INSTITUTO TECNOLOGICO
Y DE
ESTUDIOS SUPERIORES
DE MONTERREY

REDES NEURONALES

UNA APLICACION PARA LA PREDICCION DE OZONO

POR
RAUL SMITH PEREZ

TESIS

MAESTRIA EN
CIENCIAS COMPUTACIONALES
ESPECIALIDAD EN
INTELIGENCIA ARTIFICIAL

MAYO 1994

REDES NEURONALES

UNA APLICACIÓN PARA LA PREDICCIÓN DE OZONO

TESIS

MAESTRÍA EN CIENCIAS COMPUTACIONALES
ESPECIALIDAD EN INTELIGENCIA ARTIFICIAL

POR

RAÚL SMITH PÉREZ

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS
SUPERIORES DE MONTERREY

MAYO 1994

REDES NEURONALES
UNA APLICACIÓN PARA LA PREDICCIÓN DE OZONO

POR

RAÚL SMITH PÉREZ

TESIS

Presentada a la División de Graduados e Investigación
ESTE TRABAJO ES REQUISITO PARCIAL PARA OBTENER EL TÍTULO DE MAESTRO EN CIENCIAS.

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

MAYO 1994

QUIERO DAR GRACIAS A *CRISTO JESUS* QUE ME HA DADO LA VIDA Y LA OPORTUNIDAD DE CONCLUIR MIS ESTUDIOS. A ÉL ME DEBO.

A MIS PADRES *RAÚL Y CECILIA* DESEO AGRADECERLES TODO SU AMOR Y SACRIFICIO. POR ELLOS ESTOY AQUÍ.

A MI HERMANO *ARMANDO*, A QUIEN ADMIRO MUCHO, QUIERO DARLE LAS GRACIAS POR SU APOYO Y SUS CONSEJOS.

QUIERO AGRADECER DE FORMA MUY ESPECIAL A *OLIVIA* QUE ME HA ANIMADO SIEMPRE.

PARA EL *DR. CARLOS RUIZ*, MI ASESOR Y AMIGO, ESPERO LO MEJOR Y DESEO BRINDARLE MI MÁS CALUROSO AGRADECIMIENTO.

FINALMENTE DOY GRACIAS A *CONACYT*, QUE ES MÉXICO, POR HABER FINANCIADO MIS ESTUDIOS.

RESUMEN

Este trabajo presenta una investigación y un análisis sobre las capacidades y limitaciones prácticas de dos modelos neuronales convencionales: el modelo de retropropagación y la memoria asociativa bidireccional (BAM). El fundamento teórico para cada modelo antecede a una aplicación general sobre contaminación ambiental para el pronóstico del nivel de ozono en la ciudad de México. Se realizaron pruebas sobre cada uno de estos modelos, especialmente sobre la BAM. La información necesaria provino de las cinco principales estaciones del sistema RAMA (red automática de monitoreo ambiental). Las comparaciones y los resultados de estas pruebas son presentados, además de las conclusiones sobre cada uno de los modelos.

ABSTRACT

This work reports an investigation and an analysis over real advantages and disadvantages of two conventional neural models: the backpropagation approach and the bidirectional associative memory (BAM). A theoretical foundation is presented for both models along with an environmental pollution application: the Mexico City short-term forecasting of ozone. All data came from the five most complete RAMA (red automática de monitoreo ambiental) stations. Comparisons and results are presented and some conclusions are drawn.

TABLA DE CONTENIDO

TABLA DE FIGURAS

MODELO DE RETROPROPAGACIÓN

MEMORIA ASOCIATIVA BIDIRECCIONAL (BAM)

RETROPROPAGACIÓN vs BAM

RESULTADOS

CAPÍTULO 1

1 INTRODUCCIÓN

Una red neuronal, computacionalmente hablando, es un conjunto de unidades procesadoras interconectadas entre ellas. Cada unidad procesadora o neurona recibe los estímulos de ciertas unidades procesadoras, realiza un cierto proceso, y manda su propia señal a su vez a otras unidades procesadoras. La señal que emite depende de la entrada neta recibida y de una función de activación (en general no lineal) que opera sobre dicha entrada neta [1]. La manera en que este sistema es capaz de aprender es modificando paulatinamente la fuerza de las conexiones entre unidades procesadoras. Se pueden incluso eliminar o crear nuevas conexiones durante el proceso de aprendizaje.

Las redes neuronales son entes capaces de generar auto-clasificaciones a partir de información de entrada [2], de forma que la entrada de un nuevo estímulo (no entrenado) provoca una respuesta de salida correspondiente que, en distancia de Hamming (cantidad de bits en que difieren dos vectores), es el patrón almacenado más cercano a la salida correcta. La forma en que las redes neuronales **aprenden** proviene directamente del usuario o por medio de su propio conocimiento acumulado.

Los modelos neuronales computacionales surgen a partir del modelo neuronal biológico y por esto sus principios estructurales y modo de operación tienen gran analogía biológica. Además, las redes neuronales computacionales se rigen por principios físicos comprobables. Las redes neuronales se comportan como **procesos energéticos** en donde se garantizan los mínimos de energía, es decir, la convergencia siempre a una solución. La red neuronal es capaz de encontrar siempre una respuesta de salida (no se cicla eternamente), aún cuando ésta no sea la solución correcta.

La naturaleza intrínseca de los modelos neuronales es la capacidad de procesamento en paralelo [3]. Dicha cualidad se vuelve ideal para aplicaciones de procesamiento de imágenes o reconocimiento de patrones [4], por mencionar algunos. La gran explosión combinatoria espacial de estos problemas, aunado con ambientes de ruido, vuelve ineficientes e insuficientes a otros enfoques tradicionales (aún cuando el problema puede resolverse utilizando otros enfoques, se requiere de modelos muy complejos y de una gran cantidad de tiempo en la ejecución). Además la naturaleza misma de la computadora, cuyo procesamiento es secuencial, se aleja de por sí de la forma en como debe abordarse esta problemática. El procesamiento paralelo real brinda esta ventaja.

La principal y gran división de los modelos neuronales está dada por el modo en como se actualiza la matriz de conexiones [5]. Existen los modelos supervisados y no supervisados. Los modelos supervisados realizan su aprendizaje ajustando progresivamente las conexiones de acuerdo al criterio de minimización de la diferencia entre el patrón de salida real y el patrón generado por el sistema. En los modelos no supervisados este aprendizaje no se rige por la minimización de la diferencia sino busca la libre clasificación.

El submodelo de pesos-compuestos (memorias asociativas) pertenece a la clase de modelo no supervisado. La principal característica del modelo de pesos-compuestos o memorias asociativas es que los valores de sus pesos son precalculados y prealmacenados.

El modelo neuronal no supervisado incluye lo que se llama la regla de aprendizaje competitivo [8,5]. Bajo este grupo estan las redes de auto-organización [2] (self-organizing), resonancia adaptiva [9] (adaptive resonance) y neocognitrón [4]. El aprendizaje de estos modelos depende solamente de los patrones de entrenamiento. Gráficamente estos modelos van generando superficies n-dimensionales donde gradualmente se forman zonas de mayor aglutinación de datos, que corresponden a los grupos autoclasificados. Estas zonas contienen un punto máximo alrededor del cual se forma dicho grupo [10].

Al contrario, el tipo de modelo supervisado requiere de una regla de aprendizaje (basada en la minimización de la diferencia entre el patrón de salida real y el obtenido por el sistema) para el ajuste de los pesos de las conexiones. Existen los grupos basados-en-decisión y los basados-en-aproximación-optimización [6]. El primer grupo describe las redes de clasificación en donde no es trascedente la precisión del resultado, sino sólo la clasificación correcta. El segundo tipo implica reglas de aprendizaje basadas en alguna función de optimización, en donde se requiere que la salida sea exacta o aproximada al resultado correcto con cierto margen de error [10]. El modelo de retropropagación pertenece a este último grupo.

El modelo de aprendizaje supervisado requiere la introducción de reglas de aprendizaje del tipo $M_{ij}(t+1)=M_{ij}(t)+\Delta M_{ij}(t)$ [5]. El aprendizaje en este tipo de modelo se realiza actualizando los pesos de las conexiones en base a una función de diferencias de salida entre el patrón de datos real y el predicho.

Dentro de este tipo de modelo existe el procesamiento recurrente y no recurrente. El procesamiento recurrente es aquel que requiere de varias pasadas (una entrada produce una salida que a su vez vuelve a entrar al sistema, y así sucesivamente) antes de poder generar una solución. Al contrario, el procesamiento no recurrente obtiene la solución de una sola pasada.

El modelo de autoasociación de Hopfield es de este tipo y utiliza la actualización de pesos por medio de la regla de Hebb [6]. El algoritmo de esta red de Hopfield puede ser del tipo asíncrono o síncrono (modo de actualización secuencial o paralela de los pesos). El modo de procesamiento de la memoria asociativa bidireccional (BAM) es recurrente. Esta red es una extensión del modelo de Hopfield para heteroasociaciones [5,7].

Más aún, la memoria asociativa lineal y no-lineal corresponden al tipo de procesamiento no recurrente (feedforward) y pueden operar con datos discretos o continuos, y ser del tipo autoasociativo o heteroasociativo [5]. La linealidad o no linealidad del modelo lo determina el tipo de función de activación o función que se aplica sobre la entrada neta de cada unidad procesadora. Generalmente se utiliza la función de activación de tipo sigmoide (no líneal y continua). El modelo neuronal auto-asociativo es aquel en donde una entrada parcial o una entrada con ruido genera como salida dicho patrón de entrada completo y limpio de ruido.

Este tipo de modelos se utiliza frecuentemente como filtro señales y como reconocedor de patrones. Por otro lado, el modelo hetero-asociativo permite almacenar relaciones pares de forma que la entrada de un patrón A evoca (realiza un **recall**) la salida de su correspondiente asociado B.

La capacidad de operar con datos discretos y/o continuos produce otra gran división en los modelos neuronales. La función de activación juega un papel muy importante al respecto. Generalmente dicha función es continua y opera sobre el valor de la entrada neta a cada unidad procesadora; sin embargo cuando el modelo neuronal en cuestión es de tipo discreto, existen ciertas condiciones de umbral para evaluar la salida de la función de activación y así discretizar la salida.

La memoria asociativa bidireccional (BAM) tiene la cualidad de ser un modelo sencillo de comprender, fácil de codificar y tolerante al ruido. La desventaja que presenta es su limitada capacidad de almacenamiento de asociaciones (aproximadamente 15 asociaciones por cada 100 neuronas en la capa de mayor dimensión [15]).

A modo de aprovechar las capacidades de la BAM y superar la restricción de su limitada capacidad de almacenamiento, durante la evolución del presente trabajo se creó un **Sistema BAM** capaz de aprender un número ilimitado de asociaciones. Este **Sistema BAM** está compuesto por un conjunto de **BAM´s**, donde cada una se encarga de aprender sólo un cierto número de asociaciones. Así, todas juntas, forman la gran memoria del **Sistema BAM**. En este trabajo se presenta este **Sistema BAM** junto con una serie de pruebas y resultados sobre una aplicación para el pronóstico de ozono. De la misma forma, otro sistema llamado **Sistema Retropropagación** se creó y se probó sobre la misma aplicación.

El problema de la contaminación ambiental en las grandes urbes es una preocupación prioritaria a nivel internacional. La Ciudad de México no es la excepción de este problema, y por esto surgió la idea de utilizar las redes neuronales como una herramienta de pronóstico de los niveles de ozono.

Debido a que las variables que intervienen de forma significativa en la química del ozono (y por lo tanto en el pronóstico de ozono) son varias, este trabajo muestra las pruebas y los resultados de sólo tres grandes grupos de prueba formados a partir de estas variables.

PRÓLOGO

Este trabajo de tesis está orientado sobre todo hacia aquellos alumnos que, además de un interés particular sobre el tema, cuentan con las bases fundamentales sobre redes neuronales. Debido a que este trabajo hace más énfasis en la parte práctica de un par de modelos neuronales aplicados al pronóstico de ozono, se presta menor atención al desarrollo teórico fundamental del enfoque neuronal mismo, y por lo tanto algunos principios, estructuras y modos de operación se explican tan sólo de forma breve y no se describen a detalle.

En el capítulo 1 se presenta una introducción sobre lo que son las redes neuronales y sus principales divisiones. Además se introduce el **Sistema BAM** y el **Sistema Retropropagación,** sistemas que fueron creados durante este trabajo para una aplicación sobre pronóstico de ozono.

Durante el capítulo 2 se muestra el desarrollo teórico-matemático de cada uno de los dos modelos neuronales aquí utilizados: el modelo de retropropagación y la memoria asociativa bidireccional (BAM). Su estructura, modo de aprendizaje, y modo de evocación o **recall** también se presentan.

El capítulo 3 muestra las pruebas realizadas para cada uno de los modelos sobre la aplicación del pronóstico de ozono, además de una comparacion entre éstos. Los resultados de estas pruebas y de las comparaciones se presentan en el capítulo 4.

Las conclusiones de este trabajo y la posibilidad de trabajos futuros estan contenidos en el capítulo 5. Además, en forma de apéndices, se incluye un artículo internacional aceptado, producto de este trabajo, y los códigos fuente de los Sistemas BAM y Retropropagación.

Espero que al lector le sea de beneficio la lectura parcial o completa de este trabajo de tesis y le sirva como un punto de partida para lograr avances superiores y para realizar también trabajos mucho mejores que éste.

CAPÍTULO 2

2 FUNDAMENTO TEÓRICO

2.1 MODELO DE RETROPROPAGACIÓN

2.1.1 DEFINICIÓN

La red de retropropagación es un modelo neuronal capaz de resolver complejas funciones de mapeo y reconocimiento de patrones complejos [10]. Se utiliza principalmente para clasificación de patrones y como filtro de ruido para regeneración de señales. Inicialmente fue formalizado por Werbos [1], y más tarde por Parker, Rumelhart y McClelland [3].

Es una arquitectura simétrica, multicapa, y genéricamente representada por una capa de entrada, una o más intermedias, y una de salida [5]. No existen conexiones intracapa ni conexiones hacia atrás [10]. Su procesamiento es hacia adelante y utiliza el modo de aprendizaje supervisado. La figura 1 ilustra el modelo genérico.

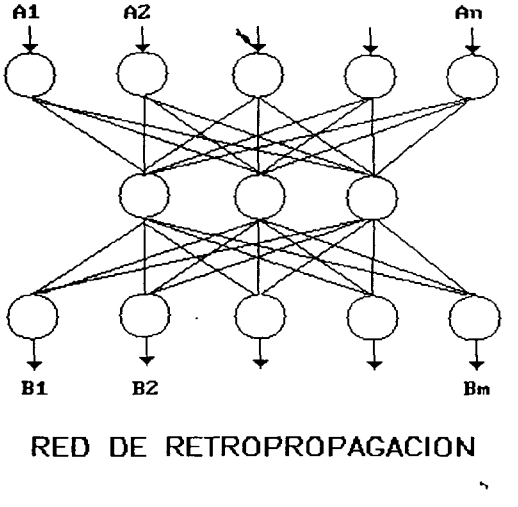


Figura 1 Esquema genérico de la arquitectura de la red de retropropagación.

2.1.2 OPERACIÓN [2]

Un conjunto de pares asociados (A_i,B_i) son presentados a la red uno a la vez. Un primer vector A_i entra como un vector estímulo y se propaga a cada una de las capas superiores, hasta que un vector de salida O_i sale del sistema. Posteriormente se obtiene una diferencia o error entre la salida deseada B_i y la actual O_i. Este error se transmite entonces hacia atrás a todas las unidades de procesamiento a modo de corregir el valor de los pesos de las conexiones y dirigir así el proceso de aprendizaje.

En un principio se corrigen los pesos de la capa de salida de acuerdo a la diferencia o error detectado. Posteriormente, basado en el error de salida, la capa intermedia corrige también sus pesos de acuerdo a la proporción de la contribución parcial que produjo dicha capa sobre la salida O_i.

De esta forma los pesos y las conexiones de las capas intermedias (puede haber más de una y el proceso es semejante) se van modificando a medida que el entrenamiento con patrones diferentes continúa. Una nueva entrada ocasiona que las capas intermedias proporcionen el patrón asociado más cercano, aún cuando exista ruido o información incompleta en el vector estímulo.

2.1.3 APRENDIZAJE: REGLA DELTA [1]

La regla de aprendizaje del modelo de retropropagación deriva de una función de minimización del error durante la fase de aprendizaje. A continuación se presenta la función de minimización, además del proceso para determinar dicha regla. Este desarrollo teórico se realiza para un modelo de retropropagación de una capa intermedia.

