

Esto se hace solamente, con la evacuación de los individuos en la función objetivo, midiendo así la aptitud de los individuos, dicha función debe ser diseñada en base a los

parámetros y como los deseamos maximizar o minimizar en su caso. Siendo esta la función que le da forma al espacio de búsqueda del algoritmo evolutivo [4].

La ventaja mas clara con el uso de esta herramienta del computo suave es el paralelismo implícito del algoritmo [5], ya que al momento de declarar la población, cada individuo es un punto en el espacio de búsqueda que evoluciona o muere en búsqueda de tener una mejor aptitud, siendo modificado en base a las reglas y operadores genéticos antes mencionados, adaptando su forma para maximizar o minimizar según sea el caso sobre una función objetivo.

En un algoritmo genético se manejan tres operadores básicos:

- Reproducción
- Cruza
- Mutación

Descripción de los Operadores Genéticos:

Reproducción

El primer operador que mencionaremos es el operador de reproducción, este se puede describir como el proceso por el cual un individuo es copiado y permanece a la siguiente población, en función de su aptitud. Por lo que un individuo puede tener una o mas copias de si mismo, hacia la siguiente generación. Esta operación es una simulación del proceso de la selección natural Darwinista, donde los individuos más aptos para explotar al medio son los que sobreviven para poder dejar descendencia. En este caso los individuos o vectores que den un mayor valor en la función objetivo son los que sobreviven, aunque todo esto tiene variaciones, ya que no es forzoso que el algoritmo sea totalmente Greedy, y otros factores como el número de veces que se reproduce un individuo, y el porcentaje de la población que se puede ver afectado por este operador.

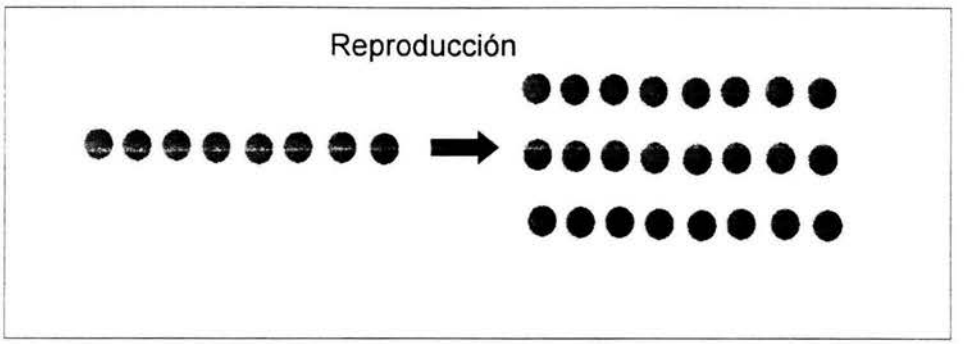


Fig. 1: Se muestra como la información de un individuo es copiada de manera exacta hacia la siguiente generación.

Operador de Cruza

Otro operador es el de cruza (crossover), este simula una reproducción sexual sobre dos individuos de la población ayudando en la expansión del espacio de búsqueda y en la creación de nuevos individuos con características similares, para realizar este proceso es necesario generar uno o más puntos de cruza (crossover points), en base a este punto de cruza, se toman la información de dos individuos padres y se intercala la información de los padres entorno a estos puntos de cruce, para generar nueva descendencia ya sea uno o dos hijos a partir de una pareja de padres [6].

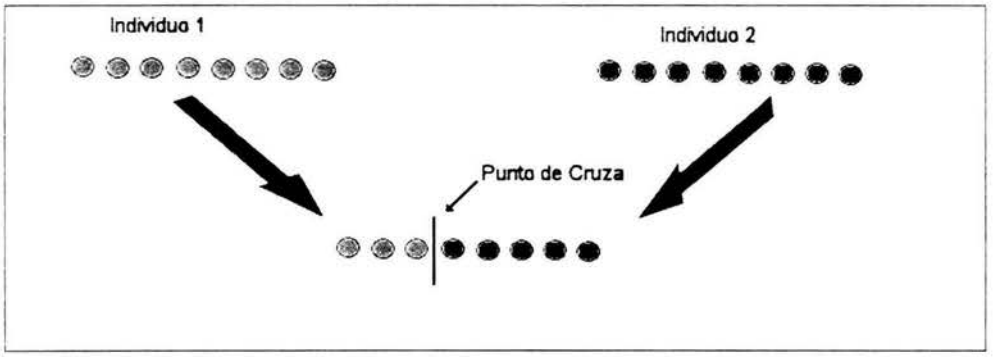


Fig. 2 Muestra la cruza entre dos individuos y como su información es unida al rededor de un punto de cruza

Algoritmos (m,l) y (m+l)

El operador de cruza puede tener varias aproximaciones diferentes, denominadas (m,l) y (m+l), diferenciándose entre si, por la manera en que cada uno de estas maneja la descendencia generada durante la cruza y si esta competirá con los padres o no durante el proceso de selección de la siguiente generación.

La característica principal de la modificación (m,l) es que procede a eliminar a los padres, ya una vez obtenida su descendencia. A fin de que los padres no compitan contra los hijos en la siguiente generación. Esta variación puede ayudar a que el algoritmo no caiga en mínimos o máximos locales, durante el proceso de optimización al recorrer el espacio de búsqueda. Pero por otro lado esta opción puede causar convergencias lentas durante el proceso evolutivo, a diferencia de un algoritmo de tipo (m+l).

El uso de un algoritmo (m,l), genera otros problemas al momento de controlar el crecimiento o decremento de la población. Obligando a que sea necesario un proceso de control del tamaño de la población durante el proceso evolutivo. (m,l) puede generar problemas de decremento del tamaño de la población, causados por la eliminación de los padres hacia la siguiente generación. Por lo que no es necesario verificar que el número de padres eliminados no sea menor al número de hijos generado. Ya que tener una diferencia entre estos podría causar variaciones no deseadas en la dinámica del tamaño de la población.

Por otro lado los algoritmos (m+l), el manejo de la población difiere a (m,l) ya que los padres sí compiten contra sus propios hijos, durante el proceso de selección de la siguiente generación, esto nos puede crear problemas, los cuales pueden afectar el rendimiento del algoritmo, ya que de tener una cantidad excesiva de elementos en la población, podría tomar demasiado tiempo al momento calcular la aptitud de cada elemento de la población en el algoritmo genético, por lo que es necesario que la dinámica del comportamiento del tamaño de la población este controlada por el programa. Otro problema que se puede generar, es que al momento de optimizar se caiga en mínimos o máximos locales, ya que un individuo con una aptitud relativamente alta, puede permanecer generación tras generación, sin que los hijos superen la aptitud de este individuo. Solo pudiéndose ver afectador por el operador de Mutación.

Mutación

El operador de mutación es el mecanismo por el cual se realiza una modificación puntual, sobre un sector de individuo, en base a una distribución de probabilidad, este operador es el encargado de evitar que durante el proceso de optimización se caigan en mínimos o máximos locales y la dinámica de la aptitud de la población se mantenga en una dinámica constante.

Fig. 3 Concepto de mutación puntual en la información de un individuo

2.2 La Neurona Biológica

Primero comencemos por definir un poco la célula en general, esta es una unidad mínima de un organismo capaz de actuar de manera autónoma. Todos los organismos vivos están formados por células, y en general se acepta que ningún organismo es un ser vivo sí no consta al menos de una célula. Algunos organismos microscópicos, como bacterias y protozoos, son células únicas, mientras que los animales y plantas están formados por muchos millones de células organizadas en tejidos y órganos. Aunque los virus y los extractos acelulares realizan muchas de las funciones propias de la célula viva, carecen de vida independiente, capacidad de crecimiento y reproducción propios de las células y, por tanto, no se consideran seres vivos. La biología estudia las células en función de su constitución molecular y la forma en que cooperan entre sí para constituir organismos muy complejos, como el ser humano [7].

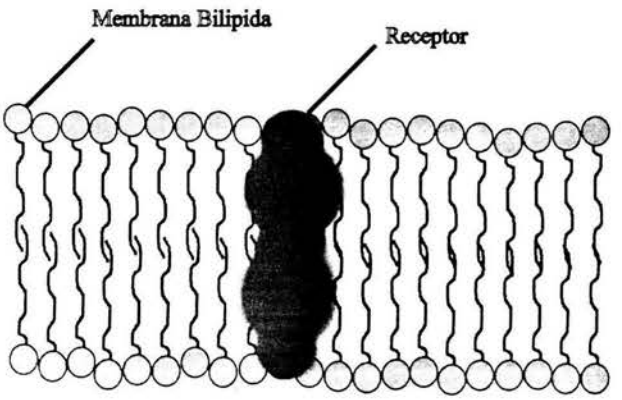
Características generales de las células

Hay células de formas y tamaños muy variados. Algunas de las células bacterianas más pequeñas tienen forma cilíndrica de menos de una micra o μm de longitud. En el extremo opuesto se encuentran las células nerviosas, corpúsculos de forma compleja con numerosas prolongaciones delgadas que pueden alcanzar varios metros de longitud. Casi todas las células vegetales tienen entre 20 y 30 μm de longitud, forma poligonal y pared celular rígida. Las células de los tejidos animales suelen ser compactas, entre 10 y 20 μm de diámetro y con una membrana superficial deformable y casi siempre muy plegada [7].

Dentro de las células, se encuentran las Neuronas, que son prácticamente iguales a las células, del resto del cuerpo, respecto a sus mecanismos y a los organelos que las componen, pero estas poseen la capacidad de comunicarse con otras neuronas.

2.2.1 Comunicación celular

La comunicación celular se realiza en la mayoría de los casos mediante substancias químicas que son producidas por una célula y se difunden por el medio extracelular hasta otra célula, sobre la que producen un efecto. Los mediadores químicos actúan uniéndose a un receptor, que es una proteína. Los receptores que también son en su mayoría una proteína, dado a que la mayoría de las moléculas son hidrofílicas, y de esta forma no pueden a atravesar la membrana, excepto que lo hagan a través de un receptor. Los cuales se encuentran incrustados en la membrana celular y la atraviesan hacia el interior [8].



En la transmisión entre células, hay dos etapas, la primera es cuando viaja el mediador y entra a la célula, una vez que llega el mediador al receptor, se produce otra molécula, en la cual la comunicación es interna de la célula, donde la información viaja por el citoplasma de la célula, a este proceso se le llama segundo mensajero, siendo los mas comunes:

Calcio, Inositol trifosfato, AMP cíclico, GMP cíclico.

2.2.2.2 Tipos de comunicación

Endócrina: El mediador se libera al torrente sanguíneo y actúa viajando a través de este. Este tipo de comunicación es a distancia y se conoce como hormonal.

Nerviosa: Es cuando la célula nerviosa transmite un impulso eléctrico y la terminación de esta célula, libera un agente químico, el cual se denomina neurotransmisor.

Paracrita: El mediador actúa solamente en las cercanías de la célula que lo libero, ya que la vida de estos mediadores es corta, de tal manera que no llegan lejos por que desaparecen antes de hacerlo.

Por contacto: Este solamente se da cuando las células se encuentran adyacentes y las proteínas en ambas membranas se unen para formar un poro, permitiendo el paso de ciertas moléculas entre las células.

2.2.3 Células Nerviosas

La célula nerviosa mejor conocida como neurona, tiene dos funciones principales siendo, la primera; la propagación del potencial de acción, o también llamado impulso o señal nerviosa, esta señal vieja a través del axón y su transmisión a otras neuronas induce así una respuesta en la célula receptora. Esta señal se logra a través del axón por medio de un fenómeno eléctrico causado por el intercambio de iones Na+ y K+. En cambio, la transmisión del impulso de una neurona a otra, depende de la acción de neurotransmisores (NT), siendo estos NT específicos dependiendo de los receptores [9].

