UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

ANÁLISIS COMPARATIVO DE MODELOS DE APRENDIZAJE DERIVADOS DE REDES NEURONALES Y APROXIMACIÓN POLINOMIAL MULTIVARIADA.

T E S I S

QUE PARA OBTENER EL GRADO DE
MAESTRO EN CIENCIAS
(COMPUTACIÓN)

P R E S E N T A :

FEDERICO JUÁREZ ALMARAZ

DIRECTOR DE TESIS:
DR. ÁNGEL FERNANDO KURI MORALES

México, D.F. 2005

Análisis Comparativo de Modelos de Aprendizaje Derivados de Redes Neuronales y Aproximación Polinomial Multivariada

Federico Juárez Almaraz.

¨Raphel mai amech isabi almi¨, empezó a gritar la fiera boca, la cual no estarían bien otras voces más suaves; y mi Guía le dijo:

-Alma insensata, sigue entendiéndote con la trompa, y desahógate con ella, cuando te agite la cólera u otra pasión. Busca por tu cuello y encontrarás la soga que la sujeta, ¡oh alma turbada!, mirala como ciñe tu enorme pecho.

Después me dijo:

- El mismo se acusa: ese es Nemrod, por cuyo audaz pensamiento se ve obligado el mundo a usar más de una lengua. Dejémosle estar, y no lancemos nuestras palabras al viento; pues el no comprende el lenguaje de los demás, ni nadie conoce el suyo.

La divina comedia, Infierno:Canto Trigésimo Primero. Dante Alighieri

Agradecimientos.

!Listo está¡, agradezco la luz de dios que guía el sendero. Agradezco a mi madre el apoyo brindado sin el cual se sería muy dificíl seguir adelante y me hace fuerte. Así como a mi hermana que pone su fé en mí.

A quien es mi tutor, Ángel Kuri; agradezco su confianza, paciencia, consejos y reprimendas sin las cuales me hubiera sido imposible terminar este trabajo. Pero sobre todo su amistad y generosidad al compartir sus conocimientos a través de los cuales tengo otra percepción del mundo [1].

A ti Liliana que estuviste conmigo tanto en las buena como en las malas, siempre con un vivaz ánimo el cual me contagió para continuar, y sobre todo que te has mantenido firme hasta el final a pesar de todos los inconvenientes compensados por los grandes momentos que hemos compartido.

También no puedo dejar de lado a mis amigos Gary y Julio, los cuales me han apoyado en lo personal como amigos muy cercanos y que estuvieron prestos para ayudarme en la realización de este trabajo.

No se me puede olvidar mi amigo Gustavo, que además de brindarme una amistad sincera, me ha escuchado y ayudado a salir de muchos lios y dificultades.

También debo de agradecer a otra mucha gente que me ha otorgado su amistad y confianza. A mi gran amigo de muchos años Alfredo Acosta, quien simpre ha creido en mi y a través de sus consejos me siento apoyado. También ha Daniel (Logan) quien se mantiene cerca para escuchar.

Mucho le agradezco a Lulú quien nos ayudo tanto a Liliana como a mi para hacer mas llevadera nuestra estancia en el IIMAS, también Ceci quien muchas veces nos ha apoyado desinteresadamente.

También agradezco a Karina, Erick y Yazmín por su amistad y cariño, no pueden faltar las nuevas amistades, Olivia y Roberto, que a pesar de los problemas, se quedaron cerca entregando su amistad.

Un agradecimiento especial a mis sinodales por el tiempo invertido en leer este trabajo.

Concluyo danto gracias a la Universidad Nacional Autónoma de México que ha sido mi hogar y a través de ella he ido consolidando mi formación, tanto academica y como persona.

[1] Y le pido una disculpa por los corajes que le hice pasar sobre todo por mis faltotas de ortografía

Índice general

Introducción.

Desde su aparición las Redes Neuronales Artificiales han despertado un gran interés. La razón: son una metodología exitosa que resuelve una gran variedad de problemas, incluidas la clasificación y la aproximación de funciones [Hayk99], [Roja96], [Terr99]. Éstas son especialmente utilizadas como un aproximador de funciones pues no requieren un conocimiento previo de la distribución de los datos de entrada, además de que se ha podido demostrar formalmente su propiedad de Aproximador Universal. Sin embargo, tienen un defecto: no es posible extraer el conocimiento explícito que explique cómo la Red Neuronal resolvió el problema. Se han tratado de encontrar métodos con los cuales se pueda extraer dicho conocimiento, pero el éxito ha sido limitado [RKZ02], [JNL96], [CS93].

Aunque es deseable poder resolver este problema, también se pueden proponer opciones. En el caso particular de este trabajo es nuestra intención poder mostrar y comparar una metodología alternativa, que sea capaz de realizar la tarea de aproximación de funciones, y que nos brinde además una expresión matemática del sistema que asimile. Esta metodología la denominamos: Aproximación Polinomial Multivariada [Kuri], [Kuri93] [Kuri02].

El propósito de este trabajo es mostrar de manera empírica las ventajas y desventajas que muestran ambas metodologías. Las cuestiones que vamos a investigar son:

- ¿Qué tan buena es la aproximación realizada por el Aproximador Polinomial Multivariado con respecto a las Redes Neuronales?
- ¿Es comparable la capacidad de generalización del Aproximador Polinomial Multivariado con respecto a las Redes Neuronales?
- ¿Habrá alguna relación entre ambas metodologías?
- ¿Qué ventajas ofrece una metodología con respecto a la otra?

1

Redes Neuronales.

Basándose en la eficiencia de los procesos llevados a cabo por el cerebro e inspirados en su funcionamiento, se han intentado desarrollar metodologías que simulen dicho comportamiento. Tal es el caso de las Redes Neuronales Artificiales (RNA). Éstas imitan la estructura y funcionamiento del cerebro humano mediante modelos matemáticos, enfocándose principalmente a tres cualidades básicas:

1. El conocimiento está distribuido sobre múltiples neuronas dentro del cerebro.
2. Las neuronas pueden comunicarse localmente con otras neuronas.
3. El cerebro es adaptable, esto es, se automodifica para afrontar nuevas circunstancias.

La terminología a través de la cual se describen las RN's está basada en las tres características anteriores, de tal manera que una red neuronal tiene los siguientes elementos:

a. Estructura de la neurona;

b. Arquitectura de la red;

c. Algoritmo de aprendizaje (adaptación).

Existen varias definiciones de RNA's, por ejemplo [Hayk99]:

Definición 1.1 *Una red neuronal es un procesador masivamente paralelo y distribuido, que es propenso por naturaleza a almacenar conocimiento experimental y hacerlo disponible para su uso.*

Este mecanismo se parece al cerebro en dos aspectos:

1. *El conocimiento es adquirido por la red a través de un proceso que se denomina aprendizaje.*

2. *El conocimiento se almacena mediante la modificación de la fuerza o pesos sinápticos de las distintas uniones entre neuronas.*

En las siguientes secciones realizaremos una breve descripción de estos conceptos básicos.

1.1. Estructura de la Red Neuronal.

1.1.1. La Neurona.

Los componentes básicos de una RNA son las neuronas las cuales pueden ser entendidas como *unidades de procesamiento*. Una neurona es un dispositivo simple que consiste de tres componentes:

1. Un *conjunto de sinapsis o ligas de conexión*, las cuales son caracterizadas por un *peso* o *potencia* propios a cada conexión. En este punto la operación que se hace es: recibir una señal x_j en la entrada de la sinapsis j, la cual se conecta a la neurona k y es multiplicada por el peso sináptico w_{kj}.

2. Una función *suma* que se encarga de sumar los productos de las señales de entrada por los pesos de las sinapsis que llegan a la neurona. La operación descrita corresponde al cálculo de una combinación lineal (se concibe como la actividad interna o actual de la neurona). El rango de amplitud para la salida de la neurona es normalizado y está normalmente definido en los intervalos $[0, 1]$ o $[-1, 1]$.

   La neurona tiene la siguiente forma:

   donde

   $$u_k = \sum_{j=1}^{p} w_{kj} x_j$$

   y

   $$y_k = \phi(u_k - \theta_k)$$

   siendo conocido θ_k como el *umbral de disparo* de la Neurona.

3. La *función de activación o transferencia*, se encarga de determinar el nuevo estado de actividad de la neurona con base a la actividad actual de la red, condicionando de este modo la amplitud de salida de una neurona.

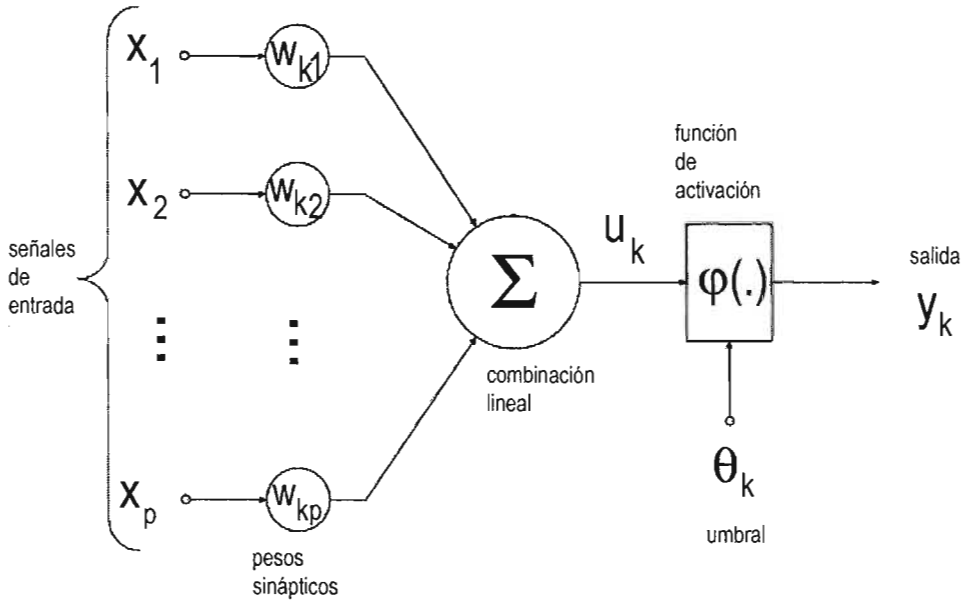


Figura 1.1: Esquema de una neurona.

1.2. Arquitectura de una Red Neuronal.

Para definir la arquitectura general de una RNA, primero enunciaremos algunos conceptos para estar en posición de definir de manera formal las RNA.

Una *gráfica de flujo de señal* es una red de ligas dirigidas que interconectan ciertos puntos llamados nodos. En particular diremos que un nodo j tiene asociado un nodo señal x_j. Una liga dirigida queda determinada como una pareja ordenada (j,k) donde j es el nodo origen y k el nodo terminal; este último a su vez tiene asociada una función de transferencia que especifica la manera en la cual la señal y_k llega al nodo k, dependiendo de la señal x_j en el nodo j. Las señales fluyen a través de la gráfica de acuerdo a tres reglas básicas:

Regla 1: El flujo de una señal es transmitido en la liga sólo en la dirección definida por el arco de la misma.

Existen dos tipo de ligas:

a) *Liga Sináptica*, la cual mantiene una relación lineal de entrada-salida, a través de una combinación lineal de señal de entrada y el peso asociado a la liga.

b) *Liga de Activación*, la cual mantiene una relación de entrada-salida no lineal; transmite la señal emitida por la función de activación de la neurona.

Regla 2: La señal de un nodo es igual a la suma algebraica de todas las señales que entran al nodo correspondiente mediante las ligas de entrada.

Regla 3 : La señal de un nodo es transmitida a cada una de las ligas de salida que surgen de dicho nodo, la transmisión se realiza independientemente de la función de transferencia de las ligas de salida.

Haciendo uso de las nociones anteriores, definimos formalmente una RNA [Hayk99].

Definición 1.2 *Una Red Neuronal Artificial es una gráfica dirigida que consiste de nodos interconectados por ligas sinápticas y de activación, las cuales están caracterizadas por las siguientes propiedades:*

1. *Cada neurona se representa por un conjunto de ligas sinápticas lineales; un umbral de aplicación externo y una liga de activación no lineal. El umbral se define como una liga sináptica cuya señal de entrada es un valor constante.*
2. *La liga sináptica de una neurona pesa o valora sus respectivas señales de entrada.*
3. *La suma de los pesos de las señales de entrada define el nivel de actividad interna de la neurona en cuestión.*
4. *La liga de activación acota el nivel de actividad interna de la neurona produciendo una salida que representa los cambios de estado de la neurona.*

A partir de la definición es como se establecen las diferentes arquitecturas y topologías de las RNA que se tienen catalogados en la literatura [Hayk99].

1.3. Reglas de Aprendizaje

.

Un *algoritmo de aprendizaje* es un método adaptable por medio del cual una RNA se autorganiza, implementando un comportamiento deseado. Dando un conjunto de muestras para la RNA, el algoritmo de aprendizaje reconfigura iterativamente los parámetros de la red hasta obtener una respuesta deseada. Un paso de corrección es ejecutado iterativamente hasta que la red aprende a producir la respuesta deseada. Esencialmente, un *algoritmo de aprendizaje* es un ciclo cerrado de presentación de ejemplos y correcciones de los parámetros de la red.

En algunos casos sencillos, los pesos de la RN pueden ser encontrados a través de una secuencia de pruebas estocásticas generando combinaciones numéricas. Sin embargo, tales algoritmos realizan una búsqueda ciega de la solución, por lo cual no se consideraría un aprendizaje.

El algoritmo de aprendizaje debe adaptar los parámetros de la RNA de acuerdo a la experiencia previa hasta que se encuentre una solución, en caso de que ésta exista. [Roja96]

Los algoritmos de aprendizaje se dividen en dos diferentes tipos: *supervisados* y *no-supervisados.*

1.3.1. Aprendizaje Supervisado.

El *aprendizaje supervisado* es un método en el cual se selecciona un conjunto de muestras del sistema a asimilar para introducirlas a la red neuronal. La salida calculada por la red es cotejada contra la salida esperada para calcular la diferencia entre ellas. Tal medida se denomina *error*. Los pesos de la RNA son corregidos de acuerdo a la magnitud del error y al criterio utilizado por el algoritmo de aprendizaje. Este conjunto de técnicas de aprendizaje se denomina *aprendizaje con maestro*, dado que el proceso de control conoce las respuestas correctas para el conjunto de muestras de entrada.

1.3.2. Aprendizaje No-Supervisado

El *aprendizaje no-supervisado* se utiliza cuando se tiene un conjunto de entradas para el cual se desconoce la salida que debe ser producida por la red neuronal. En tal caso la RN trata de aprender a distinguir las entradas que se le proporcionan en grupos (*cluster*); así las neuronas de la RN se tienen que organizar de tal suerte que, cada vez que se proporciona algún tipo de entrada cada una de éstas se activa si dicha entrada pertenece al grupo que distinguen.

En este caso no sabemos a priori qué neuronas están especializadas en cuáles grupos. En general no sabemos de antemano cuántos grupos bien definidos están presentes en el conjunto de entradas; en este caso no hay un maestro disponible. La RN se debe autorganizar para ser capaz de asociar clusters con las neuronas que disponga.

Con esta definición podemos afirmar que las RNA son en particular una poderosa herramienta de modelado de datos, que es capaz de capturar y representar complejas relaciones de entrada/salida. La motivación para el desarrollo de una metodología sobre redes neuronales estriba en el deseo de obtener un sistema artificial que pueda realizar tareas "inteligentes", similares a las que hace el cerebro humano. Las Redes Neuronales imitan al cerebro humano de dos formas:

1. Una Red Neuronal adquiere conocimiento a través del aprendizaje.

2. Una Red Neuronal almacena su conocimiento en las conexiones inter-neuronales, las cuales denominamos como pesos sinápticos.

El verdadero poder y ventaja de las RNA radica en su habilidad para representar tanto *relaciones lineales como no-lineales*, así como en su capacidad para aprender estas relaciones directamente del modelo de datos proporcionado.

Tradicionalmente, los modelos lineales para las Redes Neuronales son inadecuados cuando se llegan a modelar datos que contienen características no-lineales. Pero esta desventaja ha sido librada, aumentando la capacidad de cómputo de las unidades elementales que integran a una RNA.

Sólo resta resaltar un detalle, que es tema de conversación en contra de las RNA y comienza por lo afirmado en el punto (2.) anteriormente citado. Hasta ahora no se ha

podido desarrollar un método mediante el cual se pueda extraer información del conocimiento almacenado en las conexiones inter-neuronales. Por el momento dejaremos esta discusión aquí y volveremos a ella posteriormente.

1.4. El Perceptrón

Hasta este punto hemos estudiado el concepto de RNA en general pero mencionamos en secciones anteriores, que existen de diferentes tipos de redes neuronales; entre ellos se encuentran las Redes de Perceptrones Multicapa, las cuales serán nuestro objeto de estudio.

1.4.1. Arquitectura de la Red de Perceptrones Multicapa (RPM).

Como ya se mencionó antes, una RNA está determinada por medio de las unidades de cómputo elementales que la componen y la disposición en que están organizadas las mismas.

En el caso particular de la RPM, las neuronas están compuestas por unidades de cómputo que son denominadas *perceptrón.*

Los perceptrones están conformados de n conexiones de entrada, las cuales son los argumentos para una función de integración $\Psi : \mathbb{R}^n \to \mathbb{R}$ por medio de la cual se establece el nivel de actividad del perceptrón con respecto a los valores de entrada. La función Ψ realiza una combinación lineal de las entradas y los pesos sinápticos. El nivel de actividad es evaluado por una función de activación $\Phi : \mathbb{R} \to \mathbb{R}$ que define el valor de salida del perceptrón (neurona). Por lo regular la función de activación Φ que se utiliza para las RPM's es alguna de las siguientes: la función logística, la función Tangente Hiperbólica o una función Lineal.
A continuación definimos la arquitectura de los RPM [Roja96].

Definición 1.3 *La arquitectura de una Red de Perceptrones Multicapa es una tupla* (I, N, O, E) *donde:*

- I *es el conjunto de entradas;*
- N *un conjunto de neuronas (perceptrones);*
- O *es un conjunto de salidas;*
- E *es un conjunto de aristas dirigidas con pesos.*

Además, la arista dirigida queda determinada como (u, v, w) *donde* $u \in I \cup N$, $v \in N \cup O$ *y* $w \in \mathbb{R}$.

Notemos que los elementos del conjunto I de entradas son nodos a través de los cuales introducimos la información a la RPM, los cuales no realizan ningún cálculo. También, se dice que es una arquitectura en capas pues las neuronas que están en N se hayan dividas en l subconjuntos $N_1, N_2, \ldots, N_l$, tal que las conexiones del subconjunto N_1 van

únicamente a las unidades de N_2, las que parten de N_2 a N_3, etc. Los dispositivos de entrada solamente están conectados a las unidades del subconjunto N_1. Análogamente, las unidades del subconjunto N_l están conectadas a las unidades de salida. Los subconjuntos N_i son denominados las capas de la red.

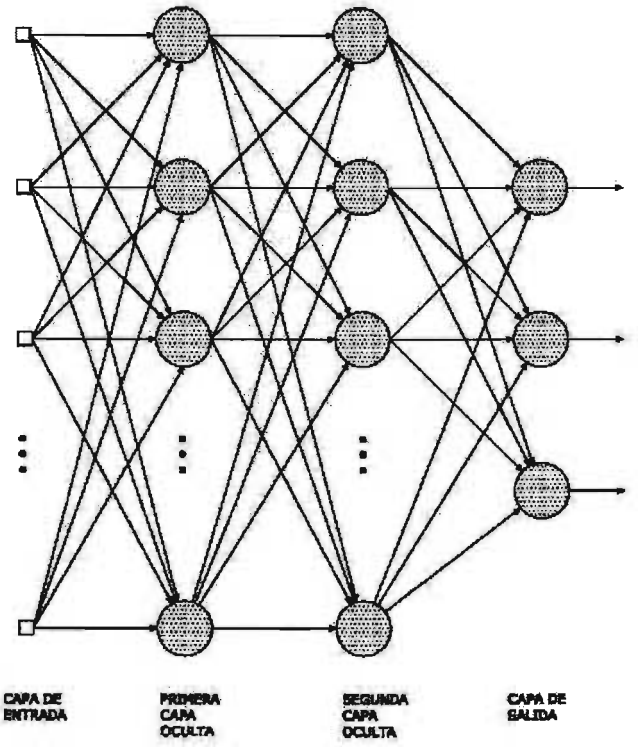


Figura 1.2: Arquitectura de un Perceptrón Multicapa

1.4.2. Aprendizaje de la Red de Perceptrones Multicapa.

El aprendizaje de la Red de Perceptrones Multicapa corresponde a un aprendizaje supervisado, pues este proceso de aprendizaje requiere de un conjunto de datos de entrada los cuales están asociados a una salida deseada. La meta de este tipo de redes es la de crear (asimilar) un modelo funcional correcto entre las entradas y sus respectivas salidas, usando

datos históricos del modelo, el cual es utilizado para producir una salida aproximada cuando se desconoce la salida deseada.

Las RPM son el tipo de RN más estudiado y utilizado. En consecuencia, se tienen en la literatura varias técnicas de entrenamiento para su aprendizaje (el más utilizado es el llamado *Algoritmo de Retropropagación*).

Algoritmo de Retropropagación.

En el *Algoritmo de Retropropagación* la idea es hacer que los datos de entrada sean repetidamente presentados a la Red Neuronal con cada presentación, la salida de la Red Neuronal es comparada con la salida deseada y se calcula la diferencia entre ambas, esta diferencia se conoce como: *error*. Éste es "enviado" de regreso a través de las conexiones de la red (*retropropagación*), utilizándolo para ajustar los pesos de tal forma que éste vaya decreciendo con cada iteración y el modelo neuronal vaya produciendo un valor cercano a la salida deseada. Este proceso es conocido como *entrenamiento.*
A continuación se muestra el algoritmo de retropropagación:

ALGORITMO DE RETROPROPAGACIÓN.

1. Inicialización. Escoger los pesos sinápticos y umbrales de una distribución uniforme cuya media es cero.

2. Presentación de los ejemplos de entrenamiento. Presentar un estado de los patrones de entrenamiento (esto es una muestra) a la red, seleccionados de manera aleatoria.