Primeramente supóngase un vector de entrada a la red de dimensión n (número de neuronas de entrada) dado por $A = \{A_1, A_2, .., A_n\}$. La dimensión de la capa intermedia se supone p y la de salida m. Las unidades de entrada distribuyen entonces los datos hacia la capa intermedia, y ésta, a su vez, hacia la capa de salida. La entrada neta hacia cada unidad intermedia es:

$$Net^h_{pj} = \Sigma^n_i M^h_{ji} A_{pi} + U^h_j$$

donde n es el número de neuronas de la capa de entrada, h denota las p unidades de la capa intermedia, y M_{ji} indica el peso de la conexión desde la unidad procesadora de entrada i a la unidad j (es importante notar que por fines prácticos este trabajo omite la inicialización de los subíndices de las sumatorias, dado que en todos los casos comienza en la unidad). El último término es el umbral (en un principio los umbrales toman valores aleatorios que se consideren apropiados [5], sin embargo, estos valores no son definitivos en el resultado). Se asume que la función F es la función de activación (generalmente sigmoide o del tipo $F(X)=X/(X^2+c)^{1/2}$) de la entrada neta a cada unidad de la capa intermedia, de forma que:

$$i_{pj} = F^h_j (Net^h_{pj})$$

representa la salida de cada una de las p unidades procesadoras de la capa intermedia. Nuevamente, la entrada neta para cada unidad de salida está dada por:

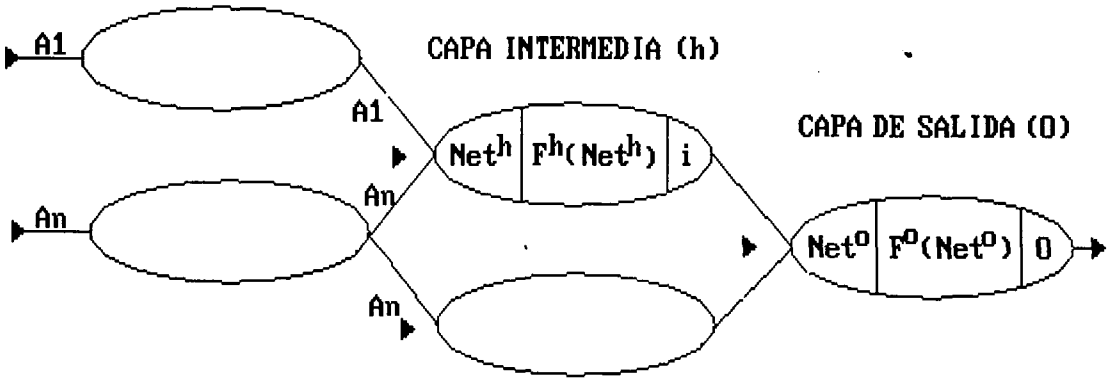
$$Net^o_{pk} = \Sigma^l_j M^o_{kj} i_{pj} + U^o_k$$

y su respectiva función de activación es:

$$O_{pk} = F^o_k (Net^o_{pk})$$

donde O_{pk} es la salida del sistema de cada una de las m unidades de salida de la red. La figura 2 ilustra gráficamente cada una de las variables anteriores.

RETROPROPAGACION
ESQUEMA ESTRUCTURAL



F

igura 2 Diagrama estructural del modelo de retropropagación de una capa intermedia.

Una vez que se conocen las entradas netas para cada unidad procesadora, se procede a evaluar el error y distribuirlo proporcionalmente entre las unidades de cada capa. Este aprendizaje se basa principalmente en el gradiente negativo de la función de error (se busca minimizar las crestas de la superficie generada por la función de error sobre un plano imaginario).

El ajuste de los pesos para cada unidad se basa en el valor del peso mismo anterior más una diferencia (positiva o negativa) de acuerdo al error detectado y proporcional a dicha unidad. La función de error es:

$$E_p = 1/2 \, \Sigma^m{}_k \, \delta^2{}_{pk} \qquad (1)$$

donde

$$\delta_{pk} = (B_{pk} - O_{pk})$$

donde B_{pk} es el valor deseado de salida y O_{pk} la salida del sistema.

A fin de simplificar el proceso, se muestra primero el desarrollo para cada unidad procesadora de la capa de salida. Se tiene lo siguiente:

$$E_p = 1/2 \, \Sigma^m{}_k \, (B_{pk} - O_{pk})^2 \qquad (2)$$

y

$$\frac{\partial E_p}{\partial M^o{}_{kj}} = -(B_{pk} - O_{pk}) \frac{\partial F^o{}_k}{\partial Net^o{}_{pk}} \frac{\partial Net^o{}_{pk}}{\partial M^o{}_{kj}} \qquad (3)$$

Si la función de activación es sigmoidal, el segundo término después de algunas operaciones, que no se incluyen aquí por fines prácticos, queda de la siguiente forma:

$$F^o{}_k' = F^o{}_k(1 - F^o{}_k) = O_{pk}(1 - O_{pk}) \qquad (4)$$

e i_{pj} se define como:

$$\frac{\partial Net^o_{pk}}{\partial M^o_{kj}} = \frac{\partial}{\partial M^o_{kj}} \Sigma^I_j (M^o_{kj}\, i_{pj} + O^o_k) = i_{pj} \qquad (5)$$

Así, de (3),(4) y (5) el ajuste de las unidades de salida está dado por:

$$M^o_{kj}(t+1) = M^o_{kj}(t) + \eta\, \delta^o_{pk}\, i_{pj}$$

y

$$\delta^o_{pk} = (B_{pk} - O_{pk})\, O_{pk}\, (1 - O_{pk})$$

El símbolo η se introduce como **coeficiente de aceleración** y se explicará más adelante.

Para obtener la regla de aprendizaje (ajuste) para las unidades de la capa intermedia se sigue un proceso similar:

$$E_p = 1/2\ \Sigma^I_k (B_{pk} - F^o_k(Net^o_{pk}))^2$$

$$= 1/2\ \Sigma^I_k (B_{pk} - F^o_k(\Sigma^m_j\ M^o_{kj}\, i_{pj} + O^o_k))^2$$

y sabemos que i_{pj} depende de los pesos de la capa intermedia. Por tanto, por medio de este término podemos ajustar los pesos de las unidades de esta capa intermedia. La parcial de E_p con respecto de M_{ji} intermedia es:

$$\frac{\partial E_p}{\partial M^h_{ji}} = -\Sigma^I_k (B_{pk} - O_{pk}) \frac{\partial O_{pk}}{\partial Net^o_{pk}} \frac{\partial Net^o_{pk}}{\partial i_{pj}} \frac{\partial i_{pj}}{\partial Net^h_{pj}} \frac{\partial Net^h_{pj}}{\partial M^h_{ji}}$$

en donde después de calcular algunos de los términos (otros ya se han calculado anteriormente) obtenemos el ajuste para las unidades de la capa intermedia:

$$M^h_{ji}(t+1) = M^h_{ji}(t) + \eta\, \delta^h_{pj}\, A_i$$

donde

$$\delta^h_{pj} = F^{h\prime}_j(Net^h_{pj})\, \Sigma^I_k \delta^o_{pk} M^o_{kj}$$

en donde puede verse que cada δ^h_{pj} está en términos de todas las δ^o_{pk}.

Un pseudo-algoritmo con los pasos principales para el aprendizaje es el siguiente:

1) Entrar un vector A_i a la red y obtener la salida O_i mediante el proceso anterior.
2) Comparar la salida O_i con el valor deseado B_i y obtener el error entre éstos.
3) Determinar la dirección (+,-) del cambio en los pesos.
4) Obtener la cantidad de cambio en los pesos por medio de las reglas de ajuste.
5) Repetir 1) hasta que el error para cada uno de los vectores de entrenamiento sea mínimo (o todos los ejemplos hayan sido entrenados).

2.2 MEMORIA ASOCIATIVA BIDIRECCIONAL (BAM)

2.2.1 DEFINICIÓN

La red neuronal de tipo BAM (Bidirectional Associative Memory) es una arquitectura de dos capas (entrada y salida) conectadas simétricamente. Su aprendizaje lo lleva a cabo de forma no-dinámica [12,11] si se conocen los patrones asociados (A_i,B_i).

Este modelo de red permite trabajar con toda clase de operaciones de asociación de patrones en el espacio o en el tiempo. Opera con patrones discretos en el espacio binario bipolar {+1,-1} con el fin de simular mejor las excitaciones (+1) y las inhibiciones (-1) neuronales [12]. La figura 3 ilustra esta arquitectura.

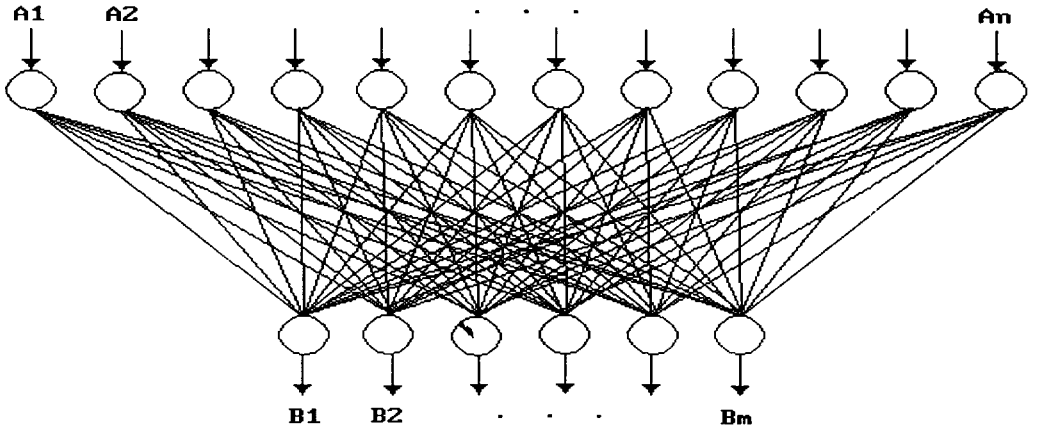


Figura 3 Arquitectura general de la Memoria Asociativa Bidireccional (BAM).

2.2.2 ASOCIATIVIDAD

Aquí se presentan los tres tipos de asociaciones más comunes que existen en el enfoque neuronal y el tipo que corresponde a la BAM. La letra ϕ denota cierta función de asociación o de mapeo.

- Memoria autoasociativa: se define por $\phi(A_i) = A_i$. Si existe una entrada A cercana en distancia de Hamming a otra almacenada A_i, entonces $\phi(A) = A_i$. Este tipo de memoria se utiliza en aplicaciones donde se desea filtrar una señal que contiene ruido o en operaciones de reconocimiento de patrones.

- Memoria asociativa interpolativa: definida como $\phi(A_i) = B_i$. Si existe una entrada $A = A_i+d$, donde A_i es una entrada ya almacenada y d una cantidad de ruido, entonces $\phi(A) = B_i+I$, donde B_i es la salida asociada a A_i, $I = Md$, y M es la matriz de memoria. Este tipo de memoria se utiliza en aplicaciones en donde se desea interpolar la salida, dada la entrada y la memoria actual.

- Memoria heteroasociativa: está dada por $\phi(A_i) = B_i$. Si existe una entrada A cercana en distancia de Hamming a otra entrada almacenada A_i, entonces $\phi(A) = B_i$. Este tipo de memoria es la que utiliza la BAM y es fuertemente resistente al ruido.

2.2.3 APRENDIZAJE [1,8]

El aprendizaje se lleva a cabo de forma que la suma de los productos externos (matrices) entre los vectores de salida y los transpuestos de los vectores de la entrada forman la memoria de la BAM. Los vectores de entrada son de dimensión n (número de neuronas enla la capa de entrada) y los vectores de salida son de dimensión m (número de neuronas en la capa de salida). Supongamos un conjunto de patrones (A_i,B_i), la memoria M se construye como:

$$M = B^t_1A_1 + B^t_2A_2 + .. + B^t_nA_n$$

y el proceso de recall de un nuevo vector A se realiza multiplicando la memoria M por el vector de entrada A de la forma siguiente:

$$\phi(A) = (B^t_1A_1 + B^t_2A_2 + .. + B^t_nA_n)A$$

2.2.4 OPERACIÓN [6]

Supóngase un par asociado memorizado (A_i,B_i); la entrada (excitación) A deberá invocar la salida B_i siempre y cuando su patrón asociado A_i sea el más cercano en distancia de Hamming (número de bits en los que difieren A_i de A) al vector almacenado A. Lo anterior sucede después de una serie de pasadas bidireccionales de la forma:

$$\begin{array}{l} A \text{ --> } M \text{ --> } B \\ A' \text{ <-- } M^t \text{ <-- } B \\ \quad\quad\quad\quad \vdots \\ A_i \text{ --> } M \text{ --> } B_i \\ A_i \text{ <-- } M^t \text{ <-- } B_i \end{array}$$

esto significa que un vector A entra al sistema (memoria M) y produce una primer salida B, que a su vez entra nuevamente con una memoria transpuesta M^t y produce una salida A'; este proceso se repite hasta que no exista cambio en la salida B_i (estado energético mínimo). Este vector B_i es la solución del sistema.

2.2.5 CODIFICACIÓN DE LA BAM [6]

Sea A la capa de entrada y B la capa de salida. Para conocer los valores de entrada netos de la capa B tenemos:

$$NET(B) = MA, \qquad Net(B_i) = \Sigma^n_j M_{ij} A_j$$

e igualmente para la capa A:

$$NET(A) = M^tB, \qquad Net(A_i) = \Sigma^m_j B_j M_{ji}$$

El valor de salida de cada unidad procesadora está dado por:

$$B_i(t+1) = \begin{cases} +1 & \text{si NET(B)} > 0 \\ B_i(t) & \text{si NET(B)} = 0 \\ -1 & \text{si NET(B)} < 0 \end{cases} \qquad A_i(t+1) = \begin{cases} +1 & \text{si NET(A)} > 0 \\ A_i(t) & \text{si NET(A)} = 0 \\ -1 & \text{si NET(A)} < 0 \end{cases}$$

2.2.6 FUNCIÓN DE ENERGÍA [1]

El aprendizaje de la BAM se lleva a cabo por medio de un proceso no-dinámico [12,11]. Una vez entrenada la red, la entrada de un nuevo patrón A_i genera un proceso dinámico de reverberación hacia la producción de un patrón asociado B_i; es decir, un proceso de pasadas bidireccionales que existen mientras no se alcance un estado energético mínimo. Por ende, es deseable que este proceso dinámico sea estable, es decir, se pueda llegar a un estado energético mínimo.

Se requiere entonces una función cuyos cambios sucesivos resulten en decrementos energéticos para tender a una solución estable [10,12]. Dicha función es:

$$E(A,B) = - B^t M A \qquad \text{o} \qquad E(A,B) = - \Sigma^m_i \Sigma^n_j \; B_i \, M_{ij} \, A_j$$

en cual debe cumplirse lo siguiente:
1) cualquier cambio en A o B resulta en decremento de E,
2) E está acotada por $E(min) = -\Sigma_{ij} |M_{ij}|$,
3) y los cambios de E son cantidades finitas.

2.2.7 DEMOSTRACIÓN [1]

Para demostrar que la función anterior E(A,B) cumple efectivamente el requerimiento de ser una función cuyos cambios energéticos resultan siempre en decrementos, dichos cambios energéticos que resultan siempre en decrementos se demostrarán primero para cualquier unidad procesadora k y luego se generalizará para todas las unidades restantes. Dado lo anterior, para una unidad k cualquiera tenemos:

$$B_k: \quad E = - \Sigma^n_j B_k M_{kj} A_j - \Sigma^m_{i \neq k} \Sigma^n_j B_i M_{ij} A_j$$

$$E^{new} = - \Sigma^n_j B_k^{new} M_{kj} A_j - \Sigma^m_{i \neq k} \Sigma^n_j B_i M_{ij} A_j$$

$$\Delta(E^{new} - E) = (B_k - B_k^{new}) \Sigma^n_j M_{kj} A_j$$

donde

$$B_k^{new} = \begin{cases} +1 & \text{si } \Sigma M_{kj} A_j > 0 \\ B_k & \text{si } \Sigma M_{kj} A_j = 0 \\ -1 & \text{si } \Sigma M_{kj} A_j < 0 \end{cases}$$

Existen dos casos que pueden producir un cambio energético y éstos suceden cuando en la unidad procesadora B_k existe un cambio de valor (pues si no hay cambio de valor la energía es cero). Las dos posibilidades son las siguientes:

A) $B_k = +1$ y $B_k^{new} = -1$:

$$B_k - B_k^{new} > 0 \Rightarrow \Sigma^n_j M_{kj} A_j < 0 \Rightarrow \Delta(E) < 0.$$

B) $B_k = -1$ y $B_k^{new} = +1$:

$$B_k - B_k^{new} < 0 \Rightarrow \Sigma^n_j M_{kj} A_j > 0 \Rightarrow \Delta(E) < 0.$$

Este proceso puede ser extrapolado a todas las unidades de la capa de salida B como:

$$\Delta(B_i) = (B_i - B_i^{new}), \qquad \Delta(E) = \Sigma^m_i \Delta(B_i) \Sigma^n_j M_{ij} A_j$$

Así se observa que $\Delta(E)$ es una suma de m términos, en donde cada uno de los términos $\Delta(B_i)$ es cero (si no hay cambio de valor) o negativo (si existe cualquier cambio de valor).