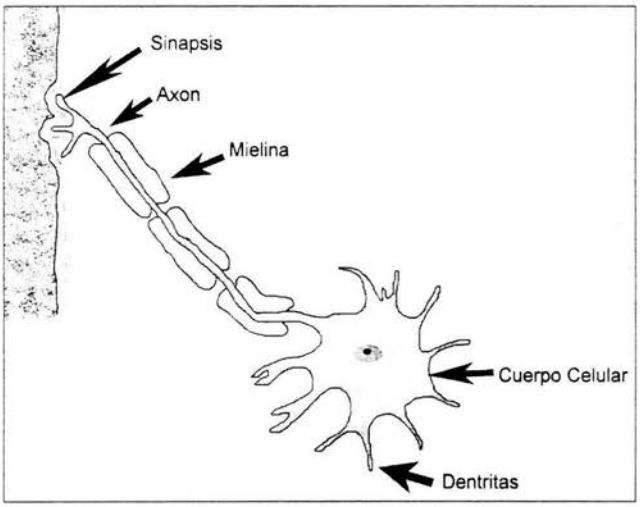


Figura. 1 Neurona biológica típica

2.2.4.1 Breve Historia de las redes neuronales

Se dice que las redes neuronales empezaron con el trabajo de McCulloch y Pill en 1943. Mas tarde Wiener describió conceptos importante sobre control, comunicaciones y mas tarde sobre aprendizaje y autoorganización. En 1949 llegó otro avancé importante con la publicación Hebb "The Organization of Behavior", donde se abordaron por primera vez los conceptos explícitos de aprendizaje psicológico y de modificaciones sinápticas, donde Hebb propuso teorías sobre la conectividad cerebral y de cómo esta cambia constantemente, para formar un modelo organizacional de aprendizaje, sufriendo modificaciones de conectividad con el aprendizaje de nuevas tareas. Con esto inspiro el avance de nuevos modelos computacionales de aprendizaje y sistemas adaptivos. Mas tarde en 1952, Sabih publico el libro "Design for a Brain: The Origin of Adaptive Behavior" donde se empezó a introducir el concepto de que los sistemas deben manejar un aprendizaje a través del entrenamiento en lugar de que este sea intrínseco del sistema y enfatizó en la dinámica de los organismos vivos. Comenzando un trabajo más formal en el área, fue el trabajo de Minsky en 1954, empezando a crear los primeros pasos en el trabajo de inteligencia artificial.

Poco a poco se empezó a migrar la técnica ya desarrollada hacia el área de la teoría de la comunicaciones con el trabajo de Gabor en 1954, mas tarde Gabor, entrenó redes alimentando muestras de procesos estocásticos. De manera simultánea hubo trabajo en el área de memoria asociativa por Taylor. Donde después se generaron aportaciones importantes en el área de la memoria asociativa por Kohonen (1972), Anderson (1972) y Nakano (1972), aportaron al trabajo de la matriz de correlación basado en *producto externo*. Dando un trasfondo más claro fue Little y Show en 1975 con el modelo probabilístico de la neurona. Llegando mas tarde avances de Hopfield del almacenaje de información en redes neuronales estables. Hasta que en 1986 fue el desarrollo de Rumelhart, Hinton y Williams, sobre el algoritmo de retropropagación, con el trabajo a priori de Parker y LeCun, que de manera separada descubrieron el aprendizaje por medio retropropagación.

2.2.4.2 Neuronas Artificiales:

La neurona es la base estructural del procesamiento de las redes neuronales, dentro de su estructura se pueden identificar tres elementos básicos [10]:

1. Un conjunto de sinapsis, donde cada uno de estas uniones esta caracterizada por un peso sináptico, donde la entrada asociada a esa sinapsis es multiplicada por el peso. Cada uno de estos pesos sinápticos W_{kj}, donde *k* es el número de la neurona y el subíndice *j* se refiere a la conexión, estos pesos pueden tomar dos rangos de valores, de ser negativo este peso, la conexión es inhibidora y en caso de ser positivo este el peso, la conexión es excitadota.
2. El sumador es donde todas las conexiones ya multiplicadas por sus respectivos pesos son conectadas, calcula la excitación o inhibición total de la neurona, en base a las entradas.
3. La función de transferencia limita y modula a la salida de la neurona en base a lo obtenido en el proceso de sumatoria. La amplitud normalizada de las funciones de transferencia es [-1,1] o en algunos casos [0,1].

Estructura general de una neurona:

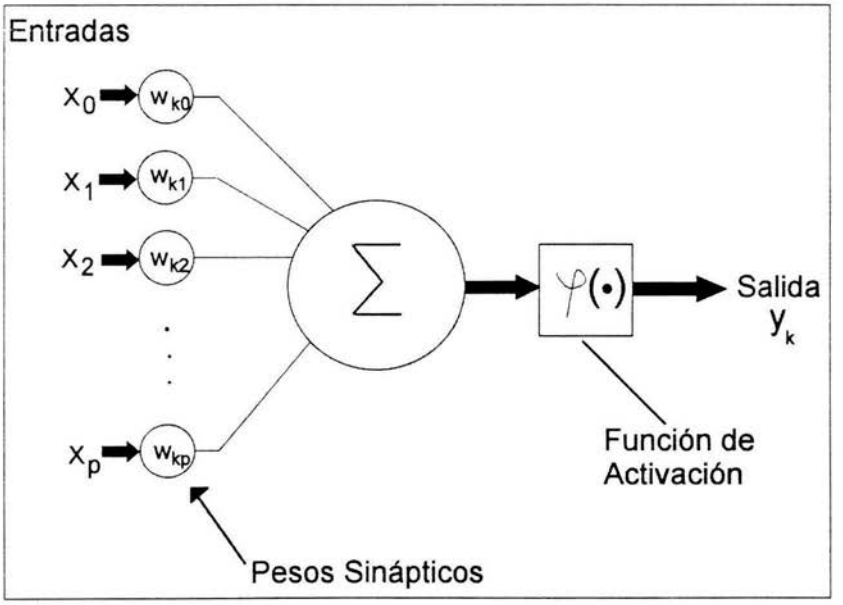


Figura. 2 Modelo una Neurona

Este modelo de la neurona [10], lo podemos describir matemáticamente como:

$$u_K = \sum_{j=1}^{P} w_{kj} x_j$$

Donde u_K es la salida de la etapa de sumado de las entadas por sus pesos respectivos.

$$y_k = \varphi(u_K)$$

2.3 Redes Neuronales Artificiales

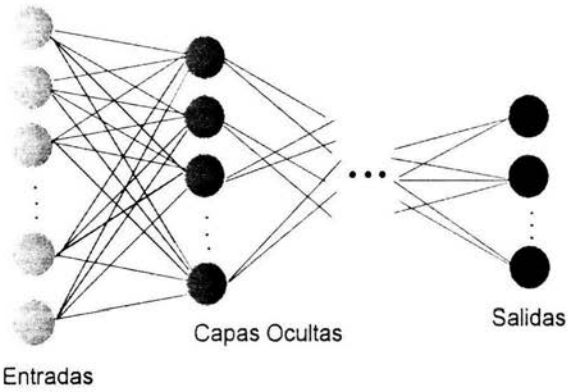
El uso de redes neuronales para la clasificación de patrones, es una herramienta ampliamente utilizada, por su capacidad de generalización.

Para hacer uso de cualquier red neuronal en la clasificación, es necesario tomar en cuenta la representación de la información y como van a ser preprocesadas la entradas de la red, para lograr una representación que sea lo mas clara posible del problema. Por lo que es necesario tomar en cuenta las siguientes reglas al momento de preprocesar la información para ser entregadas a las redes neuronales [10].

- Regla 1: Entradas similares de clases similares producen representaciones similares dentro de la red, por lo que deben pertenecer a la clasificación de la misma clase.
- Regla 2: Elementos para ser clasificados en clases diferentes, deben dársele representaciones totalmente diferentes en la red.
- Regla 3: Si una característica en particular es importante, debe de haber un número mayor de neuronas trabajando en su representación dentro de la red.
- Regla 4: Información a priori y variaciones deben de estar representadas en la diseño de la red, evitando a si la necesidad de que la red las aprenda.

El modelo de McCulloch-Pitts [11]

1. La actividad de la neurona es un proceso de todo o nada.
2. Un cierto numero de sinapsis (>1) deben ser excitados dentro de un periodo de latencia para que una neurona sea excitada.
3. El único retraso significativo dentro de la neurona es el retardo sináptico.
4. La adición de una sinapsis inhibidora absolutamente previene de la excitación de la neurona en ese momento.
5. La estructura de las interconexiones de la red no cambia con respecto al tiempo.



2.4 Retropropagación del Error

La retropropagación es la forma básica del entrenamiento de una red neuronal supervisada. Y definimos una red supervisada cuando se le indica la salida deseada por cada entrada a la red, por lo que a la red le estamos indicando las clases que deseamos que entrene.

El algoritmo de retroprogación (backpropagation), para redes de tipo feedforward, es el proceso por el cual se obtiene el vector de la gradiente en dirección opuesta al flujo de la salida en cada nodo [12], y se modifican el conjunto de pesos en la red en base a esto.

Para esto primero tenemos que obtener la expresión generada de un nodo de la red:
Donde L representa las capas donde l (l = 0, 1, ..., L) donde l = 0 representa la capa de entrada, donde i es el número de neuronas en una capa. De esta manera la representación de cada capa es $X_{l,i}$ y $f_{l,i}$ para la función del nodo, así tenemos la función:

$$X_{l,i} = f_{l,i}(x_{l-1,1}, \ldots x_{l-1,N(l-1)}, \alpha, \beta, \gamma, \ldots)$$

Donde α, β, γ, *etc.* Son parámetros de cada nodo.
El error es calculado, con la diferencia entre la salida de la red y la salida esperada

$$\{P_1, t_1\}, \{P_2, t_2\}, \ldots, \{P_Q, t_Q\}$$

donde P_q es la salida de la red y t_q es salida esperada de la red. Y como se va aplicando las entradas a la red, se va calculado la salida de la misma, teniendo así el error como la suma de los errores cuadrados.

$$ecm = \frac{1}{Q}\sum_{k=1}^{Q} e(k)^2 = \frac{1}{Q}\sum_{k=1}^{Q} (t(k) - a(k))^2$$

Este método es una modificación del método de aprendizaje de Widrow-Hoff, siendo una modificación del método de LEAST MEAN SQUARED (LMS), donde se intenta minimizar el error cuadrático medio. Y se hace realizando una modificación en los pesos en la dirección del gradiente del error. El algoritmo de retropropagación del error actualiza los pesos de la red al igual que sus bias en dirección del decremento menor del error.

$$x_k + 1 = x_k - \alpha_k g_k$$

xk = vector de pesos y bias
gk = gradiente
αk = coeficiente de aprendizaje

Existen dos diferentes tipos de modificación de los pesos, siendo uno el denominado incremental, ya que la actualización de los pesos de hace después de alimentar cada una de las entradas. La otra forma de actualización es de tipo batch donde la actualización se realiza solo una vez que todos las entradas de la red fueron analizadas, modifica el peso

usando el promedio de todas las modificaciones de cada entrada, esto reduce el tiempo de procesamiento, ya que este se lleva acabo una vez que todo el conjunto de entradas fue administrado a la red.

Algoritmo de la retropropagación [11]:

1. Aplicar un vector de entrada $Xp=(Xp1,Xp2,\ldots,XpN)^t$ a las entradas
2. Calcular los valores de la entradas de la red a la capa oculta:

$$net_{pj}^{h} = \sum_{i=1}^{N} \omega_{ji}^{h} x_{pi} + \theta_{j}^{h}$$

3. Calcular las salidas de la capa oculta:

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

4. Moverse a la capa de salidas. Calcular los valores de entrada a la red para cada unidad:

$$net_{pk}^{o} = \sum_{j=1}^{L} \omega_{kj}^{o} i_{pj} + \theta_{k}^{o}$$

5. Calcular las salidas:

$$O_{pk} = f_{k}^{o}(net_{pk}^{o})$$

6. Calcular el error para las unidades de salida:

$$\delta_{pk}^{o} = (y_{pk} - O_{pk}) f_{k}^{o\prime}(net_{pk}^{o})$$

7. Calcular el error para las unidades ocultas:

$$\delta_{pj}^{h} = f_{j}^{h\prime}(net_{pj}^{h}) \sum_{k} \delta_{pk}^{o} \omega_{kj}^{o}$$

   Los errores deben ser calculados antes de que se pueda hacer cualquier modificación en los pesos.

8. Actualizar pesos en la capa de salidas:

$$\omega_{kj}^{o}(t+1) = \omega_{kj}^{o}(t) + \eta\delta_{pj}^{h} i_{pj}$$

9. Actualizar pesos en la capa oculta:

$$\omega_{kj}^{o}(t+1) = \omega_{kj}^{o}(t) + \eta\delta_{pj}^{h} x_{i}$$

Calcular el error:

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

Donde podemos calcular la modificación del peso por medio de la siguiente ecuación [10]:

$$\begin{pmatrix} Correción \\ Pesos \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} Coeficiente \\ Aprendizaje \\ \eta \end{pmatrix} \bullet \begin{pmatrix} Gradiente \\ \delta_{j}(n) \end{pmatrix} \bullet \begin{pmatrix} Señal \\ de \\ Entrada \\ Neurona_{j} \\ y_{i}(n) \end{pmatrix}$$

El coeficiente de aprendizaje en el algoritmo de retropropagación del error, provee el paso de desplazamiento para modificar los pesos sinápticos, entre más pequeño que sea este parámetro, menor será la modificación de los pesos, en cada iteración y mas suave será la trayectoria en el espacio de los pesos.