3. Cálculo hacia adelante. Sea $(x(t), d(t))$ un ejemplar de entrenamiento

$$y_j^{(L)} = o_j(t)$$

$$e_j(t) = d_j(t) - o_j(t)$$

4. Cálculo de retropropagación.

   Calcular los valores de corrección las δ gradientes locales.

$$\delta_j^{(l)}(t) = \begin{cases} e_j^{(L)}(t) f'(v_j^{(L)}(t)) & \text{para la neurona } j \text{ de la capa de salida } L \\ f'(v_k(t)) \sum_k \delta_k(t) w_{kj}(t) & \text{para la neurona } j \text{ de la capa oculta } l \end{cases}$$

   Ajustar los pesos de la red de la capa l de acuerdo con

$$w_{ji}^{(l)(t+1)} = w_{ji}^{(l)}(t) + \alpha[w_{ji}^{(l)}(t-1)] + \eta \delta_j^{(l)} y_i^{l-1}(t)$$

5. Iteración. Iterar los cálculos hacia adelante y hacia atrás, presentando nuevas épocas de ejemplos de entrenamiento hasta que el criterio de paro se cumpla. El orden de presentación de los ejemplos de entrenamiento se debe hacer al azar entre estados (épocas).

1.5. Teorema del Aproximador Universal.

Al hablar de que las RN's tienen la capacidad de Aproximador Universal, desde el punto de vista matemático, se entiende que todo modelo matemático (funciones/operadores) se puede expresar en forma aproximada en términos de los modelos que representan la RNA (que también es una composición de funciones).

Cuando aparecieron las primeras definiciones e implementaciones prácticas de las RN's, se produjeron muchas interrogantes acerca de sus capacidades. En consecuencia aparecieron preguntas como: utilizando las redes neuronales ¿es posible resolver cualquier problema, al menos numérico?.

Se han realizado múltiples investigaciones entorno a descubrir el potencial de las RN's. En el contexto de las Redes de Perceptrones Multicapa, Cybenko fue uno de los primeros en demostrar de manera rigurosa que una RPM con una sola capa oculta es suficiente para aproximar de manera uniforme cualquier función continua acotada en un hipercubo (en el año de 1988)[Hayk99]. A continuación se enuncia tal teorema:

Teorema 1.1 *(Teorema del Aproximador Universal).*

Sea $\phi(\cdot)$ una función continua, no-constante, acotada y monótonamente creciente. Sea I_p un hipercubo unidad p-dimensional $[0,1]^p$. El espacio de funciones continuas en I_p es denotado por $C(I_p)$. Entonces dada cualquier función $f \in C(Ip)$ y $\varepsilon > 0$, existe un entero M y un conjunto de constantes α_i, θ_i y $w_{i,j}$ donde $i = 1, \ldots, M$ y $j = 1, \ldots, p$ tales que podemos definir

$$F(x_1, \ldots, x_p) = \sum_{i=1}^{M} \alpha_i \phi(\sum_{j=1}^{p} w_{i,j} x_j - \theta_i) \tag{1.1}$$

entendiendo por esto que F es una aproximación de la función $f(\cdot)$, esto es:

$$|F(x_1, \ldots, x_p) - f(x_1, \ldots, x_p)| < \epsilon$$

para cualquier $x_1, \ldots, x_p \in I_p$.

Este teorema es directamente aplicable a un RPM [Hayk99]. Si utilizamos una función de activación logística $\frac{1}{[1+\exp(-v)]}$ como una función no-lineal en el modelo neuronal para la construcción de la RPM (la función logística es una función no-constante y acotada), se satisfacen las condiciones impuestas a la función $\phi(\cdot)$. Además, observemos que la ecuación (1.1) representa la salida de un RPM descrita así:

1. La red tiene p nodos de entrada y está constituida por una sola capa oculta de M neuronas; las entradas se denotan por $x_1, x_2, \ldots, x_p$.

2. La neurona i tiene pesos sinápticos $w_{i,1}, \ldots, w_{i,p}$ y un umbral θ_i,

3. La salida de la red es una combinación lineal de las salidas de las neuronas ocultas, con $\alpha_1, \ldots, \alpha_M$ definiendo los coeficientes de esta combinación.

El teorema del aproximador universal es un *teorema de existencia* en el sentido de que establece la justificación matemática para la aproximación de una función continua arbitraria sin establecer su representación exacta. La ecuación (1.1) que es la parte medular del teorema nos dice que se generalizan las aproximaciones a funciones reales por series de Fourier finitas [Hayk99]. En consecuencia, el teorema establece que una sóla capa es suficiente para que una RPM calcule una aproximación uniforme ε, entrenándola con un conjunto representativo de entradas $x_1, \ldots, x_p$ y un conjunto de salidas deseadas (objetivo) $f(x_1, \ldots, x_p)$.

Este punto es relevante para nosotros pues de aquí partimos para el desarrollo de nuestra investigación. De lo anterior se observa la capacidad que poseen las RNA para realizar aproximaciones (por lo menos de funciones reales). Ahora bien, existe un detalle el cual es un amplio tema de estudio con respecto a las RNA y es el que concierne a la extracción del conocimiento almacenado en la red neuronal. Este problema es denominado *Problema de la Caja Negra.* Para su solución se han propuesto Algoritmos para la Extracción de Reglas de las redes neuronales como:

- Rule Extraction As Learning. [CS93]
- Relevance Information.[JNL96]

Sin embargo, estas técnicas no son del todo efectivas y mucho menos eficientes. En muchos casos sólo se pueden probar en dominios muy pequeños. Ninguna de las técnicas se ha podido definir como óptima ni para que tipo de problemas es mejor.

En particular vamos a estudiar una metodología que es capaz de aproximar (aprender) funciones reales continuas. Entonces podremos comparar ésta en efectividad con la RPM de una capa. Esta metodología no sólo puede aproximar (y bajo algunas restricciones generalizar) sino también da por resultado una expresión matemática del sistema que aproximó algo que, como se puede observar hasta el momento no se ha logrado con la RPM.

Para lograr nuestro objetivo es indispensable atacar un problema de optimización no lineal. Apelamos entonces a la computación evolutiva, como se discute en el siguiente capítulo.

2

Optimización y Algoritmos Genéticos

En este capítulo se expondrán las ideas básicas sobre la optimización de funciones. En particular se estudiará una técnica que resulta por demás útil, comúnmente denominada *Algoritmos Genéticos* (en adelante AG). El interés radica en el hecho de que los AG no sólo realizan la optimización de funciones, sino que también son capaces de proporcionar la caracterización (forma) de dicha optimización, la cual será crucial para la metodología que vamos a estudiar.

2.1. Optimización de Funciones.

Para las matemáticas el término optimización se refiere al estudio de problemas que tienen la siguiente forma:

Dada una función $f : A \to \mathbb{R}$ para algún conjunto de números reales A, encontrar un elemento $x_0 \in A$ tal que $f(x_0) \leq f(x)$ para toda $x \in A$ (minimización); o que x_0 satisfaga $f(x_0) \geq f(x)$ para toda $x \in A$ (maximización).

Tal formulación es algunas veces llamada programación lineal. Éste es un tema ampliamente estudiado pues muchos problemas teóricos pueden ser modelados en este marco general.

En la mayoría de las ocasiones se plantea que A sea un subconjunto del espacio euclidiano $\mathbb{R}^n$. Lo cual no necesariamente es cierto pues cabe la posibilidad de que trabajemos con otros dominios (como veremos más adelante), los cuales en ocasiones están especificados por un conjunto de restricciones. Pueden pensarse como un conjunto de igualdades o desigualdades que los miembros de A tienen que satisfacer. Los elementos de A se denominan *soluciones factibles.* La función f es llamada *función objetivo* o *función de costos.* Finalmente una solución factible que minimiza (o maximiza si es el caso) la función objetivo es llamada una *solución óptima.*

Una característica más de este problema consiste en definir las soluciones parciales de las soluciones locales, esto es, en general hay varias soluciones que son valores mínimos o máximos locales. Un mínimo local x^* se define como un punto tal que para algún valor $\delta > 0$ y para toda x, se cumple:

$$|x - x^*| \leq \delta \text{ si } f(x^*) \leq f(x)$$

El máximo local es definido de manera similar. Es común encontrar mínimos o máximos locales por lo cual una dificultad adicional al problema de optimización es el de garantizar que la solución encontrada realmente es un mínimo o máximo global.

2.2. Algoritmos Genéticos.

Una vez analizado el problema de optimización continuaremos hablando sobre una técnica que se ocupa de optimizar funciones (denominada *Algoritmos Genéticos*), la que ha demostrado ser eficaz y sobre todo una poderosa herramienta práctica, pues no sólo es capaz de hallar la solución óptima de funciones de valores reales, sino que también es capaz de optimizar otro tipo de problemas cuya representación no necesariamente tiene que ser llevada a un dominio de valores reales.

2.2.1. Modelo e Inspiración.

Puesto que la gama de problemas relacionados con la optimización es muy amplia, a lo largo de la historia de las matemáticas se han desarrollado diferentes técnicas para resolverlos. Sin embargo, desde tiempos remotos la humanidad ha observado un modelo el cual ha demostrado ser efectivo y eficiente, tal modelo es la Naturaleza.

Podemos citar varios ejemplos de estudiosos de la naturaleza que nos muestran dicha afirmación. Por ejemplo, Leonardo Da Vinci escribió: *Aunque el ingenio humano puede lograr infinidad de inventos, nunca ideará ninguno mejor, más sencillo y directo que los que hace la naturaleza ya que en sus inventos no falta nada y nada es superfluo*[1]. Se vuelve claro el hecho de que en la naturaleza los seres vivientes están perfectamente adaptados al medio que los circunda. Diría Leibniz[2]: *cuanto más informados e iluminados estemos acerca de las obras de Dios, más inclinados estaremos a encontrarlas excelentes y totalmente conformes a cuanto se hubiera podido desear.*

Así que la naturaleza genera seres óptimos, seres perfectamente adaptados a su entorno, constituido por una infinidad de circunstancias: temperatura, presión atmosférica, precipitación, velocidad del viento, altitud, nivel de insolación, depredadores, alimentos, etc. Al observar a los seres vivos, concluimos que su adaptación es perfecta porque a lo largo de miles de generaciones se ha perfeccionado a pequeños saltos infinitesimales. Lo que vemos hoy en día es el resultado acumulado de miles de experimentos exitosos que se han ido refinando paulatinamente a partir de alguna creación primigenia. Los experimentos fallidos, algunos órdenes mayor que los exitosos, ya no los vemos. Los individuos

[1] Cuaderno de Notas, la vida y estructura de las cosas, el embrión
[2] Discours de Metaphysique

resultantes perecieron compitiendo al lado de otros más aptos para sobrevivir. *La selección natural obra solamente mediante la conservación y acumulación de pequeñas modificaciones heredadas, provechosas todas al ser conservado*, scribió Darwin y dio nombre a este proceso[3].

Estas modificaciones heredadas, señaladas por Darwin como las generadoras de organismos mejores, son llamadas mutaciones hoy en día y constituyen el motor de la evolución. Un organismo mutante ha sufrido una modificación que lo hace diferente al resto de sus congéneres. Esta modificación puede ser un inconveniente para él, pero también puede suceder que le confiera alguna cualidad que le permita sobrevivir más fácilmente que al resto de los individuos de su especie. Este organismo tendrá mayor probabilidad de reproducirse y heredará a sus descendientes la característica que le dío ventaja. Con el tiempo gracias a la competencia, los organismos que en un principio eran raros se volverán comunes a costa de la desaparición del *modelo anterior*. Se habrá dado entonces un paso en el proceso evolutivo.

En este marco general sobre la evolución de los seres vivos, podemos identificar los pasos generales que sigue la naturaleza para realizar dicha evolución:

- En la naturaleza las características de los seres vivos, incluso aquéllas que los hacen óptimos para habitar en su medio, están determinadas por las proteínas que producen. A su vez, estas proteínas se codifican en el material genético en cada una de las células del individuo. Así pues, la naturaleza ha mapeado cada posible solución al problema de crear un individuo óptimo en una secuencia particular de *bases* que producirá ciertas proteínas, o sea, ha *codificado* el dominio del problema.

- Se tiene una primera población de seres vivos, los cuales buscan sobrevivir al medio ambiente que los rodea. La naturaleza establece para cada especie algún mecanismo con el que prueba (evalúa) qué tan buena es su adaptación al medio (qué tan fácil es que lo preden, que tan rápido es, si ahorra energía en sus movimientos, etc).

- Una vez que la población ha sobrevivido (adaptación al medio), los que mejor estén adaptados serán *seleccionados* para reproducirse.

- Se realiza la *cruza* de los individuos que fueron seleccionados para ese propósito, de tal suerte que sus características sean propagadas en sus descendientes.

- Un punto que no resultó explícito pero que se manifiesta con una baja regularidad es la *mutación*. Este proceso lo que suscita es que algunos individuos manifiesten una modificación "repentina" en su información genética, lo cual provoca que haya nuevas posibilidades de modificación que puedan resultar exitosas. En la mayoría de los casos no es así pero es una manera de explorar nuevas posibilidades de adaptación.

- Bajo diversos criterios los nuevos individuos resultantes son integrados a la población para que nuevamente se inicie el ciclo.

[3]El origen de las especies, selección natural

Con estas ideas, en el año de 1970, John Holland introdujo la primera propuesta de un algoritmo de optimización basado en los puntos anteriores. Actualmente dicho proceso es denominado *Algoritmo Genético Simple*, definido de la siguiente manera:

ALGORITMO GENÉTICO SIMPLE:

1. Codificar el dominio del problema. Es frecuente que el código de los elementos del dominio del problema utilicen un alfabeto binario (0's y 1's)
2. Generar un conjunto aleatorio (población inicial) de N posibles soluciones codificadas al problema. A éste se le denomina población actual.
3. Calificar cada posible solución (individuo) de la población actual.
4. Seleccionar dos individuos de la población actual con una probabilidad proporcional a su calificación.
5. Lanzar una moneda al aire (con probabilidad p_c de caer cara).
6. Si cayó cara, mezclar los códigos de los dos individuos seleccionados para formar dos híbridos, los cuales son llamados *nuevos individuos.*
7. Si cayó cruz entonces los individuos seleccionados serán los nuevos individuos.
8. Por cada bit de cada nuevo individuo lanzar otra moneda al aire (con probabilidad p_m cae cara).
9. Si cae cara, cambiar el bit en turno por su complemento.
10. Si cae cruz, el bit permanece inalterado.
11. Incluir a los dos nuevo individuos en una *nueva población.*
12. Si la nueva población tiene ya N individuos, llamarla *población actual* y regresar al paso tres, a menos que se cumpla alguna condición de terminación.
13. En caso contrario, regresar al paso 4.

Una descripción básica sobre el comportamiento de los individuos de la población a lo largo de la ejecución de un algoritmo genético está dada por el *Teorema de Esquema.* El teorema originalmente fue pensando para una codificación binaria; de esta forma, un esquema es una cadena de símbolos tomados del alfabeto $\{0, 1, \#\}$. El caracter $\#$ es un símbolo *comodín*, por lo cual, un esquema puede representa varias cadenas de bits.

Dos magnitudes importantes que se tienen de un esquema son: *el orden* O de un esquema, el cual es el número de símbolos que no son comodines $(\#)$ y la *longitud de definición* L, que es la distancia que hay entre la primera y la última posición explicita en la cadena.

Con base en estos elementos se enuncia el teorema de esquema, el cual predice cómo el número de ejemplares de un esquema en una población varía de una generación a la siguiente. El teorema se expresa como sigue:

$$m(H,t+1) \geq m(H,t)\frac{\overline{f}(H)}{\overline{f}}((1-p_c)(p_r - O(H))p_m)$$

Donde:

- m(H,t) es el número de cadenas que coinciden con el esquema H en la generación t.
- $\overline{f}$ es el valor promedio de adaptación de la población de generación t.
- $\overline{f}(H)$ el valor de adaptación promedio de los representantes del esquema H;
- p_c la probabilidad de cruza, p_m la probabilidad de mutación por bit.
- $p_r = (1 - p_c f(H)/(l-1) - p_m O(H))$.
- $m(H,t+1)$ es el valor esperado del número de cadenas que coinciden con el esquema H en la generación $t+1$.

El teorema de esquema predice que cuando se tienen individuos con un orden alto y una longitud característica larga, el número de individuos que coinciden con el esquema H aumenta exponencialmente con el paso de las generaciones.

Al realizar un análisis más detallado sobre el comportamiento y efectividad del AGS se pueden destacar dos problemas principales:

1. Existen algunos problemas denominados *engañosos.*[Kuri02]
2. La correlación *spuria.* [Kuri02]

Básicamente de lo que trata el punto (1) es de la existencia de problemas, los cuales por la codificación de su dominio resulta que lejos de ayudar al AGS hallar la solución, tiende a *confundir* o extraviar dicha solución.

Esto sucede porque el esquema solución es de una característica muy específica que puede perderse fácilmente por la sucesiva aplicación de los operadores de Cruza y Mutación. En consecuencia el AGS no es capaz de encontrar la solución óptima.

Veamos un ejemplo: Supongamos que la función de adaptación para el problema que está resolviendo el AGS es caracterizado de la siguiente manera:

$$f(H) = \begin{cases} 2 & \text{si } 111\#\#\# \ldots \\ 1 & \text{si } 0\#\#\# \ldots \\ 0 & \text{en otro caso} \end{cases}$$

también supongamos que la población está constituida por individuos que están representados por los esquemas que están en la tabla 2.1.

Si observamos los valores de adaptación para cada esquema de la tabla 2.1, tenemos que: $f(H_1) = 2$, $f(H_2) = 1$, $f(H_3) = 0$, $f(H_4) = 0$ y $f(H_5) = 0$. Luego entonces,

No. Esq.	Esquema
H_1	$111\# \ldots \#$
H_2	$0\ \# \ldots \#$
H_3	$100\# \ldots \#$
H_4	$101\# \ldots \#$
H_5	$110\# \ldots \#$

Cuadro 2.1: Ejemplo de Problema Engañoso

si calculamos el valor promedio de adaptación de los esquemas de la forma $1\# \ldots \#$ tenemos que:

$$f_{prom}(1\# \ldots \#) = 1/2$$

que comparado con la adaptación promedio de los esquemas de la forma $0\# \ldots \#$:

$$f_{prom}(0\# \ldots \#) = 1$$

lo anterior implica que el AGS comenzará a propagar favorablemente a individuos de la forma del esquema H_2 que no son individuos cuya forma sea semejante a la solución óptima.

El punto (2) se refiere a que una vez que el AGS ha encontrado un representante con alta calificación de un esquema de orden alto. Éste último se propaga copiosamente y con rapidez en la población. Esto en consecuencia ocasiona que se disminuya la proporción de otros esquemas que son necesarios para la construcción de la solución correcta [MHF94], [Kuri02].

Es posible optar por otro modelo diferente para explicar el comportamiento de la población a lo largo del proceso realizado por un AG. Esto se logra por medio de las *Cadenas de Markov* (CM).

Las cadenas de Markov son procesos estocásticos de los cuales la probabilidad de que el proceso esté en el estado j en el tiempo t depende sólamente del estado i en el tiempo $t-1$. El *estado* de una población de un AG se refiere a todas las posibles poblaciones de tamaño n. Éstas pueden ser enumeradas en algún orden canónico indizado por i. Podemos representar la i-ésima de tales poblaciones con un vector $\vec{p}$ de longitud 2^l. El i-ésimo elemento de $\vec{p}$ es el número de apariciones de la cadena i en la población P_i. Bajo un AGS la población actual P_j depende sólo de la población en la generación previa. *Por ello el algoritmo genético puede ser modelado por una Cadena de Markov* [Kuri02].
A partir de modelar el comportamiento de la población tras la ejecución de un AG se tienen establecidos los siguientes resultados [Kuri02] [RUD94]:

Teorema 2.1 *El Algoritmo Genético Simple no converge al óptimo global.*

Teorema 2.2 *En una cadena de Markov homogénea, el tiempo de transición entre un estado inicial i y cualquier otro estado j es finito, independientemente de los estados i y j.*

En la siguiente sección, hablaremos de como podemos evitar las dificultades que muestra el AGS.

2.3. El Algoritmo Genético Idealizado. (AGI)

Intuitivamente podemos pensar en muchas modificaciones al AGS para evadir los problemas que se presentan, de tal manera que la ejecución de un AG realmente encuentre la solución óptima. El hecho notable para explicar el proceso seguido por el AG para encontrar la solución óptima se hace por medio de la Hipótesis de Bloques Constructores (HBC) [Mitch96]:

Hipótesis HBC: Los AG's parten de bloques de corta longitud y poca especificidad para ir integrando bloques más específicos de longitud mayor.

A partir de esta hipótesis se plantea cómo sería el arquetipo de AG. Éste es denominado simplemente como AG idealizado y trabaja de acuerdo con la HBC. A continuación se muestra el algoritmo AGI [Mitch96]:

ALGORITMO GENÉTICO IDEALIZADO:

1. En cada paso, elija una nueva cadena al azar con probabilidad uniforme en cada bit.
2. La primera vez que se encuentre una cadena que contenga uno o más de los esquemas deseados debe secuestrarse dicha cadena.
3. Cuando figure una cadena que contiene uno o más de los esquemas aún no descubiertos, instantáneamente debe cruzarse con la cadena secuestrada de forma tal que ésta contenga todos los esquemas deseados que se han descubierto hasta el momento.

Con lo discutido hasta aquí, estamos en posición de decir cuándo un AG se aproxima a un AGI:

- Las muestras son independientes. La población debe ser suficientemente grande, el proceso de selección debe ser suficientemente lento y la tasa de mutación lo suficientemente alta, de manera que ningún locus se mantenga en un valor fijo para una mayoría significativa de las cadenas de la población.
- Los esquemas deseados deben secuestrarse. La selección debe ser suficientemente fuerte para preservar los esquemas deseados que han sido descubiertos, pero también suficientemente lenta para evitar la correlación espuria en algunos esquemas altamente eficientes.
- El cruzamiento debe ser *instantáneo.* La taza de cruzamiento debe ser tal que el tiempo de cruzamiento que combine dos esquemas deseados sea pequeño con respecto al tiempo que toma descubrir dichos esquemas.

La justificación del segundo punto antes citado se deriva de la conclusión del siguiente teorema, el cual se deduce de modelar los AG's como Cadenas de Markov [Kuri02] [RUD94]:

Teorema 2.3 *El AGS que mantiene la mejor solución encontrada en el tiempo después de la selección, converge al óptimo global.*

2.4. El Modelo de Vasconcelos.

Sirviéndonos del AGI, podemos determinar cuando alguna variante de AG puede ser considerada como efectiva o mejor que el AGS; con este criterio podemos exponer un modelo del AG que ha resultado más efectivo comparado con el AGS.

Empezaremos por discutir algunas variantes en cuanto al control y selección de los individuos.

2.4.1. El Elitismo.

Se puede pensar, partiendo de la idea de la HBC, que si nosotros preservamos en la población al mejor individuo que se va manifestando en cada generación, sea posible garantizar alguna convergencia del AG, ya que sobre este individuo se empezarían a identificar los bloques que componen al individuo solución. De esta idea se puede desplegar una estrategia para poder controlar las soluciones que va encontrando el AG.