CAPÍTULO 3

3 DESARROLLO PRÁCTICO

Con el fin de aplicar los modelos de retropropagación y la memoria asociativa bidireccional (BAM), el tema sobre la contaminación ambiental por emisiones de gases en la ciudad de México pareció una buena alternativa práctica para este trabajo de investigación. Debido a la facilidad para adquirir la información que se requería y al conocimiento de personas del área química (que trabajan ya al respecto), se tornó factible formar un equipo de trabajo sobre este tópico.

3.0.1 PLANTEAMIENTO DEL PROBLEMA [13,14]

La contaminación en la Ciudad de México, y sobre todo, el alto nivel de ozono, es producto de la interacción entre un conjunto de variables atmosféricas y la presencia de ciertos elementos contaminantes.

Con el fin de medir los índices de contaminación ambiental, en la ciudad de México existe una red de 25 estaciones (RAMA) ubicadas en distintos puntos de la ciudad. Sólo 5 estaciones miden hasta 9 parámatros por minuto (promediados cada hora) cada 24 horas, y por esto son consideradas como las más completas. Estas estaciones son: Xalostoc (noreste), Tlalnepantla (noroeste), Merced (centro), Cerro de la Estrella (sureste), y Pedregal (suroeste).

Las variables que miden son: dirección del viento (DV) y velocidad del viento (VV), temperatura (T), humedad relativa (HR), dióxido de sulfuro (SO_2), monóxido de carbono (CO), ozono (O_3), dióxido de nitrógeno (NO_2), y óxidos de nitrógeno (NO_x).

La producción de ozono depende principalmente de dos factores: la cantidad de NO_2 en la atmósfera y el flujo actínico (en un rango de 280-420 nanómetros) en el límite de la capa terrestre. El primer parámetro lo mide RAMA, pero no el segundo. Para estimar el segundo parámetro se utiliza un modelo teórico validado y utilizado anteriormente sobre las condiciones atmosféricas de la Ciudad de México [22].

Por otro lado, la dispersión del ozono es una función no lineal de la dirección del viento (DV), de la velocidad del viento (VV), de la temperatura (T), y de la relación de óxidos de nitrógeno (NO_x/NO_2), entre otros. La razón NO_x/NO_2 indica el estado de la mezcla de los componentes anteriores.

El parámetro de humedad relativa (HR) juega un papel importante en la reactividad total del sistema, afectando ya sea reacciones terminales de cadena o por la producción de aerosoles húmedos (que a su vez afectan el flujo actínico UV). El CO es el contaminante menos reactivo e interviene poco en la química del O_3. Sin embargo, debido a esta propiedad, es llevado a grandes distancias en la atmósfera sin reaccionar con otros componentes y se convierte entonces en una medida indirecta de la corriente de viento.

Finalmente el SO_2, en la presencia de H_2O_2 y de aerosoles húmedos, participa en la química del ozono. No obstante, durante la temporada seca (que es la considerada en este trabajo) el SO_2 juega el mismo papel del CO.

Durante los inicios de la investigación sobre algunos paradigmas neuronales tradicionales, se planteó la necesidad de construir un sistema capaz de PRONOSTICAR EL NIVEL DE OZONO EN LA CUIDAD DE MEXICO basado en el cúmulo de datos que las estaciones de monitoreo aportaban y de un cierto modelo neuronal adecuado.

3.0.2 OBJETIVO

El objetivo es entonces construir un sistema neuronal capaz de ser entrenado con grandes volúmenes de datos y de generar un pronóstico de nivel de ozono cercano a los niveles medidos reales. Posteriormente se buscará predecir los niveles de ozono para horas y días futuros.

El sistema tendrá que ser fuertemente resistente al ruido debido, primero, a que los datos medidos en las estaciones de monitoreo traen de por sí error en la medición o incluso no son medidos en algún momento, y segundo, porque la gran explosión combinatoria de valores (de aproximadamente 50 variables) representa en sí misma ruido.

3.0.3 CRITERIOS DE EVALUACIÓN DE PRUEBAS

Durante las pruebas y los resultados de este trabajo se utiliza nativamente la palabra **recall** para evitar confusiones o impresiciones con su significado en español (quizá la palabra más adecuada sea **evocación** pero esto queda al gusto del lector). El **recall** entonces se entenderá en adelante como el proceso de evocación de una salida almacenada B ante el estímulo de una entrada A.

Por otro lado, durante el trabajo se utilizó como criterio de evaluación de pruebas de recall al rango de +/- 30 imecas como un porcentaje tolerable de error (30 imecas representa aproximadamente el 10% del valor del rango del ozono: desde 0 hasta 350 puntos imecas).

MODELO NEURONAL. Dos modelos neuronales que son motivo del presente trabajo constituyen la base de la aplicación para el pronóstico del ozono.

SELECCIÓN DE MUESTRAS. El cúmulo de información provino de las 5 estaciones más completas y correspondieron a los meses de enero a mayo de 1992. Los datos se colectaron de cada estación de monitoreo de contaminantes y se proporcionaron las muestras de monitoreo promediadas cada hora. Se conformaron archivos de entrenamiento y archivos de prueba. El criterio de selección de muestras para estos archivos fue aleatorio. Cada línea de archivo se constituyó por la información de entrada (valores de las variables que intervienen en la química del ozono) y de salida (ozono medido) en este orden. Se realizaron pruebas y comparaciones. Las muestras para las pruebas y para las comparaciones entre modelos fueron seleccionadas de forma aleatoria.

En cada prueba se midieron los tiempos de aprendizaje y de recall por muestra. Debido al número de variables de entrada involucradas en la química del ozono, se formaron tres grandes bloques de pruebas (12, 18, y 47 variables de entrada) bajo el criterio de prioridad o importancia de la variable en la química del ozono.

Dentro de cada uno de estos grupos de prueba se variaron los siguientes parámetros:

- tamaño del archivo de entrenamiento: número de líneas o muestras;

- criterio de selección de muestras de prueba: el primer criterio fue selección aleatoria y el segundo consistió en detectar muestras con un nivel de ozono similar y entrenar al sistema con sólo la mitad de ellas para luego probarlo con la otra mitad no entrenada;

- porcentaje de ruido (información incompleta) en las muestras de prueba no entrenadas: variables con valor 0 (variables que no se midieron en esa muestra);

- porcentaje de ruido (información incompleta) en las muestras de prueba entrenadas: variables con valor 0 (variables que no fueron medidas para esa muestra).

3.1 MODELO DE RETROPROPAGACIÓN

3.1.1 SOLUCIÓN

Aún cuando en la parte del desarrollo teórico se presentó un modelo de retropropagación de una capa intermedia, el **Sistema Retropropagación** desarrollado aquí no tiene capa intermedia. Esto es por razones de comparación, ya que la BAM (el modelo neuronal utilizado en este trabajo) no contiene capa intermedia.

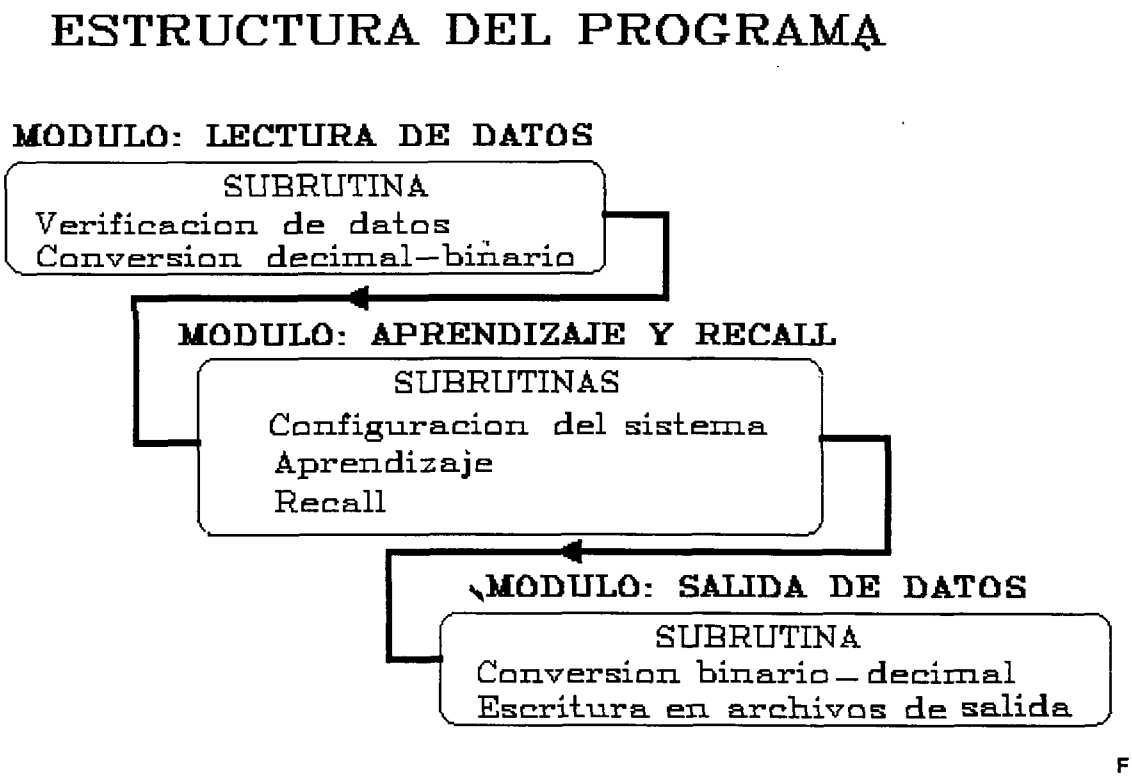
El sistema de retropropagación de dos capas está constituido por una sóla memoria de dimensiones limitadas por la computadora y el compilador disponibles. Bajo dichas dimensiones, el sistema cuando mucho es capaz de aprender algunas decenas de asociaciones. Los patrones de entrada son vectores correspondientes a los valores de las variables que influyen en la química del ozono. Los patrones de salida corresponden similarmente a los valores del nivel de ozono medido.

El número de neuronas de la capa de entrada y de salida corresponden a la mínima cantidad de bits necesarios para representar los valores máximos de las variables de entrada y de salida. Los pesos y los umbrales son inicializados de forma aleatoria (+/- 0.5).

3.1.2 SISTEMA DE RETROPROPAGACIÓN

Se desarrolló un programa que contiene todas las funciones necesarias para llevar a cabo, de manera automática, el proceso de pronóstico de nivel de ozono utilizando el modelo de retropropagación de dos capas.

Lo único que requiere el sistema es el archivo de entrenamiento y los archivos de prueba. El código fuente está escrito en lenguaje C y corre sobre cualquier PC. La figura 4 muestra los principales módulos del programa.



F

igura 4 Diagrama de módulos del Sistema Retropropagación.

FORMATO DE LOS ARCHIVOS. Los archivos de entrenamiento y de pruebas deben estar en formato texto. Cada línea de archivo debe contener tanto los datos de entrada como los datos de salida en este orden.

Los principales módulos del programa son los siguientes:

- **Configuración.** Se encarga de generar, a partir del archivo de entrenamiento, la configuración nxm mínima suficiente de la red de retroprogagación para que sea posible entrenar todas las muestras. Para generar la configuración mínima suficiente, un módulo se encarga de obtener los valores máximos de cada variable en el archivo de entrenamiento y los archivos de prueba.

Una vez obtenidos los máximos, otro módulo más proporciona el número de bits necesarios para soportar dichos valores. De esta forma no tiene que declararse un arreglo ixj de tamaño fijo y grande para la memoria del modelo, sino que el tamaño está dado por los valores máximos de las variables, y las dimensiones de la memoria son dinámicas.

También valida el tipo de datos del archivo de entrenamiento y verifica que no falten o sobren datos en el mismo. Tiene la opción de realizar lo anterior de dos formas: leyendo los datos anteriores de un archivo (el usuario determina la configuración deseada) o de forma automática (el programa asigna esta configuración).

- **Conversión a binario.** Realiza la transformación entero-binaria del archivo real de entrenamiento a un archivo binario. Es aquí donde las entradas y salidas, que en el archivo real de entrenamiento formaban una sola línea, se separan ahora por líneas, es decir, la entrada forma una línea binaria y la salida correspondiente forma la siguiente línea binaria. Este archivo binario se utiliza en la fase de aprendizaje.

- **Aprendizaje.** Ejecuta el proceso de aprendizaje basado en el archivo binario de entrenamiento y de la configuración asignada. Esta fase utiliza el aprendizaje por muestras alternadas, es decir, se entra al sistema una primera muestra al tiempo t_1; se entra la siguiente muestra a entrenar al tiempo t_2 aún cuando no se haya aprendido la anterior. Este proceso se repite hasta entrar todas las muestras por entrenar. Al terminar de entrar todas las muestras se vuelve a tomar la primer muestra y el proceso se repite hasta que el sistema aprenda todas las muestras.

El otro enfoque es entrar una misma muestra varias veces hasta que el sistema la aprenda y hasta entonces entrar la muestra siguiente. La primera forma demostró ser más rápida.

- **Recall.** Realiza la función de recall para cada muestra de prueba. Tiene la opción de leer las muestras de un archivo o de aceptarlas interactivamente. El formato para construir el archivo de pruebas de recall es igual al de el archivo de entrenamiento.

Cada vector de muestra lo traduce a un formato binario, ejecuta el recall, y luego lo traduce nuevamente al formato decimal para darlo como resultado. Dicho resultado lo escribe sobre un archivo de salida (opcionalmente). Este archivo de salida contiene además información adicional de la eficiencia total del recall para dicha prueba.

3.1.3 PRUEBAS

3.1.3.1 *12 VARIABLES DE ENTRADA*

En este bloque de pruebas se pretende pronosticar el nivel de ozono de una sola estación de monitoreo dadas sólo las 12 variables más significativas medidas en la misma estación. Las variables son: Mes, Día, Hora, DV, V-V, T, HR, SO_2, CO, NO_2, NO_x y O_3. Para esto se seleccionó un grupo de 32 muestras. De este grupo se seleccionaron a su vez 16 muestras que eran similares a las restantes 16 muestras. El sistema fue entrenado con las primeras 16 muestras y probado con las otras16. **El recall de las 16 muestras entrenadas fue de 100%.**

Fueron seleccionadas 11 muestras no entrenadas como muestras de prueba. La figura 5 muestra el resultado de esta prueba de recall.

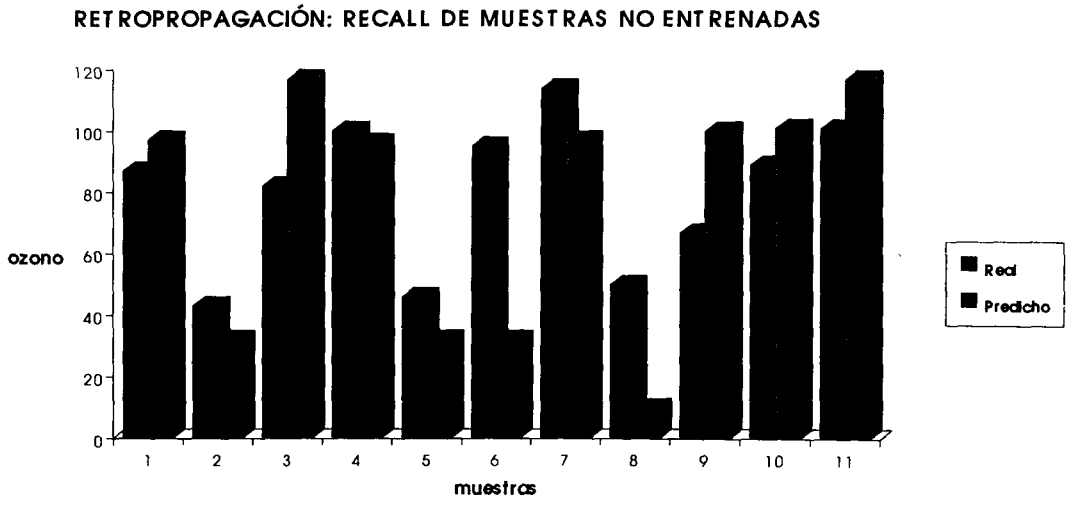


Figura 5 Recall de 11 muestras no entrenadas dado un entrenamiento de 16 muestras (12 variables de entrada).

La eficiencia del recall anterior alcanza el 73% bajo un margen de error de +/-30 imecas. Para esta prueba, el número de muestras con un error de recall de más de 50 imecas es de 2 (18%). El ejemplo anterior sólo se entrenó con 16 muestras.