Capítulo 3.

Desarrollo:

3.1 Análisis del problema:

Decidir la mejor arquitectura para una red neuronal, no es una tarea trivial, ni que se pueda tomar a la ligera, ya que si solo se considera como parámetro en el diseño, el número de entradas y el número de salidas, sin tomar en cuenta otros factores, como el número de capas internas y las funciones de activación de cada capa, se obtiene una red, que aunque sea funcional no otorga el mejor desempeño posible.

Por lo que es una etapa común en la implementación de una red neuronal, es la inversión de tiempo en el diseño de una arquitectura para la misma. Siendo necesario generar muchos tipos de arquitecturas y realizar pruebas para elegir finalmente la mejor. Por lo que el uso de algún tipo de evolución en la generación de diversas configuraciones, ya es un tema tratado en la literatura aunque escasamente [13].

Por otra parte también se planteo el problema, de cómo seleccionar las entradas a una red, ya que no siempre que se pretende generalizar un problema por medio de una red, se conoce que entradas o parámetros del problema son los más indicados para lograr el mejor entrenamiento de la red. Este tipo de circunstancias suelen suceder cuando se intenta hacer uso de variables económicas y trabajo de modelos financieros. Ya que la forma mas común para eliminar estas variables, que no corresponden al modelo o que usarlas en este puede causar malos resultados, es por medio de un análisis de componente principal. Donde distinguimos la importancia de cada variable y de decide si se prosigue con su eliminación. Pero la aproximación que se lleva a cabo en este trabajo evita que este tipo de análisis sea requerido. Ya que el algoritmo genético realiza ajustes sobre las entradas usadas, de la misma manera que realiza ajustes sobre la arquitectura de la red. Obteniendo así un arreglo de redes cuyas entradas y arquitectura de la red, por si mismas dan el menor error a la salida de la red.

3.2 Programación:

Podríamos definir un algoritmo general para el uso de evolución en la generación de arquitecturas de redes neuronales es el siguiente [13].

a) Codificar cada individuo de una generación en una arquitectura
b) Entrenar cada red con la arquitectura codificada para ese individuo, según lo parámetros codificados.
c) Calcular la aptitud de cada individuo, esto puede ser por ejemplo: el error cuadrático medio, la capacidad de generalización, el tiempo de entrenamiento, la complejidad de la arquitectura, etc.
d) Reproducir un número de individuos de la generación actual a la siguiente basada en su aptitud o evaluación.
e) Aplicación de operadores genéticos, como mutación, cruza, suma, inversión, etc.

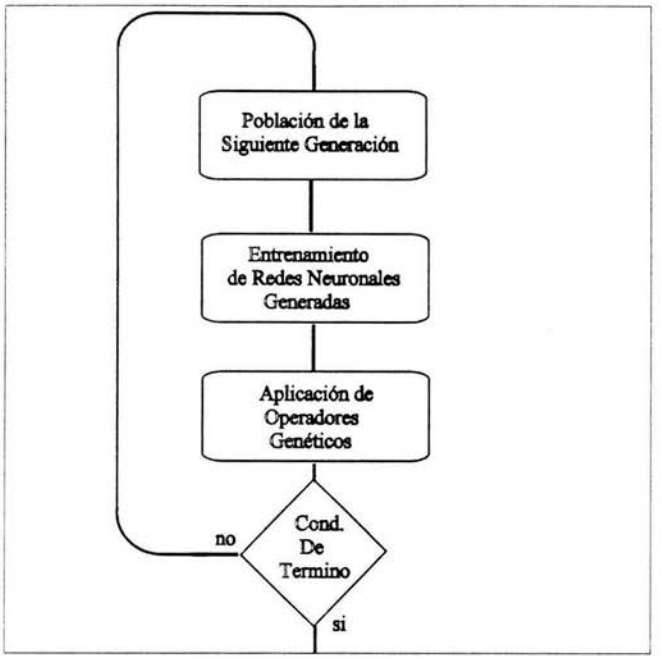


fig. 4 Ciclo del Algoritmo

Por esto, lo primero que se tiene que hacer es decidir la forma de codificar el problema. Para este trabajo se decidieron que los parámetros a optimizar, en el caso básico, en donde solo se realiza el proceso evolutivo sobre la arquitectura de la red, fueran los siguientes:

- Número de entradas
- Número de Capas ocultas
- Elementos de cada capa
- Función de activación de las neuronas de cada capa

Al momento de declarar el algoritmo genético, es necesario indicarle al programa la cantidad máxima de capas ocultas y la cantidad máxima de elementos en las capas, ya que de no suministrar estos parámetros, se generarían redes de complejidades mayores a las deseadas en el proceso de creación de la población inicial. Para este fin, se hizo uso de un

archivo de configuración (config.ini[1]), donde se modifican los parámetros[2] tanto de las redes neuronales, como:

- El número de capas ocultas
- Número máximo de neuronas por capa
- Número máximo de funciones de activación usadas (default: 10)[3]

Y del parámetro del algoritmo genético como:

- Número de generaciones
- Número de individuos (tamaño de la población)
- Porcentajes de aplicación de los operadores genéticos.

Por lo que una población inicial se ve de la siguiente manera:

[1] Apéndice A
[2] Más sobre estos parámetros de configuración de encuentran en el Apéndice B.
[3] Apéndice C

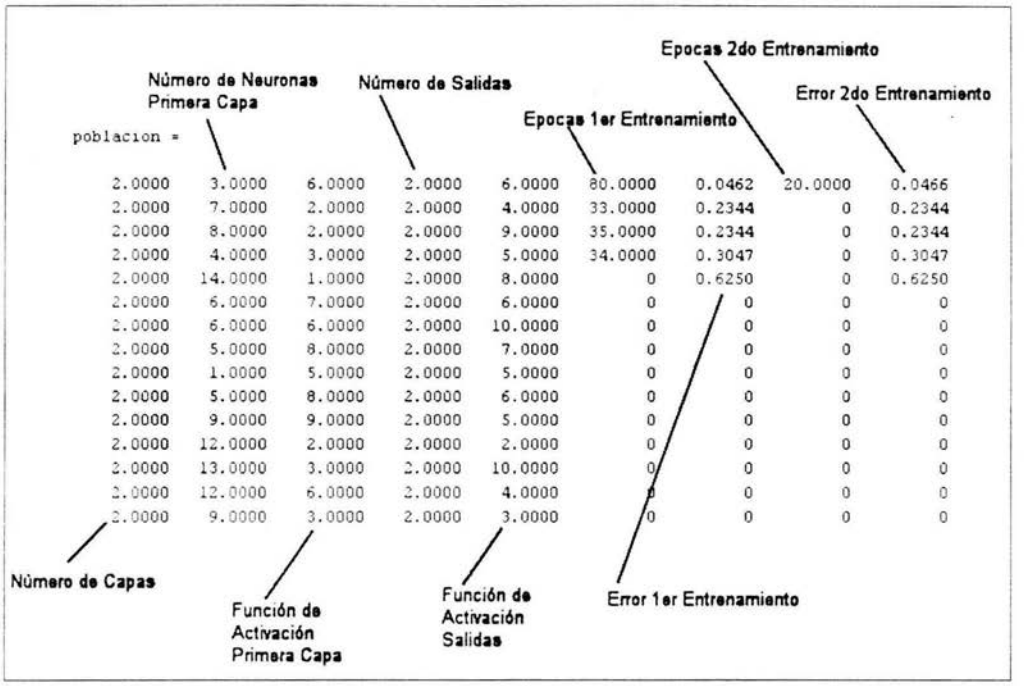


fig. 5 Matriz de población

Esta tabla, puede tener diversas dimensiones pero siempre siguiendo esta regla de generación:

(capas,Entradas,fcapa1,#capa1,fcapa2,#capa2,...,fcapanN,#capaN,fit1,fi2)

El número de salidas no se indica en el vector, ya que la última capa de nuestra red se toma como las salidas, lo que en la figura 5, se puede ver como se mantiene fijo el número de salidas, y no es afectado por ningún operador genético.

Selección del tamaño de la población

La selección del tamaño de la población se hace, por medio de un análisis de los esquemas posibles que puede tomar un individuo, ya que es necesario tener un tamaño adecuado de la población para poder cubrir el espacio de búsqueda de una manera adecuada.

Las pruebas realizadas con diversos tamaños de poblaciones, mostraron la necesidad de poblaciones mayores o iguales a 80 individuos. Asumiendo que la distribución de la población sea homogénea, dado las características del generador de números aleatorios de Matlab.

Generador de números aleatorios de Matlab

El generado de números aleatorios de Matlab hace uso de un generador de Fibonacci, combinado con un registro de corrimiento, dando la posibilidad de generar 2^{1492} números, teniendo un ciclo de repetición de números muy grande, el único problema es que la semilla que usa sigue siendo fija, por cada vez que se ejecuta el Matlab [14]. Por lo que es necesario generar una nueva semilla a partir del reloj interno de la computadora. Esto se hace con el comando:

```
rand('seed',sum(100*clock))
rand('state',0)
```

Esta semilla es guardada, dentro de las variables de entorno y así conocer la semilla inicial de cada uno de los procesos evolutivos, pudiendo así recrear el proceso al volver a poner todos los parámetros en su estado original, e inicializar el contador con la misma semilla.

Esto es muy útil cuando es necesario repetir resultados de alguna simulación, así como saber que no se esta generando poblaciones iguales, cuando no se requieran así.

Un problema asociado a una mala distribución de los individuos de la población, causaría que el operador crossover y la Mutación no serían capaz de generar individuos en otras zonas del espacio de búsqueda, dado que todos estos se encontrarían muy cerca uno del otro.

Selección del conjunto de entrenamiento de las Redes Neuronales.

La primera parte al momento de plantear, el conjunto de entradas que tendrá la red, es contemplar que estas tienen que ser menores al conjunto total de posibles entradas del problema, ya que de darle a la red todas las entradas posibles o un números excesivos de las mismas, no se estaría aprovechando la capacidad de generalización de la red. Este es el caso de vectores de entrada no conocidos por la red, y que se encuentren entre dos clases ya entrenadas previamente en la red. Se busca una salida asociada que represente una interpolación entre estas dos clases. Por lo que la red nos puede entregar valores de salida sobre datos no entrenados o conocidos por la red.

Generalmente se utiliza un subconjunto de las posibles entradas del problema, reservando así otro subconjunto de este espacio, para realizar pruebas de la red, con valores no entregados a esta durante su etapa de entrenamiento.

Selección Evolutiva de las variables de entrada a la Red

En problemas multivariable, el proceso de discernir cuales variables son las indicadas para ser usadas en el modelo, no es una tarea fácil, ya que no siempre es necesario usar todas las variables de las cuales se disponen, y utilizarlas implicaría un modelo mas grande y complejo, ya que podría darse el caso que se usen variables que pudieran parecer muy útiles en el análisis inicial, pero al momento de entrenar las redes neuronales causen confusión y perjudiquen el entrenamiento y la generación de las clases dentro de la red.

Para este caso el programa, cuando se usa la modificación "gia_on" (parámetro que se explicará mas adelante en la descripción de la estructura del código) el proceso evolutivo no solo se lleva acabo entorno a la arquitectura de las redes neuronales sino también modifica evolutivamente la mascara de variables de entrada de cada red. Esto modifica ligeramente la codificación de cada individuo que se explico anteriormente, ya que en el primer elemento del vector encontramos, una representación en decimal de una mascara binaria, asociada a las entradas de la red. Donde cada uno de los elementos del número binario, indica en el caso de ser "1" o "0", si la respectiva variable de entrada será usada o no, en la arquitectura de la red representada por el resto del renglón dentro de la matriz poblacional.

Por ejemplo si tenemos, un sistema con 6 entradas, las mascaras serán un número de 6 bits: "110101", donde las variables de entrada número "1,2,4,6" serán las usadas al momento de entrenar la red. De esta manera cada individuo tiene un conjunto de variables que usará como entradas a la arquitectura de la red que codifique.