Así, podemos definir el *elitismo* para los AG's como la selección de los mejores individuos de cada generación que va encontrando el AG, y que son utilizados para localizar al mejor individuo final. En consecuencia se tienen identificadas dos variantes del modelo elitista. El *elitismo parcial* y *elitismo total.* Por elitismo parcial se entiende que en una población de tamaño n mantenemos una copia de los mejores $\tau < n$ individuos hasta la generación k. Por otra parte decimos que existe un elitismo total si mantenemos una copia de los mejores n individuos hasta la generación k [Kuri02].

En la figura 2.1 mostramos el caso del elitismo total. Notemos que en la generación k hemos elegido los mejores n individuos posibles del total de los nk individuos considerados hasta ese punto. Al hacer esto artificialmente forzamos a la población, orientándola hacia un conjunto focalizado de posibles soluciones.

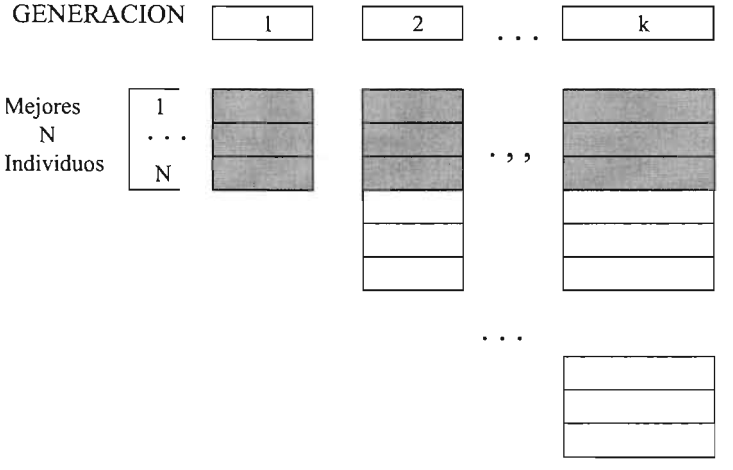


Figura 2.1: Elitismo Total

En este contexto, un hiperplano es el espacio de búsqueda de las posibles soluciones al problema. El riesgo que se corre es que restringimos el espacio de búsqueda de manera

que sesgue el AG excesivamente. Para evitar estos problemas modificaremos la forma de selección, para que no sea como en AGS (es decir una selección proporcional).

En consecuencia de lo anterior, al considerar un modelo elitista para el AG, debemos tomar en cuenta que hay que sacar ventaja de hacer una elección de este tipo. De manera que, como hemos realizado una selección premeditada de los individuos, también se considerará un operador de selección determinístico. En particular ([Kuri02]) se tienen contemplados dos modos de selección, ambos contrastantes; el primero es el de favorecer que los genes de los mejores individuos se crucen entre ellos mismos. En el otro favorecemos que los mejores individuos se crucen con los peores.

2.4.2. Definición del Modelo de Vasconcelos.

Con la ayuda del elitismo es posible definir el Modelo de Vasconcelos (MV), el cual ha sido considerado para el desarrollo de este trabajo de investigación.

El MV considera que los individuos son seleccionados bajo el criterio del elitismo total, se adopta la estrategia de seleccionar determinísticamente el individuo i para cruzarlo con el individuo $n-(i+1)$ determinísticamente, esto es, estamos forzando a que se crucen los mejores i individuos encontrados con los peores $n-(i+1)$ individuos de manera deliberada. Con esta estrategia inmediatamente salta a la vista el hecho de que la idea de seleccionar los individuos de tal manera va en contra de lo que dicta el AGI. Esto es porque tal estrategia destruye, de manera superficial, características deseables de los mejores individuos cruzándolos deliberadamente con los peores. Sin embargo, cuando se hace en conjunción con elitismo total, la estrategia conlleva al análisis implícito de una más amplia variedad de esquemas (es decir, maximiza la exploración del espacio de soluciones) [Kuri02]. La exploración de tal espacio se enfoca mediante el elitismo total implícito en el modelo (recordemos que tratamos de evitar el denominado problema de la correlación espuria). Se tienen documentadas ideas similares [Kuri02], proponiendo buscar la variedad haciendo un tipo de cruzamiento llamado de *recombinación destructiva.*

El MV permite al AG atrapar los mejores esquemas sin restringir el espacio de búsqueda. Ahora podemos finalmente comparar los requerimientos del AGI con el MV:

a) Ningún locus se debe fijar a un valor en un número grande de las cadenas de la población. El MV lo logra gracias a que determinísticamente estamos afectando a los posibles locus problemáticos con una operación de cruza; mejor contra peor individuo.

b) La selección debe ser suficientemente fuerte para que preserve a los esquemas deseables pero también debe evitar la correlación espuria en los esquemas altamente efectivos.

Debido a que trabajamos con elitismo, preservamos los esquemas deseados y, como antes, evitamos la correlación espuria al cruzar individuos disímiles.

c) La tasa de cruzamiento debe ser tal que el tiempo de cruza que combine a dos esquemas sea pequeño con respecto al tiempo que toma descubrir dichos esquemas.

El elitismo total garantiza que, aún en el caso del MV, el peor individuo de la población k está en el estrato que corresponde al $1/k$ porcentual. Por ejemplo, si $n = 50$ y observamos la vigésima población, los individuos en ésta están dentro del mejor 5 % del total de los individuos analizados. Esta característica es incrementalmente expuesta a medida que el AG evoluciona [Kuri02].

Como comprobaremos, este modelo es muy efectivo para hallar el óptimo global, que es el objetivo principal que se persigue con el AG.

3

Aproximación Polinomial Multivariada.

3.1. El objetivo.

Hasta este punto se han expuesto las bases que dan pauta al motivo de estudio de este trabajo. Ya fue expuesta la primera tecnología, las Redes Neuronales y un modelo particular, las Redes de Perceptrones Multicapa; y también una herramienta de optimización, los Algoritmos Genéticos.

En este capítulo enfocaremos nuestra atención en mostrar otra metodologíla cual en principio es capaz, de un modo restringido, realizar las tareas que puede desarrollar una RPM. Esta es denominada: *Aproximación Polinomial Multivariada* (en adelante APM). Dicha metodología es interesante pues también manifiesta propiedades de aprendizaje semejantes a la del RPM, pero con la ventaja de que el APM explicítamente devuelve un modelo del sistema bajo análisis.

3.2. El algoritmo de ascenso (FAA).

Como primera intención vamos a plantear el problema del que partiremos para ilustrar las ideas.Supongamos que los datos que caracterizan a un sistema bajo estudio están disponibles en forma de una tabla. En dicha tabla existe una columna por cada uno de los parámetros que caracterizan al sistema. Supongamos que dicha tabla contiene $1, \ldots, p+1$ columnas donde las columnas $1, \ldots, p$ representan variables independientes del sistema y la columna $p+1$ es una variable dependiente de las anteriores. Por ejemplo en la tabla 3.1 vemos que la tabla consta de 300 renglones ($n = 300$) y $p+1$ columnas (parámetros) donde p es el numero de variables independientes. En este caso $p = 4$.

Ahora bien nuestra intención es la de expresar una de las variables en función de las otras p. Por convención consideramos que la $p+1$-ésima variable es la variable dependiente

	ν_1	ν_2	ν_3	ν_4	$f(\nu_1, \nu_2, \nu_3, \nu_4)$
1	195,553	41,823	254,815	65,516	1,252,293
2	198,678	42,752	267,550	65,080	1,306,383
3	205,439	44,848	277,830	69,302	1,364,631
...	...	...	...	...	...
300	343,410	118,329	934,544	255,576	4,092,231

Cuadro 3.1: Tabla de valores característicos

y las otras p variables son independientes.

Elegimos a priori la forma del aproximante. En particular supondremos una forma polinomial completa :

$$f(\nu_1, \ldots, \nu_p) = \sum_{i_1=0}^{g_1} \cdots \sum_{i_p=0}^{g_p} C_{i_1,\ldots,i_p} \nu_1^{i_1}, \ldots, \nu_p^{i_p} \tag{3.1}$$

en donde g_i denota el máximo grado que puede tomar la i-ésima variable independiente. Finalmente se elige a priori, una métrica que nos permita definir la distancia entre el aproximante y los valores de la tabla.

3.3. Fundamentos matemáticos.

Generalmente, en los problemas de interés los sistemas están *sobredeterminados*. Es decir, existen más datos que parámetros. Por ello, los procesos de aproximación arrojan ajustes parciales y es necesario medir qué tan lejos estamos de la curva que representa la relación entre las variables independientes y la variable dependiente. En este caso la norma elegida es la norma L_∞ o *mínimax*. Es decir, deseamos encontrar los coeficientes $C_{0,\ldots,0}, C_{0,\ldots,1}, \ldots, C_{g_1,\ldots,g_p}$, tales que se minimice la diferencia:

$$\varepsilon_i = (f_{1\ldots p})_i - (\sum_{i_1=0}^{g_1} \cdots \sum_{i_p=0}^{g_p} C_{i_1,\ldots,i_p} \nu_1^{i_1}, \ldots, \nu_p^{i_p})_i \tag{3.2}$$

para $i = 1, \ldots, m$

3.3.1. Solución del conjunto M

Elegimos arbitrariamente m vectores de la tabla, en donde

$$m = (g_1 + 1)(g_2 + 1)....(g_p + 1) + 1$$

esto es:

$$m = 1 + \prod_{i=1}^{p} (g_i + 1) \tag{3.3}$$

De (3.2) podemos escribir:

$$(\sum C\overline{\nu})_i + \varepsilon_i = f_i \tag{3.4}$$

donde:

$$\begin{aligned} f_i &= (f_{1\ldots p})_i \\ C &= C_{i_1,\ldots,i_p} \qquad (3.5) \\ \overline{\nu} &= \nu_1^{i_1},\ldots,\nu_p^{i_p} \qquad (3.6) \\ (\sum C\overline{\nu})_i &= (\sum_{i_1=0}^{g_1}\cdots\sum_{i_p=0}^{g_p} C_{i_1,\ldots,i_p}(\nu_1^{i_1},\ldots,\nu_p^{i_p})_i \qquad (3.7) \end{aligned}$$

(3.8)

Sean $\varepsilon_\theta = \text{máx}\{|\varepsilon_1|,\ldots,|\varepsilon_m|\}$ y $\varepsilon_i = \eta_i\varepsilon_\theta$. Claramente [Kuri], $\eta_i \leq 1$. Es ε_θ lo que queremos minimizar. De (3.4) podemos inferir

$$(\sum C\overline{\nu})_i + \eta_i\varepsilon_\theta = f_i \tag{3.9}$$

Al escribir el sistema de (3.9) en forma matricial, tenemos:

$$\begin{bmatrix} \eta_1 & (\nu_1^0\ldots\nu_p^0)_1 & \ldots & (\nu_1^{g_1}\ldots\nu_p^{g_p})_1 \\ \vdots & \vdots & \vdots & \vdots \\ \eta_m & (\nu_1^0\ldots\nu_p^0)_m & \ldots & (\nu_1^{g_1}\ldots\nu_p^{g_p})_m \end{bmatrix} \begin{bmatrix} \varepsilon_\theta \\ C_{0\ldots0} \\ \vdots \\ C_{g_1,\ldots,g_2} \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix} \tag{3.10}$$

lo cual podemos escribir como

$$\mathbf{AC} = \mathbf{f}$$

De la regla de Cramer, el valor de ε_θ es

$$\varepsilon_\theta = \frac{\begin{vmatrix} f_1 & (\nu_1^0\ldots\nu_p^0)_1 & \ldots & (\nu_1^{g_1}\ldots\nu_p^{g_p})_1 \\ \vdots & \vdots & \vdots & \vdots \\ f_m & (\nu_1^0\ldots\nu_p^0)_m & \ldots & (\nu_1^{g_1}\ldots\nu_p^{g_p})_m \end{vmatrix}}{\begin{vmatrix} \eta_1 & (\nu_1^0\ldots\nu_p^0)_1 & \ldots & (\nu_1^{g_1}\ldots\nu_p^{g_p})_1 \\ \vdots & \vdots & \vdots & \vdots \\ \eta_m & (\nu_1^0\ldots\nu_p^0)_m & \ldots & (\nu_1^{g_1}\ldots\nu_p^{g_p})_m \end{vmatrix}} \tag{3.11}$$

Si denotamos el numerador de (3.11) con Δ_f y los cofactores de la primera columna del denominador por Δ_i podemos escribir la ecuación ec. (3.11) como

$$\varepsilon_\theta = \frac{\Delta_f}{\eta_1\Delta_1 + \ldots + \eta_m\Delta_m}$$

Sea $\sigma = \text{sgn}(x)$ el signo de x. Puesto que deseamos minimizar el valor absoluto de ε_θ debemos escoger los valores que maximicen el valor absoluto del denominador de ((3.11)).

Esto se logra: a) escogiendo el valor máximo absoluto de las η_i y b) Haciendo $\sigma_i = sgn(\Delta_i)$ o $\sigma_i \neq sgn(\Delta_i)$. Sabemos que máx$\{\eta_1, \ldots, \eta_m\} = 1$. Por lo tanto, el ajuste minimax para el conjunto implica que las ε_i's tienen el mismo valor absoluto y que los signos de tales ε_i son todos iguales (o todos diferentes) a los signos de las Δ_i's. Es decir se minimiza si

$$\varepsilon_\theta = \frac{\Delta_f}{\sigma_1\Delta_1 + \ldots + \sigma_m\Delta_m}$$

En los casos en los que hubiese una sola variable independiente ($p = 1$) las condiciones señaladas se logran trivialmente ordenando los vectores en orden estrictamente ascendente por el valor numérico de la variable independiente. Esto se sigue de que el determinante del denominador de ec. (3.11), para ese caso es un determinante de Van der Monde [Kuri02]. Sin embargo, es imposible lograr un ordenamiento de ese tipo si $p > 1$. Por tanto, la manera más sencilla de calcular las σ_i's es como sigue:

Teorema 3.1 *Si los elementos de una columna de un determinante se multiplican por el cofactor de una columna diferente y se suman, el resultado es cero.*

Si denotamos los elementos en el determinante del denominador en ec. (3.11) como $\delta_{11}, \ldots, \delta_{1m}$; $\delta_{21}, \ldots, \delta_{2m}$; ...; $\delta_{m1}, \ldots, \delta_{mm}$ podemos poner:

$$\begin{aligned} \Delta_1\delta_{12} + \Delta_2\delta_{22} + \ldots + \Delta_m\delta_{m2} &= 0 \\ \Delta_1\delta_{13} + \Delta_2\delta_{23} + \ldots + \Delta_m\delta_{m3} &= 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots &\ldots \ldots \\ \Delta_1\delta_{1,m} + \Delta_2\delta_{2,m} + \ldots + \Delta_m\delta_{m,m} &= 0 \end{aligned} \tag{3.12}$$

Este es un sistema de m incógnitas y solamente $m - 1$ condiciones. Podemos elegir, sin embargo, el valor de cualquiera de las Δ_i's (digamos Δ_m) arbitrariamente y resolver el sistema (3.12) para las $m - 1$ Δ_i's restantes. Para que esto sea posible debemos suponer que todas las Δ_i 's son no singulares. Cuando un sistema manifiesta esta independencia lineal, se dice que satisface la *condición de Haar.*

Si hacemos que $\Delta_m = -1 = \mathcal{C}_m$ el sistema (3.12) se convierte en

$$\begin{bmatrix} (\nu_1^0 \ldots \nu_p^0)_1 & \ldots & (\nu_1^{g_1} \ldots \nu_p^{g_p})_{m-1} \\ \vdots & \vdots & \vdots \\ (\nu_1^{g_1} \ldots \nu_p^{g_p})_1 & \ldots & (\nu_1^{g_1} \ldots \nu_p^{g_p})_{m-1} \end{bmatrix} \begin{bmatrix} \mathcal{C}_1 \\ \vdots \\ \mathcal{C}_{m-1} \end{bmatrix} = \begin{bmatrix} (\nu_1^0 \ldots \nu_p^0)_m \\ \vdots \\ (\nu_1^{g_1} \ldots \nu_p^{g_p})_m \end{bmatrix} \tag{3.13}$$

en donde $\mathcal{C}_i = k\Delta_i$. Es decir, no obtenemos el valor de los cofactores, sino el valor de los mismos multiplicado por una constante k, la cual puede tomar cualquier valor positivo o negativo distinto de cero. Las σ_i se obtienen, entonces de

$$\sigma_i = \begin{cases} sgn(\mathcal{C}_i) & i \neq m \\ sgn(\Delta_m) & i = m \end{cases}$$

Sin embargo, sin importar qué valor se asigna a Δ_m (y de cual k depende), las σ_i's permanecerán constantes dependiendo de σ_m. Que σ_m puede ser arbitraria se sigue del

hecho de que solamente es relevante el signo de $\mathcal{C}_i$, y no su magnitud y de que, por otra parte, el denominador de (3.3.1) puede maximizarse escogiendo ya sea $\sigma_i = sgn(\Delta_i)\forall i$ o $\sigma_i \neq sgn(\Delta_i)\forall i$.
Podemos entonces escribir la matriz $\mathbf{A}$ de (3.10) como

$$\mathbf{A} = \begin{bmatrix} \sigma_1 & (\nu_1^0 \dots \nu_p^0)_1 & \dots & (\nu_1^{g_1} \dots \nu_p^{g_p})_1 \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_m & \nu_1^0 \dots \nu_p^0)_m & \dots & (\nu_1^{g_1} \dots \nu_p^{g_p})_m \end{bmatrix} \tag{3.14}$$

Y los coeficientes resultantes de $\mathbf{C} = \mathbf{f}\mathbf{A}^{-1}$ son los coeficientes minimax que buscamos para un conjunto de m vectores.

3.4. Solución del sistema n

Lo que buscamos es el conjunto de coeficientes $\mathbf{C}$ que minimice ε_θ para los n elementos del conjunto original. En la discusión precedente, como ya se anotó, supusimos que los m vectores elegidos son arbitrarios. Para encontrar la ε_θ global, es decir, aquella que resuelva el ajuste minimax para los n vectores originales, planteamos un algoritmo iterativo de ascenso, análogo al algoritmo de Remes [Kuri02]. El algoritmo es el siguente:

ALGORITMO DE ASCENSO

1. Hacer $i \leftarrow 1$.
2. Elegir un conjunto de vectores arbitrarios M_i de tamaño m. A este conjunto lo llamamos el *conjunto interno*; a los vectores no incluidos en este conjunto los agruparemos en otro conjunto E_i que llamamos el *conjunto externo* y que es, claramente, de tamaño $n - m$.
3. Encontrar la matriz A_i de acuerdo con la ecuación 3.14.
4. Encontrar los coeficientes minimax $\mathbf{C_i}$ y el máximo error de ajuste $[\varepsilon_\theta]_i$ para el conjunto interno.
5. Calcular el máximo error de ajuste $[\varepsilon_\theta]_i$ para el conjunto externo. En donde

$$[\varepsilon_\phi] = \text{máx}\{|(f_{1,\dots,p}) - (\sum_{i_1=0}^{g_1} \dots \sum_{i_p=0}^{g_p} C_{i_1,\dots,i_p} \nu_1^{i_1}, \dots, \nu_p^{i_p})_j|\}$$

   donde $\overline{\nu} \in E_i$
6. Si $[\varepsilon_\theta]_i \geq [\varepsilon_\phi]_i$, fin del algoritmo;
7. Si $[\varepsilon_\theta]_i \leq [\varepsilon_\phi]_i$ Intercambia un vector en M_i por un vector en E_i tal que $[\varepsilon_\theta]_{i+1} \geq [\varepsilon_\phi]_i$.
8. Hacer $i \leftarrow i + 1$;

9. Ir al paso 3.

10. *Fin algoritmo de ascenso.*

Es claro que el algoritmo converge ya que, por definición tenemos que

$$i < \tau \Rightarrow |[\varepsilon_\theta]_i| < |[\varepsilon_\phi]_i|$$

en donde τ denota el índice del conjunto objetivo (M_τ). Esta noción puede formailizarse en el siguiente teorema.

Teorema 3.2 *(Teorema de Intercambio.)*
Sea $\{A^1, \ldots, A^{m+1}\}$ *un conjunto de vectores en el espacio* R^m *que satisface la condicion de Haar. Si* $\mathbf{0}$ *está en la envolvente convexa de* $\{A^1, \ldots, A^{m+1}\}$, *entonces hay un índice* $j \leq m$ *tal que esta condición se mantiene cuando* A^j *es reemplazado por* A^{m+1} *[Chen66].*

En el algoritmo de ascenso el paso crítico es el 7. Determinar si $|[\varepsilon_\theta]_{i+1}| > |[\varepsilon_\theta]_i|$ (la condición de intercambio o CI) puede lograrse directamente reemplazando una por una de las filas de la matriz $\mathbf{A}$ y calculando cada una de las Δ 's y sus signos de manera que $|[\epsilon_\theta]_{i+1}| > |[\epsilon_\theta]_i|$. Tomando en cuenta que la solución de un determinante tiene un costo de $O(m^3)$ flops, el costo promedio para determinar la CI es $\frac{m}{2}O(m^3) = O(m^4)$. Es decir, hay que ejecutar $O(m^4)$ flops por iteración. Si suponemos que el número de iteraciones crece de manera cuadrática con n, entonces el costo promedio del algoritmo de ascenso (C_{asc}) puede ser aproximado por:

$$C_{asc} = O(n^2 m^4) > O(m^6) \tag{3.15}$$

$$C_{asc} = km^6$$

Por ejemplo , si $m = 60$

$$C_{asc} \approx 60^6 \approx 6^6 \times 10^6$$
$$C_{asc} \approx 1{,}7 \times 10^{12}$$

Para lograr un sistema más eficiente, es posible usar técnicas numéricas y heurísticas que nos permitan disminuir C_{asc} hasta $O(km^2)$ [Kuri], [Kuri02].

3.5. Solución de Sistemas Singulares.

El método presentado en la sección anterior se basa en la obtención de un mapeo de cada uno de los vectores del problema original de $\mathbb{R}^p$ a $\mathbb{R}^m$ (en nuestro ejemplo cada vector consta de cuatro elementos, es decir $p = 4$)y en la posterior obtención de los σ_i para conformar $\mathbf{A}$. Del teorema del intercambio se sigue que cada uno de los conjuntos de vectores de $\mathbf{A}$ cumple con la condición de Haar. La pregunta interesante es: ¿Cuál es la posibilidad de que alguno de los conjuntos de vectores $\mathbf{A}$ no cumpla con esta condición?. Y

si este fuera el caso ¿Existe alguna forma de encontrar la solución a los llamados *sistemas singulares*?