La figura 6 ilustra gráficamente el recall de las mismas 11 muestras de prueba y el mismo entrenamiento de 16 muestras, solo que cada una de las muestras de prueba contiene ahora un 20% de información incompleta. La introducción de dos valores cero por muestra (20%) fue hecha de forma aleatoria. Lo anterior se introdujo para probar la robustéz del modelo en presencia de ruido.

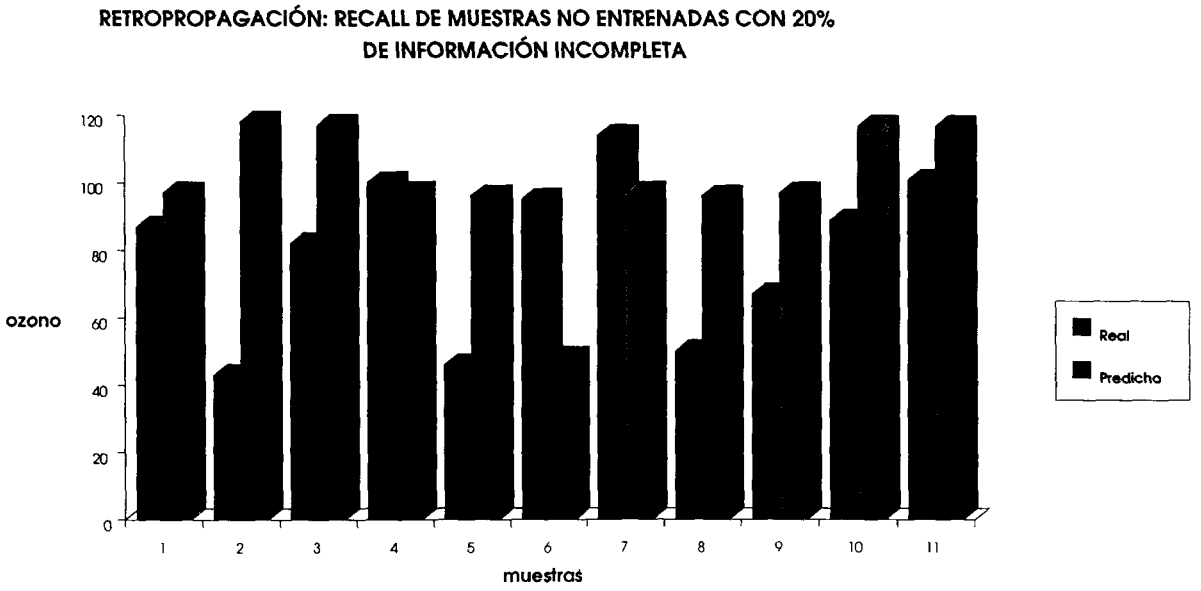


Figura 6 Recall de 11 muestras no entrenadas con dos variables de valor cero. El entrenamiento fue de 16 muestras.

La eficiencia del recall fue de 55% (disminuyó un 25% con respecto al recall anterior). Además el número de muestras con error de recall de más de 50 imecas fue de 4 (aumentó un 50%). También la media del error de recall de las muestras dentro del rango +/-30 imecas aumentó un 25%.

La prueba siguiente muestra la distorsión que sufre la función de recall de muestras entrenadas a medida que se agregan variables con valor cero a dichas muestras (producto de la ausencia de medición).

La figura 7 presenta el recall de 16 muestras entrenadas con 20% de información incompleta. Ha de aclararse que el recall de las 16 muestras entrenadas anteriores con un 10% de información incompleta (sólo una variable de valor cero por muestra) fue de 100%.

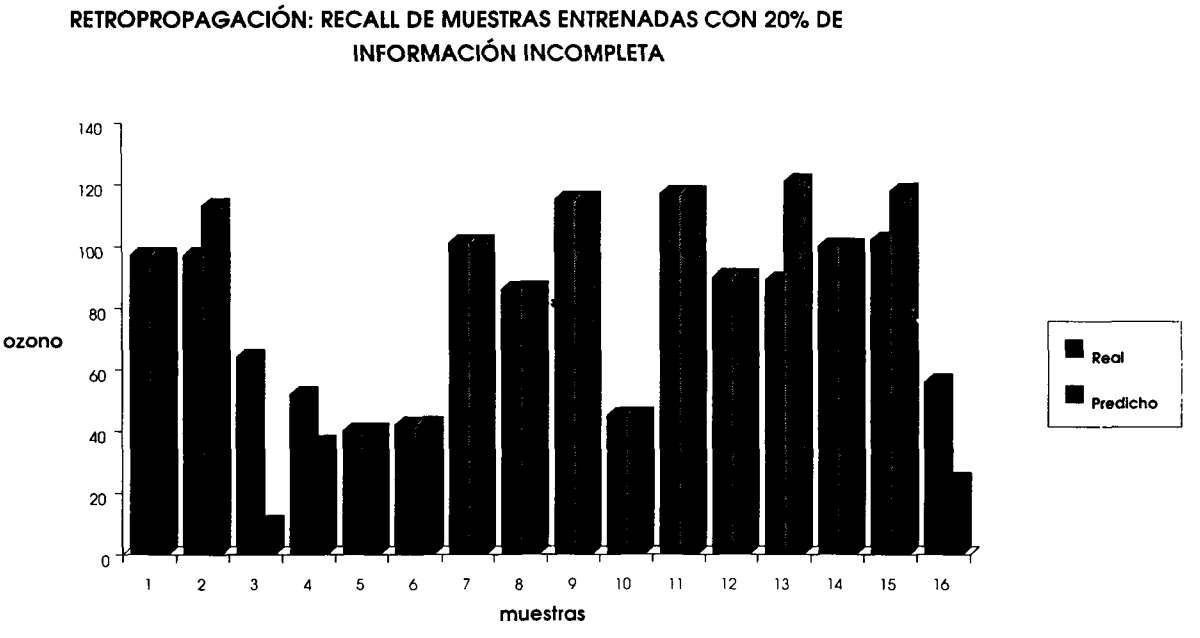


Figura 7 Recall de 16 muestras entrenadas con dos variables de valor cero. El entrenamiento fue de 16 muestras.

Como puede observarse, la eficiencia del recall fue de 88% en el rango de +/-30 imecas. El número de muestras con error del recall de más de 50 imecas fue de 1 (6%).

El modelo de retropropagación presenta una gran diferencia entre el tiempo de aprendizaje y el tiempo de recall. A medida que aumenta el número de muestras de entrenamiento, el tiempo de aprendizaje se eleva considerablemente. No así el tiempo de recall, que permanece exactamente igual. La Figura 8 ilustra esta diferencia para una prueba de entrenamiento y de recall con 16 muestras.

La Figura 9 ilustra similarmente los tiempos crecientes de aprendizaje para entrenamientos que contenían desde 4 hasta 16 muestras. Es claro que mientras aumenta el número de muestras de entrenamiento el tiempo de aprendizaje crece de forma incontrolable.

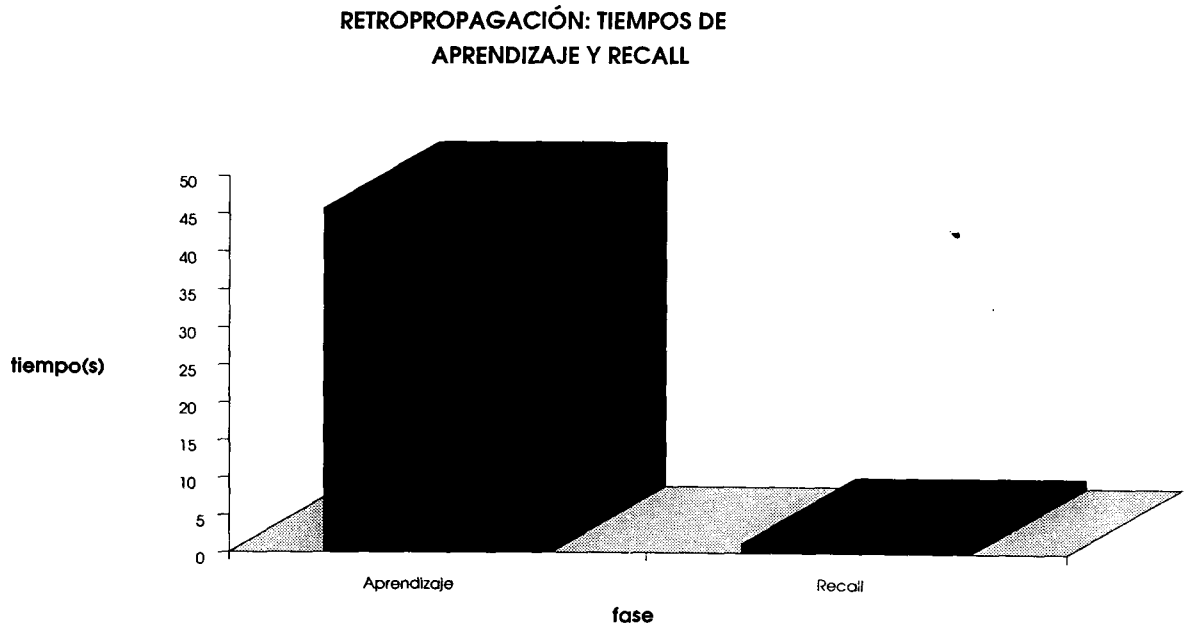


Figura 8 Tiempos de aprendizaje y de recall

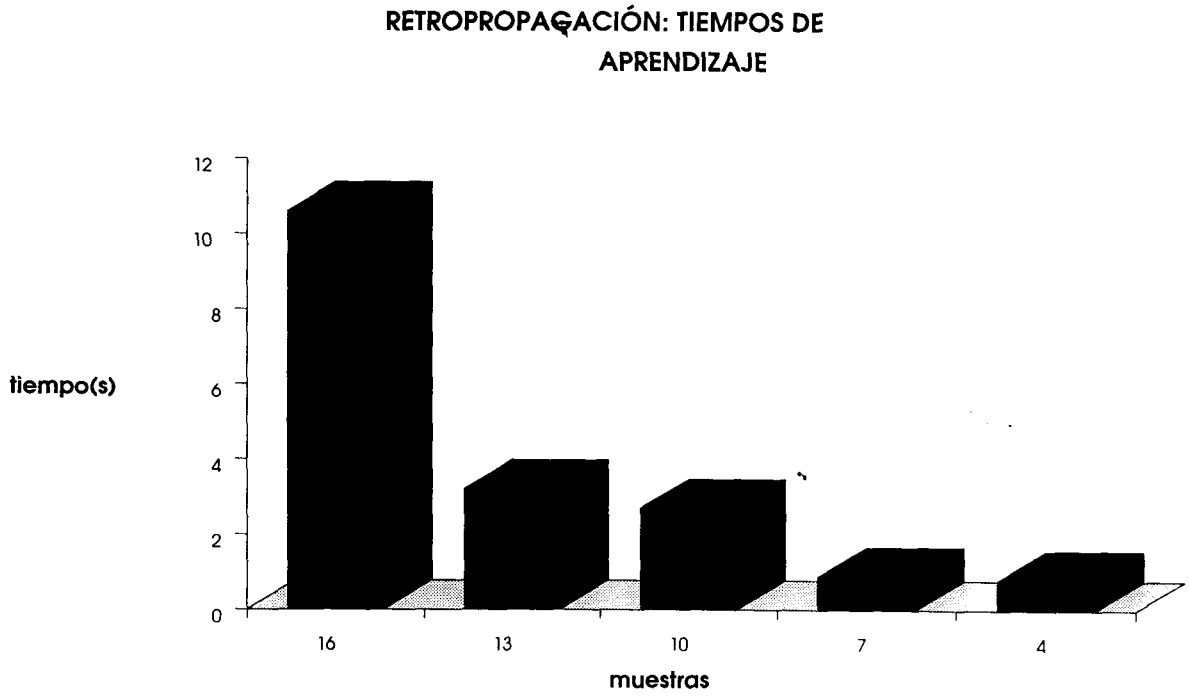


Figura 9 Comparación de tiempos de aprendizaje.

El valor del coeficiente de aceleración tiene gran trascendencia en el tiempo de aprendizaje total del sistema. Generalmente este coeficiente toma valores en el rango 0.1-0.3; sin embargo, dentro de este mismo rango, la diferencia en tiempo de aprendizaje puede ser bastante significativa. La figura 10 presenta esta comparación.

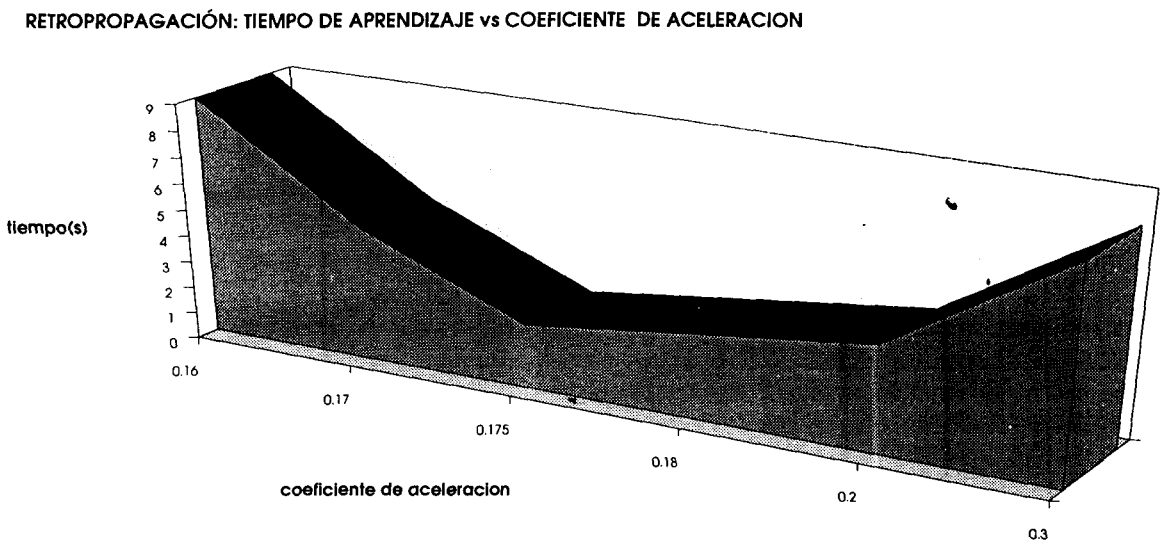
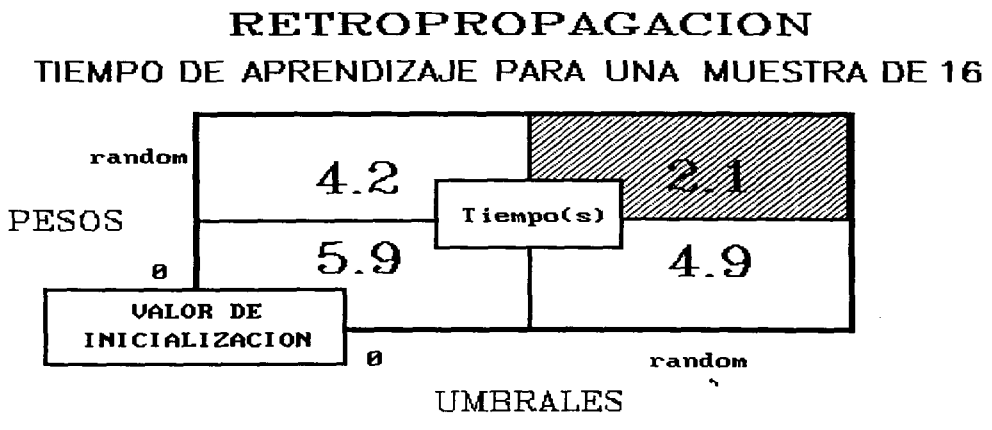


Figura 10 Tiempos de aprendizaje de 16 muestras con diferentes coeficientes de aceleración (12 variables de entrada).

Por otro lado, la inicialización adecuada del valor de las conexiones y de los umbrales (generalmente valores aleatorios entre +/-0.5) es significativa en el tiempo de aprendizaje. La tabla 1 ilustra claramente el tiempo de aprendizaje para una prueba de 16 muestras no entrenadas dadas las diferentes combinaciones de valores de inicialización.



T

ab. 1 Tiempos de aprendizaje para las diferentes combinaciones de inicialización.

3.1.4 CONSIDERACIONES PRÁCTICAS [1]

Generalmente no es necesario entrenar la red con todo el conjunto de muestras de entrenamiento, sino sólo con algunas de ellas. El resto puede constituir un conjunto de prueba. Además, es conveniente entrenar la red con muestras que contengan ruido dado que esto facilita la convergencia de la red. Es también aconsejable entrenar la red con muestras alternadas (el significado de **alternadas** ya se explicó anteriormente) de forma

que en un corto tiempo de aprendizaje todas las muestras por entrenar hayan sido presentadas al sistema al menos una vez.
El parámetro η es el coeficiente de aceleración para la velocidad de convergencia de la red. Los valores de este parámetro cercanos a 1 producen un retardo en la solución del sistema.