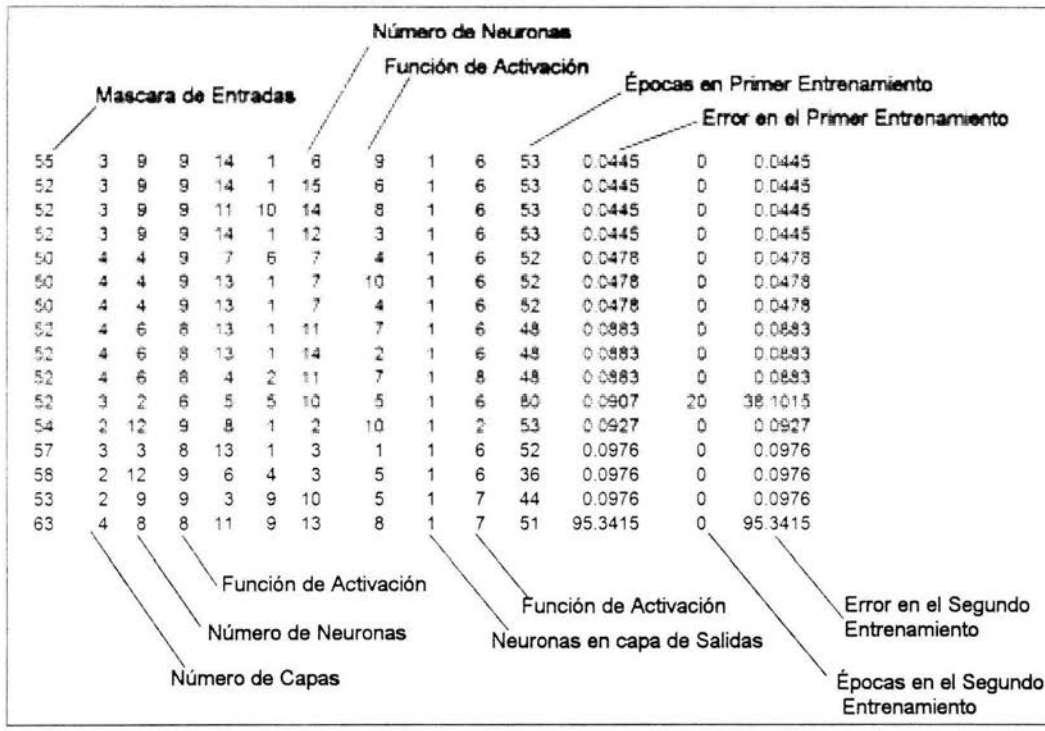


fig. 6 Matriz Poblacional con mascaras de variables de entrada

El análisis tradicional para descartar las variables en base a si afectan o no un modelo, es generando un análisis de componente principal, sobre este modelo, en cada una de las variables. Y de esta forma buscar eliminarlas cuando estas no aporten una parte significativamente representativa al modelo. Pero en esta aproximación la decisión de la eliminación de la variable es en función del desempeño en el entrenamiento, ya que se genera un algoritmo genético en torno la máscara de entradas (archivo de configuración 'config.ini', variable 'siempre_mask'). La forma en que se realiza el manejo de los operadores genéticos entorno a la máscara de entradas es de la misma manera, que sobre

todo el vector del individuo.

Solo se diferencía debido a que el manejo de esta variable es con código binario, que representa a la máscara, y una vez aplicados los operadores genéticos el almacenaje se lleva acabo en su equivalente en decimal, y se anexa en la matriz poblacional.

Por otra parte el proceso de evolución también puede causar la eliminación de una entrada, debido al elemento heurístico intrínseco en los algoritmos genéticos. Pero este elemento heurístico en la eliminación de una variable se puede observar, al momento de comparar los mejores individuos de cada población, o realizando varias corridas del proceso del evolutivo y comparando las salidas arrojadas.

Entrenamiento de las redes neuronales:

Una vez que el algoritmo crea la población y la empieza a evaluar renglón por renglón, generando una red neuronal, en base a los parámetros de individuo, se procede a entrenar la red neuronal obtenida por la representación del individuo, el entrenamiento se realiza en dos partes, alimentando a la red las variables de entrada seleccionadas al momento de modificar la mascara de entradas, la primera parte del entrenamiento tiene un número de épocas diferente a las utilizadas en la segunda parte del entrenamiento. Y la cantidad de épocas que cada sección del entrenamiento es modificable a través del archivo de configuración, ya que la cantidad de épocas que le tome a una red entrenarse depende del tipo de problema que se quiera abordar. En el archivo de configuración (config.ini) se modifican las variables 'epochs' y 'epochs_dec'. Por lo que la cantidad de épocas de cada entrenamiento se calculan de la siguiente manera:

Épocas del primer entrenamiento:

$$Ep_1 = epochs - epochs \times epochs\_dec$$

Épocas del segundo entrenamiento:

$$Ep_2 = epochs \times epochs\_dec$$

Con esto logramos conocer, el comportamiento del entrenamiento, en dos momentos diferente en la red, y así poder conocer si la red todavía cuenta capacidad de aprendizaje [15]. Esto nos ayuda a descartar redes, que al parecer tengan un muy buen aprendizaje, y aun cuando hayan llegado al error deseado, ya no tengan la misma capacidad de aprendizaje, debido a características intrínsecas de su arquitectura. En la figura 7 se logra ver como el error de la red, desminuye a lo largo del entrenamiento y como este llega a ser menor al deseado, pero con una simple inspección ocular en la gráfica, podemos concluir que esta red ya no cuenta con capacidad de aprendizaje, y que el desempeño pobre al momento del aprendizaje debe perjudicar en su cálculo de la aptitud asociada a este individuo. Esto nos es extremadamente útil al momento de realizar el proceso evolutivo en sus primeras etapas, ya que podemos saber si la red todavía tiene posibilidades de aprender, aun cuando el error de la misma todavía no es aceptable. Este parámetro del doble entrenamiento, también nos permite saber si en la red se ha generado un sobre entrenamiento, representándose este problema en un aumento del error en la segunda etapa del entrenamiento en lugar de una disminución del mismo. Esta falla es muy común que se de en etapas mas avanzadas del proceso evolutivo, cuando el error puede crecer mucho debido a un sobre entrenamiento.

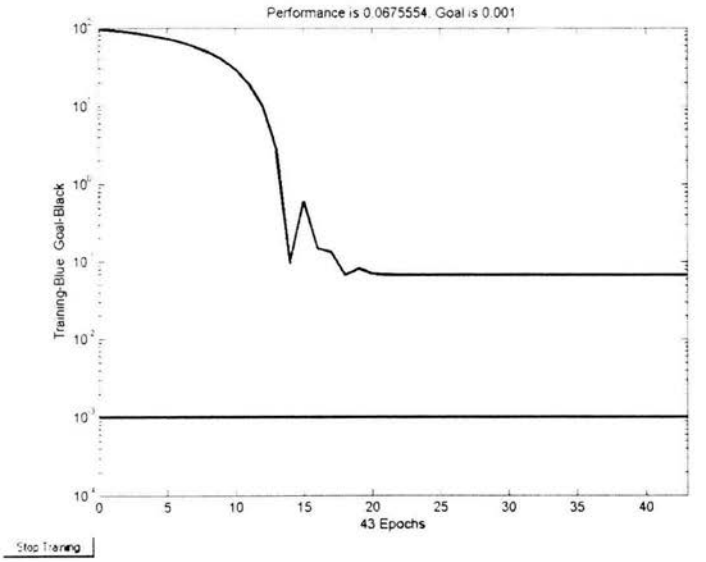


fig. 7 Curva de Aprendizaje de una Red

El entrenamiento de las redes se hace por medio del entrenamiento por lotes (*batch traning*), donde la actualización de los pesos se realiza una vez que todas la entradas de una época son alimentadas. Dónde el error es calculado de la siguiente manera:

$$\varepsilon_{av} = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in C} e_j^2(n)$$

Y la actualización de los pesos es por medio de la regla delta:

$$\Delta W_{ji} = -\eta \frac{\delta \varepsilon_{av}}{\delta w_{ji}}$$

$$= -\frac{\eta}{N} \sum_{n=1}^{N} e_j(n) \frac{\delta \varepsilon_j(n)}{\delta w_{ji}}$$

Descripción de la estructura del código:

El programa fue implementado bajo el ambiente Matlab [16], y las corridas se realizaron en una computadora Pentium a 766Mhz con 256Mb RAM.

El uso de programa, consta de una línea principal de ejecución la cual, es mas sencilla que la línea general el usar las redes tipo feedforward de matlab, y solo cuenta con un parámetro aparte de las entradas y salidas que son totalmente necesarias en una red neuronal con un entrenamiento supervisado, este parámetro nos define el tipo de evolución, la primera opción de este parámetro es "gia_off", indicando que solo se lleve a cabo el proceso evolutivo en la codificación de la arquitectura de la red:

[poblacion,ganadores,best_array]=gann(input,output,'gia_off');

Mientras que el segundo valor que puede tomar este parámetro es "gia_on", indicando que se realice un proceso evolutivo sobre la arquitectura de la red, pero de igual manera sobre las variables de entrada usadas para el entrenamiento de la red misma:

[poblacion,ganadores,best_array]=gann(input,output,'gia_on');

Las entradas tanto como las salidas, en cualquiera de las dos opciones de evolución son parámetros forzosos y deben tener el siguiente formato: (N x inputs) y (N x outputs), donde inputs y output, son el número de variables que tendrá el problema, mientras que N es el número de elementos que se usara para el entrenamiento, N debe de ser un subconjunto de las posibles entradas del problema, que se intenta que generalice o aprenda la red.
Los parámetros entregados por el algoritmo al momento de terminar la ejecución son: *[poblacion, ganadores, best_array, seed]*, donde:

- *Población* es el estado de la población en su última generación.
- *Ganadores* es un arreglo con los mejores elementos de la última población, siendo esta una submatriz de la matriz *Población.*
- *Best_array* es un arreglo cronológico del mejor individuo de cada

generación, donde el primer renglón es el mejor individuo de la primera generación, y así sucesivamente, por lo que la cantidad de elementos de este arreglo nos indica el numero de generaciones que se ejecuto el algoritmo genético.

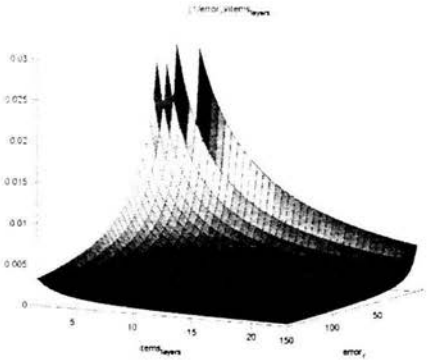
- *Seed:* La semilla del generador de números aleatorios de Matlab.

A la salida del proceso evolutivo se devuelve el arreglo "Best_array", el cual nos permite ver como se ha ido modificando el mejor individuo de la población y como su error ha ido disminuyendo generación tras generación, después de verse afectada la población por los operadores del algoritmo evolutivo.

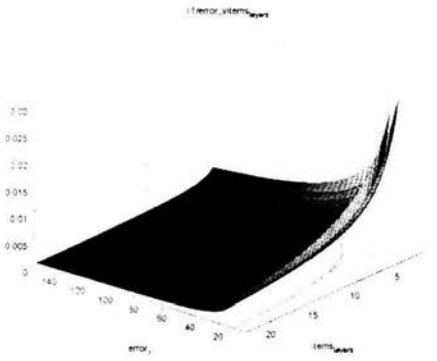
Función Objetivo:

La función objetivo, muchas veces se ha denominado como el elemento principal, dentro del concepto de la computación evolutiva, en el cual se define quienes son los individuos más aptos dentro de la población. Dando así la pauta para la aplicación de los operadores evolutivos.

Durante la búsqueda de una función objetivo correcta, se plantearon varias posibles aproximaciones de funciones objetivo a usarse, para el análisis de las mismas se generaron las superficies descritas por estas funciones, encontrando gradientes muy altas en una u otra dimensión de las funciones objetivo. Un ejemplo son las siguientes gráficas, donde se puede apreciar el gradiente de las superficies (ver gráficas de gradiente excesiva 1,2):



Gradiente Excesiva 1



Gradiente Excesiva 2

Este tipo de funciones objetivo no se consideraron como viables ya que la aptitud de los individuos al ser evaluados por esta función cambian de manera muy drástica, aun cuando los puntos sobre la superficie, que representan individuos de la población se encuentren muy cerca uno del otro.