Una vez conformada la matriz **A** cuyos vectores están en $\mathbb{R}^m$, ésta no satisface la condición de Haar si

$$A^i = KA^j; (i,j) \leq p; i \neq j; K \equiv \text{constante.}$$

ni cuando

$$A^j = KA^j; (i,j) \leq p; i \neq j; K \equiv \text{constante.}$$

Si podemos garantizar que ninguna de tales filas o columnas sean linealmente dependientes la condición de Haar puede mantenerse. Para lograr esto inducimos perturbaciones aleatorias en los vectores $\nu_i = [(\nu_1)_i, ..., (\nu_p)_i, (\nu_p + 1)_i]$ del conjunto original N haciendo

$$(V_1) = (\nu_1)_i(1 + \delta_i)$$
$$(V_2) = (v_2)_i(1 + \delta_{n-i})$$
$$\cdots$$
$$(V_{p+1}) = (v_{p+1})_i(1 + \delta_{np-i})$$

con $\delta_i = \delta_H \times \rho_i$

donde $\rho_i = rand()$ para $i = 1, \ldots, n(p+1)$ y

$$0 < \rho_i < 1$$

$$\delta_H \ll 1$$

En este esquema δ_H es constante, por lo tanto, cada vector ν_i en el conjunto original N es reemplazado por otro vector $V_i = [(V_1)_i, (V_2)_i, \ldots, (V_{p+1})_i]$ el cual exhibe un desplazamiento porcentual aleatorio relativo a ν_i que está acotado por δ_H. El conjunto resultante N^* satisface la condición de Haar para una δ_H apropiada aún si N no lo hace.

Una interpretación geométrica del método descrito consiste en visualizar cada uno de los puntos de R^m de manera que ningún subconjunto de ellos sea colineal en ninguna de las m dimensiones. Para evitar lo anterior desplazamos ligeramente cada punto de manera aleatoria. Tal desplazamiento aleatorio evita que se mantenga una relación lineal entre los vectores [Kuri].

Así, la sustitución de ν por V pretende resolver el problema de la dependencia lineal, como ya se mencionó. Sin embargo, esto es solamente un heurístico. En algunos casos no se logra la independencia lineal deseada de manera directa. Por esta razón, en la práctica se emplea el siguiente algoritmo[Kuri].

ALGORITMO DE CONDICIONAMIENTO

for i=1 **to** η

while true

leer los N datos originales.
Obtener N^* a partir de N
if El sistema es linealmente dependiente
$\delta_H \leftarrow \delta_H$.
if $\delta_H > \mu_\pi$
$\rho \leftarrow rand()$ $(0 < \rho < 1)$
$\Phi \leftarrow \Phi \times (1 + p)$
$\delta_H \leftarrow (\delta_H)_0 \times \Phi$
loop for
end if
else
exit for
end if

end while

end for

Se reciben cuatro parámetros de entrada:

a. El valor de factor de pertubación inicial (δ_H), en donde $\delta_H \ll 1$

b. La máxima perturbación aceptable (μ_π).

c. Un factor de ajuste (Φ).

d. El máximo número de ciclos de intento de estabilización (η).

El algoritmo empieza perturbando el conjunto de vectores y verifica que el sistema resultante haya cumplido con la condición de Haar. Si este no es el caso, aumenta el factor de perturbación por cada cantidad constante y obtiene el nuevo conjunto N^*.

Típicamente $(\delta_H)_0 = 10^{(-6)}$ y $\Phi = 2$. Esto significa que, en iteraciones sucesivas, $\delta_H = 2 \times 10^{(-6)}, \ldots$. Este proceso continúa mientras no se logre la estabilidad del sistema. Sin embargo, si el sistema excede un valor máximo de perturbación aceptable μ_π (por supuesto que no es deseable que los vectores se perturben mas allá de cierto punto) el algoritmo intenta un nuevo factor de ajuste cuyo valor es aleatorio (lo cual cierra un ciclo) y el proceso se repite. Este ciclo de repeticiones está acotado por η.

El algoritmo anterior es un heurístico que no garantiza que se logre la estabilidad deseada. Esto depende, básicamente de μ_π, pero en prácticamente todos los casos en que ha sido probado, ha logrado sistemas estables. Esto es muy interesante ya que, por la

forma del aproximante definida en 3.1 es claro que la dependencia lineal es altamente probable [Kuri93].

Sólo resta mencionar qué sucede cuando se realiza la aproximación del conjunto N sobre el nuevo conjunto N^*. En ese caso se tiene demostrado [Kuri] que la diferencia entre el error minimax obtenido del polinomio aproximador ε_θ para N y el error minimax ε_θ^* para el conjunto condicionado N^* es proporcional a δ_H. Dado que el valor de δ_H es por definición muy pequeño, el aproximante que se obtiene del conjunto perturbado aleatoriamente es muy cercano al aproximante obtenido de N.

3.6. Solución de sistemas completos.

Usando el método señalado en la sección anterior, es factible caracterizar un sistema de datos arbitrariamente obtenidos. Lo anterior es fundamentalmente importante porque, típicamente, se imponen ciertas demandas a la forma de los datos para lograr una expresión funcional, como la de (3.1). Cuando logramos encontrar dicha expresión, lo que estamos haciendo es sintetizar de manera muy compacta nuestro conocimiento del sistema. En ese sentido es que el método *aprende* de la experiencia pasada (los datos históricos). Como puede observarse el método de aproximación atiende a una selección a priori de la forma del aproximante y de la norma de aproximación. Aquí queremos consignar que el método descrito se puede aplicar sin modificaciones a otro tipo de aproximantes [Kuri].

En la exposición anterior hemos omitido un hecho práctico de fundamental importancia. En un aproximante de la forma de 3.1, el número de coeficientes crece exponencialmente con el número de variables independientes y los grados máximos estipulados. Por ejemplo, para tres variables y $g_1 = g_2 = g_3 = 2$ el número de coeficientes es $5^3 = 125$; para diez variables y $g_1 = g_2 = g_3 = 6$ el número de coeficientes es $7^{10} \approx 7,2832{,}475 \times 10^6$. Evidentemente, una solución cuya caracterización implica tal número de datos es inútil. Adicionalmente, la solución de los sistemas de ecuaciones de gran tamaño induce problemas de estabilidad numérica que invalidan los resultados obtenidos cuando los rangos dinámicos en los sistemas de la forma (3.10) son grandes. Para evitar el problema mencionado, reemplazamos el sistema de (3.1) por el siguiente:

$$f(\nu_1, \ldots, \nu_p) = \sum_{i_1=0}^{g_1} \cdots \sum_{i_p=0}^{g_p} \mu_{i_1,\ldots,i_p} C_{i_1,\ldots,i_p} \nu_1^{i_1}, \ldots, \nu_p^{i_p} \tag{3.16}$$

En donde las variables $\mu_{i_1,\ldots,i_p}$ solamente pueden tomar los valores 0 ó 1 y su propósito es excluir/incluir el coeficiente del cual son factor. Inducimos un nuevo parámetro en el problema: el numero máximo de coeficientes deseados (γ).

Es decir, ahora el problema consiste en:

a) Encontrar los γ coeficientes mas convenientes para minimizar ε_θ,

b) Encontrar los valores de los coeficientes elegidos.

Buscamos, ahora, dos vectores de solución: el vector μ y el vector $\mathbf{C}$. El vector μ tiene

$1 + \prod_{i=1}^{p}(g_i + 1)$ elementos; el vector $\mathbf{C}$ tiene γ elementos. Éstos son problemas altamente no lineales y **NP**-completos. Ambos pueden ser resueltos mediante un algoritmo genético como se describe a continuación.

3.6.1. Algoritmo genético de orden.

El problema que nos preocupa puede ser resuelto si establecemos un genoma en el cual observemos la siguiente convención:

a) El genoma tiene una longitud m igual a número de coeficientes de la expresión (3.16).

b) Cada uno de los bits del genoma corresponden a un coeficiente de (3.16). Los coeficientes se indizan de manera canónica de izquierda a derecha de manera que los índices corren desde 0 hasta m.

Por ejemplo, si $p = 3$ y $g_1 = 1$, $g_2 = 2$ y $g_3 = 2$, entonces en la siguiente tabla se muestra como quedaría constituido el genoma, el primer renglón corresponde a las potencias asociadas a cada coeficiente y el segundo a los índices de los coeficientes:

000	001	002	010	011	012	020	021	022	100	101	102	110	111
1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	0	0	0	0	1	0	1	0	0	0	0	1

112	120	121	122
15	16	17	18
0	1	0	1

c) Existen γ posiciones en las cuales el genoma toma el valor 1 y $m - \gamma$ posiciones en las cuales toma el valor 0. Al número de 1´s del genoma lo denominaremos el orden del genoma y será representado por ω.

d) Un 1 en el genoma significa que el coeficiente correspondiente existe ($\mu = 1$); por el contrario, un 0 en el genoma significa que el coeficiente no existe $\mu = 0$.

Para el caso que se muestra en la tabla anterior, el tecer renglón es un ejemplo de genoma que representa un aproximante de la siguiente forma:

$$f(\nu_1, \nu_2, \nu_3) = C_{000} + C_{001}\nu_3 + C_{020}\nu_2^2 + C_{022}\nu_2^2\nu_3^2 +$$
$$C_{101}\nu_1\nu_3 + C_{111}\nu_1\nu_2\nu_3 +$$
$$C_{122}\nu_1\nu_2^2\nu_3^2.$$

Sean $\Gamma = (g_1, g_2, \ldots, g_p)$ un conjunto de grados máximos de cardinalidad p y $f(\nu_1, \ldots, \nu_p)$ el aproximante minimax de 3.16. Como sabemos, la longitud del genoma está dada por

$\prod_{i=1}^{p}(g_i+1)$. Dado el índice de los coeficientes del genoma, es posible encontrar las potencias asociadas a los coeficientes de la siguiente relación [Kuri]:

$$\tau_i = \left\lfloor \frac{(N-1) - \sum_{j=i+1}^{p} \tau_j \prod_{k=1}^{j-1}(g_k+1)}{\prod_{j=0}^{i-1}(g_j+1)} \right\rfloor$$

en donde $a > b$ implica $\sum_{i=a}^{b} x \equiv 0$ y $\prod_{i=a}^{b} x \equiv 1$. Los términos de las potencias $(\tau_p, \tau_{p-1}, \ldots, \tau_1)$ corresponden a las posiciones $p, p-1, \ldots, 1$ de la potencia.

De manera análoga, dados los términos $\tau_p, \ldots, \tau_1$ es posible encontrar el índice correspondiente de la expresión

$$N = 1 + \sum_{i=1}^{p} \tau_i \prod_{j=0}^{i-1}(g_j+1)$$

Estos genomas son susceptibles de ser tratados genéticamente de manera que la función de ajuste sea el valor de error ε_θ y el proceso es un proceso de minimización de dicho error. En cada generación se derivan dos componentes para cada individuo:

a) El máximo error de ajuste (ε_θ)

b) El vector de coeficientes $\mathbf{C}$ para el algoritmo de ascenso.

De esta manera, la población evoluciona hacia la combinación de coeficientes que minimice ε_θ bajo la condición de que el número de 1´s en el genoma sea igual a γ. Es importante hacer notar que el algoritmo genético, en este caso, tiene una función objetivo en la cual, además de minimizar el error ε_θ, debe mantener un genoma cuyo número de 1´s sea siempre igual a γ $(\omega = \gamma)$. Es decir, los operadores genéticos de cruzamiento y mutación están limitados en el sentido de que éstos no dan origen a genomas válidos en todos los casos. Consecuentemente, es preciso adoptar una de las siguientes estrategias:

a) Establecer una medida de castigo para aquellos genomas resultantes de los operadores genéticos.

b) Modificarlos para que el orden del genoma se mantenga constante.

En este último caso, debe introducirse un algoritmo de reparación, de manera que los genomas que violen la restricción $\omega = \gamma$ sean reemplazados por otros que no la violen.

La primera estrategia es más general pero la segunda, obviamente es más eficiente. En este contexto, debemos preferir la segunda sobre la primera por razones de economía.

Como se ve, la aproximación polinomial multivariada no busca la solución del problema (el vector $\mathbf{C}$) de manera directa. La búsqueda de la forma de la solución (el mejor individuo

de orden γ), sin embargo, nos conduce a las solución del problema de fondo, que es encontrar el conjunto de coeficientes que mejor ajusten a los datos.

Para concluir, es importante destacar que la metodología descrita funciona debido, básicamente, al método de perturbación aleatoria. Sin ésta, las dependencias lineales que se presentan con alta probabilidad en los datos muestrales no podrian ser resueltas.

4

Implementación

Terminado el planteamiento del marco teórico, es necesario hablar de la implementación de los algoritmos que hemos discutido, claro está, que van a ser las herramientas con las que vamos a trabajar.

En este caso se seleccionó como simulador de las RPM's el software *DataEngine* Versión 2.10.012 [DE97], que nos permite diseñar este tipo de redes de manera sencilla y supervisar el entrenamiento de la misma. La discución con respecto a la arquitectura, métodos de entrenamiento y demás detalles se presentan en la sección 5.1.

4.1. Implementación del APM

Para la implementación de los algoritmos del APM utilizamos el lenguaje de programación Matlab versión 6.5, ya que ofrece múltiples facilidades para el desarrollo de aplicaciones de cálculo técnico, incluyendo un ambiente de graficación sencillo.

Se planearon dos fases del desarrollo, en la primera se desarrolló la implementación del Algoritmo de Ascenso Rápido (FAA) y la heurística denominada Algoritmo de Condicionamiento. Mientras que en la segunda etapa se implementaron los algoritmos evolutivos, básicamente el Algoritmo Genético de Orden.

En los siguientes listados presentamos las implementaciones correspondientes a la primera etapa.

Implementación del Algoritmo FAA

```
function M = aproximacionTemporal(tabla, tablaFuncion, tablaPotencia, exponentes)

[rengT, colmT] = size(tabla);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Parametros de la Perturbacion    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
deltaI = 0.0000001;
muPi = 0.5;
phi = 2;
eta = 10;

p = length(tablaPotencia);
m = 0;
Producto = 1;
eleccionInc = zeros(1, m);
epsilonTheta = 0;
epsilonPhi = 0;
maxEpsilonPhi = 0;
sigmaPhi = 0;
indiceCambio = 0;
for i=1:p
    Producto = Producto*(tablaPotencia(i) + 1);
end
m = Producto;
A = zeros(m+1,m);
datosGral = zeros(rengT, colmT + 1);
conjInt = zeros(m+1, colmT);
conjExt = zeros(rengT-(m+1), colmT);
conjIntFun = zeros(m+1,1);
conjExtFun = zeros(rengT-(m+1),1);
sigma = ones(m+1,1)*(-1);
vectorMaxExt = zeros(1, colmT);


datosGral = gralDatos(tabla, tablaFuncion,
            tablaPotencia, exponentes, deltaI, muPi, phi, eta);


conjInt = datosGral(1:m+1, 1:colmT);
conjExt = datosGral(m+2:rengT, 1:colmT);
conjIntFuncion = datosGral(1:m+1, colmT+1);
conjExtFuncion = datosGral(m+2:rengT, colmT+1);


A = calculaMatrizM(conjInt, exponentes);


S1 = A(m+1, :);
X = S1';

aReducida = A(1:m,:);
fprintf('A reducida %d \n', rank(aReducida));
S = aReducida\X;

for i=1:m
    if S(i,1) >= 0
```

```
        sigma(i,1) = 1;
    else
        sigma(i,1) = -1;
    end
end
sigma;
solucionA = [sigma, A];
rank(A)

inversaB = inv(solucionA);
matrixExt = calculaMatrizM(conjExt, exponentes);
%Coeficientes y error
c_C = ones(1, m+1);
coeficientes_C = ones(m,1);
size(matrixExt);
%%%%Aqui es el ciclo de aproximacion%%%%%
valoresFuncion = conjIntFuncion;
while 1 == 1
    %Asigna los valores de los coeficientes de los
    %polinomios y cacula el error.
    c_C = valoresFuncion' * inversaB;
    epsilonTheta = c_C(1);
    coeficientes_C = c_C(2:m+1)';
    vector_epsilonPhi =  conjExtFuncion - matrixExt*coeficientes_C;
    [epsilonPhi, posicionEpsPhi] = max(vector_epsilonPhi);
    epsilonPhi
    epsilonTheta
    if abs(epsilonPhi) <= abs(epsilonTheta)
        fprintf('sali del ciclo \n');
        break;
    else
        if (epsilonPhi == 0)
            sigmaPhi = 1;
        else
            sigmaPhi = abs(epsilonPhi)/epsilonPhi;
        end
    end
    lambdaJ = matrixExt(posicionEpsPhi,:)*inversaB(2:(m+1), :) + inversaB(1,:);
    [maxBeta, posicionMaxBeta] = max((sigmaPhi*lambdaJ)./inversaB(1,:));
    KKK = inversaB;
    for k=1:(m+1)
        inversaB(k,posicionMaxBeta)= inversaB(k, posicionMaxBeta)/lambdaJ(posicionMaxBeta);
    end
    for k=1:m+1
        for j=1:m+1
            if(posicionMaxBeta ~= j)
                inversaB(k,j) = inversaB(k,j) -
                                (lambdaJ(j) * inversaB(k, posicionMaxBeta));
            end
        end
    end
    posicionEpsPhi;
    valoresFuncion(posicionMaxBeta, 1) = conjExtFuncion(posicionEpsPhi, 1);
```

Implementación del Algoritmo de Condicionamiento.

```
function R = gralDatos(Datos, datosFun, potencias, exponentes,
                       perturbacionDeltaH, maxPerturbacionMu,
                       ajustePhi, estabilizacionEta)

c = clock;
k = c(2)*c(3)*c(4)*c(5)*c(6);
rand('seed', k);

[datosR, datosC] = size(Datos);
m = 1;
fuga = 0;
rango = 0;
deltaH = perturbacionDeltaH;
Phi = ajustePhi;
m = potencias;
eleccionM = zeros(datosR, datosC);
salidaN = EligeVectores(Datos, datosFun, m);

salidaN = [Datos, datosFun];
salidaNTmp = PerturbaDatos(salidaN(:, 1:datosC), deltaH);
eleccionM = EligeVectores(salidaNTmp(:, 1:datosC), salidaN(:, datosC+1), m);
Phi = ajustePhi;
for i=1:estabilizacionEta
    while (1 == 1)
        rango = MatrizHaar(eleccionM(1:m+1, 1:datosC), m+1, exponentes);
        if(rango == 0)
            deltaH = deltaH*Phi;
            if deltaH > maxPerturbacionMu
                Phi = ajustePhi * (1 + rand(1));
                deltaH = perturbacionDeltaH * Phi;
                fuga = 1;
                break;
            end
            salidaNTmp = PerturbaDatos(salidaN(:, 1:datosC), deltaH);
            eleccionMTmp = EligeVectores(salidaNTmp(:, 1:datosC), salidaN(:, datosC+1), m);
            eleccion0 = sub_Set_Initial_Data(eleccionMTmp(1:m+1, :));
            eleccionM = [eleccion0; eleccionMTmp(m+2:datosR, :)];
            fuga = 0;
        else
            fuga = 2;
            break;
        end
    end
    if (fuga == 1)
        if i >= estabilizacionEta
            R = eleccionM;
        end
        fuga = 0;
```

```
    elseif(fuga == 2)
        R = eleccionM;
        break;
    else
        R = eleccionM;
        fuga = 0;
        break;
    end
end
```

Con las implementaciones del Algoritmo de Ascenso y la heurística de Condicionamiento de Datos es posible mostrar las capacidades de la combinación de ambos algoritmos. En la figura 4.1 se presenta el resultado de aplicar ambas implementaciones para obtener un polinomio minimax para las variables x y y, cuyo grado máximo es tres. Tal polinomio aproxima la función

$$F(x, y) = sin(x) + cos(y)$$

definida en el intervalo $[-2, 2] \times [-2, 2]$.

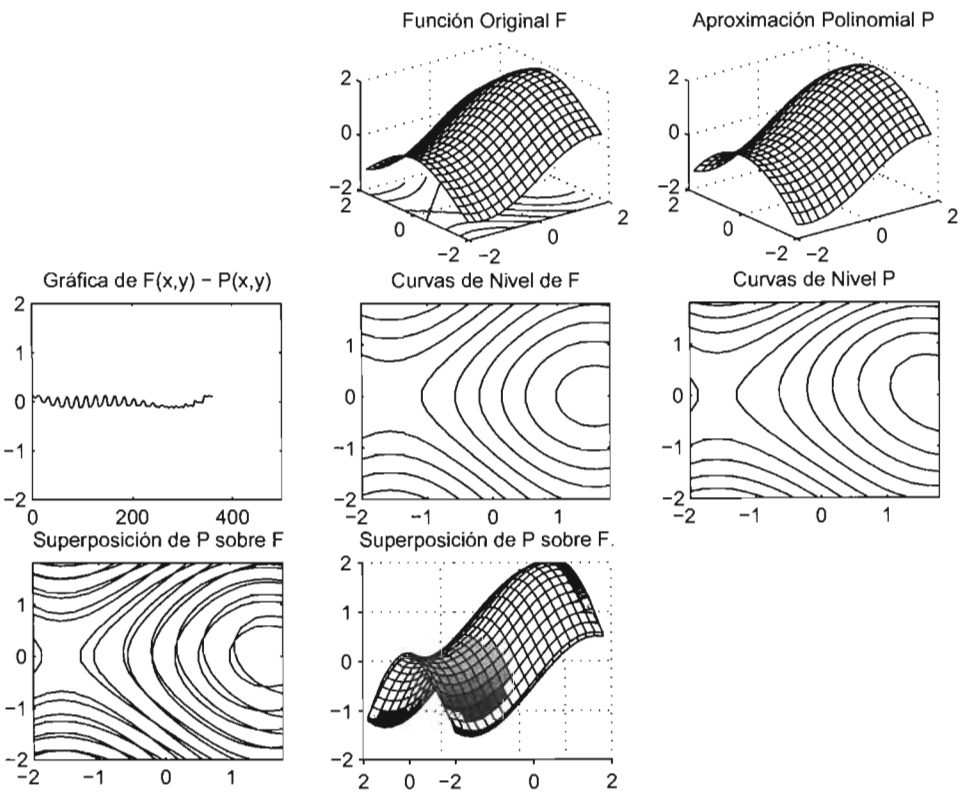


Figura 4.1: Aproximación realizada por el Algortimos de Ascenso y Condicionamiento

Seguimos ahora con la implementación de una heurística denominada *set Initial Data*, la cual se ocupa de acelerar el tiempo de ejecución del Algoritmo de Ascenso. Sobre esta heurística hablaremos en la sección 5.2.