Gráficamente esto sucede debido a que se generan grandes saltos que abandonan alguna solución posible drásticamente (si la solución se encontró en algún momento). Por el contrario, valores pequeños de este parámetro disminuyen la rapidez de convergencia del sistema, pero es más probable que se encuentre una mejor solución.

Es práctica común que la selección de los pesos iniciales sea de forma aleatoria en el rango +/-0.5. Los umbrales suelen considerarse como otro peso conectado a una unidad ficticia y de valor de conexión 1. Durante este trabajo fueron inicializados de forma aleatoria bajo el rango de +/- 0.5.

3.2 MEMORIA ASOCIATIVA BIDIRECCIONAL (BAM)

3.2.1 SOLUCIÓN

Algunas pruebas piloto mostraron que la BAM presentaba una gran tolerancia al ruido (ideal para esta aplicación) pero una limitada capacidad de almacenamiento. Esta desventaja se superó por medio de la construcción de un sistema BAM que contiene un conjunto de subBAM´s que hacen, todas juntas, las veces de una gran memoria. Lo anterior se pudo realizar única y exclusivamente sobre este modelo gracias a su cualidad bidireccional de regeneración de patrones asociados.

El sistema BAM está constituído por un conjunto de subBAM's en donde cada una de ellas se encarga de aprender cierta cantidad de asociaciones. Los patrones de entrada son vectores bipolares de las variables que intervienen en la química del ozono y los patrones de salida corresponden a los valores del nivel de ozono medido. Un patrón de entrada y otro de salida forman una asociación.

El número de subBAM´s que el sistema requiere es directamente proporcional al total de muestras a entrenar, ya que cada subBAM almacena el mismo número de asociaciones, en este caso de 3. De la misma forma, el tiempo de aprendizaje y el tiempo de recall por muestra son proporcionales al tamaño de los archivos a entrenar (total de asociaciones).

La forma en que opera el recall por muestra es la siguiente: ante un estímulo o vector de entrada A, el sistema genera tantos vectores de salida B_k como subBAM's contiene el sistema. De todo este conjunto de vectores de salida sólo un vector B_i es el más adecuado, debido a que el producto interno de A y A_i (vector asociado de B_i) es el de mayor magnitud (el producto interno de un vector bipolar consigo mismo es igual a la dimensión

de éste; así, un vector A_i más cercano en distancia de Hamming a un vector de entrada A generará el producto interno de mayor magnitud).
Dicho vector B_i es la respuesta del sistema . La figura 11 ilustra gráficamente el modo del recall en el Sistema BAM.

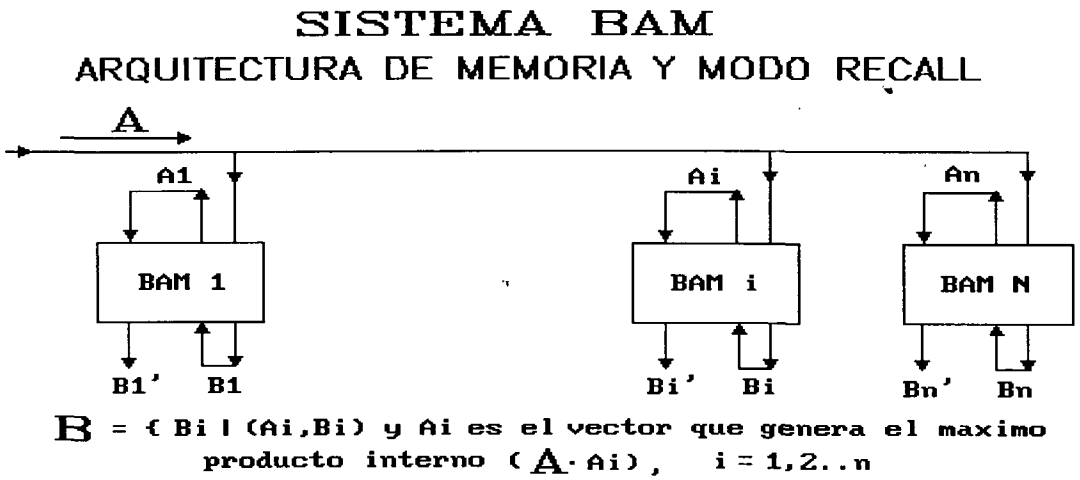


Figura 11 Esquema genérico del modo del recall en el Sistema BAM.

3.2.2 SISTEMA BAM

Se desarrolló un programa que contiene todas las funciones necesarias para llevar a cabo, de manera automática, el proceso de pronóstico de nivel de ozono por medio del modelo de la memoria asociativa bidireccional (BAM). Lo único que requiere el sistema es el archivo de entrenamiento y los archivos de prueba. El código fuente está escrito en lenguaje C y corre sobre cualquier PC. La figura 12 muestra el diagrama de módulos del programa.

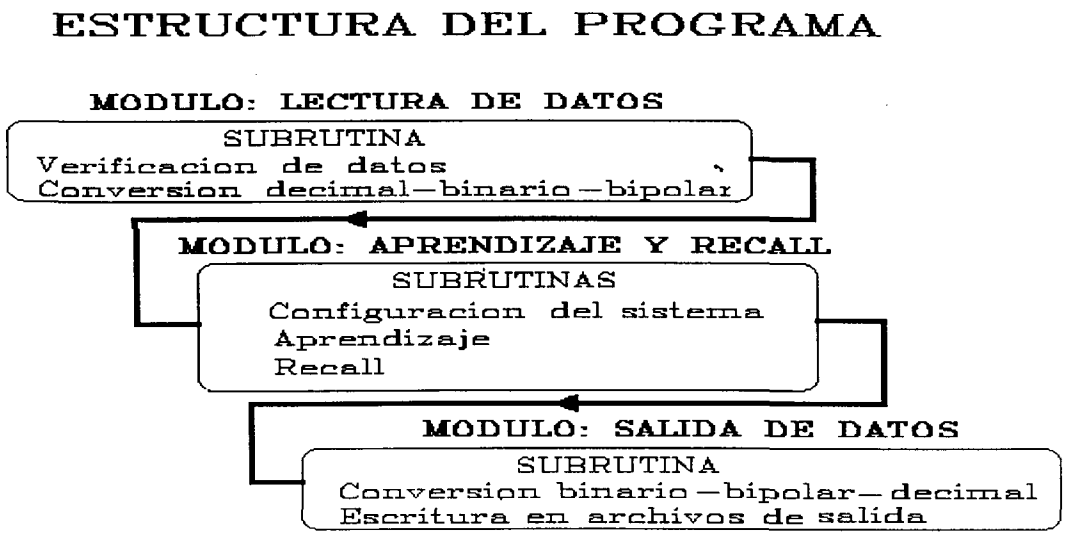


Figura 12 Diagrama de módulos del Sistema BAM.

FORMATO DE LOS ARCHIVOS. Los archivos de entrenamiento y de pruebas deben estar en formato texto. Cada línea de archivo debe contener tanto los datos de entrada como los datos de salida en este orden.

Los principales módulos del programa son los siguientes:

- **Configuración.** Se encarga de generar, a partir del archivo de entrenamiento, la configuración nxm mínima suficiente de la red de tipo BAM para que sea posible entrenar todas las muestras. Para generar la configuración mínima suficiente, un módulo se encarga de obtener los valores máximos de cada variable en el archivo de entrenamiento y los archivos de prueba. Una vez obtenidos los máximos, otro módulo más proporciona el número de bits necesarios para soportar dichos valores. De esta forma no tiene que declararse un arreglo ixj de tamaño fijo y grande para la memoria del modelo, sino que el tamaño está dado por los valores máximos de las variables y las dimensiones de la memoria son dinámicas.

También valida el tipo de datos del archivo de entrenamiento y verifica que no falten o sobren datos en el mismo. El proceso de configuración lo realiza de dos formas: leyendo los datos anteriores de un archivo (el usuario determina la configuración deseada) o de forma automática (el programa asigna esta configuración).

- **Conversión a binario.** Realiza la transformación entero-binaria-bipolar del archivo real de entrenamiento a un archivo binario-bipolar. Es aquí donde las entradas y salidas, que en el archivo real de entrenamiento formaban una sola línea, se separan ahora por líneas, es decir, la entrada forma una línea binaria-bipolar y la salida correspondiente forma la siguiente línea binaria-bipolar. Este archivo binario-bipolar se utiliza en la fase de aprendizaje.

- **Aprendizaje.** Ejecuta el proceso de aprendizaje basado en el archivo binario-bipolar de entrenamiento y de la configuración asignada. Este módulo genera cada una de las subBAM´s (para este caso cada subBAM aprende 3 muestras) que conforman la memoria del sistema.

- **Recall.** Realiza la función de recall para cada muestra de prueba. Tiene la opción de leer las muestras de un archivo o de aceptarlas interactivamente. El formato para construir el archivo de pruebas de recall es igual al de el archivo de entrenamiento. Cada vector de muestra lo traduce a un formato binario-bipolar, ejecuta el recall, y luego lo traduce nuevamente al formato decimal para dar el resultado. Este resultado lo escribe sobre un archivo de salida (opcionalmente). Este archivo de salida contiene además información adicional de la eficiencia total del recall para dicha prueba.

3.2.3 PRUEBAS

3.2.3.1 *12 VARIABLES DE ENTRADA*

El fin de este primer bloque de pruebas es predecir el nivel de ozono de una sola estación de monitoreo dadas las variables medidas en ésta. Sólo se incluyeron las 12 variables más significativas. Estas son: Mes, Día, Hora, DV, V-V, T, HR, SO_2, CO, NO_2, NO_x y O_3.

El sistema se entrenó con 100 muestras seleccionadas aleatoriamente. **El recall de las 100 muestras entrenadas fue de 100%.**

Fueron seleccionadas 14 muestras no entrenadas como muestras de prueba. La figura 13 muestra el resultado de esta prueba de recall.

Figura 13 Recall de 14 muestras no entrenadas dado un entrenamiento de 100 muestras (12 variables de entrada).

La gráfica anterior muestra claramente que sólo el 42% de las muestras anteriores tuvieron un recall satisfactorio dentro de un rango de +/-30 imecas. Por otro lado, el número de muestras cuyo error de recall fue mayor a 50 imecas asciende a 7 (50%).

La prueba siguiente utiliza el mismo archivo de entrenamiento de 100 muestras pero ahora 13 muestras de prueba seleccionadas con un criterio diferente. El criterio fue seleccionar un grupo de 26 muestras tal que las primeras 13 fueran similares a las 13 restantes. Sólo la mitad se entrenó. El resultado de este recall lo ilustra la figura 14.

Figura 14 Recall de 13 muestras no entrenadas dado un entrenamiento de 100 muestras (12 variables de entrada).

Si se observa la figura anterior podrá notarse que la eficiencia del recall mejoró, aún cuando el número de muestras de prueba disminuyó (13). Un 77% de las muestra anteriores tuvieron un recall satisfactorio dentro del rango +/-30 imecas (mejoró un 83% respecto del anterior).

Además, el número de muestras cuyo error de recall fue mayor a 50 imecas fue de sólo 3 (mejoró un 133%). Aún más, la media del error de recall de las muestras con recall satisfactorio mejoró un 50%.

Lo anterior parece indicar que si las muestras de prueba son seleccionadas de manera aleatoria es probable que la eficiencia del recall sea un tanto menor que cuando se selecciona un grupo tal, que dos subgrupos de éste, de igual número de muestras, sean similares. Esto es un tanto evidente y afortunadamente no caótico; es decir, si varias muestras presentan un nivel de ozono similar y entrenamos al sistema con un subgrupo de estas muestras, es correcto suponer que las pruebas con las muestras no entrenadas generen niveles de ozono parecidos a aquellos de las muestras entrenadas (ya que el grupo original presentaba un ozono similar en todas sus muestras) si es que existe alguna interrelación, hasta cierto punto predecible, entre las variables que toman parte en la química del ozono.

Si dicha interrelación no existe, o sea, la producción de ozono es de forma caótica, no habrá sistema alguno capaz de predecir dichos niveles. Afortunadamente las pruebas demuestran que sí existe cierto grado de interrelación.

Otro tipo de pruebas interesantes son aquellas en donde puede apreciarse el grado de distorsión de la función de recall de muestras que contienen cierto porcentaje de información incompleta. Esto se debe a la ausencia previa de medición de dicho valor. A continuación se presentan un par de gráficas que muestran el deterioro de la función de recall a medida que se agregan variables con valor cero en las muestras de prueba.

La figura 15 presenta la eficiencia de la función de recall para una prueba de 13 muestras (las mismas del ejemplo anterior) no entrenadas con una variable de valor cero cada una (10% de información incompleta). El entrenamiento se realizó con 100 muestras.

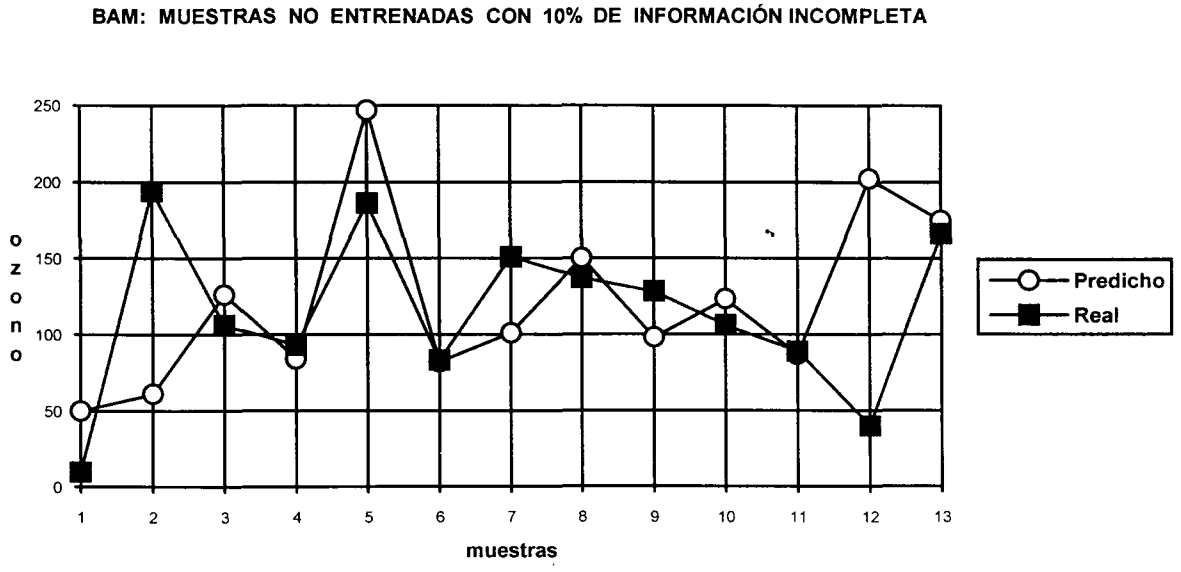


Figura 15 Recall de las muestras del ejemplo anterior con una variable de valor cero cada muestra (12 variables).

La eficiencia del recall ha sido de 61% (disminuyó un 20%) en el rango de +/-30 imecas. A pesar de que la media del error de recall de las muestras de recall satisfactorio no se modificó significativamente (apenas alcanzó el 15%), el número de muestras con error de recall de más de 50 imecas ascendió a 4 (aumentó un 33%).

Una prueba más nos ratifica el deterioro del recall cuando las muestras de prueba no entrenadas contienen un número creciente de variables con valor cero. La figura 16 ilustra la función de recall para las mismas 13 muestras, solo que ahora cada muestra contiene dos variables con valor cero (20% de información incompleta).

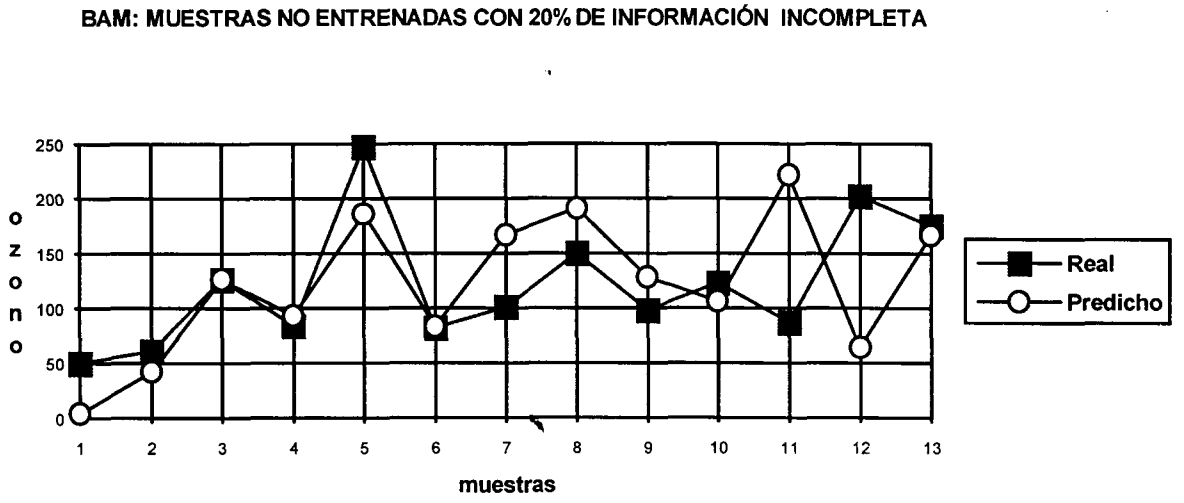


Figura 16 Recall de las mismas 13 muestras pero con dos variables de valor cero cada una (12 variables de entrada).