Tras varias pruebas se optó finalmente por el uso de un hiperparaboloide de 4 y 5 dimensiones, este cambio de las dimensiones depende, si se esta haciendo uso de la evolución de la máscara de variables de entrada o no. La forma de la superficie (ver figuras gradiente menor 1,2) ayudo a que cualquier modificación causada por los operadores no represente variaciones muy fuertes en la aptitud de los individuos.
Siendo la función objetivo para el caso general la siguiente:

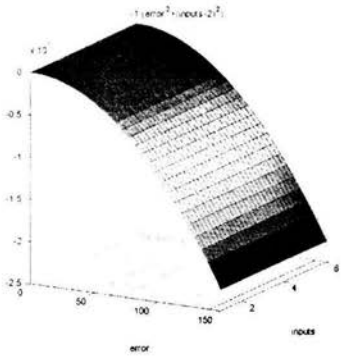
$$fitness = -1\left(40 \times error\_r^2 + \left(\frac{\max\_i + items\_layers}{\max\_i}\right) + \left(\frac{\max\_e + epocas}{\max\_e}\right)^2\right)$$

Y su equivalente al hacer evolucionar la máscara de variables de entrada:

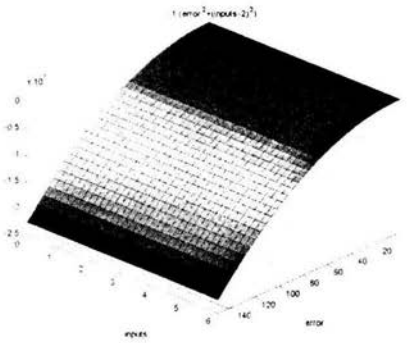
$$fitness = -1\left(3000 \times error\_r^2 + \left(\frac{\max\_i + items\_layers}{\max\_i}\right) + \left(\frac{\max\_e + epocas}{\max\_e}\right)^2 + \left(\frac{\max\_inputs - inputs}{\max\_inputs}\right)^2\right)$$

Donde:

- Error_r: Es el error a la salida de la red
- Max_i: Número máximo de capas definido en la configuración.
- Ítems_layer: Número de capas del individuo
- Max_e: Número de épocas máximo definido en la configuración
- Epocas: Cantidad de épocas de entrenamiento del individuo
- Max_inpus: Cantidad máxima de variables usadas como entradas.



Gradiente Menor 1



Gradiente Menor 2

La gráfica de la superficie generada solo representa, la función objetivo en tres dimensiones, teniendo en los ejes 'X' y 'Y', al error y número de entradas, y en el eje 'Z', la aptitud.

La modificación de la concavidad en alguna de las dimensiones de la ecuación, es necesaria para lograr algún interés en particular en algún aspecto de la arquitectura de la red, ya sea en el error, o en las dimensiones de la misma, ya que de no contar con una función objetivo que especifique los parámetros de interés a optimizar, o que no se encuentre bien ajustada, podría causar que se optimizara demasiado la complejidad de la red, y no se logre mejorar el desempeño de la misma en términos de reducir el error. O alguna relación poco deseada, entre la arquitectura y el desempeño de la red.

Por lo que obtenemos el término general de funciones objetivo:

$$fitness = (ke \times error)^2 + \left(ki \times \frac{\max\_items + item\_layers}{\max\_items} \right)^2 + \left(kep \times \frac{\max\_epochs + epochs}{\max\_epochs} \right)^2 + \left(kin \times \frac{\max\_inputs + inputs}{\max\_inputs} \right)^2$$

Donde la magnitudes de las constantes de cada uno de los término depende de que tan critico se considere el parámetro asociado a la constante, en general la constante que afecta al error (ke), debe ser mas grande que las otras constantes, ya que es el error el aspecto de diseño que mas nos podría importar en una red neuronal funcional.

Control de la población:

El control del tamaño de la población es un proceso que siempre se tiene que llevar a cabo, pero que rara vez es descrito en la bibliografía como una parte importante, aun cuando juega un papel crítico al momento de medir los recursos computacionales y el desempeño del algoritmo, por lo que nos interesa que la población no se encuentre creciendo de manera descontrolada al momento de aplicar el operador de crossover, por lo que es necesario vigilar, que no se encuentre la población en un decremento fuerte al momento de aplicar el operador de selección. Lo que se busca en el control de la población es que se encuentre dentro de un rango aceptable, alrededor del tamaño de la población inicial, y de ser el caso de generarse incremento de la misma, el programa sea capaz de detectarlo y autorregular el tamaño de la población, manteniéndola en una dinámica oscilatoria.

El control de la población juega un papel muy importante, ya que la dinámica del tamaño modifica el paralelismo intrínseco del algoritmo, ya que definimos a cada individuo de la población como un punto sobre el hiperplano de la función objetivo, dicho individuo se encuentra buscando su camino hacia el máximo o el mínimo de la función, por lo que una cantidad de individuos reducida, nos genera una cantidad igualmente pequeña de puntos sobre este hiperplano, de manera que encontrar el optimo puede tomar mas tiempo o puede ser que nunca se lleve a cabo si la dispersión de los mismos no fue la adecuada. Por otro lado si tenemos un exceso de individuos causados por una sobrepoblación, esto podría parecer muy bueno al momento de pensar en un montón de puntos dispuestos en un hiperplano, pero nos afecta directamente el desempeño del algoritmo, ya que el tiempo de procesamiento crece. Dado a que el tiempo de procesamiento tiene la siguiente función:

$$T_p = Tm_p * TUE * N_G + (To * N_G * Tm_p)$$

Donde:

- T_p :Tiempo de procesamiento
- Tm_p: Tamaño de la población
- TUE: Tiempo unitario de entenamiento
- N_G : Número de generaciones
- To: Tiempo de aplicación de operadores

3.3 Pruebas:

3.3.1 Descripción del problema:

Los sistemas de computo suave, cada vez toman mas fuerza en las áreas de negocios, ya que suelen ser una forma de disminuir los costos de operación dentro de las empresas, estos sistemas aprenden las relaciones entre ciertos atributos o características que son tomados como variables independientes y alguna salida representada por la variable dependiente [17]. Sistemas inteligentes como las redes neuronales y algoritmos genéticos, son usados por su habilidad para procesar modelos de negocios o financieros a partir de datos históricos, donde la relación precisa entre las variables se desconoce.

Para realizar la prueba del proceso evolutivo completo, tanto de la máscara de variables de entrada como de la arquitectura de la red, se utilizó un modelo financiero de predicción del Índice de Precios y Cotizaciones de la bolsa Mexicana de Valores (IPC-BMV), para el análisis de series de tiempo [15, 18, 19].

Sabemos que el flujo de información en el mundo se realiza muy rapidamente causando que una variación en alguna economía mundial, pudiera afectar a otra economía, al lado opuesto del mundo en segundos, por lo que es necesario contemplar diferentes variables al momento de intentar aproximar un modelo econométrico de este tipo.

La información para las entradas a la red de comportamiento bursátil, fueron obtenidas del Banco de México [20] y del histórico de DowJones [21].

Se creo una tabla de datos, con 190 días de información Bursátil con de diferentes variables para el entrenamiento y proceso evolutivo, y una sección de 48 días para las pruebas de la red. mismas variables que se muestran a continuación:

1	Tipo de cambio pesos por dólar E.U.A. Interbancario a 48 horas
2	Índice de precios y cotizaciones Cierre (BMV)
3	Volumen operado (BMV)
4	TIIE a 28 días
5	Tasa de fondeo bancario, Promedio ponderado
6	DowJones
7	Cantidad de CETES a 28 días
8	Interés de CETES a 28 días
9	Variable aleatoria generada por Matlab
10	Tasa de fondeo gubernamental promedio ponderado

Tabla 1. Variables usadas para el entrenamiento

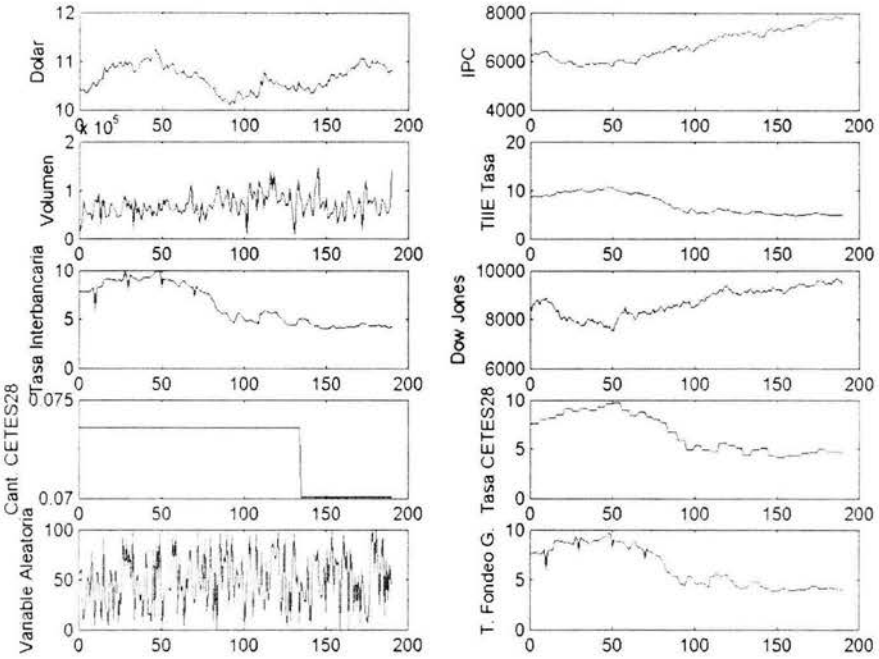


fig. 8 Gráficas de Variables Utilizadas

A la tabla de datos, se le realizó un ajuste, ya que los días de operación de la bolsa Mexicana y la Bolsa DowJones, no son siempre los mismos, debido a las diferencias de los días feriados en cada país. Para estos casos se tuvo que repetir, el valor de los índices e indicadores del día anterior de operación y así mantener una continuidad de los mismos. De igual forma las entradas fueron previamente normalizadas, para evitar una saturación en las funciones de activación de la red.

Como primera etapa para diseñar una red consta en tener el conjunto de variables de entrada y salidas, que se pretende que esta aprenda. Aunque en el caso de modelos econométricos, es difícil el planteamiento de como o cuales deben de ser las variables de entrada, y como deben de acomodarse en la ecuación del modelo econométrico. Se busco que esta tarea fuera sustituida haciendo uso del algoritmo planteado en la tesis, ya que el modelo econométrico, se genera al entrenar la red neuronal y al escoger su arquitectura por

medio de la evolución, y la selección de las entradas o variables del modelo, es modificado por el algoritmo genético, al momento de generar variaciones en la máscara de entradas que usara cada red neuronal.

3.4 Resultados

Debido a que las pruebas se pueden realizar en muchos diferentes aspectos de la evolución, los resultados se presentan en cuatro secciones:

3.4.1 Capacidad de Selección de variables

El algoritmo por su naturaleza, tiene la capacidad de seleccionar las variables por su aportación al problema, para realizar este análisis se introdujo una variable aleatoria generada en Matlab, dentro de la base de datos de variables financieras y bursátiles utilizadas en el entrenamiento y evolución del algoritmo.

A continuación se muestran dos pruebas con parámetros diferentes, donde se pueden ver de manera clara la distinción de las variables de poca significancia en el problema.

Cabe resaltar que se pueden encontrar redes donde se encuentren como entrada variables de poca importancia y aun así tengan un buen rendimiento, pero cabría realizar un análisis posterior, sobre los valores de los pesos de las conexiones donde son recibidas estas variables de poca importancia o aportación baja al modelo y analizar su conexión o desconexión en la red neuronal.

Se corrió el algoritmo genético con redes neuronales, haciendo uso de las variables descritas en la tabla 1, para que este encontrara y distinguiera la variable aleatoria dispuesta de manera intencional, y la aportación de las demás variables.

Las gráficas y el análisis de la utilización de las variables de entrada, se realizo usando la máscara de variables de entrada a cada una de las redes escogidas, y verificando la cantidad de veces que cada una de las variables de la máscara fue utilizada, las redes tomadas para este conteo fueron los individuos con la mejor aptitud en cada generación. Lo que se podría expresar como la suma vertical de un arreglo como el siguiente, por cada ventana de

análisis, siendo el bit de la izquierda la variable uno, y así consecutivamente.