Set Initial Data.

```
function N = set_Initial_Data(data, num_M)
%Calcula de manera distribuida los datos
%para elegir conrespecto a la distancia euclidiana
%los M vectores iniciales.....................

[rID, cID] = size(data);
nulo = zeros(1,cID);
for i=1:cID
    nulo(i) = inf;
end
dist_Origen = zeros(1,rID);
min_Vector = zeros(1,cID);
posicionVectores = ones(1,rID);
distanciasOrdenadas = ones(2, rID);
distanciasVect = zeros(1,rID);
indexSeleccion = zeros(1, num_M);

for i = 1:rID
    dist_Origen(i) = norm(data(i, :));
end

[mm, posicionMin] = min(dist_Origen);
min_Vector = data(posicionMin, :);
for i=1:rID
    distanciasVect(i) = norm(data(i,:)- min_Vector);
end

posicionDistancias = 1:rID;
distanciasOrdenadasTmp  = sortrows([distanciasVect', posicionDistancias'], 1)';
distanciasOrdenadas = distanciasOrdenadasTmp(1, :);
posicionVectores = distanciasOrdenadasTmp(2, :);
%selecciono vectores

paso = ceil(rID/(num_M-1));
indexSeleccion(1) = posicionVectores(1);
indexSeleccion(num_M) = posicionVectores(length(posicionVectores));
xj = 1 + paso;
for i = 2:num_M-1
    j = xj;
   indexSeleccion(i) = posicionVectores(xj);
   xj = xj + paso;
end
sizeIndexSeleccion = length(indexSeleccion);
%indexSeleccion;
interior = ones(sizeIndexSeleccion, cID);
exterior = ones(rID-sizeIndexSeleccion, cID);
%size(interior)
```

```
%size(exterior)


indiceElegido = 1;
indexExt = 1;

for i=1:sizeIndexSeleccion
    interior(i,:) = data(indexSeleccion(i), :);
    data(indexSeleccion(i), :) = nulo;
end
cont = 1;
for i=1:rID
    if data(i,:) < nulo
        exterior(cont,:) = data(i,:);
        cont = cont + 1;
    end
end
%interior;
%exterior;
N = [interior; exterior];
```

Continuando con el desarrollo de los algoritmos presentamos las implementaciones de la segunda etapa, la cual consiste de la implementación de dos programas: el Algoritmo Genético Orden y la función GAFAA. Esta última contiene la implementación de la interacción de todos los algoritmos.

Operador de Reparación $\Omega\Pi$

Esta función se encarga de restaurar el equilibrio de los genomas para el Algoritmo de Ascenso después de aplicar los operadores genéticos de Cruza y Mutación. La variable *tablasGenomas* contiene los genomas de la población, la variable *gamma* contiene el número de 1's que se desean de antemano en el genoma [Kuri].

```
function genomaRestaurado = Operador_ReparacionOmegaPi(tablaGenomas, gamma)

%Variables de soporte para el algoritmo
[numRenGen, numColGen] = size(tablaGenomas);
ordenOmega = zeros(1,numRenGen);
s = -1;
t = -1;

%Variables auxiliares
conteo = 0;
c = clock;
k = c(2)*c(3)*c(4)*c(5)*c(6);
rand('seed', k);

for i=1:numRenGen
    ordenOmega(i) = contador_Omega(tablaGenomas(i,:));
end

for i=1:numRenGen
```

```
    if ordenOmega(i) ~= gamma
        if(ordenOmega(i) > gamma)
            s = 1;
            t = 0;
        else
            s = 0;
            t = 1;
        end
        while(1 == 1)
            pivote = rand(1);
            posicionK = ceil(numColGen * pivote);
            if(posicionK == 0)
                posicionK = 1;
            end
            contadorPosicion = posicionK;
            while(contadorPosicion <= numColGen)
                if(tablaGenomas(i, contadorPosicion) == s)
                    tablaGenomas(i, contadorPosicion) = t;
                    break;
                end
                contadorPosicion = contadorPosicion + 1;
                if((contadorPosicion + 1) > numColGen)
                    contadorPosicion = 1;
                end

            end
            if(contador_Omega(tablaGenomas(i,:)) == gamma)
                break;
            end
        end
    end
end
genomaRestaurado = tablaGenomas;
```

Es importante mostrar la implementación de las fórmulas para calcular la posición en el genoma con respecto a los exponentes del monomio dado y su análogo, la función que calcula los exponentes correspondientes con respecto a su posición en el genoma. En seguida se encuentra el código que se implementó.

Algoritmo Calcula Posición

```
function N = CalculaPosicion(g, tau)

producto = 1;
suma = 0;
tamPotencias = length(g);
tamExponentes = length(tau);

for j=1:tamPotencias
    for k=j+1:tamExponentes
        producto = producto*(g(k) + 1);
```

```
    end
    suma = suma + (tau(j) * producto);
    producto = 1;
end
N = suma + 1;
```

Algoritmo Calcula Potencia Asociada

```
function P = CalculaPotencias(N, g)

p = length(g);
tau = zeros(1,p);
i = 0;
j = 0;
k = 0;
jj = 0;
productoDenominador = 1;
suma = 0;
producto = 1;

for t=1:p
    i = t;
    %calcula Suma
    j = 1;
    if(j > i-1)
        suma = 0;
    else
        suma = 0;
        for j=1:i-1
            %calcula producto
            k = j+1;
            if (j+1) > p
                producto = 1;
            else
                producto = 1;

                for k = j+1:p
                    producto = producto * (g(k) + 1);
                end
            end
            suma = suma + (tau(j) * producto);
        end

    end
    %calcula productoCociente
    jj = i+1;
    if jj > p
        productoCociente = 1;
    else
        productoCociente = 1;
        for jj=i+1:p
            productoCociente = productoCociente *(g(jj) + 1);
        end
    end
```

```
    %calcualo final
    tau(t) = floor(((N-1) - suma)/ productoCociente);
end
P = tau;
```

Para concluir, se presenta la implementación del Aproximador Polinomial Multivariado, el cual incluye una gran variedad de funciones auxiliares tanto para el Algoritmo Genético, como para ajustar el funcionamiento del Algoritmo de Ascenso. Cabe notar que en realidad no existe una restricción, al menos desde el punto de vista teórico, para limitar el número de monomios que debe componer nuestra aproximación. Eso es una variable que debemos ajustar, según los resultados que se vayan obteniendo.

Implementación GAFAA

```
function [varargout] = GAFAA(varargin)
NIND = varargin{1};
MAXGEN = varargin{2};
GGAP= varargin{3};
PRECI = 1;
Pc = varargin{4};
Pm = varargin{5};
DATAFUN = varargin{6};
MAXCOEFICIENTES = varargin{7};
GRADOMAX = varargin{8};
for i=1:length(GRADOMAX)
    PRECI = PRECI * (GRADOMAX(i)+1);
end
BESTCOEFICIENTES = ones(MAXGEN,MAXCOEFICIENTES);
arregloGenes = ones(MAXGEN, PRECI);
arregloError = ones(MAXGEN,1);
Chrom = crtbp(NIND, PRECI);
fprintf('Procesando : %d, %d \n', size(Chrom));
Chrom = Operador_ReparacionOmegaPi(Chrom, MAXCOEFICIENTES);
%incio contador
gen = 0;                        % contador de generaciones
[dataFunR, dataFunC] = size(DATAFUN);

evalCoeficientes = evaluaFAA(Chrom, GRADOMAX, MAXCOEFICIENTES,
DATAFUN(:, 1 :dataFunC-1),DATAFUN(:, dataFunC));
ObjV =  evalCoeficientes(:,1);
[Min, pos] = min(ObjV');
BESTCOEFICIENTES = evalCoeficientes(pos, 2:MAXCOEFICIENTES+1);
MinFinal = Min;
genCoeficientes = Chrom(pos, :);

while gen < MAXGEN,
    fprintf('Generacion: %d ', gen+1);
    FitnV = ranking(ObjV,[2, 1]);
    SelCh = selectMV(Chrom, FitnV, GGAP, indexSelect);
    SelCh = recombin('xovdp',SelCh, Pc);
    SelCh = mutaSelc(SelCh, Pm);
    Chrom = integra(Chrom,SelCh);
```

```
    Chrom = Operador_ReparacionOmegaPi(Chrom, MAXCOEFICIENTES);
    evalCoeficientes = evaluaFAA(Chrom, GRADOMAX, MAXCOEFICIENTES, DATAFUN(:, 1:dataFunC-1),DATAFUN(:,
    ObjV =  evalCoeficientes(:,1);
    [Min, pos] = min(abs(ObjV'));
    [Max, posMax] = max(abs(ObjV'));
    BESTCOEFICIENTES(gen+1, :) = evalCoeficientes(pos, 2:MAXCOEFICIENTES+1);
    arregloGenes(gen+1, :) = Chrom(pos, :);
    arregloError(gen+1, :) = evalCoeficientes(pos, 1);
    [minError, posError] = min(arregloError);
    Chrom(posMax,:) = arregloGenes(posError,:);
    ObjV(posMax, :) = minError;
    fprintf(' MinError ---> %e \n', arregloError(gen+1, :));
    gen = gen + 1;
end

[minError, posError] = min(arregloError);
fprintf('<------------------- RESULTADO FINAL ---------------------->\n');
fprintf('Error final ----> %e \n', minError);
fprintf('[');
for ww = 1:length(arregloGenes(posError,:))
    fprintf('%d,', arregloGenes(posError, ww));
end
fprintf('] \n');
impresionPotencias = calculaPotenciasChrom(arregloGenes(posError, :), GRADOMAX, MAXCOEFICIENTES);
fprintf('Coeficientes \t Potencias \n');
for ww = 1:length(impresionPotencias)
    fprintf('%g \t ', BESTCOEFICIENTES(posError,ww));
    for jw= 1:length(GRADOMAX)
        fprintf('%d ', impresionPotencias(ww,jw));
    end
    fprintf('\n');
end

errorRMS = DATAFUN(:,
dataFunC)-calculaPolinomioResultante(BESTCOEFICIENTES(posError,:),
DATAFUN(:, 1 :dataFunC-1),
MAXCOEFICIENTES, GRADOMAX,arregloGenes(posError,:));
RMS = sqrt(sum(errorRMS.^2)/dataFunR);
fprintf('Error RMS: %g \n', RMS);
dibuja(DATAFUN, BESTCOEFICIENTES(posError, :), impresionPotencias);
varargout = {RMS,minError, BESTCOEFICIENTES(posError, :), arregloGenes(posError, :)};


function d = dibuja(dataFuncion, coefPolinomio, potencias)
[rD, cD] = size(dataFuncion);
matrizData = calculaMatrizM(dataFuncion(:,1:cD-1), potencias);

Poly = matrizData*coefPolinomio';
cla;
clf;
base = 1:1:rD;
subplot(1,3,1)
plot(base, dataFuncion(:, cD)','.');
```

```
xlabel('x'); ylabel('F(x)');
subplot(1,3,2)
plot(base, Poly','-');
xlabel('x'); ylabel('P(x)');
subplot(1,3,3)
plot(base, dataFuncion(:, cD)', ':');
hold on
plot(base, Poly', '-');
hold off
```

5

Experimentando con APM y RPM

5.1. Planteamiento.

Hasta este punto se han expuesto los argumentos teóricos que conforman las metodologías de Redes Neuronales y el Aproximador Polinomial Multivariado (APM). Ahora bien, como el próposito de este trabajo es realizar una comparación entre el APM y las Redes Neuronales, es necesario comenzar por determinar nuestro punto de comparación.

En el capítulo 3, describimos la forma de trabajar del APM, y se puede observar que es funcional, pues dado un conjunto de datos dispuestos en una tabla, establecemos una de las columnas de la tabla en función de las restantes.

Una vez proporcionados los datos del sistema bajo estudio en una tabla de $p+1$ columnas y q renglones, definimos a priori que los datos correspondiente a la $p+1$-ésima columna están en función de las p columnas restantes, respectivamente a cada reglón. Además, por cada renglón definimos dos puntos $\bar{x}_i = (x_1, \ldots, x_p) \in \mathbb{R}^n$ y $y_i \in \mathbb{R}$, de tal manera que el APM utiliza el Algoritmo de Ascenso para encontrar un polinomio que aproxime a cada punto $\bar{x}_i$ al punto y_i, respectivamente. De tal manera que el APM obtiene una expresión polinomial predeterminada a través de un Algoritmo Genético.

Por otro lado, por el Teorema de Aproximador Universal (sección 1.5), una RPM es capaz de aproximar una función f, definida en el intervalo $[0,1]^n$, a un punto objetivo y. En otras palabras, dados los conjuntos de puntos X y Y, definidos como:

$$X = \{(x_1, \ldots, x_n) \mid x_i \in [0,1],\ 1 \leq i \leq n\}$$

y

$$Y = \{y \mid y \in [0,1]\}$$

Donde a cada punto $\bar{x}_j \in X$ le corresponde un sólo punto $y_j \in Y$. La RPM aproxima

cada punto $\bar{x}_i$ al valor y_j correspondiente. Es decir,

$$f_{RN}(\bar{x}_j) = y_j$$

Observemos que en general, tanto el APM como la RPM son capaces de resolver problemas de aproximación, teniendo un cierto rango de error ε. Así que en principio es razonable pensar en una comparación de ambos métodos. A continuación, enunciaremos los puntos que deseamos cotejar:

- Nos interesa saber si son capaces de resolver los mismos problemas, además de las discrepancias que existen entre ellos.
- Si observamos que resuelven los mismos problemas, habrá alguna relación entre ellos.
- También estamos interesados en las ventajas o desventajas que presenta cada uno de los métodos frente al otro.

Una vez definidas las cuestiones medulares que nos interesa analizar, el siguiente paso es establecer los criterios con los cuales realizaremos los experimentos para llevar a cabo la comparación.

1. Para realizar las pruebas elegimos una suite de funciones: 14 funciones con restricción, esto con el propósito de saber que tan bien generalizan. Además se utilizaron dos funciones continuas y se empleó una tabla de datos históricos sobre PIB de México y cuatro sectores que lo integran desde 1960 a 1992.
2. Ambas técnicas están definidas en dominios diferentes, por tanto todas las muestras obtenidas de los problemas que se utilizaron para los experimentos, fueron proyectadas al intervalo $[0, 1]$ o $[-1, 1]$, dependiendo bajo cual de estos dominios el RPM manifestó un mejor entrenamiento.
3. La arquitectura de la RPM que empleamos es la siguiente:
   - Una capa de entrada (el número de entradas queda determinado por el problema en cuestión)
   - Una única capa oculta, el número de neuronas es determinado en principio por la siguiente heurística:
     $$H = \frac{S - 3O}{3(I + O + 1)}$$
     donde H representa el número de neuronas ocultas, S es el tamaño de la muestra de entrenamiento, O el número de neuronas de salida e I el número de entradas. A partir de este valor se fueron ajustando el número de neuronas según los resultados de error del entrenamiento.
   - Una neurona en la capa de salida.
   - La función de activación; para la capa oculta se utilizó la función tangente hiperbólica y para la capa de salida la función logística.

- El algoritmo de entrenamiento que se empleó fue el algoritmo de retropropagación combinado con moméntum. La tasa de aprendizaje establecida variaba entre 0.9 y 0.8, mientras que la tasa momentum va de 0.1 a 0.2.

4. Para el caso del APM se tomaron los siguientes criterios:

   a) El número de coeficientes que integran el polinomio aproximador están considerados en proporción al número de pesos que conforman el APM.

   b) Los grados para el polinomio aproximador son considerados a prueba y error[1], en promedio no fueron ocupados grados máximos mayores a 8.

Después de describir las condiciones anteriores y la arquitectura de la red que utilizamos para realizar los experimentos, en la siguiente sección presentamos los pasos que se siguieron para el desarrollo.

5.2. Desarrollo y Resultados

Se realizaron dos experimentos para comparar RPM y APM. En el primero se utilizaron las funciones listadas en la tabla 5.1.

Para el segundo experimento se utilizaron datos del PIB de México, a partir del año 1960 y hasta 1993 tomando cuatro sectores productivos del país los cuales lo integran. Los datos se presentan en la tabla 5.2.

En una primera etapa se realizó la implementación del Algoritmo de Ascenso, así como su complemento, el Algoritmo Genético de Orden (bajo el esquema de Vasconcelos). Ambos fueron probados para verificar su efectividad. Además, también se implementaron heurísticas para la aceleración del proceso:

- El algoritmo para la selección del conjunto de datos iniciales (Initial Data Set Algorithm), esta heurística pretende ordenar los puntos generados por los datos de entrada con respecto a sus distancias, con el propósito de que los puntos que conforman el conjunto interior inicial sean aquellos que muestran un error ε_θ máximo (Capítulo 3) [Kuri].

- Se optó por utilizar un criterio más flexible de paro para el Algoritmo de Ascenso. Según se dijo en el capítulo 3, el Algoritmo de Ascenso termina cuando $|\varepsilon_\theta| \geq |\varepsilon_\phi|$. En general sucede que ε_ϕ se va decrementando de manera no monótona, mientras que ε_θ crece monótonamente. En consecuencia, el tiempo para alcanzar la condición de paro original se puede tornar lento. Para reparar este defecto se considera primero tomar el valor $\xi = 1 - |\varepsilon_\theta/\varepsilon_\phi|$, y un valor κ que sirva como *zona de aproximación* con un ancho de $\kappa = 0.05$. Así el criterio de paro queda establecido como $\xi \leq \kappa$. Acelerando con ello la ejecución del algoritmo en una proporción de 20-25 % [Kuri].

[1] Lamentablemente no existe ningún criterio para definirlo

- No puede faltar la heurística para la perturbación de los datos, cuya importancia es crucial, según se explicó en el sección 3.5.

Una vez implementadas estas heurísticas, se procedió a obtener los datos de entrenamiento y prueba para cada una de las funciones propuestas. Los puntos fueron generados de manera aleatoria. Se hizo la implementación de las funciones, después se generaron proximadamente mil puntos los cuales fueron probados por las funciones, y sólo se preservaron aquellos que cumplían con las restricciones, en el caso particular de los datos stadísticos simplemente se proyectaron al intervalo $[0, 1]$.

En las paginas siguientes se muestran gráficamente los resultados. Las gráficas muestran para cada punto en el eje x (el cual corresponde a un punto de la tabla de valores de ntrenamiento), cual es el valor de salida para la RPM, el APM y el valor de la función e entrenamiento, con lo cual podemos observar que tanto se parece la salida de ambas.