Para esta prueba la eficiencia del recall fue de tan sólo 53% (cayó un 13% más), y el deterioro total del recall es de 31% con respecto a la eficiencia original del 77%. El número de muestras con error de recall de más de 50 imecas es ahora de 5 (aumentó un 66% con respecto al original). Lo anterior sugiere que a medida que las muestras contienen un número creciente de variables con valor cero la eficiencia del recall disminuye.

Como es claro, la disminución de la eficiencia del recall depende primeramente del número de variables con valor cero que contengan las muestras de prueba, y segundo, de la cantidad de muestras entrenadas y de la distribución de sus valores. Lo anterior significa que a medida que se agregan variables con valor cero a las muestras, la eficiencia del recall disminuye. Además, si el número de variables con valor cero en cada una de las muestras es constante, a medida que se aumenta el tamaño del archivo de entrenamiento (mayor número de muestras) disminuye también la eficiencia del recall.

Un par de pruebas más ilustran y complementan la gran capacidad de tolerancia al ruido del modelo BAM para 12 variables de entrada y una variable de salida. La figura 17 muestra la función de recall para 14 muestras entrenadas y seleccionadas aleatoriamente, con 6 variables de valor cero cada una (50% de información incompleta). El entrenamiento se realizó con 100 muestras.

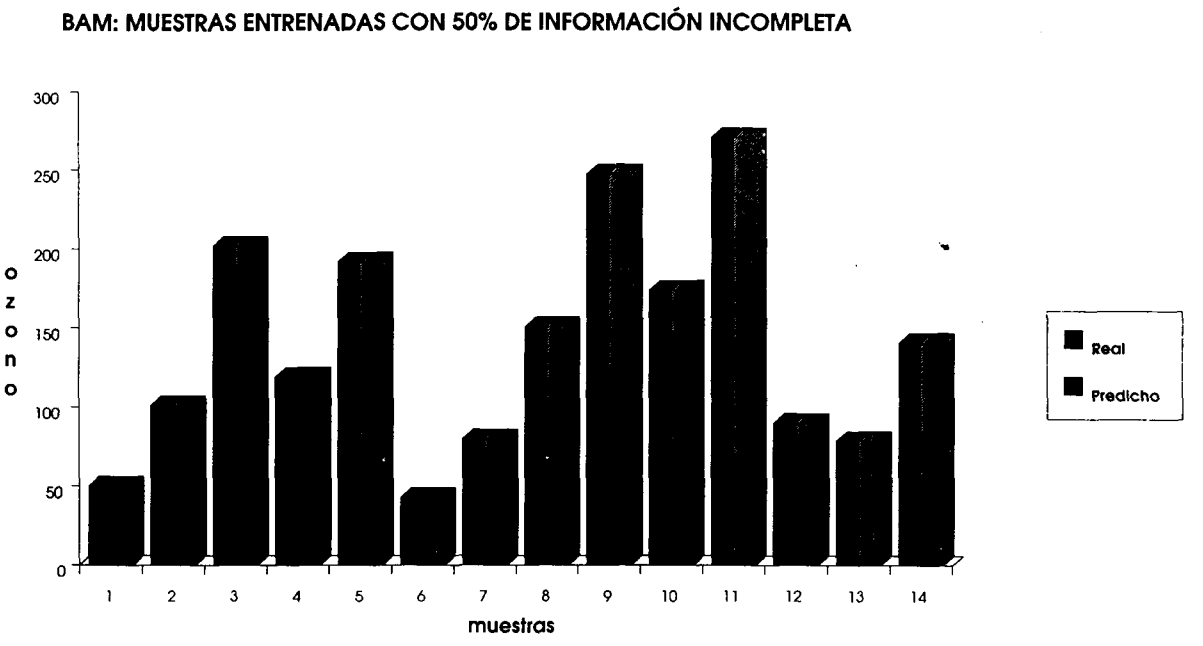


Figura 17 Recall de 14 muestras entrenadas con 6 variables de valor cero. Entrenamiento de 100 muestras (12 variables).

Como puede observarse el recall es de 100%. La información incompleta en cada una de las muestras anteriores alcanza el 50%. La eficiencia del recall para este caso dependerá claramente del número de muestras totales de entrenamiento (en pruebas con muestras que contengan variables de valor cero, a mayor número de muestras entrenadas menor la tolerancia al ruido). La figura 18 presenta el recall para las muestras del ejemplo anterior, ahora con 7 variables de valor cero por muestra (58% de información incompleta).

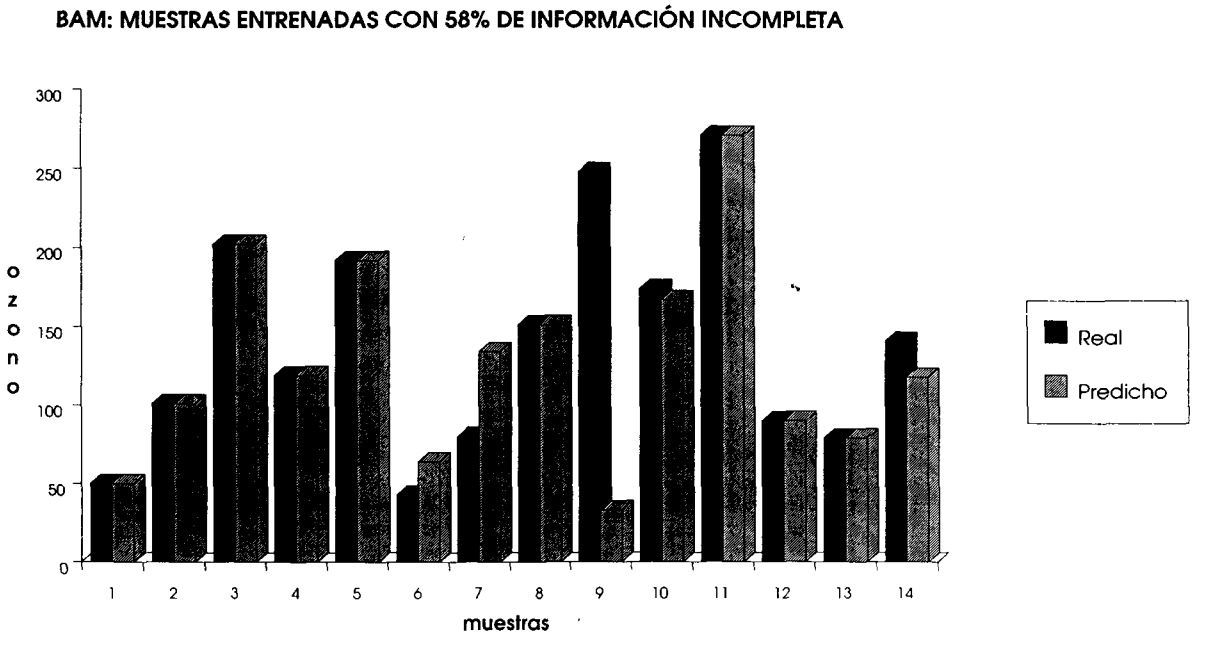


Figura 18 Recall de las muestras anteriores ahora con 7 variables de valor cero (12 variables de entrada).

El recall ha disminuido al 85% (se decrementó 15%). Un 7% de muestras tiene un error de recall de más de 50 imecas. El error medio del recall de las muestras en el rango +/-30 imecas ahora es del 8% (contra el 0% anterior). Como se mencionó antes, a medida que aumenta el número de variables con valor cero en las muestras, la eficiencia del recall disminuye.

3.2.3.2 *18 VARIABLES DE ENTRADA*

En este segundo bloque de pruebas se incluyeron las 9 variables de dos estaciones de monitoreo para pronosticar el nivel de ozono de sólo una de ellas, buscando alguna posible correlación entre sus valores. No se incluyó el Mes, Día y Hora. El sistema fue entrenado con 267 muestras. **El recall de las 267 muestras entrenadas fue de 100%.**

El primer tipo de prueba que para este segundo grupo de muestras se presenta es la concerniente a la variabilidad de la eficiencia del recall de acuerdo al número de muestras entrenadas. La figura 19 presenta la función de recall para una prueba de 21 muestras no entrenadas dado entrenamientos previos de 60, 180, y 267 muestras.

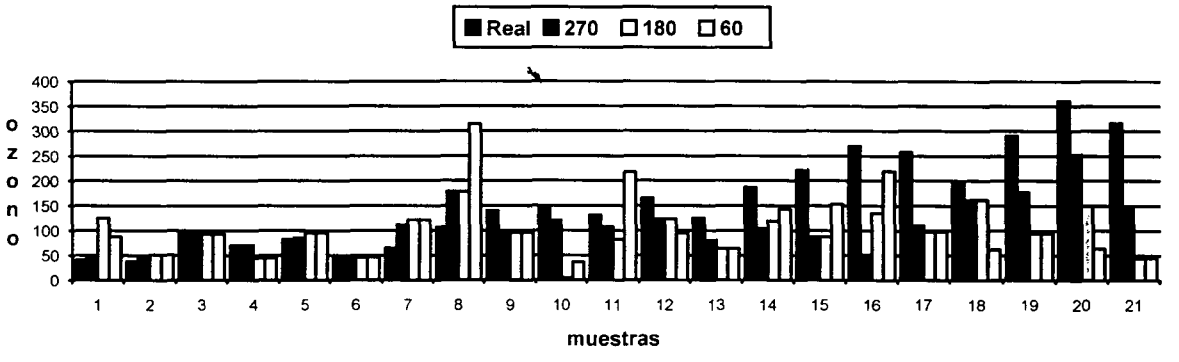


Figura 19 Recall de 21 muestras no entrenadas y diferentes tamaños en el entrenamiento (18 variables de entrada).

Es importante hacer notar que el criterio de selección de las muestras de prueba no entrenadas fue diferente para los entrenamientos con 60, 180 y 267 muestras respectivamente. Este criterio se basó en la cantidad de muestras con ozono similar, donde una parte de estas se entrenaba y la otra parte era para prueba.

Para la prueba con 60 muestras entrenadas se tuvo un recall de 23% en el rango +/-30 imecas. El número de muestras con error de recall de más de 50 imecas fue de 14 (66%). Para la prueba con 180 muestras entrenadas el recall fue de 38% (hubo un aumento de 60% con respecto al anterior).

Una tercera prueba de las mismas muestras, ahora dado un entrenamiento de 267 muestras, arrojó un recall de 86% en el rango +/-30 imecas (aumentó la eficiencia un 72% respecto del peor recall). El número de muestras con error de recall de más de 50 imecas fue sólo de 3 (disminuyó un 79%).

Una vez más las gráficas siguientes presentan la distorsión que sufre la función de recall a medida que las pruebas de muestras entrenadas agregan variables con valor cero o a medida que crece el número de muestras de entrenamiento, aún cuando el número de variables con valor cero sea pequeño. La figura 20 ilustra el recall de una prueba de 10 muestras entrenadas con 10 variables de valor cero (60% de información incompleta), dado un entrenamiento de 180 muestras.

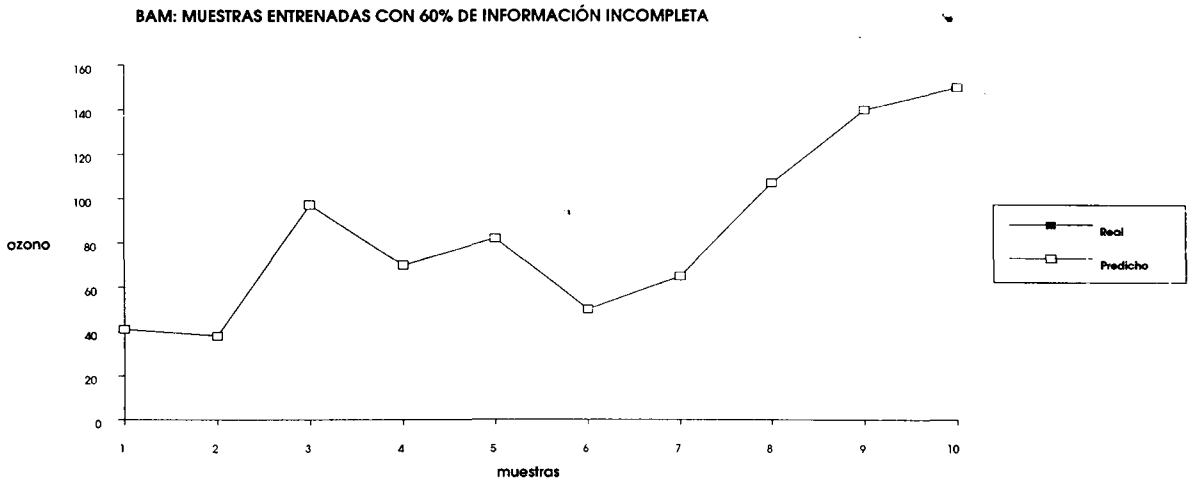


Figura 20 Recall de 10 muestras entrenadas con 10 variables de valor cero dado un entrenamiento de 180 muestras.

Como puede apreciarse el recall es de 100%. Esta vez el ruido total en las muestras de prueba fue de 60%. La figura 21 muestra ahora la misma prueba pero para un total de 267 muestras entrenadas (un aumento de 48% en el número de muestras entrenadas). Las muestras de prueba tienen el mismo porcentaje de información incompleta anterior (60%).

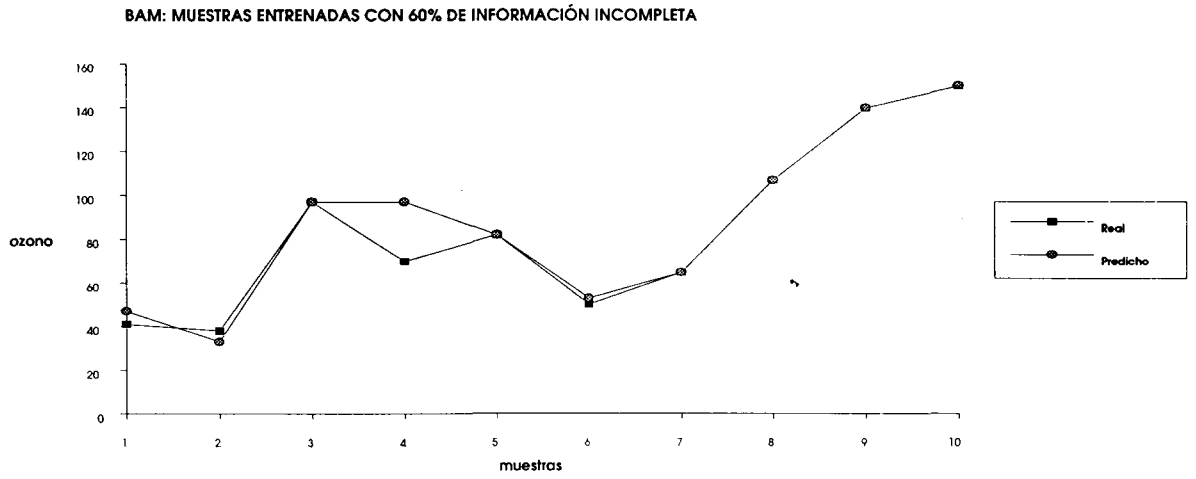


Figura 21 Recall de 10 muestras entrenadas con 10 variables de valor cero dado un entrenamiento de 267 muestras.

La eficiencia de este recall ha sido de 90% (diminuyó un 10%). Obsérvese que la máxima distorsión que sufre la función de recall, dado un entrenamiento de 267 muestras y en una prueba de 10 muestras entrenadas con 60% de información incompleta, es de sólo 10%.

3.2.3.3 *46 VARIABLES DE ENTRADA*

Para este tercer bloque de pruebas los 9 parámetros de las 5 estaciones de monitoreo más completas se incluyeron para pronosticar el nivel de ozono de cada una de las estaciones del sur (Cerro de la Estrella y Pedregal). Esto hace un total de 45 variables, más el Mes y el Día, menos la variable Ozono de la estación a pronosticar (46 variables en total). Lo anterior es debido a que en la mayor parte de las ocasiones los vientos circulan de norte a sur, y encontrar alguna posible correlación en este sentido sería más probable. Se incluyeron las variables Mes y Día.