1100000000
1101100001
1100000000
1111001010
1111001110
1101011110

Primera Prueba:

Características de la prueba:

- Población: 100
- Generaciones: 100
- Tiempo de Ejecución: 9 horas 7 minutos
- Máscara: 1100000000

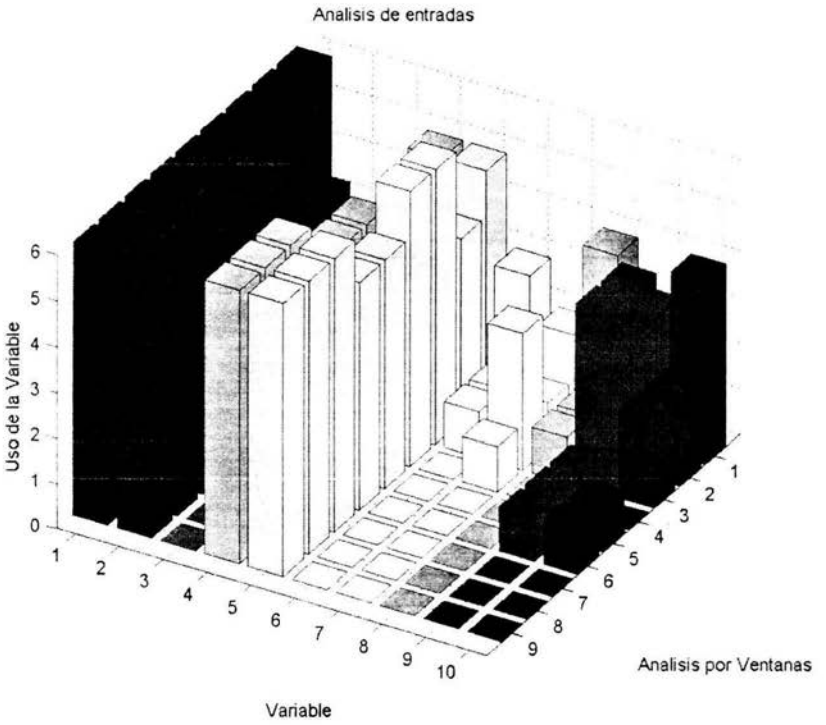


fig. 9 Análisis de Entradas

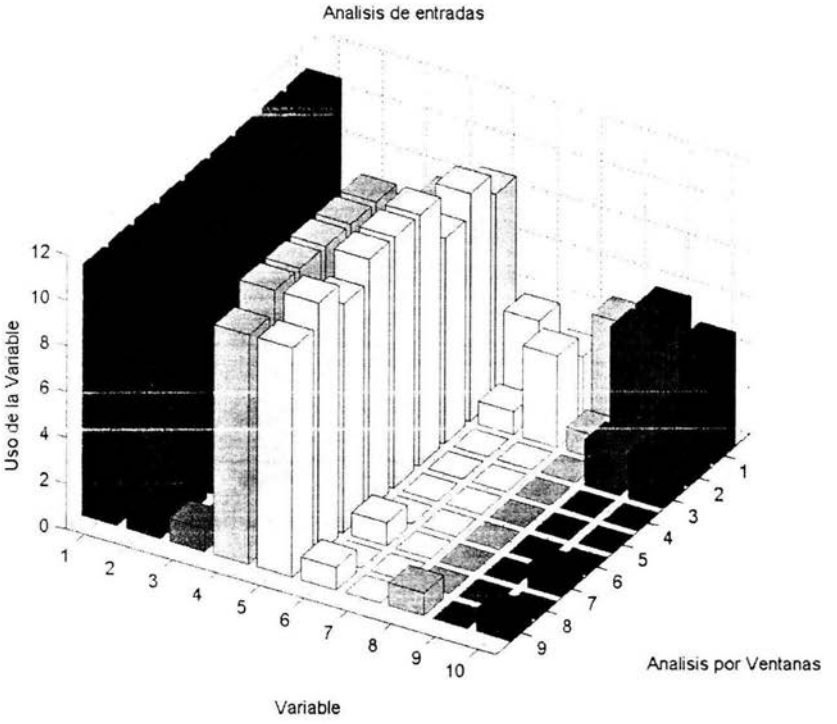
Las entradas con una magnitud de 12 en el eje del 'uso de variable' en el histograma, se debe a que son variables obligatorias, delimitadas en "siempre_mask".

Por lo que podemos ver en el histograma, que la variable novena es usada menor número de veces, siendo esta la variable Aleatoria, dispuesta en la base de datos de manera intencional.

Segunda Prueba:

Características de la prueba:

- Población: 80
- Generaciones: 60
- Tiempo de Ejecución: 7horas 24 minutos
- Máscara: 1000000000



Aquí también se muestran resultados similares en una segunda prueba, donde también se puede ver de manera clara, que la variable desechada por el algoritmo es la novena, siendo esta la variable aleatoria. Pudiendo así analizar la significancia de cada una de las variables entorno a un problema, por la rapidez en que se desecha cada variable durante el proceso del algoritmo genético.

Obteniendo como variables más significativas, las siguientes:

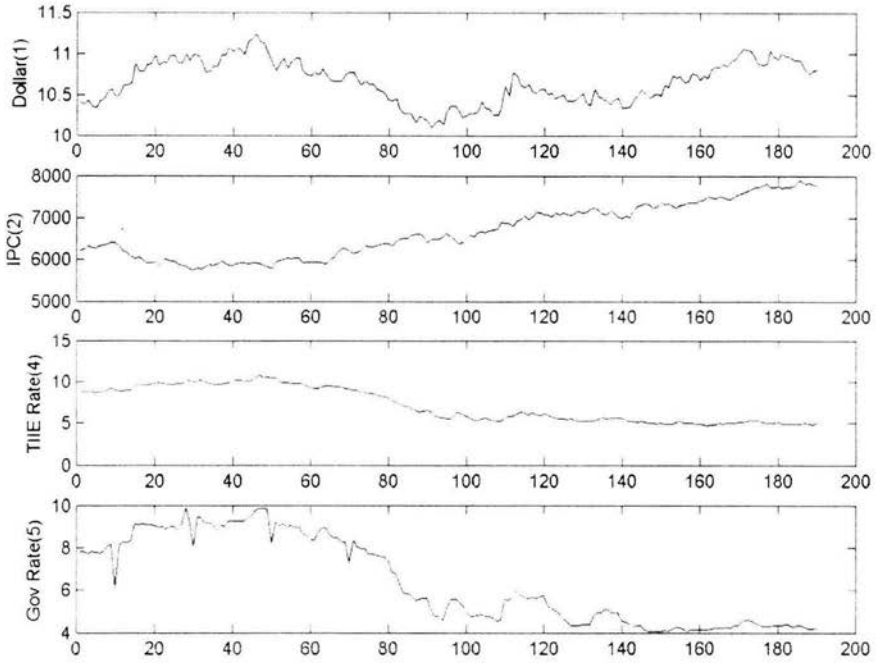


fig. 10 Variables de mayor significancia

Este proceso de evolución y adaptación de la máscara de entradas, se lleva acabo ya una vez que la configuración de la red neuronal, ya ha encontrado una forma adecuada. A partir de este momento, la configuración de la red neuronal permanece dentro de una estructura relativamente fija, mientras que la evolución se torna alrededor de la mascara de las entradas, esto se puede ver en la figura 11.

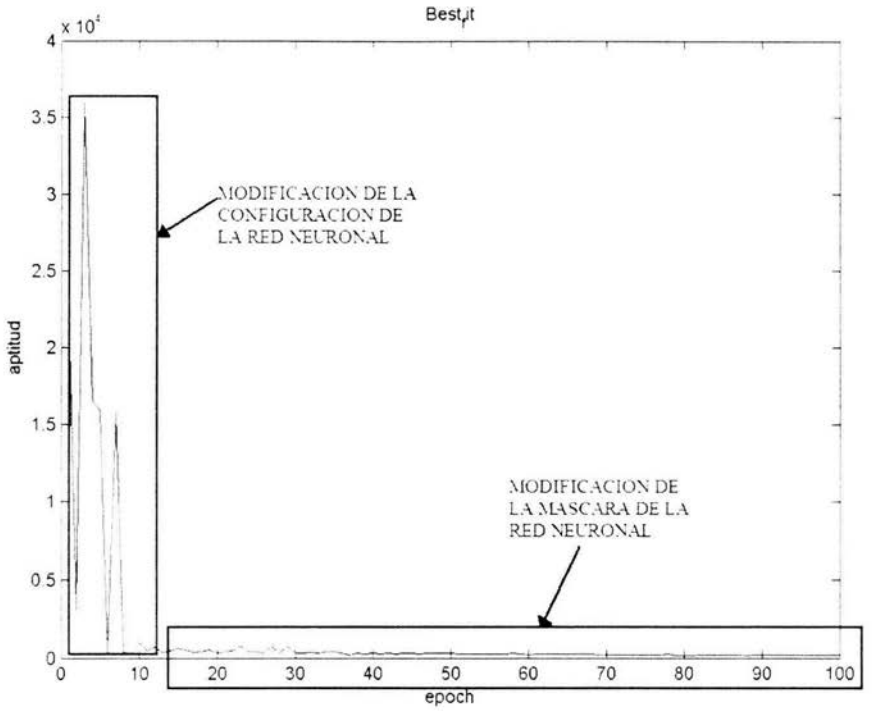


fig. 11 Evolución y Adaptación de las mascaras

3.4.2 Análisis de decremento del Error:

Otro aspecto en el análisis de los resultados que entrega el algoritmo, es el decremento del error de salida de las redes neuronales, este se ve afectado y decrementa generación tras generación, disminuyendo desde un valor cercano a 0.0976, llegando a mínimos de 0.0180, teniendo así un decremento del error de un 81.5574 %. Lo que nos aumenta realmente el desempeño de la efectividad de la red, al momento de modificar su arquitectura.

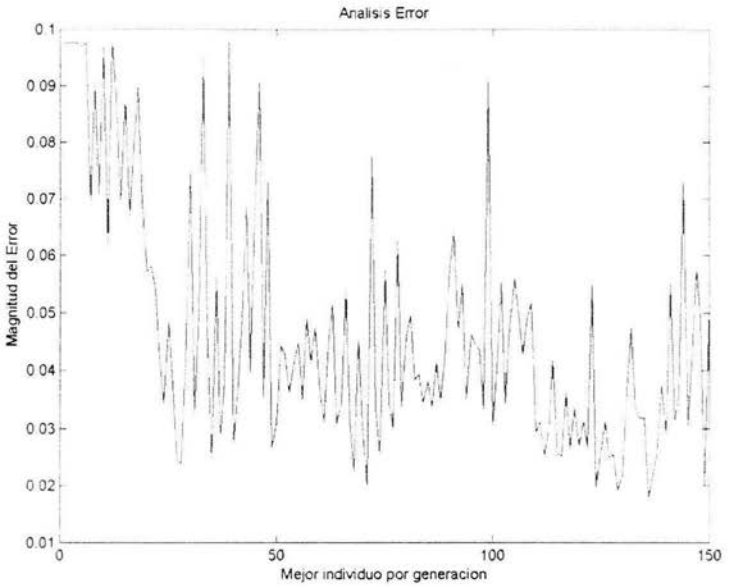


fig. 12 Análisis de Error de Salida de cada red.

En otras pruebas realizadas se encontraron decrementos del error de la red, en un rango entre el 79 y 82%:

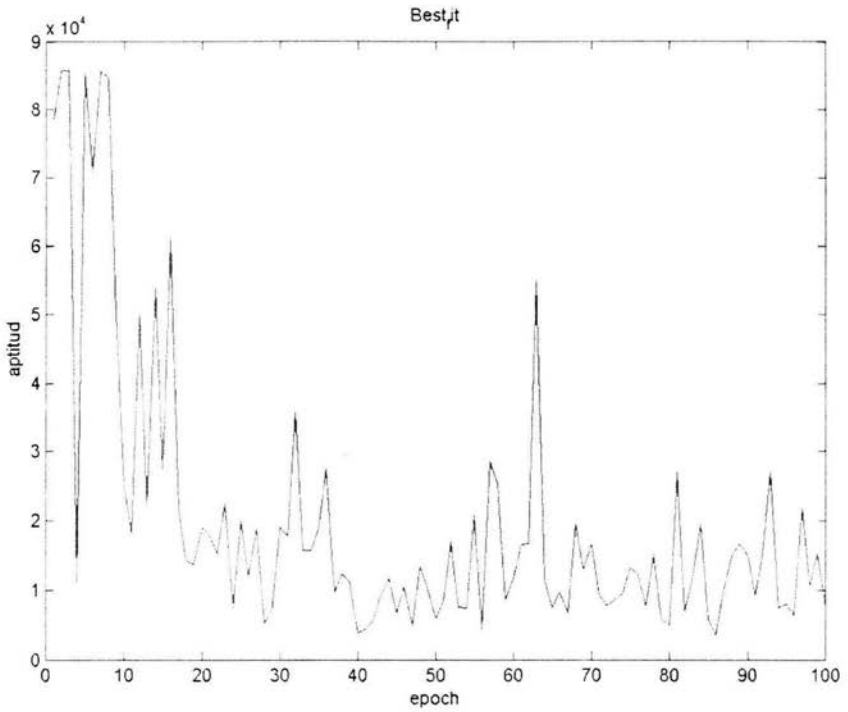


fig. 13 Gráfica de decremento de aptitud según función objetivo

La gráfica anterior muestra un decremento en el error del 0.0976 a 0.0197, siendo estó una mejora de 79.81%, con una evolución de 100 generaciones.