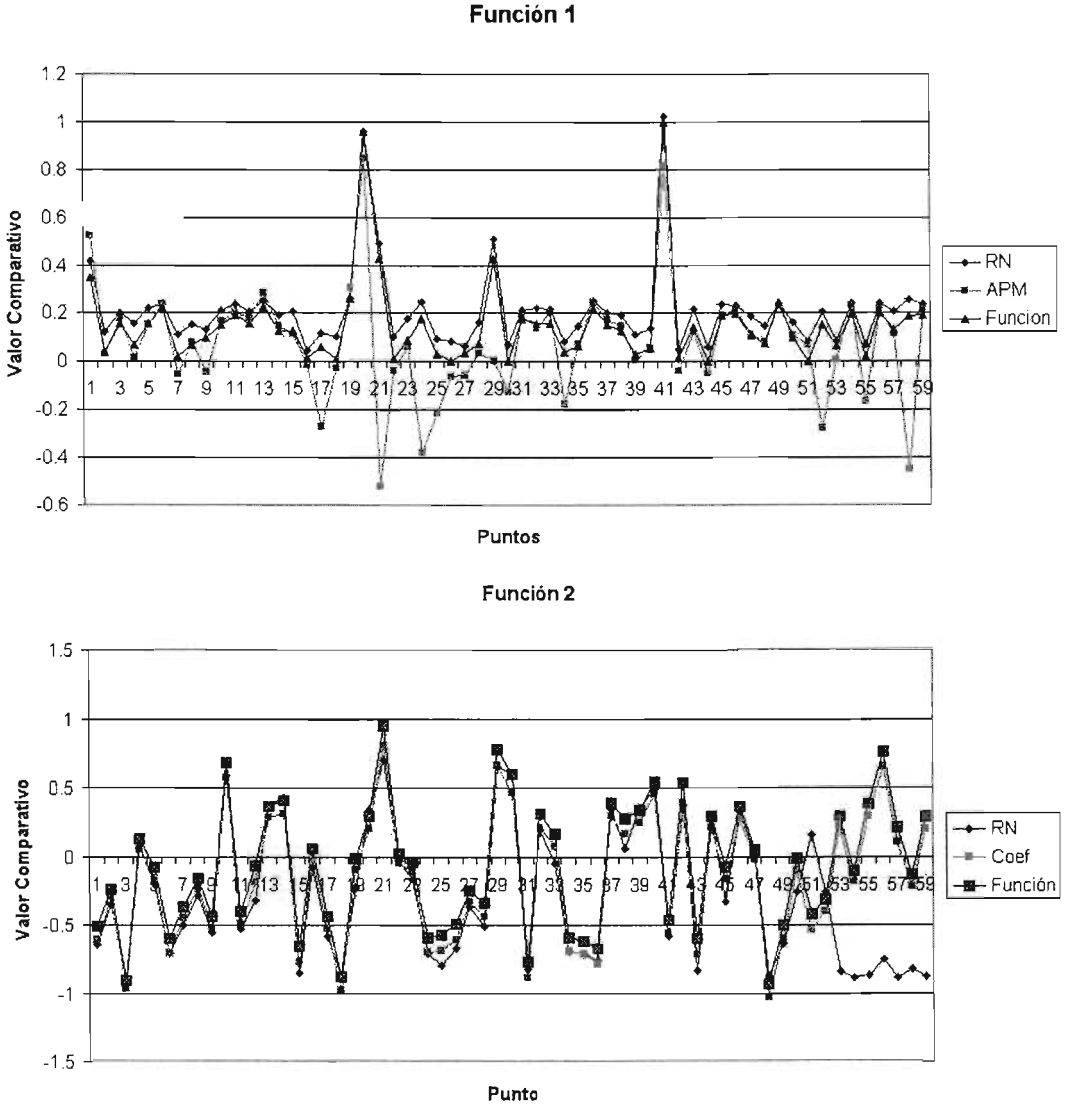


Figura 5.1: Resultados Función 1-2

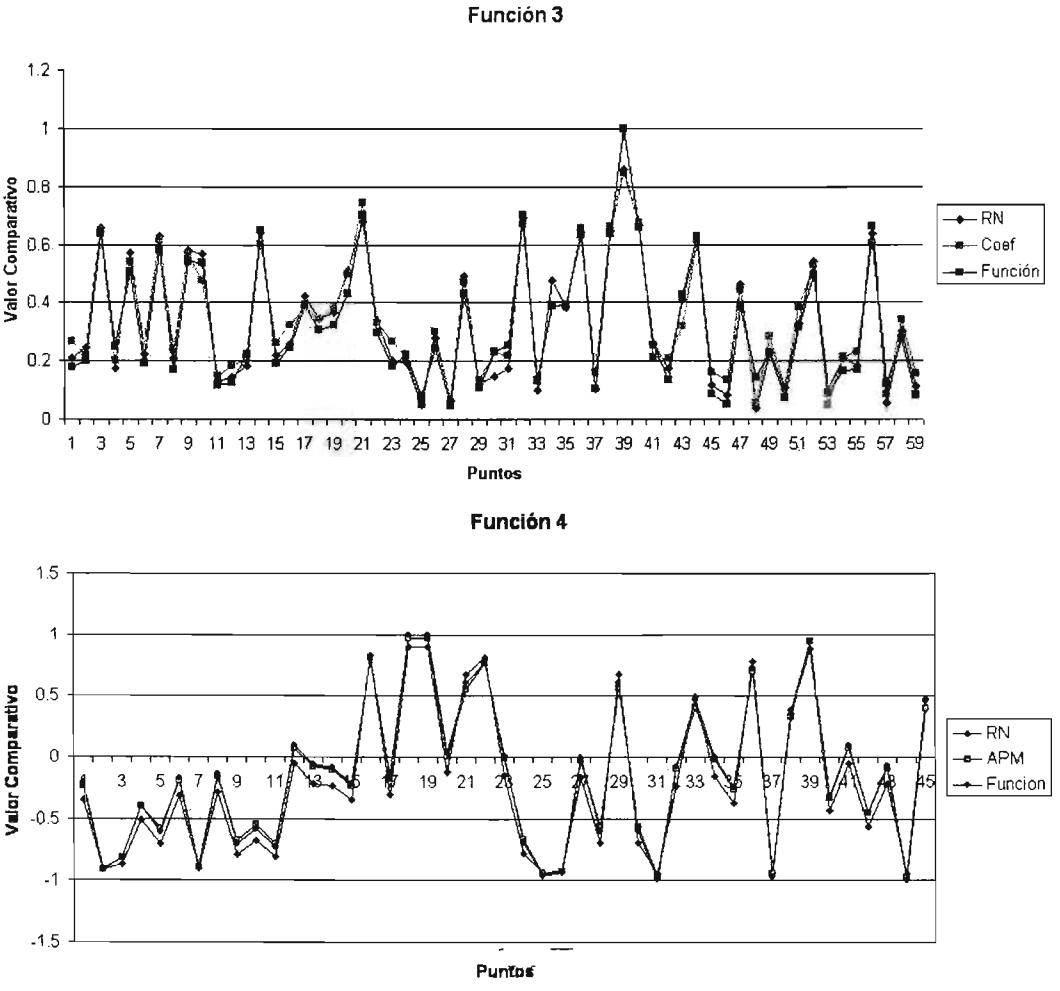


Figura 5.2: Función 3-4

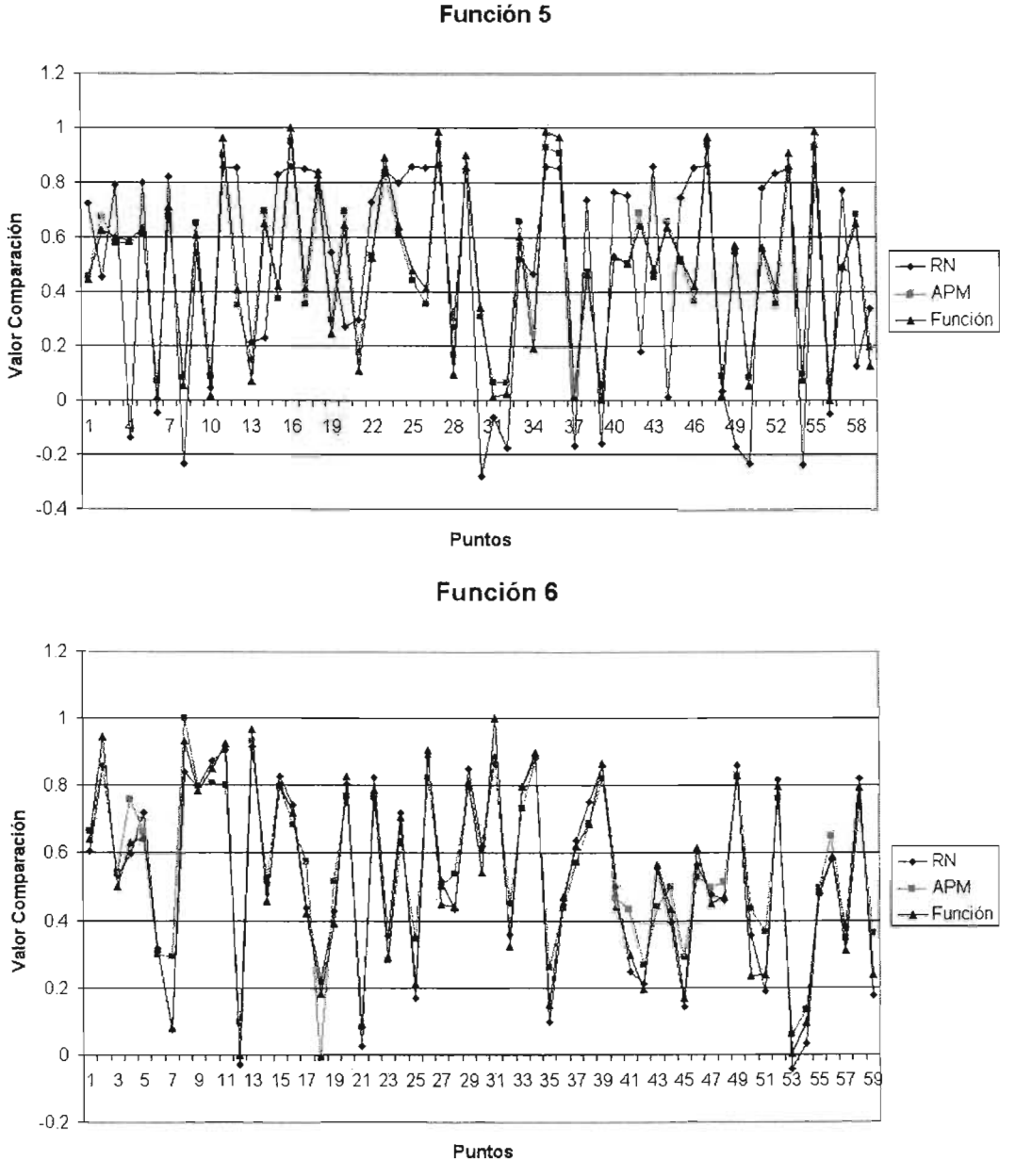


Figura 5.3: Función 5-6

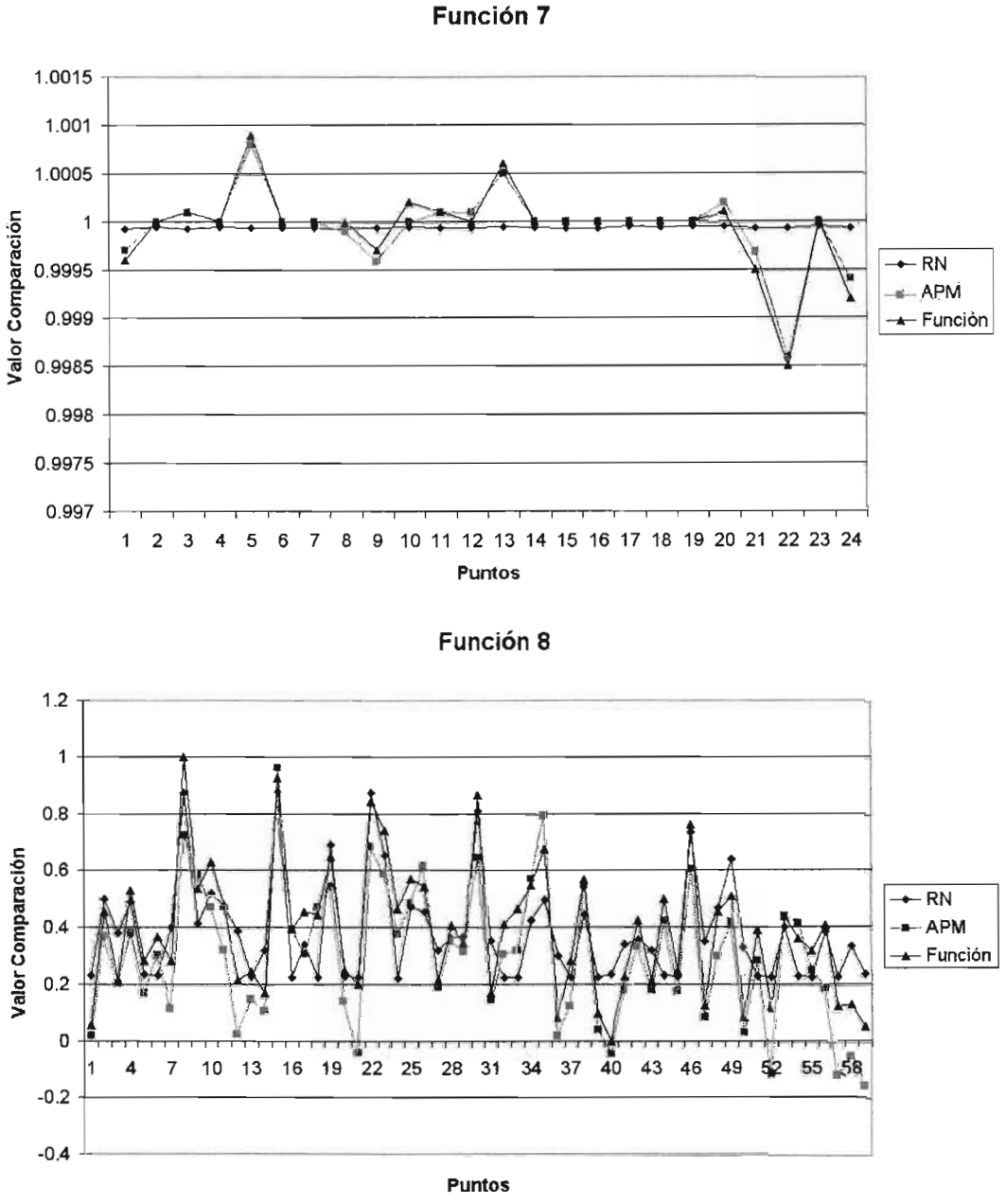


Figura 5.4: Función 7-8

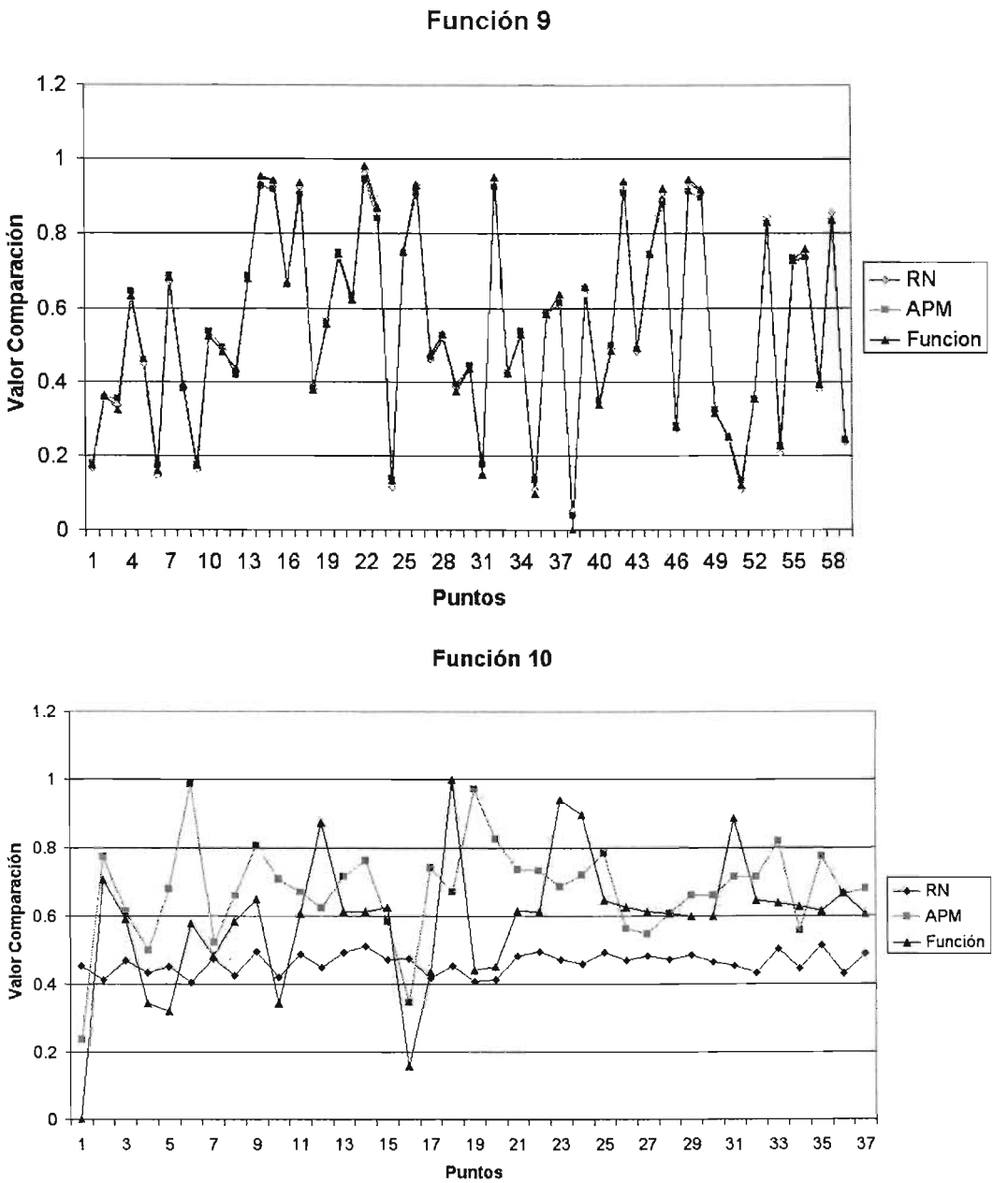


Figura 5.5: Función 9-10

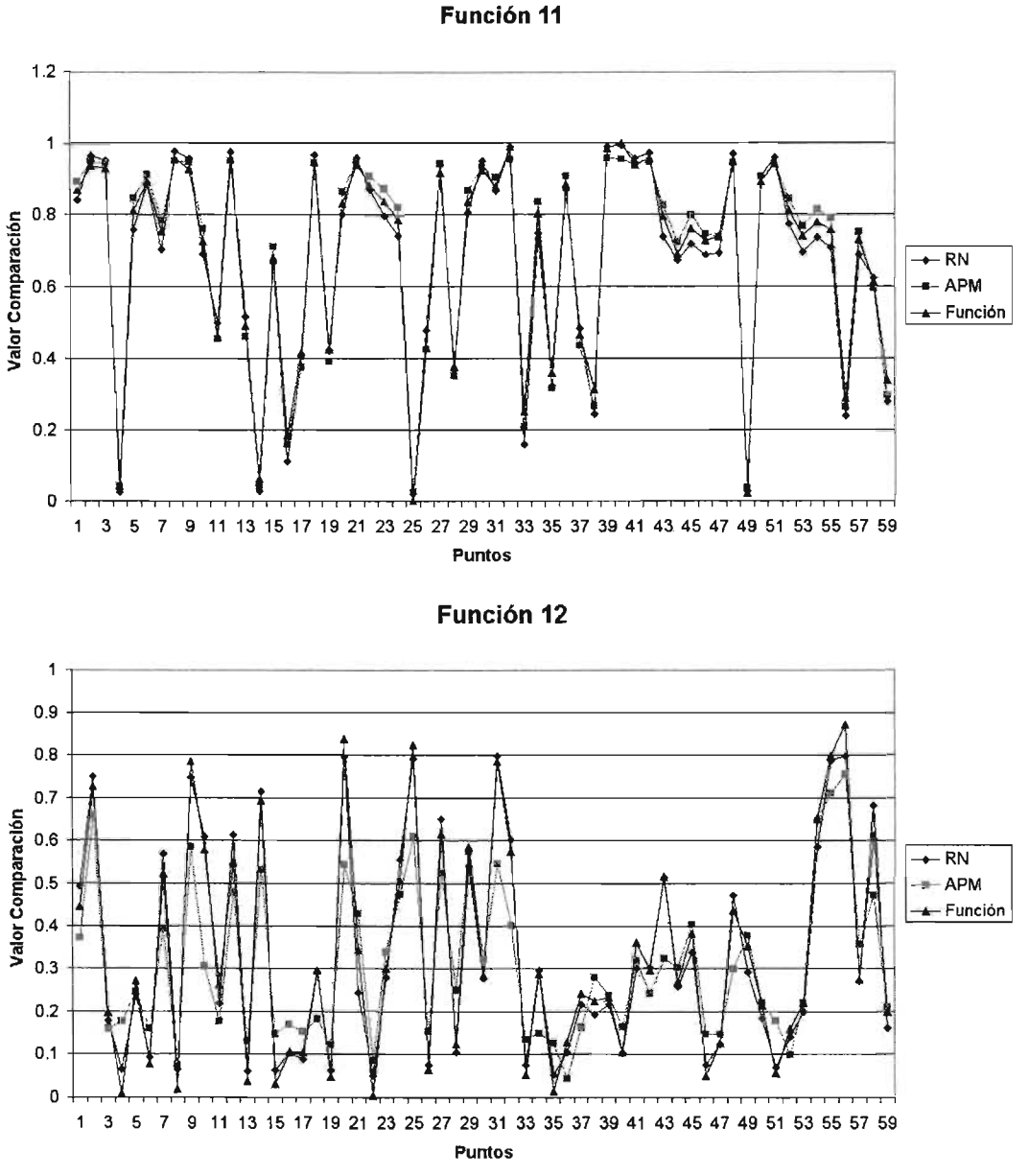


Figura 5.6: Función 11-12

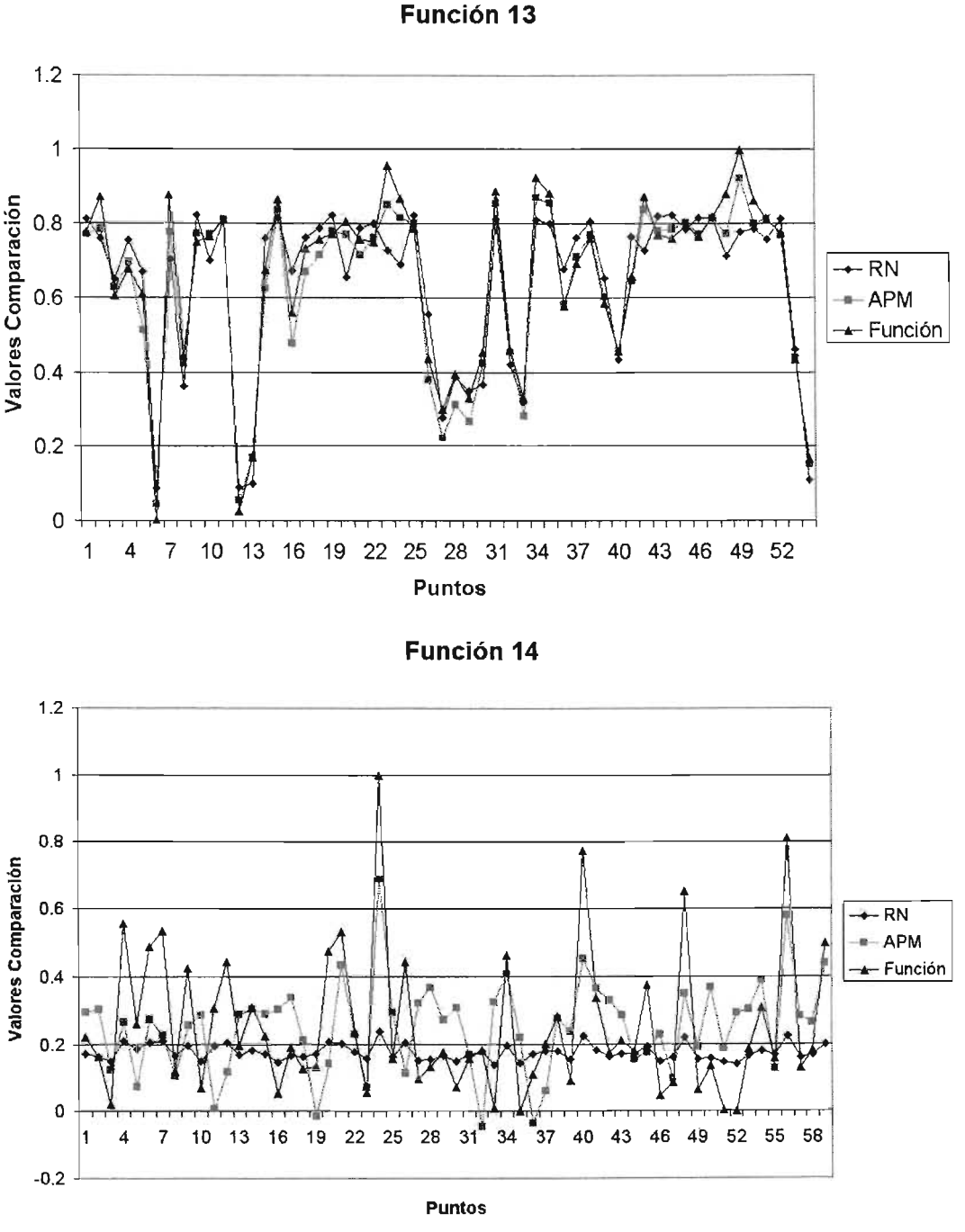
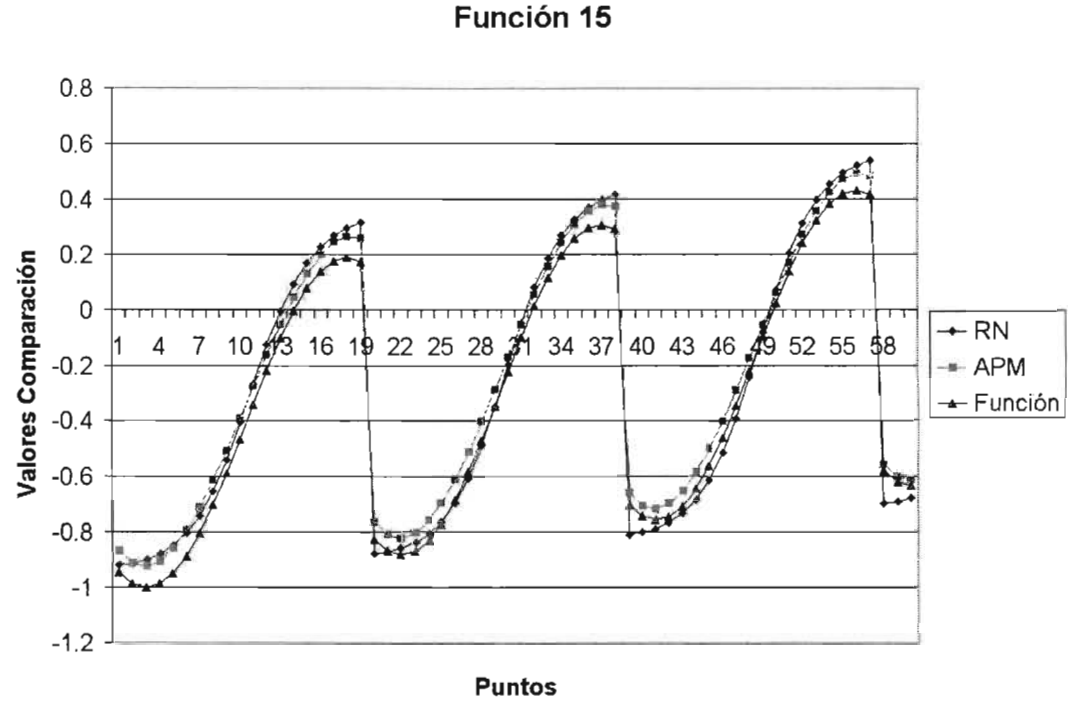