Es importante notar que para el pronóstico del nivel de ozono de la estación, digamos, Pedregal, las variables de ésta se incluyeron en la entrada al sistema pero no el valor del ozono (pués se buscaba pronosticar dicho valor y no proporcionarlo como entrada). No se incluyó la variable Hora. El sistema fue entrenado con 430 muestras que contenían dos variables con valor cero (4% de información incompleta). **El recall de las 430 muestras entrenadas fue de 100%.**

La prueba siguiente ilustra el recall de 50 muestras no entrenadas dado un entrenamiento de 430 muestras y 4% de información incompleta. La selección de muestras de prueba se llevó a cabo de forma aleatoria. Cada muestra contiene 46 variables de entrada. Obsérvese que el entrenamiento con 430 muestras implica ya un almacenamiento de 5 meses de monitoreo si consideramos, por ejemplo, tres muestras por día.

La siguiente prueba muestra el recall de 50 muestras no entrenadas dado un entrenamiento total de 430 muestras. La selección fue también realizada de forma aleatoria. El fin de esta primer prueba es pronosticar el nivel de ozono de la estación Pedregal en un tiempo t dados a diferentes valores en el resto de las estaciones al tiempo t-1. Lo anterior significa que la prueba confirmaría o no la relación directa entre el nivel de ozono medido en el sur de la cuidad a un tiempo t, con respecto a los valores de ozono medidos en las estaciones restantes al tiempo t-1. La figura 22 muestra el resultado de esta prueba.

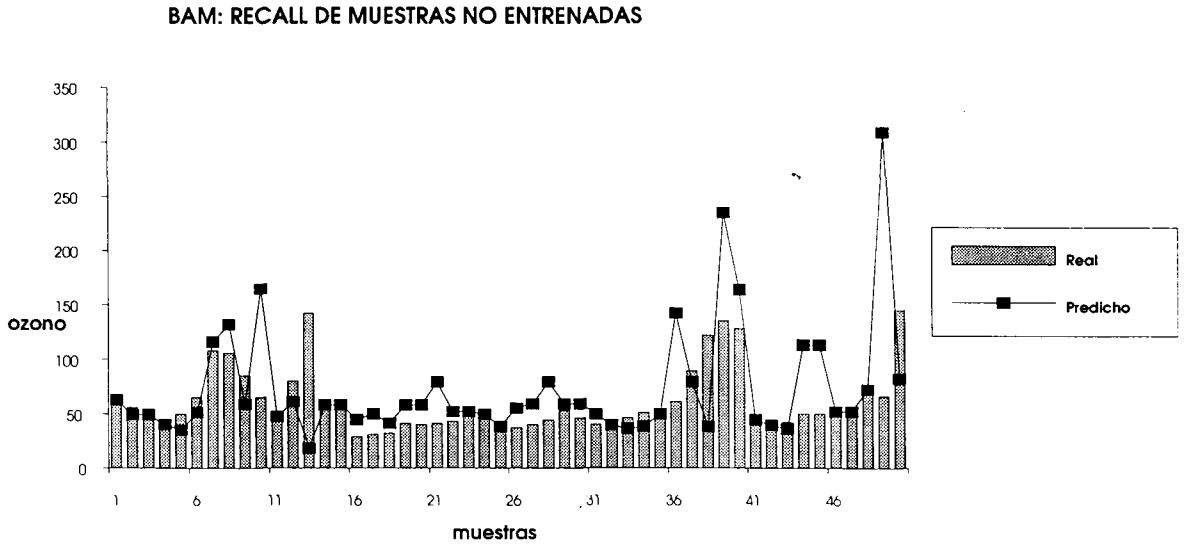


Figura 22 Recall de 50 muestras no entrenadas dado un entrenamiento de 430. Pronóstico de la estación Pedregal.

La figura anterior permite observar de forma clara el índice de eficiencia de la función de recall, que es de 78% en el margen de +/-30 imecas. En esta misma gráfica, si consideráramos un rango de error de +/-40 imecas como satisfactorio, el recall sería de 82%, y si el rango fuera de +/-10 imecas, lo sería de 60%. Sólo un 18% de las muestras (9) tuvieron un error de recall de más de 50 imecas.

De acuerdo a los resultados de estas pruebas puede suponerse la existencia de cierta correlación entre los niveles de ozono de estaciones diferentes al Pedregal y ésta. La realidad confirma un tanto esta hipótesis ya que efectivamente existen corrientes de viento que van de norte a sur, y no al contrario, en la mayor parte de las ocasiones.

Otras pruebas de pronóstico de ozono en estaciones del norte, dados niveles de ozono en estaciones del sur, mostraron un recall deficiente (apenas un 35%). Esto se puede constatar realmente dado que sólo en pocas ocasiones existen corrientes de viento que van de sur a norte.

El resultado de una prueba más para pronóstico del nivel de ozono de la estación Cerro de la Estrella se muestra gráficamente en la figura 23. La prueba se realizó con 50 muestras no entrenadas y un entrenamiento previo de 430 muestras. La selección de las muestras de prueba fue de forma aleatoria. Nuevamente, las variables del resto de las estaciones intervienen como variables de entrada para el pronóstico del nivel de ozono de la estación Cerro de la Estrella.

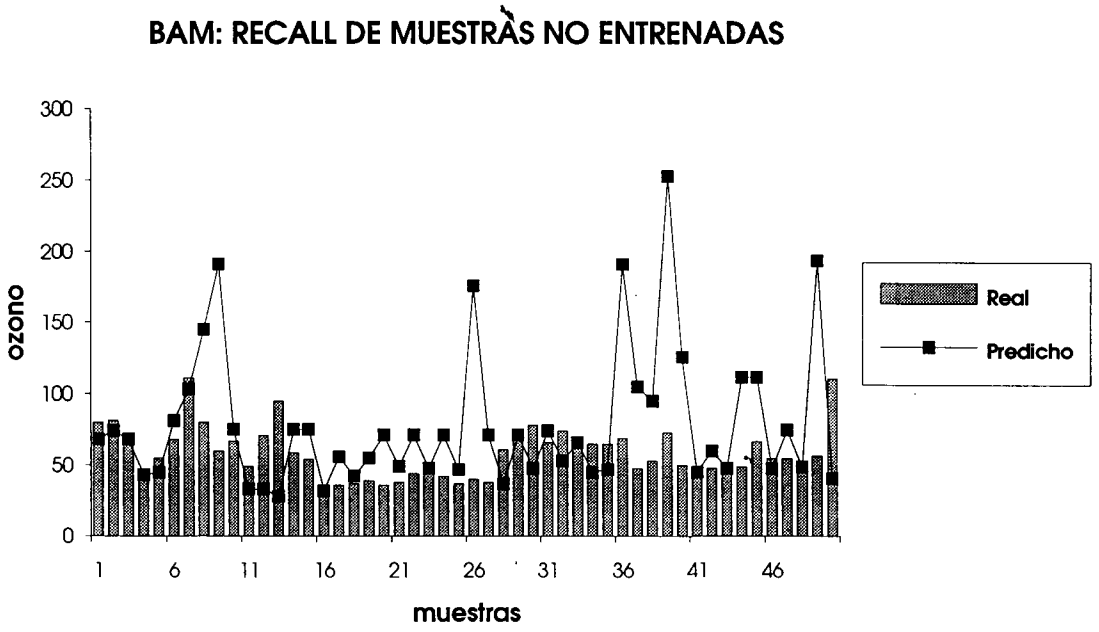


Figura 23 Recall de 50 muestras no entrenadas. Entrenamiento de 430. Pronóstico de la estación Cerro de la Estrella.

El recall que muestra la gráfica anterior es de casi 73% en el rango +/-30 imecas. Al igual que antes, si el rango de error satisfactorio fuera de +/-40 imecas la eficiencia del recall sería de 78%, y si lo fuera de sólo +/-10 imecas, la eficiencia sería de 50%. El error de recall de más de 50 imecas se elevó a 24% (12 muestras).

La relación que existe entre el número de muestras de entrenamiento y el tiempo de aprendizaje de la BAM se ilustra gráficamente en la figura 24. En esta figura se presentan los tiempos de aprendizaje para las tres configuraciones de cada uno de los bloques de prueba. Dichas configuraciones estan indicadas en la forma nxm, donde n = número de neuronas de entrada, y m = número de neuronas de salida mínimas suficientes. El número de muestras entrenadas es de 100 para cada uno de los tres casos.

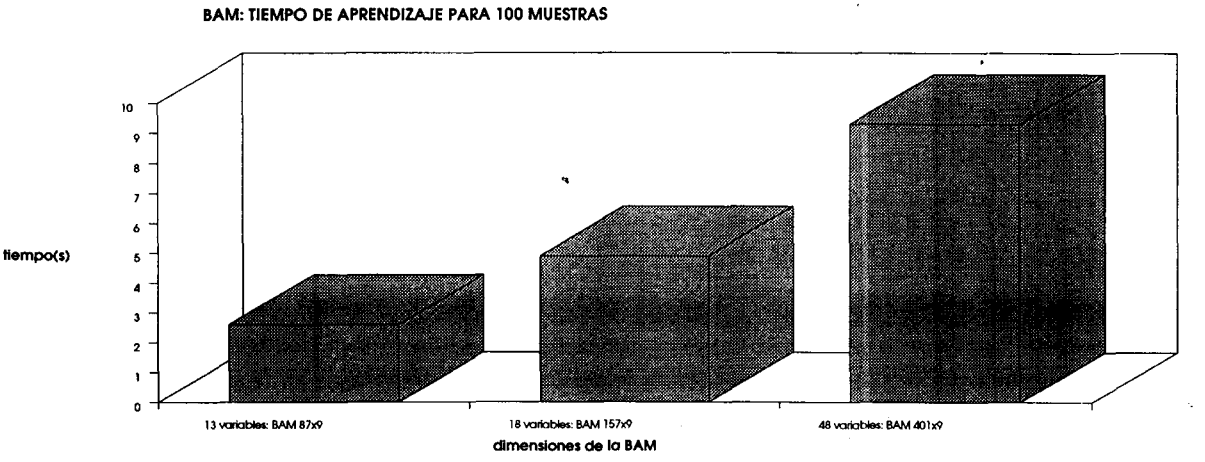


Figura 24 Comparación de tiempos de aprendizaje para cada una de las diferentes configuraciones de prueba.

La razón del aumento del tiempo de aprendizaje es debido al aumento de la dimensión de la matriz M (p. ej., si una matriz M de dimensión 100x10 memoriza 100 muestras en 3s, otra matriz M de dimensión 200x20 aprenderá 100 muestras en 12s).

Una última gráfica nos muestra los tiempos de recall de cada una de las configuraciones de los distintos bloques de prueba. Estos tiempos de recall por muestra guardan igualmente una relación líneal contra el número de muestras de entrenamiento (ya que la entrada de una nueva muestra provoca el llamado del total de subBAM's para obtener la mejor solución). Es también un tanto intuitivo pensar que, al igual que lo anterior, los tiempos de recall dependan de la dimensión de la matriz M. La figura 25. ilustra gráficamente lo anterior.

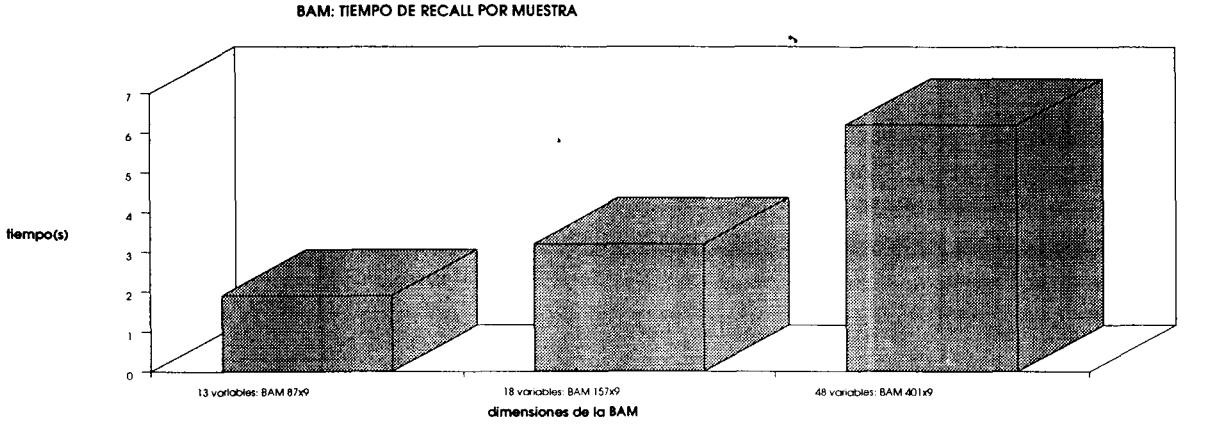


Figura 25 Comparación de tiempos de recall para cada una de las diferentes configuraciones de prueba.

Como ha podido observarse a lo largo de estas pruebas, en cualquiera de los tres grupos de pruebas la eficiencia del recall ha sobrepasado el 77% (12 variables: 77%, 18 variables: 86%, 47:variables: 82%), dados entrenamientos respectivos de 100, 267 y 430 muestras. Los resultados anteriores fueron medidos en el rango de error de recall de +/-30 imecas.

Si consideramos que el número de muestras es aún pequeño (comparado con el volumen de datos anual con que este sistema debiera entrenar) los resultados son alentadores. Pruebas contundentes debieran entrenar con al menos 4 muestras diarias durante casi 10 meses (aproximadamente 1200 muestras). Es muy probable que la eficiencia del recall aumente.

3.2.4 CONSIDERACIONES PRÁCTICAS

Un artículo de Simpson [15] dedica un espacio al cálculo aproximado de la capacidad de memoria de la BAM. Según éste, la BAM puede almacenar hasta 15 pares asociados (siempre y cuando la distribución de las muestras sea uniforme) por cada 100 neuronas en la capa de dimensión menor de la BAM. Es decir, si cierta BAM tiene una dimensión de 400x200 podrá almacenar un total de 30 pares asociados [15,16]. La capacidad de la BAM depende solamente de la dimensión de la misma y la distribución de las muestras, pero no del tamaño de las muestras.

Lo cierto, según este trabajo, es que dicho número es menor en la práctica (probablemente un 20%) . El desarrollo del trabajo actual indicó que, bajo datos ordinarios, este coeficiente de 15 por cada 100 neuronas se reduce a 12 ó 13. Sin embargo, cuando los vectores-salida de entrenamiento se encuentran perfectamente distribuidos (gráfica de campana), la capacidad aumenta a 14 ó 15 efectivamente. En el curso de este trabajo los datos que se manejaron no presentaron la distribución idónea, luego entonces la capacidad de la BAM se vió disminuida.

Otra cuestión importante es que mientras aumente el número de pares asociados aprendidos por una sola BAM, menos tolerante al ruido se vuelve la misma. Este trabajo manejó 3 pares asociados por subBAM (nótese que este número está cercano al máximo, pues la dimensión de la capa menor de la BAM fue de sólo 9 neuronas de salida en los tres bloques de prueba).

Otras pruebas con el modelo BAM revelaron también que no importa la dimensión de la subBAM (p. ej., 1000x500), cuando dos o más vectores de salida tienen una distancia de Hamming entre ellos de exactamente 1, el recall de dichas muestras entrenadas ya no es completo (100%). Además, si la distancia de Hamming entre ellas es de exactamente 2, la eficiencia del recall de dichas muestras entrenadas depende del número de muestras aprendidas por dicha subBAM.

Si la limitante anterior existe (distancias de Hamming de 1 y 2 en los vectores de salida) y el número de muestras aprendidas alcanza el límite de Simpson [15], el recall de dichas muestras no es del 100%; si al contrario, está por debajo del límite, el recall *sí* es del 100%.

3.3 COMPARACIÓN: RETROPROPAGACIÓN vs BAM

Como ya ha podido apreciarse, es considerablemente mayor el tiempo de aprendizaje que requiere el modelo de retropropagación en comparación con la BAM. La figura 26 muestra esta diferencia. Los datos representan el tiempo de aprendizaje de cada modelo para una conjunto de 16 muestras.