3.4.2 Variación del Porcentaje de aplicación de los operadores Genéticos

La variación del coeficiente de aplicación de los operadores genéticos, afecta directamente el desempeño del algoritmo. Como se muestra en las siguientes gráficas, donde se puede observar la variación brusca en la aptitud y como esta se reduce al disminuir el porcentaje de aplicación del operador de Mutación:

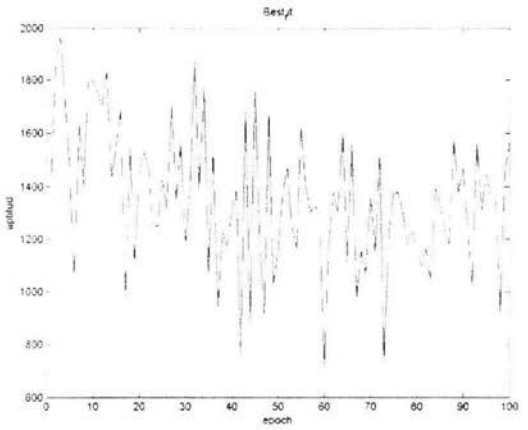


fig. 14 Porcentaje de Mutación 0.1

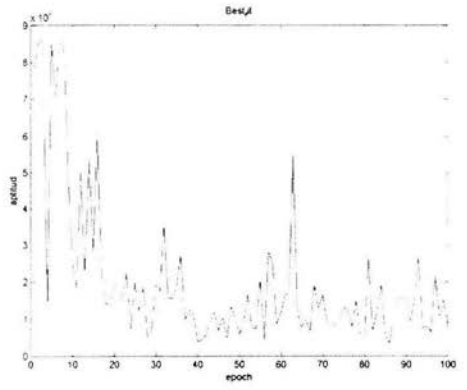


fig. 15 Con porcentaje de mutación 0.05

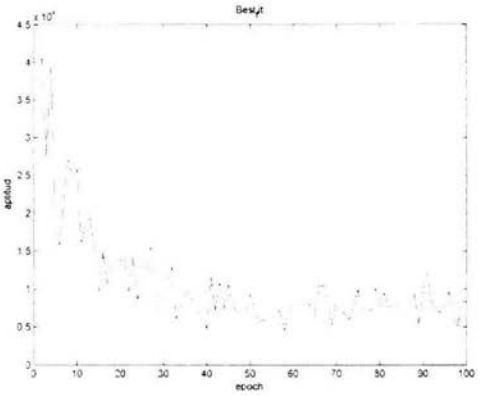


fig. 16 Con porcentaje de mutación 0.005

Con la variación de estas gráficas podemos ver de manera clara como el algoritmo responde a la variación de los parámetros. Donde la alteración se vuelve menos brusca entre mas bajo es el porcentaje de mutación.

3.4.3 Seguimiento del comportamiento bursátil haciendo uso de la red.

El comportamiento de la salida de la red neuronal ya evolucionada y entrenada, se verificó contra el conjunto de datos de prueba, con las mismas 10 variables usadas (tabla 1), aunque cada red neuronal utiliza las variables asociadas a su máscara de entradas.

Para lograr realizar las predicciones a corto plazo, se uso un conjunto de pruebas de 25 días, donde la preedición es realizada para el día siguiente, en donde el resultado del día anterior es alimentado a la red neuronal para su reentreamiento. En esta etapa de predicción es necesario un reentrenamiento, ya que son datos nuevos que la estructura de la red neuronal desconoce.

Se logró una mejora de las predicciones considerable, en comparación de las predicciones hechas por las redes de las primeras evoluciones como se muestra en las siguientes gráficas.

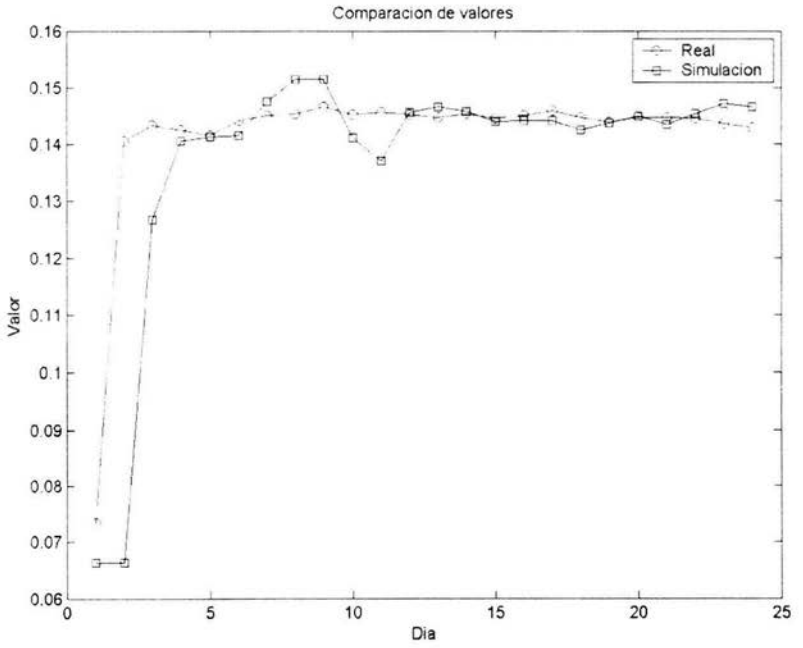


fig. 17 Seguimiento de la Red en la Primera Generación del Algoritmo Genético

Donde se encuentran que las variaciones fuertes en el precio del dólar, son anunciadas por la red neuronal. Por lo que podemos decir que el patrón de comportamiento a corto plazo ha sido aprendido de mejor manera por las redes neuronales de últimas generaciones. Por otra parte la amplia variación de los parámetros y su comportamiento poco lineal, requiere de un constante reentrenamiento, en el cual se debe considerar eliminar datos históricos demasiado antiguos, ya que el comportamiento de los mismos y la dependencia varía conforme al tiempo.

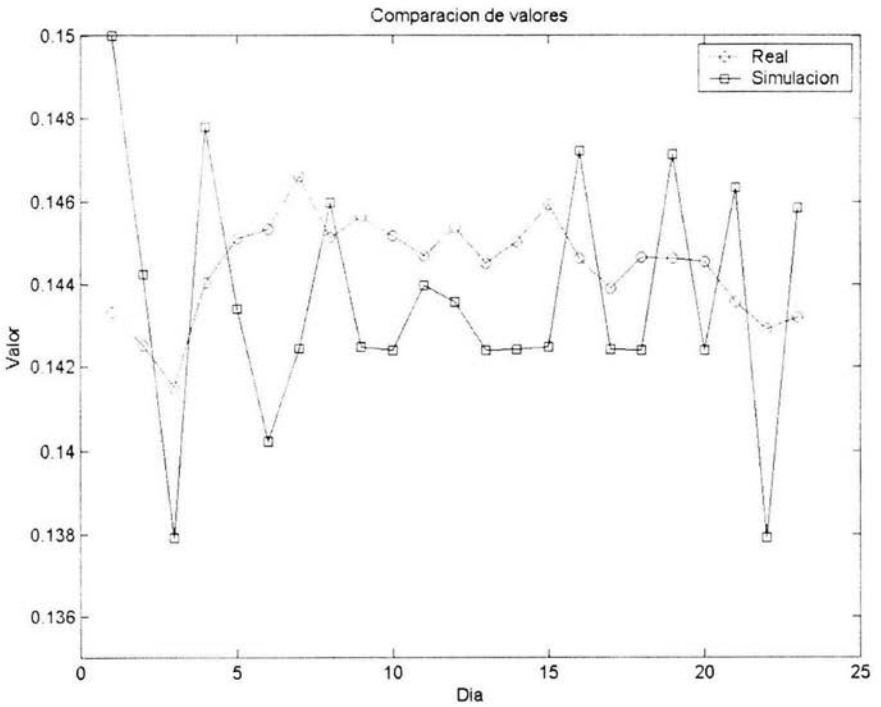


fig. 18 Seguimiento de la Red en la última generación del Algoritmo Genético

Capítulo 4.

Conclusiones:

Se logró crear una herramienta sencilla de usar, para el manejo de un proceso evolutivo entorno a diversas características de una red neuronal, para generar una optimización en base a su rendimiento, complejidad u otros aspectos de su arquitectura y desempeño.

La evolución de redes neuronales y la selección de las variables de entrada por procesos evolutivos, es una solución factible al momento de atacar problemas que contengan una amplia gama de variables, donde se desconozca la relación entre ellas y que al aproximarla con una Red Neuronal de tipo Feedforward no sea capaz de lograr los mejores resultados posibles.

Se vió que el rendimiento de una red neuronal aumenta, con una correcta selección de las variables que le son suministradas, de la misma manera que una selección de variables con una aportación menor al problema evita el entrenamiento satisfactorio en la red, por lo que también el algoritmo nos permite analizar el nivel de significancia de las variables en un modelo. Por lo que esta herramienta es ideal para la búsqueda de redes neuronales en el aprendizaje de modelos económicos o modelos que requieran un análisis de las variables a utilizar. Se logró realizar predicciones sobre series de tiempo financieras a corto plazo, donde la evolución de las redes neuronales ha logrado mejorar los resultados, por lo que se consideraron alcanzadas las expectativas de la tesis.

Bibliografía:

[1] Goldberg David E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company, Inc., 1989.

[2] Konar, Amit., Artificial intelligence and soft computing: behavioral and cognitive modeling of the human brain, CRC Press, Inc., 2000

[3] Koza, John R. Genetic programming: on the programming of computers by means of natural selection. Massachusetts Institute of Technology, 1992.

[4] Tesis de Licenciatura de Rodolfo Sánchez Guzmán, Implementación de Estrategia Evolutiva, Faculta de Ingeniería, 2003.

[5] John J. Grefenstette, James E. Baker, How Genetic Algorithms Works: A Critial Look at Implicit Parallelism, Proceedings of the Third International Conference on Genetic Algorithms, June 4-7 1989.

[6] Gilbert Syswerda, Uniform Crossover in Genetic Algorithms, Proceedings of the Third International Conference on Genetic Algorithms, June 4-7 1989.

[7] ¿Cómo funciona una célula?, Peña, Antonio, México : Fondo de Cultura Económica : ILCE, 2000 QH631 P45

[8] Comunicación Celular, Angél Luis Garcia villalón, Facultad de medicina, Dept de fisiología, universidad autónoma de Madrid.

[9] Las Células de la mente, Ricardo Tapia, La ciencia para todos, Fondo de cultura Económica, 2001.

[10] Simon Haykin, Neural Networks a comprehensive foundation, First Ed., MacMillan Publishing Company, 1994.

[11] James A. Freeman, David M. Skapura, Neural Networks algorithms, applications, and Programming Techniques, Addison-Wesley Publishing Company. 1992

[12] Jang, Jyh-Shing Roger, Neuro-fuzzy and soft computing: a computacional approach to learning and machine intelligence, Prentice-Hall, Inc, 1997.

[13] Lance D. Chambers, The practical handbook of the genetic algorithms, applications, 2nd. Ed. Chapman & Hall/CRC 2001.

[14] Solution Number: 8542, What is the random number generator in MATLAB?, TheMathWork Suport, http://www.mathworks.com/support/solutions/data/8542.shtml

[15] Abdullah, M.H.L.B.; Ganapathy, V., Neural network ensemble for financial trend prediction, TENCON 2000. Proceedings, Volume: 3 . 24-27 Sept. 2000, Pages:157 - 161 vol.3

[16] Matlab, Manuals and online support. http://www.mathworks.com

[17] Suran Goonatilake, Philip Treleaven, Intelligent systems for finance and business, John Wiley & Sons, Inc., NY USA, 1995.