Figura 5.7: Función 13-14



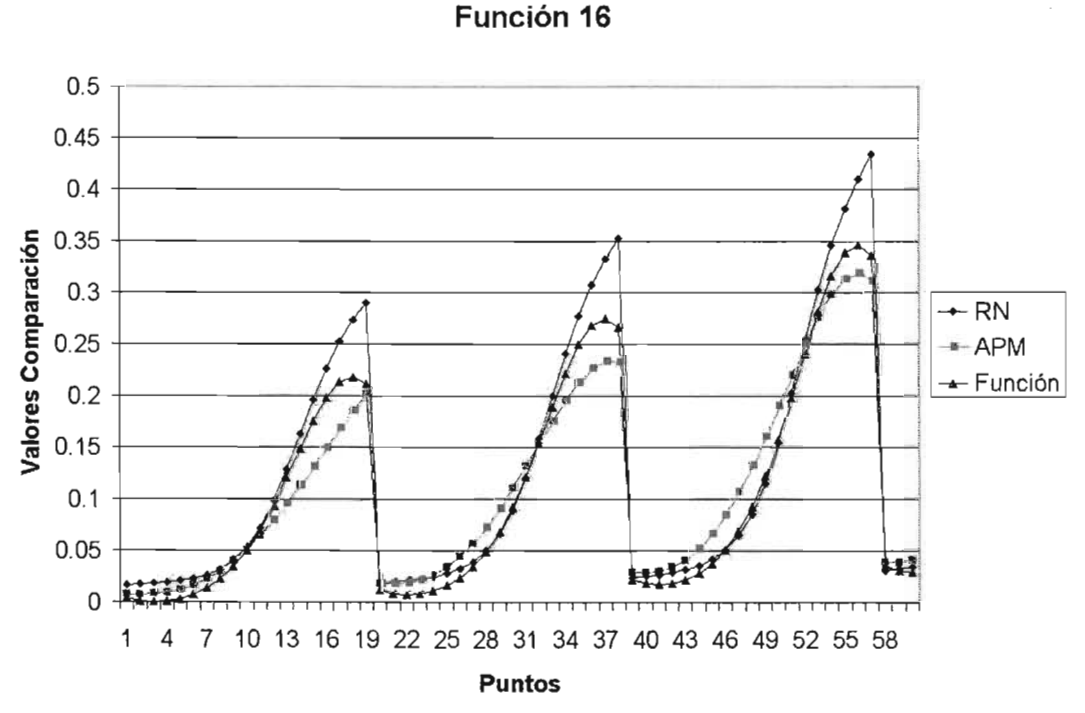


Figura 5.8: Función 15-16

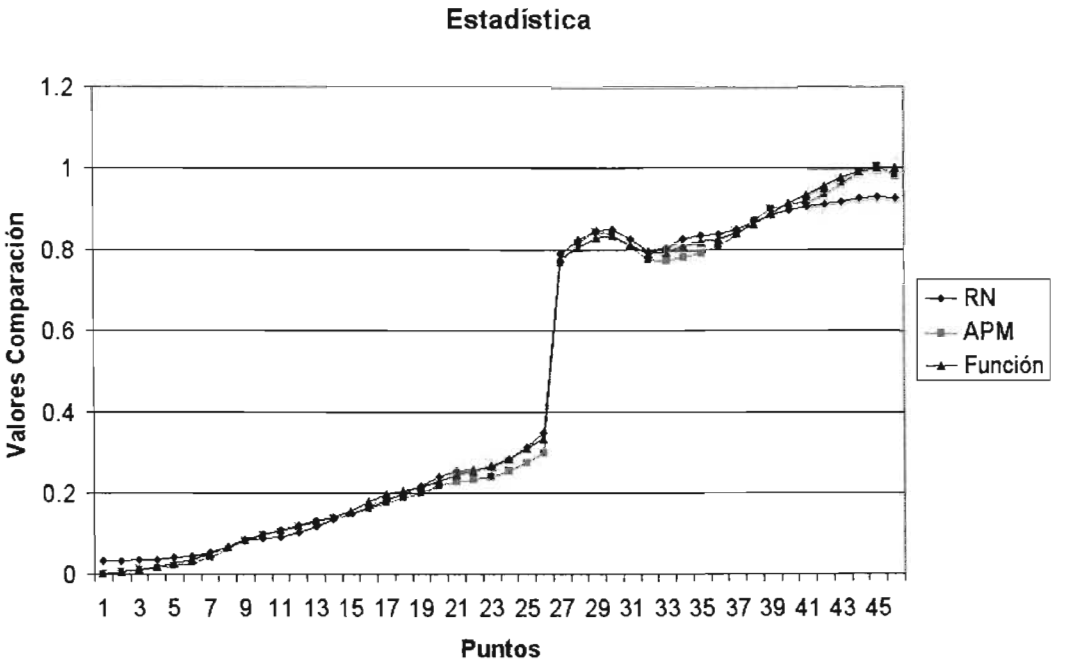


Figura 5.9: Aproximación a Datos Estadísticos

Cocluimos la sección haciendo destacar los resultados obtenidos de Máximo Error de Entrenamiento, Error RMS de Entrenamiento y sus contrapartes: Máximo Error de Prueba y Error RMS de Prueba para ambas metodologías en la tabla 5.3. Estos resultados se volverán importantes para el análisis que se realizará en el capítulo siguiente.

No.	Fución	Restricción
1	$(x_1 - x_2)^2 + (x_2 - x_3)^4 + 1$	$x_1 + x_1x_2^2 + x_3^4 = 3$
2	$x_1^2 + 4x_2^2$	$\frac{3}{5}x_1 + \frac{4}{5}x_2 \geq \frac{13}{5}$
3	$9 - 8x_1 - 6x_2 - 4x_3 + 2x_1^2$ $+2x_2^2 + x_3^2 + 2x_1x_2 + 2x_1x_3$	$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0$ $-x_1 - x_2 - x_3 \geq -3$
4	$(x_1 - 2)^2 + (x_2 - 1)^2$	$x_1 - 2x_2 + 1 = 0$ $-\frac{x_1^2}{4} - x_2^2 + 1 \geq 0$
5	$1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1x_2 - x_1x_3$	$x_1^2 + x_2^2 + x_3^2 - 25 = 0$ $8x_1 + 14x_2 + 7x_3 - 56 = 0$ $x_i \geq 0$ para $i = 1, 2, 3$
6	$100(x_2 - x_1^2) + (1 - x_1)^2 + 90(x_4 - x_3^2)^2$ $+(1 - x_3)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2)$ $+ 19.8(x_2 - 1)(x_4 - 1) + 1$	$-10 \leq x_i \leq 10$ para $i = 1, 2, 3, 4$
7	$e^{x_1x_2x_3x_4x_5}$	$x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 = 10$ $x_2x_3 - 5x_4x_5 = 0$ $x_1^3 + x_2^3 = -1$ -2.3 $\leq x_i \leq$ 2.3 para $i = 1, 2$ -3.2 $\leq x_i \leq$ 3.2 para $i = 3, 4, 5$
8	$3x_1^2 + 2\text{sen}(x_1)\cos(x_2) + 2x_2^2$ $+ \cos(x_1)\text{sen}(2x_2)$- 27.5	$3x_1^2 + 2\text{sen}(x_1)\cos(x_2) - 10.2 \geq 0$ $2x_2^2 + \cos(x_1)\text{sen}(2x_2)$ -17.3 ≥ 0
9	$-(x_1 + \lvert\text{sen}(32x_1)\rvert - \lvert\cos(11x_2)\rvert)$	$0 \leq x_1 \leq \pi$ para $i = 1, 2$
10	$-\frac{\text{sen}^3(2\pi x_1)\text{sen}(2\pi x_2)}{x_1^3(x_1 + x_2)}$	$0 \leq x_i \leq 10$ para $i = 1, 2$ $x_1^2 - x_2 + 1 \leq 0$ $1 - x_1 + (x_2 - 4)^2 \leq 0$
11	$-(e^{-x_1} + x_1 + (x_2 + 1)^2)$	$x_1^2 + x_2 \leq 3$ $x_1 \geq 0; x_2 \geq 0$
12	$\text{sen}(x_1x_2)\cos(x_3x_4) + \text{sen}(x_3x_4)\cos(x_1x_2)$ $2x_1\text{sen}(\pi x_3) + x_3\cos(\pi x_1)$ $+\sum_{i=1}^{4}\text{sen}(\pi x_i) + 4x_1^2 - x_2 + x_4$ -6.807418876198037	$\text{sen}(x_1x_2)\cos(x_3x_4) + \text{sen}(x_3x_4)\cos(x_1x_2)$ $+0.807418876 \geq 0$ $\sum_{i=1}^{4}\sin(\pi x_i) \geq 0$ $4x_1^2 - x_2 + x_4 - 5 \geq 0$ $2x_1\text{sen}(\pi x_3) + x_3\cos(\pi x_1) - 1 \geq 0$
13	$e^{x_1}\cos(x_2) + e^{x_2}\cos(x_1) + \ln(x_1) + x_2^2$ $+ 1.12330837$	$e^{x_1}\cos(x_2) + e^{x_2}\cos(x_1) - 2{,}18871249 \geq 0$ $\ln(x_1) + x_2^2 + 2.063816542 \geq 0$
14	$(x_1^2 + x_2^2) \times \lvert\text{sen}(10(x_1^2 + x_2^2))\rvert$ $\times\lvert\cos(10(x_1^2 + x_2^2))\rvert + \lvert(x_1^2 + x_2^2 + 3)/2\rvert$	$x_1 + x_2^2 + 1 \geq 0$ $x_1^2 + x_2 + 1 \geq 0$ $-2 \leq x_1 \leq 2; -2 \leq x_2 \leq 2$
15	$\text{sen}(x_1) + \cos(x_2)$	$x_i \in [-2, 2]$ para $i = 1, 2$
16	$e^{\text{sen}(x_1)+\cos(x_2)} + x_1x_2$	$x_i \in [-2, 2]$ para $i = 1, 2$

Cuadro 5.1: Funciones ocupadas en el experimento

	Agropecuaria y Pesca	Minería	Industria Manufacturera	Construcción	Total
1960	195,553	41,823	254,815	65,516	1,252,2
1961	198,678	42,752	267,550	65,080	1,306,3
1962	205,439	44,848	277,830	69,302	1,364,6
...	...	...	...	...	...
1991	412,742	189,491	1,252,246	274,308	5,462,7
1992	408,643	192,898	1,280,655	295,720	5,615,9
1993	343,410	118,329	934,544	255,576	4,092,2

Cuadro 5.2: Tabla de Valores del PIB

	Máx Error Ent		RMS Entrenamiento		Max Error Prueba		RMS Prueb	
Función	RPM	APM	RPM	APM	RPM	APM	RPM	AP
1	0.09146	0.07450	0.03175	0.03380	0.06684	0.06000	0.04563	0.0
2	0.54194	0.159371	0.10147	0.097800	0.33099	0.231900	0.09058	0.03
3	0.09547	0.091924	0.02177	0.046500	0.13805	0.186100	0.03535	0.06
4	0.19568	0.059448	0.06558	0.026300	0.16622	0.095430	0.09465	0.10
5	0.20690	0.072448	0.07423	0.046559	0.18723	0.096000	0.07311	0.04
6	0.26535	0.214802	0.06158	0.084898	0.36950	0.258900	0.13367	0.12
7	0.00140	0.000246	0.00020	0.000096	0.03280	0.180500	0.00700	0.02
8	0.29231	0.277640	0.12105	0.123294	0.12105	0.123294	0.19124	0.09
9	0.05142	0.042782	0.00895	0.018609	0.00899	0.019880	0.00899	0.01
10	0.51712	0.251721	0.12889	0.174820	0.52496	0.251700	0.19794	0.17
11	0.09142	0.049032	0.02913	0.026726	0.10607	0.061800	0.04975	0.03
12	0.16469	0.297117	0.04172	0.110960	0.17503	0.379200	0.04923	0.16
13	0.20975	0.108598	0.07617	0.046809	0.20493	0.092700	0.10100	0.05
14	0.59603	0.335111	0.18237	0.185829	0.74799	0.511600	0.18778	0.21
15	0.37180	0.093753	0.08084	0.042782	0.94186	0.715500	0.40632	0.30
16	0.13292	0.092571	0.02233	0.026535	0.11315	0.054100	0.02970	0.02
Est. PIB	0.07240	0.052094	0.01790	0.027366	0.08646	0.100936	0.04824	0.06

Cuadro 5.3: Errores de Aproximación de APM y RPM

¡¡Las dificultades aumentan conforme se aproxima uno al fin!!

Goethe

6

Conclusiones

A la luz de los experimentos, estamos en posibilidad de realizar el análisis entre la Red de Perceptrones Multicapa (RPM) y el Aproximador Polinomial Multivariado (APM). Las cuestiones que vamos a discutir son:

a. ¿Qué tan *buena* es la aproximación realizada por APM con respecto a RPM?

b. ¿Es comparable la capacidad de generalización del APM con respecto a RPM?

c. Si es posible observar que ambas metodologías son capaces de resolver los mismos problemas ¿Será posible encontrar algún tipo de relación entre ellas?

d. ¿Qué ventajas ofrece el APM sobre RPM?

e. ¿Qué posibles aplicaciones se le pueden dar a la metodología propuesta?

(a). Para responder a esta pregunta dirigiremos nuestra atención a los valores obtenidos en los errores de entrenamiento. Observemos el comportamiento de estos valores en las gráficas 6.1 y 6.2

A primera vista se puede apreciar que la diferencia que existen entre los valores de cada criterio de error no es muy grande; en cuanto al Error Máximo de Entrenamiento (EME) la diferencia promedio es menor a 0.2 (tabla 6.1), mientras que la diferencia promedio del Error RMS de Entrenamiento (RMSEnt) es menor a **0.1** (tabla 6.2).

Recordemos que el error RMS se obtiene calculado:

$$E_{RMS} = (\frac{1}{N}\sum_{n=1}^{N} e_n^2)^{1/2}$$

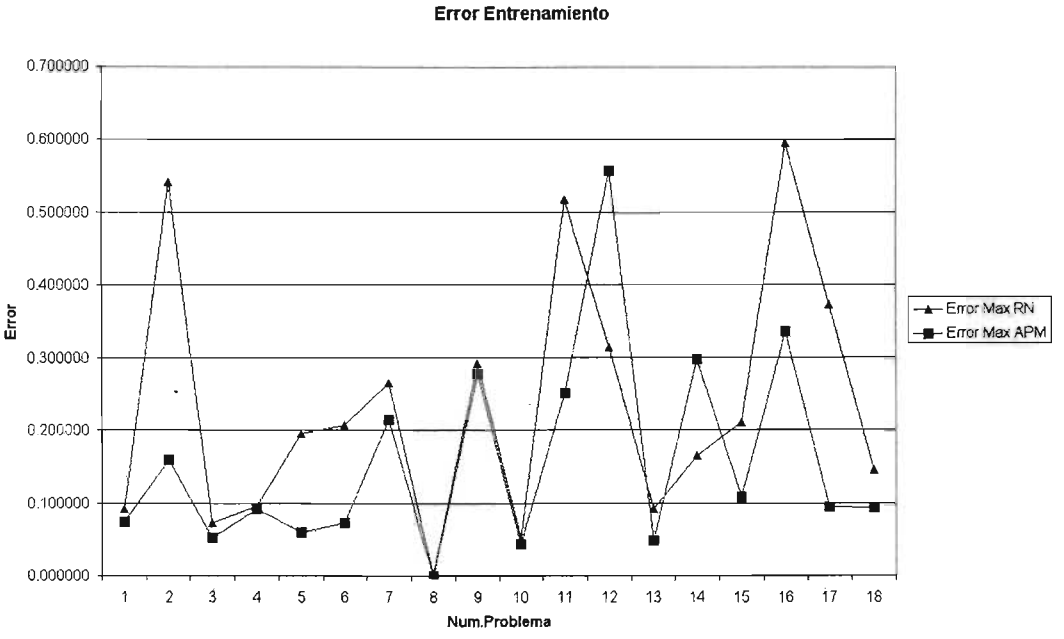


Figura 6.1: Gráfica del Error Máximo de Entrenamiento de APM y RPM

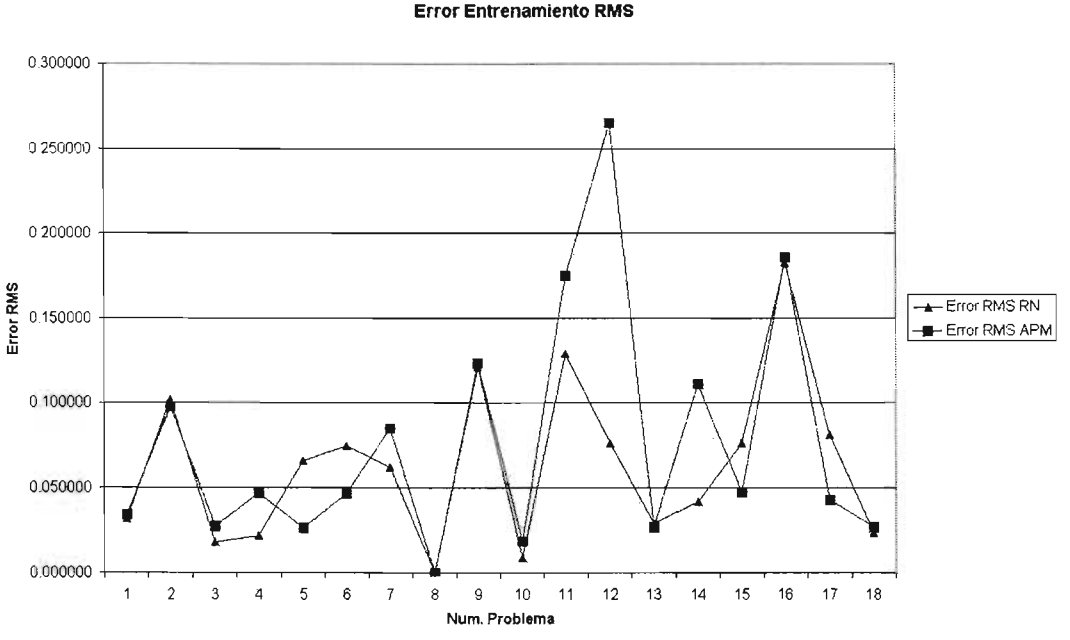


Figura 6.2: Gráfica de Error RMS de Entrenamiento del APM y RPM

donde: N es el número de ejemplares de entrenamiento; $e_n = d_n - o_n$, es decir, e_n es el error obtenido de la diferencia entre la salida deseada (d_n) y el valor que se obtiene (o_n) ya sea del APM o la RPM.

Así que el Error RMS de Entrenamiento mide el promedio del error sobre todas las muestras con los que fueron entrenados los sistemas, sin importar si es obtenido por el APM o la RPM.

Por lo que se observa en la tabla 6.2, la diferencia promedio que existe con respecto al Error RMS de Entrenamiento, entre el APM y RPM nos lleva a pensar que la capacidad de aproximación (podríamos decir aprendizaje) del APM es *del mismo orden* que la de RPM.

	Error Máximo Entrenamiento	Error Máximo Prueba
Diferencia Promedio	0.16563	0.21027

Cuadro 6.1: Relaciones Errores-Máximos del APM y RPM

	Error-RMS Entrenamiento	Error-RMS Prueba
Diferencia Promedio	0.052005	0.055568

Cuadro 6.2: Relaciones Error-RMS del APM y RPM

(b). En cuanto a la cuestión sobre si el APM generaliza tan bien como la RPM, podemos argumentar que sí. Una vez más observando las relaciones que existen entre el Error Máximo de Prueba y el Error RMS de Prueba, tenemos que la diferencia promedio entre los Errores Máximos de Prueba es muy pequeña, tan sólo 0.2 (ver tabla 6.1). Un dato interesante es el que observa al analizar el Error RMS de Prueba (ver tabla 6.2), donde vemos que la diferencia promedio de ambas secuencias de Error RMS, tanto del APM como de la RPM, es tan sólo **0.055568**.

Lo anterior nos muestra que la capacidad de *generalización* del APM es similar a la que manifiesta el RPM. En el experimento la mayoría de las funciones que probamos eran funciones con restricciones, las cuales son difíciles de caracterizar. A pesar de esto en los conjuntos de prueba (valores que no formaron parte del entrenamiento) el APM fue capaz de aproximarlos muy satisfactoriamente. Claro está que la capacidad de generalización del APM está limitada a la variedad de datos que le fueron proporcionados. Si tales datos no dan información relevante del sistema que deseamos aproximar, la generalización se verá muy afectada. Esto pasa de modo análogo con la RPM [Terr99].

(c). La respuesta a la tercera cuestión es negativa y responde a argumentos técnicos:

La RPM según el Teorema del Aproximador Universal es capaz de generalizar aproximaciones por series de Fourier finitas [Hayk99]. Por otro lado, el APM realiza aproximaciones resolviendo sistemas de ecuaciones mediante las cuales minimiza el error máximo, resultando una expresión polinomial que posee la misma capacidad expresiva.

A pesar de ello, en la práctica (como ya se mostró en las funciones aproximadas) es posible tener capacidades expresivas similares. Enfatizamos que esta propiedad es casuística

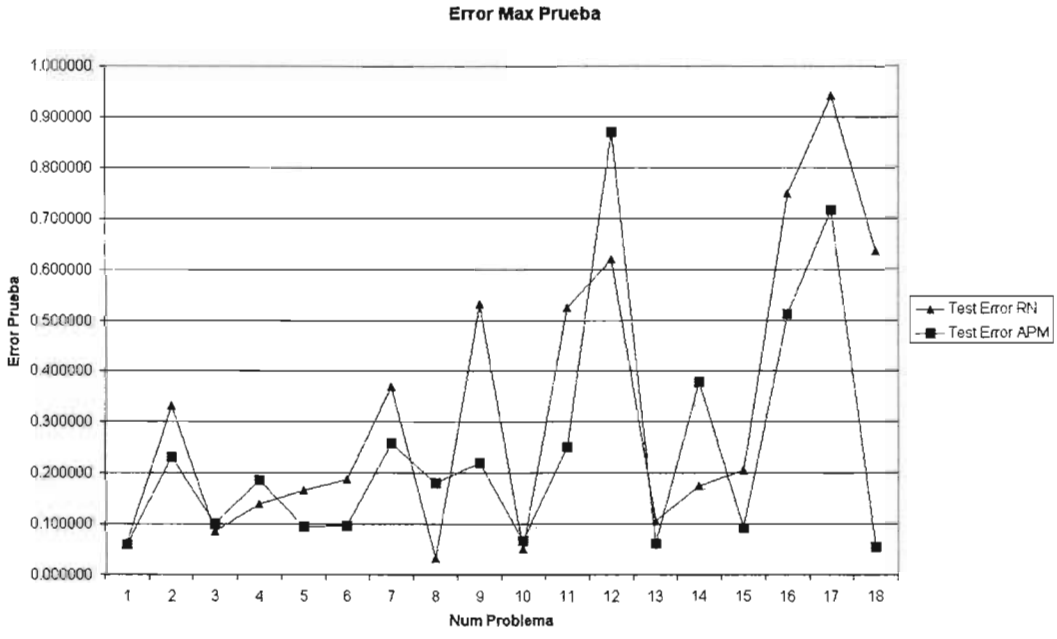


Figura 6.3: Gráfica de Error Máximo de Prueba del APM y RPM

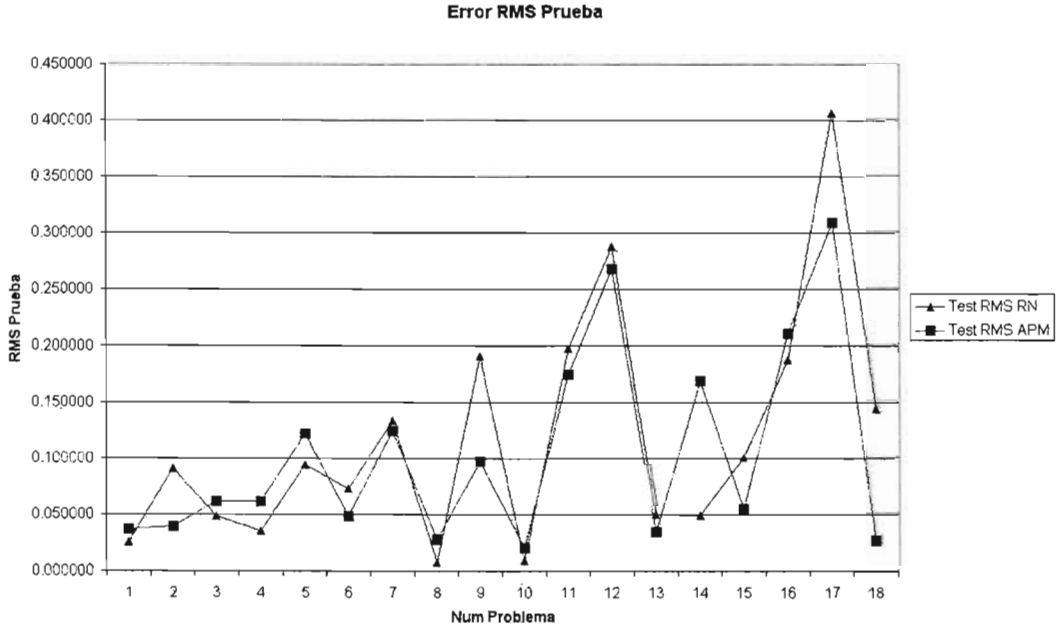


Figura 6.4: Gráfica de Error RMS de Prueba del APM y RPM

y no puede generalizarse. **(d.)** Para observar qué ventajas ofrece una metodología sobre la otra, veamos un ejemplo:

Tenemos una tabla de datos con la información demográfica de los países del mundo del año 2002, los datos relevantes se muestran en la tabla 6.3.

País	Población Total	Crecimiento Anual	% Población 60+	Tasa Fertilidad
Afghanistan	22 930	3.8	4.7	6.8
Albania	3 141	-0.4	9.5	2.3
...	...	...	...	...

Cuadro 6.3: Datos Demográficos Mundial

Esperanza Vida (EV)	Incertidumbre EV
42.6	32.0 - 49.5
70.4	65.9 - 68.7
...	...

Cuadro 6.4: Datos Demográficos Mundial (cont.)

Y supongamos que es nuestro deseo saber qué relación guardan los siguientes datos :

- Población Total.
- Tasa de Crecimiento Anual.
- Porcentaje de población mayor a 60 años.
- Tasa Promedio de Fertilidad.

con respecto a la *Expectativa de Vida* (EV). Para poder tener un sistema que modele el comportamiento de EV en función de los cuatro parámetros anteriores, es posible utilizar una RPM que modele el sistema o generar un modelo con el APM.

Contando con esta información y con base a los resultados obtenidos de los experimentos, comenzaremos con la misma mécanica con la que fueron realizados estos últimos, es decir:

- Primero proyectamos los datos al intervalo $[0, 1]$.
- Elegimos un conjunto de muestras aleatoriamente. El conjunto total está conformado por 197 datos, de los cuales tomamos un 60 % de los datos como muestras de entrenamiento.
- Una vez más la arquitectura de la RPM, es de cuatro neuronas en la capa de entrada, cuatro neuronas en la capa oculta y una neurona de salida.

- Se eligió el algoritmo de Retropropagación con Moméntum para el entrenamiento, con una tasa de aprendizaje: 0.9, tasa de Moméntum: 0.2 y un criterio de paro por épocas.

- Para el APM, claramente el problema se ajusta al tipo de problemas que resuelve. Primero se entrenó la RPM para determinar cuántas variables deseamos que conformen al polinomio aproximador, resultando que la RPM estaba formada por 25 pesos, razón por la cual, se decidió utilizar un polinomio que estuviera conformado de 25 monomios. Los grados máximos fueron ajustados y resultó que un polinomio de grado máximo tres era suficiente.

En la figura 6.5 y 6.6 se muestran el resultado del entrenamiento. Nuevamente lo que observamos en la gráfica es, en el eje x se grafica el número de posición que tiene asociado el país en la tabla, y en el eje de las ordenadas está graficado tanto el valor de EV y el valor de la aproximación dada por el APM y la RPM respectivamente.

En la tabla 6.5, se muestran los pesos obtenidos para la RPM, así como los monomios que conforman la aproximación del APM. También vienen incluidos los valores de los Errores Máximos y RMS tanto de entrenamiento como de prueba para ambas metodologías.

Después de haber realizado el experimento, estamos en posibilidad de discutir las siguientes cuestiones:

- ¿Qué tipo de información me proporcionan estas metodologías?

- ¿Cuál es la ventaja de utilizar la una con respecto a la otra?

Habiendo mostrado que la capacidad de generalización del APM y la RPM son similares en cuanto al tipo de problemas que pueden resolver. Existe, sin embargo, una ventaja extra por parte del APM con respecto a la RPM.

En la Tabla 6.5, tenemos los valores de los pesos de la RPM y los coeficientes del APM. En la RPM no hay forma de saber qué modelo está contenido en ella. Mientras el APM explícitamente nos brinda un modelo del sistema que aproximó. Este modelo es sencillo: un polinomio. Y más aún, podemos deducir una Regla de Inferencia:

> *Cada monomio del polinomio aproximador resultante, establece una regla de asociación para cada una de las variables* $\nu_1, \ldots, \nu_p$ *de la tabla, de las cuales,* ν_{p+1} *es una variable dependiente.*

En el caso particular de nuestro ejemplo la Esperanza de Vida(EV), está en función de los siguientes datos: Población Total (x_1), Crecimiento Anual (x_2), Porcentaje de Población mayor de 60 años (x_3) y la Tasa de Fertilidad (x_4). Esto se puede observar en la tabla 6.6. La conclusión de la regla establecida en la tabla 6.6, nos dice que el resultado de calcular $F_{APM}(x_1, x_2, x_3, x_4)$ es un elemento del intervalo de incertidumbre de la EV (IEV).Dicha regla nos brinda una descripción de bajo cuáles variaciones de los valores de entrada x_1, x_2, x_3, x_4 se puede ver afectada la EV de cada país dentro del rango marcado por IEV.

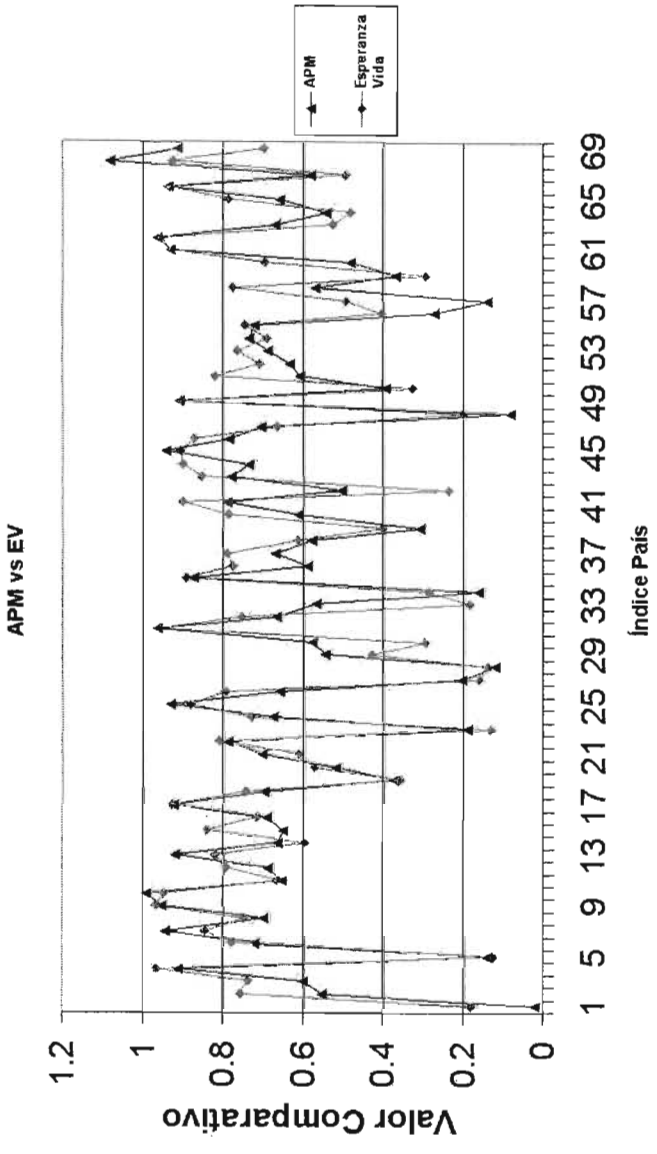


Figura 6.5: Gráficas de Ajuste del APM

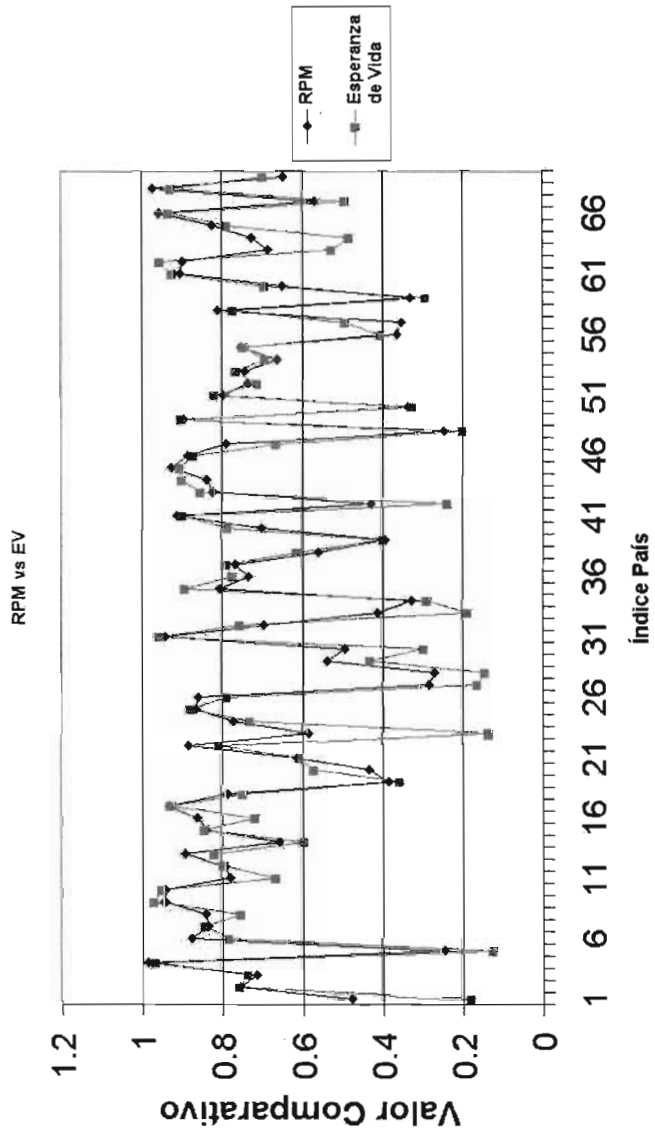


Figura 6.6: Gráficas de Ajuste de RPM

También, de manera tácita, nos proporciona una regla de clasificación. Si etiquetamos cada tupla de las variables dependientes con un índice que indique el renglón que le corresponde en la tabla de datos. Tenemos que para cada $(x_1, x_2, x_3, x_4)_i$, podemos clasificar a cada país del mundo en un IEV.

Sólo resta resaltar el hecho de que, si aplicamos APM a diferentes combinaciones de datos, intercambiando los papeles entre cuáles son variables independientes y cuál dependiente, es posible realizar diferentes análisis desde distintos puntos de vista.

(e). Cuando el APM resuelve un problema de funciones con restricciones, a partir de una muestra representativa de dicho sistema, el APM mapea el Espacio de Puntos Factibles de la función original F que está restringida, a un espacio que APM aproxima a través del polinomio minimax P obtenido.

Este hecho representa una posible ventaja en el área de optimización de funciones, puesto que estamos modelando el espacio factible de un problema con un polinomio. Con lo cual, eliminamos las restricciones impuestas al conjunto original. De esta manera, nos encontramos en posición de enriquecer el análisis de un problema de optimización. No olvidemos el hecho de que la familia de funciones polinomiales son convexas [Chen66]. En consecuencia podemos analizar problemas de optimización caracterizados por P a través de herramientas del cálculo.

Por lo tanto, la herramienta desarrollada nos permite atacar el problema de la optimización de funciones con restricciones mapeando los puntos de la región factible (muestreados aleatoriamente) a un espacio polinomial. Este último puede analizarse y manipularse con evidentes ventajas.

La re-expresión y síntesis de sistemas estadísticamente interesantes constituye posiblemente el problema más interesante de la Inteligencia Artificial. La herramienta desarrollada nos permite atacarlo eficientemente y nos otorga la posibilidad de manipular el modelo con facilidad.

RN	Pesos	Conexion		APM	Coeficientes	Potencias
1	-4.092068	B	H1		-16.735900	0 0 0 3
2	-5.440994	B	H2		121.755000	0 0 1 3
3	-1.879705	B	H3		1.654760	0 0 2 0
4	2.507832	B	H4		-19.899400	0 0 2 1
5	0.087529	B	O1		392.830000	0 0 2 2
6	0.608945	I1	H1		-753.220000	0 0 2 3
7	0.145930	I1	H2		-1.027460	0 0 3 0
8	-0.556843	I1	H3		21.322400	0 0 3 1
8	-2.102637	I1	H4		-349.423000	0 0 3 2
9	4.917166	I2	H1		22.951700	0 1 0 3
10	8.764093	I2	H2		16.935000	0 2 1 0
11	-0.324153	I2	H3		-56.749200	0 2 1 1
12	-5.458061	I2	H4		-101.877000	0 2 1 3
13	2.700059	I3	H1		-5.294710	0 2 3 0
14	4.149881	I3	H2		4.878830	0 3 0 0
15	-0.768904	I3	H3		-3.234780	0 3 0 1
16	-3.015589	I3	H4		-5.740720	0 3 0 3
17	-3.209902	I4	H1		-24.636300	0 3 1 0
18	-4.999513	I4	H2		1.831750	0 3 2 1
19	-1.997092	I4	H3		479.732000	0 3 2 2
20	-2.283683	I4	H4		0.611315	1 0 0 0
21	1.147822	H1	O1		-3.876390	1 0 10
23	1.558210	H2	O1		62.203400	1 0 2 1
24	1.472819	H3	O1		0.506657	1 1 0 1
25	1.241296	H4	O1		-377.800000	3 3 1 2
Error Entremamiento	0.442926				0.355399	
Error RMS	0.093527				0.143265	
Error Prueba	0.450754				0.492145	
RMS Prueba	0.433500				0.523457	

Cuadro 6.5: Comparativo de Pesos y Monomios

IF	-16.735900 $x_1^0 x_2^0 x_3^0 x_4^3$
and	121.755000 $x_1^0 x_2^0 x_3^1 x_4^3$
and	1.654760 $x_1^0 x_2^0 x_3^2 x_4^0$
and	-19.899400 $x_1^0 x_2^0 x_3^2 x_4^1$
and	392.830000 $x_1^0 x_2^0 x_3^2 x_4^2$
and	-753.220000 $x_1^0 x_2^0 x_3^2 x_4^3$
and	-1.027460 $x_1^0 x_2^0 x_3^3 x_4^0$
and	21.322400 $x_1^0 x_2^0 x_3^3 x_4^1$
and	-349.423000 $x_1^0 x_2^0 x_3^3 x_4^2$
and	22.951700 $x_1^0 x_2^1 x_3^0 x_4^3$
and	16.935000 $x_1^0 x_2^2 x_3^1 x_4^0$
and	-56.749200 $x_1^0 x_2^2 x_3^1 x_4^1$
and	-101.877000 $x_1^0 x_2^2 x_3^1 x_4^3$
and	-5.294710 $x_1^0 x_2^2 x_3^3 x_4^0$
and	4.878830 $x_1^0 x_2^3 x_3^0 x_4^0$
and	-3.234780 $x_1^0 x_2^3 x_3^0 x_4^1$
and	-5.740720 $x_1^0 x_2^3 x_3^0 x_4^3$
and	-24.636300 $x_1^0 x_2^3 x_3^1 x_4^0$
and	1.831750 $x_1^0 x_2^3 x_3^2 x_4^1$
and	479.732000 $x_1^0 x_2^3 x_3^2 x_4^2$
and	0.611315 $x_1^1 x_2^0 x_3^0 x_4^0$
and	-3.876390 $x_1^1 x_2^0 x_3^1 x_4^0$
and	62.203400 $x_1^1 x_2^0 x_3^2 x_4^1$
and	0.506657 $x_1^1 x_2^1 x_3^0 x_4^1$
and	-377.800000 $x_1^3 x_2^3 x_3^1 x_4^2$
THEN	$F_{APM}(x_1, x_2, x_3, x_4) \in IEV$

Cuadro 6.6: Regla derivada para la Demografía Mundial

Bibliografía

[Chen66] Cheney, E W; *Introduction to Approximation Theory*, Mc Graw-Hill Book Company, 1966.

[CS93] Craben , M.W, and Shavlik J.W. (1993) "Learning Symbolic Rules Using Artificial Neural Networks". Proceedings of the Tenth International Conference on Machine Learning, pp. 73–80, Amherts, MA.

[DE97] DataEngine. MIT GmbH, Germany, 1997.

[Hayk99] Haykin, S; *Neural Networks: A Comprehensive Foundation*, Pratice Hall, 1999.

[JNL96] Johnson, G., Nealon, J.L. and Lindsay, R.O. (1996) Üsing relecance information in the acquisition of rules from a neural network". In Proceedings of the Rule Extraction from Trined Artificial Network". In Proceedings of the Rule Estraction from Trained Artificial Networks Workshop, pages 68-80, Queensland University of Technology.

[Kuri93] Kuri, A., *Heuristic Techniques for the Generalization and Acceleration of an Algorithm for Multivariate Minimax Approximation*, AGROCIENCIA, 1993, pp. 94-95.

[Kuri02] Kuri, M. A. , Galaviz, J. *Algoritmos Genéticos*, Instituto Politécnico Nacional, Universidad Nacional Autónoma de México, Fondo de Cultura Económica, Serie: Ciencia de la Computación; México 2002.

[Kuri] Kuri, M. A., *A Comprhensive Approch to Genetic Algorithms: Theory and Applications*, Vol 2: Aplications (to be published).

[Mitch96] Mitchel, M; *An idealized Genetic Algorithm*, Introduction to Genetic Algorithms MIT Press, 1996, pp. 132-138.

[MHF94] Mitchell, M., J.Holland y S.Forrest, *Where Will A Genetic Algorithm Outperform Hill Climbing?*, Advances of Neural Information Processing Systems, No. 6, Morgan Kaufmann, 1994, pp. 51-85.

[Roja96] Rojas, R.; *Neural Networks: A systematic Introduction*; Springer-Verlag Berlin Heidelbetg, New York 1996.

[RUD94] Rudolph, G., *Convergence Analysis of Canonical Genetic Algorithms*, IEEE Transactions on Neural Networks, 5(1):96-101, enero 1994.

[RKZ02] Setiono, R., Kheng, L., Zurada J. M., *Extractor of rules from Neural Networks for Nolinear Regression.* IEEE Transactions on Neural Networks, Vol. 13, No. 3, May 2002.

[Shoi97] Shoichiro, N., *Análisis Numérico y Visualización Gráfica con Matlab*; Pearson Educación, México 1997.

[Terr99] Terrence L. F.; *Feedforward Neural Network Methodology*, Springer-Verlag, New York 1999.