[18] Yong Liu, Evolving Neural Network for Hang-Seng Stock Index Forcast, Evolutionary Computation, 2001. Proceedings of the 2001 Congress on , Volume: 1 , 27-30 May 2001.

[19] James E. Bowen, Using Neural Nets to Predict Several Sequential and Subsequent Future Values from time Series Data, Artificial Intelligence on Wall Street, 1991. Proceedings., First International Conference on , 9-11 Oct. 1991

[20] Banco de México, Información Financiera y Económica, Datos Históricos del 2003, http://www.banxico.gob.mx/

[21] DownloadQuotes, Información del histórico del índice indicador de Dow Jones de 2003, http://www.downloadquotes.com/en/

Apéndices:

Apéndice A:

****

Archivo de configuración del usado por matlab, para pasarle todos los parámetros necesarios:

Config.ini

numpop	20
max_layers	2
max_items_layers	15
complex	5
generaciones	7
funciones_transferencia	10
mut_proba	0.1
crossover_por	0.3
kill_por	0.6
epochs	100
epochs_dec	9
train_goal	0.001
epochs_squeeze	50
goal_param	1
siempre_mask	100000
inputs_size	6
ml	0

****

Apéndice B:

Archivo de README del uso de archivo de configuración:
README.txt

```
%*****************README***************
Estas son las variables de configuración que son modificables, dentro del
archivo
'config.ini'.

Nota: no se debe de modificar el formato del texto, ya que de esto
depende que el programa lo lea correctamente.

Explicación de cada variable para saber, como modificar cada uno de los
parámetros.

%*************************************************************************
NUMPOP
Descripción:
Tamaño de la población. Este tamaño es fijo durante toda la evolución
Al cambiar este parámetro a un numero muy grande se puede volver muy
lenta la evolución ya que en cada un de las generaciones, es necesario
procesar y entrenar cada uno de los vectores (individuos) de la
población.

numpop                  25          %Tamano de la población


%*************************************************************************
MAX_LAYERS
Descripción:
Este es el número máximo de capas ocultas, con las que son declaradas las
redes neuronales del algoritmo evolutivo, es necesario que este numero
sea por lo menos igual o mayor a 2
(max_layers >= 2).

max_layers        2           %maximo numero de capas ocultas


%*************************************************************************
MAX_ITEMS_LAYERS
Descripción:
Este es el numero máximo de elementos procesadores (neuronas), que puede
tener cada capa. Crecer este numero es aconsejable cuando, el tipo de
problema que se quiere aprender es de una dificultad mayor.

max_items_layer         15          %maximo numero de neuronas en capas
ocultas
```

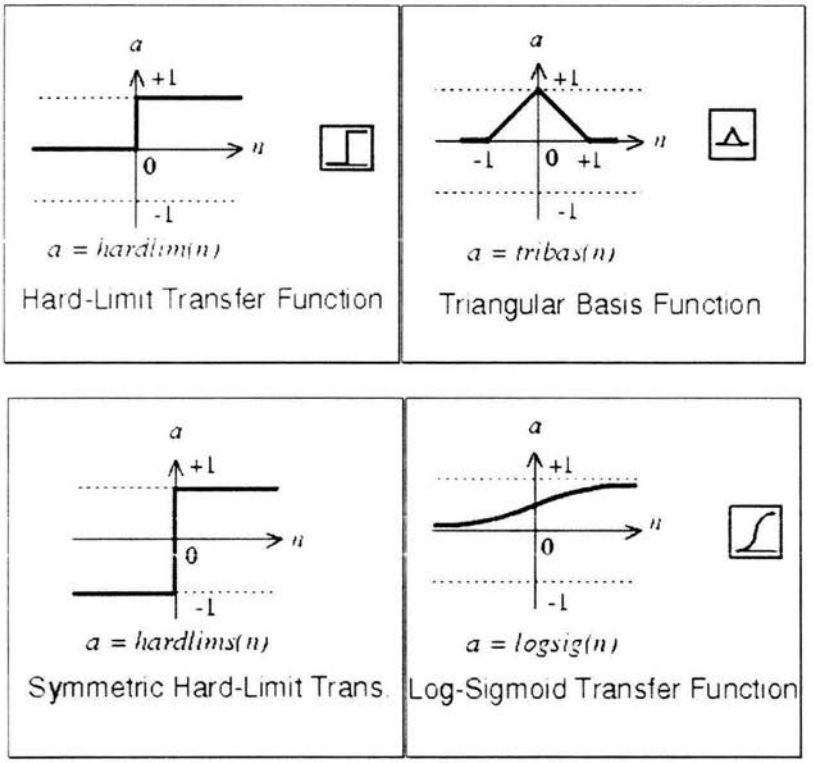
```
**********************************************************************
COMPLEX
Descripción:
No actualmente usado en el código

complex                 5           %Complejidad de la red a resolver


**********************************************************************
GENERACIONES
Descripción:
Este es el numero de generaciones, las cuales correrá el algoritmo
genético, de crecer mucho este numero puede tardarse demasiado en
ejecutase, aunque esto también depende del tamaño de la población.

generaciones            2           %Numero de generaciones


**********************************************************************
FUNCIONES DE TRANSFERENCIA
Descripcion:
Es el numero de funciones de transferencia utilizados en el código, de
tener un numero mayor (max=10), la opción de usar diferentes tipos de
funciones de transferencia, de los cuales se asignara de manera
aleatorio, a cada una de las diferentes capas de la red neuronal.

funciones_transferencia 10          %Cantidad de funciones de
transferencia usar


**********************************************************************
MUT PROBA
Descripción:
Este número indica la cantidad de mutaciones, con una distribución de
probabilidad normal, de que se vean afectados alguno parámetros de la
arquitectura de las redes neuronales.

mut_proba         0.1         %Probabilidad con la cual se efectúa una
mutación


**********************************************************************
CROSSOVER_POR
Descripción:
Este nos indica el porcentaje de la población, que generara descendencia.
Ejemplo: en caso de que la población sea de 100 individuos, y un
crossover_por de 0.3 esto generar 30 parejas, para crear descendencia.

NOTA: para que la población se mantenga constante es neceseario que el
numero de nuevo individuos, se igual el numero de individuos que son
eliminados en kill_por.

crossover_por           0.5         %poblacion la cual genera decendencia
```

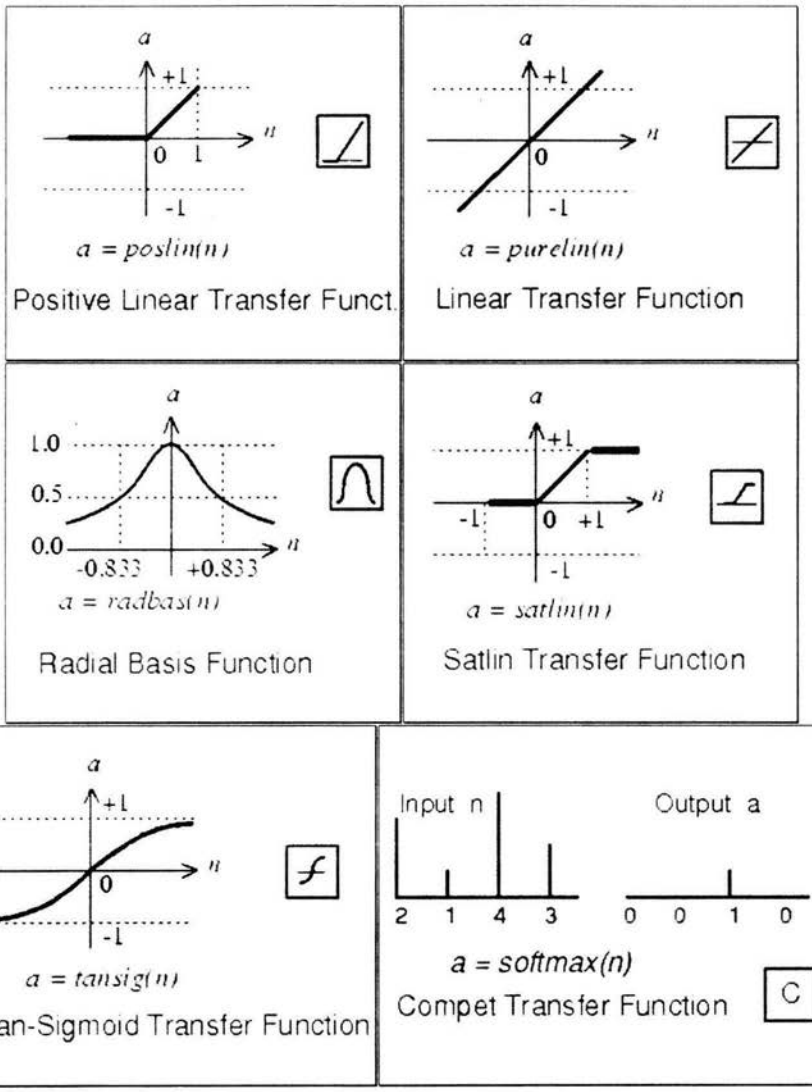
```
**********************************************************************
KILL_POR
Descripción:
Nos indica el porcentaje de la población que será eliminado.
Este porcentaje, los peores individuos según su aptitud

kill_por          0.6          %Porcentaje de la población que se elimina


**********************************************************************
EPOCHS
Descripción:
Indica el número de épocas, máximo que se utilizaran para entrenar las
redes.


epochs                    100          %Epocas de entrenamiento



**********************************************************************
EPOCHS_DEC
Descripción:
El decremento de las épocas se usa en el segundo entrenamiento, y asi
poder calcular si la red todavía cuenta con capacidad de aprendizaje.

epochs_dec        9            %Decremento de épocas


**********************************************************************
TRAIN_GOAL
Descripción:
Este parámetro se usa como el error buscado al momento de entrenar las
redes neuronales n caso de tener un error muy chico es más difícil llegar
a la meta del entrenamiento e las redes neuronales.

train_goal        0.001        %Error esperado en el entrenamiento de las
redes


**********************************************************************
EPOCHS_SQUEEZE
Descripción:
Es el número máximo de épocas, en lo que deseamos que se entrene la red
neuronal


epochs_squeeze            50           %Numero mínimo de épocas de
entrenamiento


**********************************************************************
GOAL_PARAM
Descripción:
Este parámetro tiene la opción de ser '1' o '0' solamente
```

```
En caso de que sea 1, el parámetro indica que se deben de cumplir tanto
el epochs_squeeze como el train_goal.

Si el parámetro es '0', solamente se busca que se cumpla el train_goal,
que es el error deseado, pero no importa el número de épocas que te tome
el entrenamiento

goal_param        1          %Delimita si busca train_goal AND
epochs_squeeze


**********************************************************************
SIEMPRE_MASK
Descripción:
Es la representación de un número binario, el cual nos indica que
entradas son
forzosas al momento de introducirlas a la red neuronal (indicador en
'1').
Las entradas que sean puestan en '0', son entradas opcionales, las cuales
puede ser o no escogidas por el algoritmo genético, al momento de
evolucionar este parámetro.


siempre_mask            100011          %Mascara para entradas que no
mutan


**********************************************************************
INPUTS_SIZE
Descripción:
Este parámetro le indica el numero de entradas, en caso de que la cadena
usada en siempre_mask sea menor al numero de entradas, esto sucede cuando
se tiene ceros ala izquierda en el numero binario de siempre_mask
(ejemplo:  000100)


inputs_size       6          %Numero de inputs usados


**********************************************************************
ML
Descripción:
Se indica el tipo de manejo de la población si esta va a ser:
      0     (m,1)-Es se eliminan los padres
      1     (m+1)-Es se conservan los padres

Nota: cuando se hace el (m,1)-Es, se crean dos hijos por cada pareja de
padres.

ml                      1
```

Apéndice C:

Funciones de transferencia usadas




a
+1
n
0
1
-1
a = poslin(n)
Positive Linear Transfer Funct.
a
+1
n
0
-1
a = purelin(n)
Linear Transfer Function
a
1.0
0.5
0.0
n
-0.833
+0.833
a = radbas(n)
Radial Basis Function
a
+1
n
-1
0
+1
-1
a = satlin(n)
Satlin Transfer Function
a
+1
n
0
-1
a = tansig(n)
Tan-Sigmoid Transfer Function
Input n
Output a
2 1 4 3
0 0 1 0
a = softmax(n)
Compet Transfer Function
C