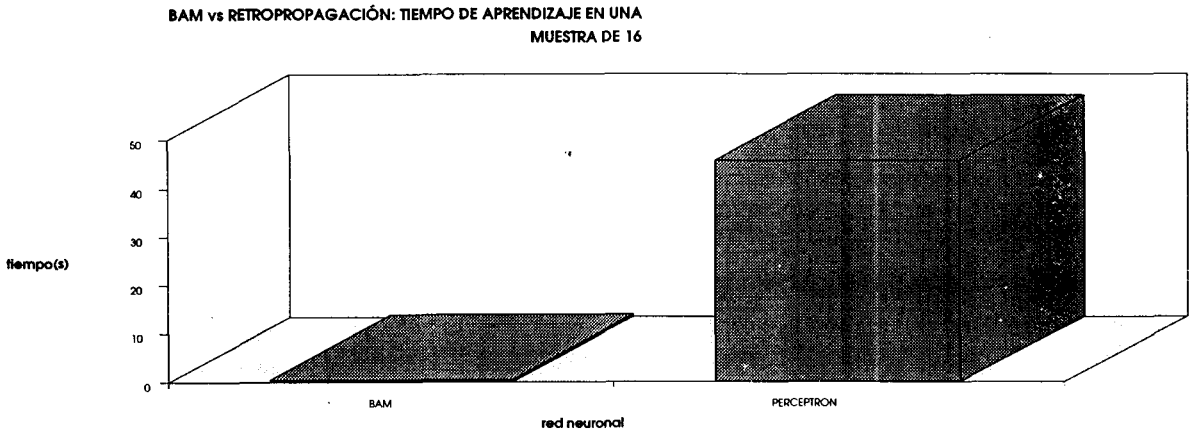


Figura 26 Comparación de tiempos de aprendizaje para un conjunto de 16 muestras (12 variables de entrada).

El modelo de retropropagación requiere mucho mayor tiempo de aprendizaje que la BAM. Bajo una configuración fija, el tiempo de aprendizaje del modelo de retropropagación aumenta considerablemente con respecto a un número creciente de muestras entrenadas.

De otra forma, si la configuración es variable, basta con aumentar la dimensión de alguna de las capas, ambas, o agregar capas intermedias, a medida que se agregen muestras, para continuar manteniendo un tiempo aceptable en la fase de aprendizaje (de otra forma, si la configuración es constante, al aumentar el número de muestras a entrenar el tiempo de aprendizaje aumenta considerablemente). La figura 27 presenta el tiempo de aprendizaje en ambos modelos para pruebas de número de muestras crecientes y una configuración fija. El color fuerte corresponde a los datos del modelo de retropropagación.

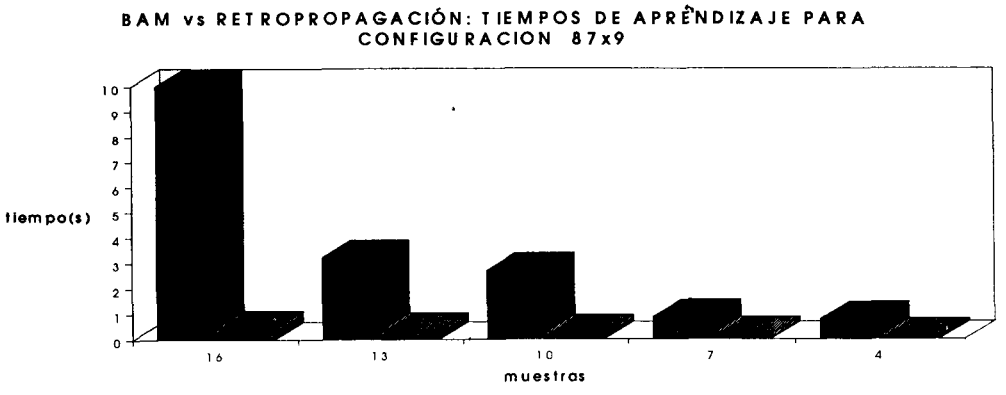


Figura 27 Tiempos de aprendizaje para muestras crecientes y configuración fija (12 variables de entrada).

En estas pruebas, la ventaja del modelo de retropropagación sobre la BAM es el tiempo de recall por muestra; una vez que el modelo de retropropagación ha sido entrenado, su tiempo de respuesta es particularmente corto. Esto se muestra en la figura 28. Nuevamente se recuerda al lector que el modelo de retropropagación aqui utilizado es de sólo 2 capas y sin capas intermedias. Es claro que un modelo de retropropagación de al menos una capa intermedia puede mostrar mayores ventajas que las que aquí se ilustran.

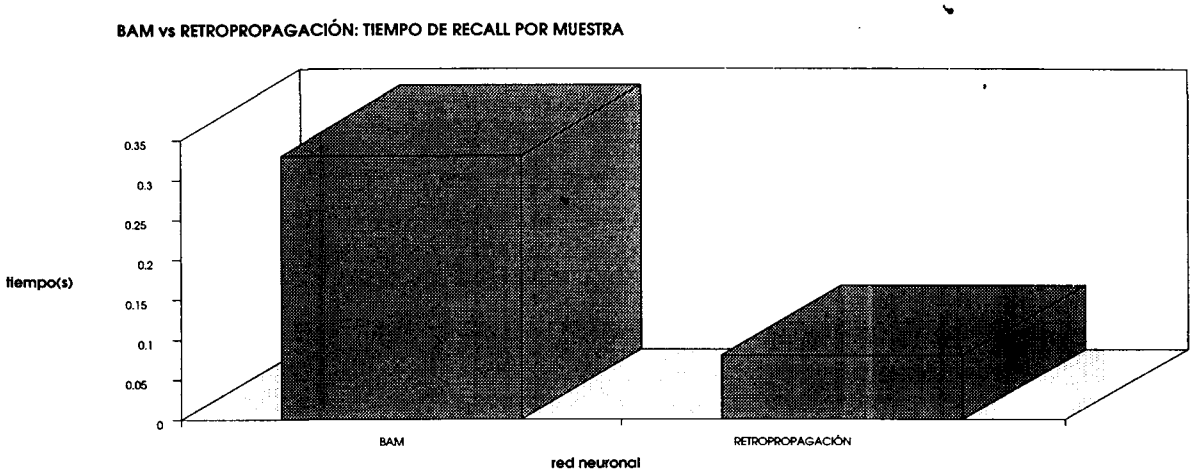


Figura 28 Tiempos de recall para una prueba de 16 muestras no entrenadas dado un entrenamiento de 16 muestras.

Por otro lado, los resultados del recall de muestras no entrenadas reflejan la superioridad de la BAM bajo cualquier condición en la entrada. La figura 29 nos permite visualizar la eficiencia del recall de cada modelo para una prueba de 11 muestras no entrenadas dado un entrenamiento previo de sólo 16 muestras.

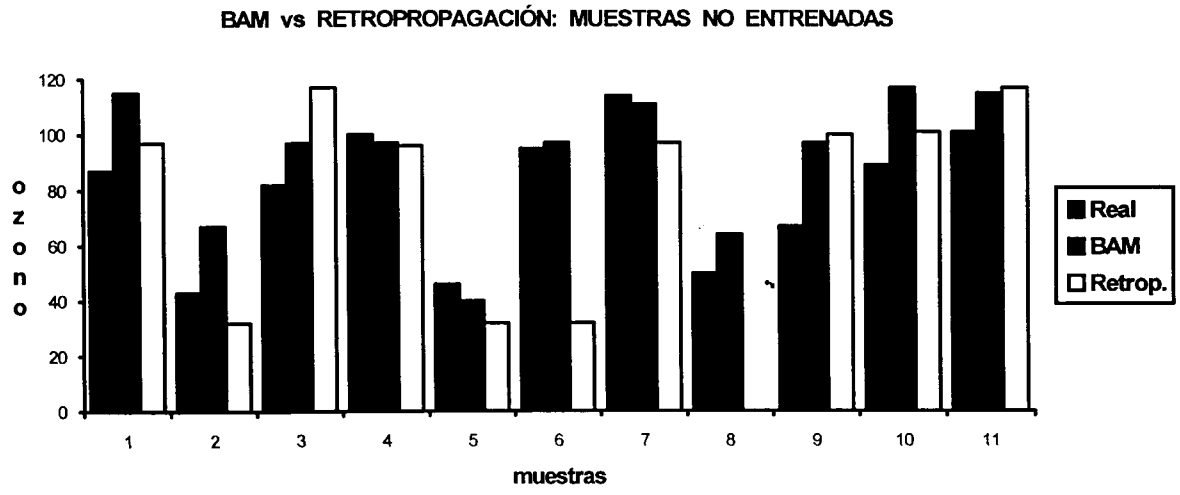


Figura 29 Recall de 11 muestras no entrenadas dado un entrenamiento de 16 muestras (12 variables de entrada).

Además del ruido implícito en las muestras no entrenadas (donde ruido significa todo aquel valor en las variables de entrada diferente de algún valor ya entrenado), en muchas ocasiones estas muestras de prueba presentan alguna variable con valor cero (en algunos casos hasta tres).

Lo anterior se debe a la ausencia de medición de dicha variable. La figura 30 muestra la eficiencia del recall de cada modelo para una prueba de 11 muestras no entrenadas con dos variables de valor cero cada una (17% de información incompleta). El entrenamiento previo fue de 16 muestras.

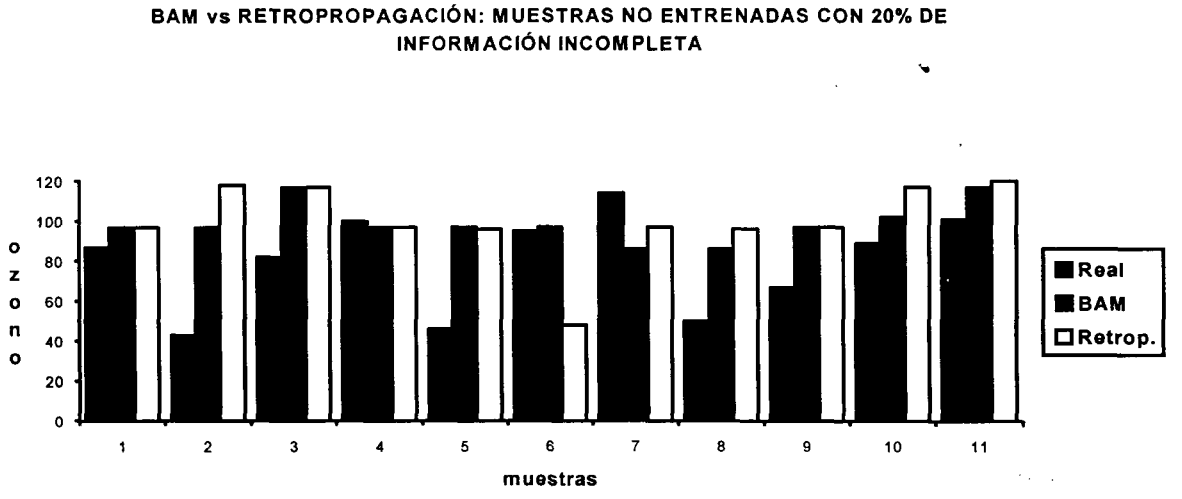


Figura 30 Recall de 11 muestras no entrenadas con dos variables de valor cero dado un entrenamiento de 16 muestras.

A fin de mostrar de forma más completa el grado de tolerancia al ruido que ambos modelos presentan, la figura 31 ilustra la eficiencia del recall de 16 muestras entrenadas con dos variables de valor cero (17% de información incompleta). Con sólo dos variables de valor cero el modelo de retropropagación comenzó a mostrar distorsión.

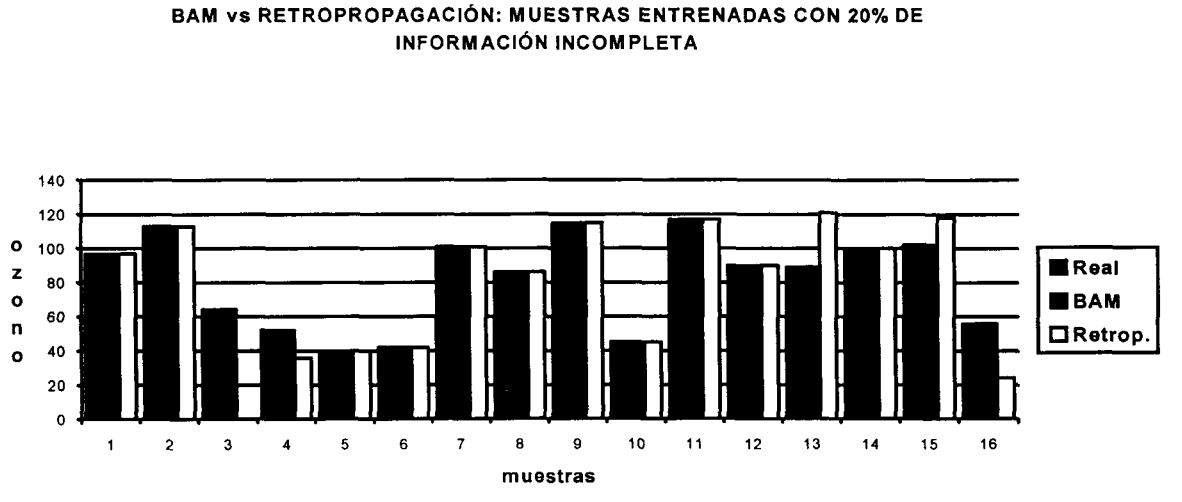


Figura 31 Recall de 11 muestras entrenadas con dos variables de valor cero dado un entrenamiento de 16 muestras.

El fin práctico de probar al sistema con muestras no entrenadas y algunas variables con valor cero, es que el sistema, además de proporcionar la salida más cercana a dicha entrada, regenere el valor de las variables con valor cero (obtenga la información que en un principio estaba incompleta).

CAPÍTULO 4

4 RESULTADOS

4.1 MODELO DE RETROPROPAGACIÓN

Para el bloque de pruebas con muestras de 12 variables de entrada, pruebas de 11 muestras no entrenadas y dado un entrenamiento de 16 muestras, la eficiencia del recall alcanzó 73%. Apenas un 20% de información incompleta en muestras no entrenadas produce, en promedio, una baja de la eficiencia del recall mayor al 20% en entrenamientos de sólo 16 muestras.

Para entrenamientos de 16 muestras (12 variables de entrada) la tolerancia a la información incompleta en muestras entrenadas apenas alcanza un 15% antes de que la función recall comience a presentar distorsión.

4.2 MEMORIA ASOCIATIVA BIDIRECCIONAL (BAM)

En cualquiera de los tres grandes bloques de prueba (dados por el número de variables de entrada) la eficiencia del recall siempre fue mayor a 77% y alcanzó un máximo de 86%, lo anterior dentro de un margen de error satisfactorio de +/-30 imecas. Los resultados anteriores se midieron sobre el recall de muestras no entrenadas. La figura 32 ilustra gráficamente los resultados de las diferentes pruebas de recall para los tres grupos de prueba.

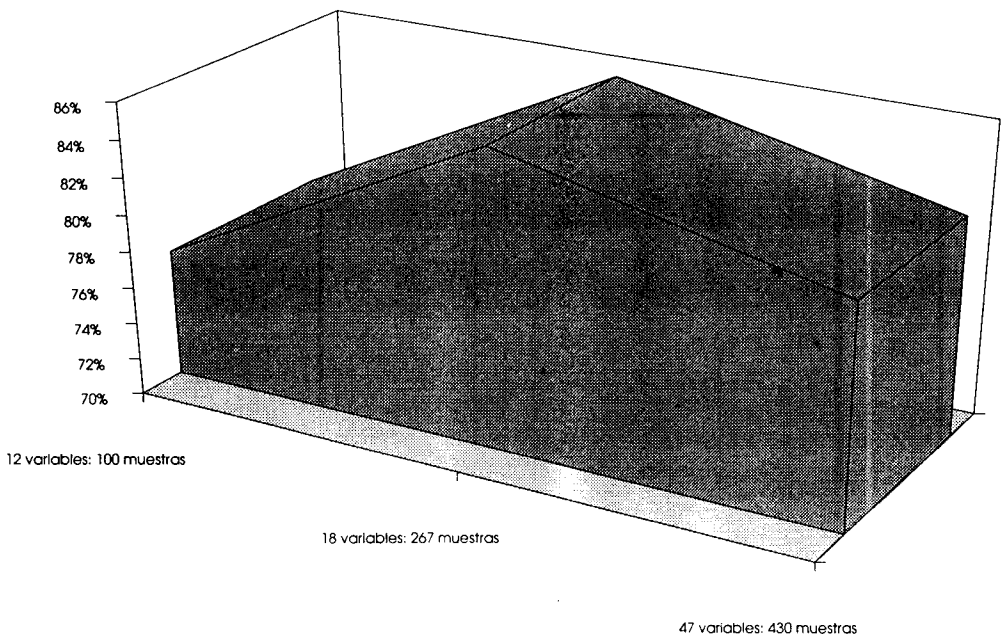


Figura 32 Comparación de eficiencias de la función de recall para diferentes configuraciones.

La menor tolerancia a la información incompleta presentada en muestras no entrenadas fue de 20% (sin sufrir aún distorsión alguna). El entrenamiento previo fue de 430 muestras. En pruebas de muestras entrenadas y entrenamientos de hasta 430 muestras, la menor tolerancia a la información incompleta fue de 50% y la mayor tolerancia de 60%.

Las pruebas anteriores indican que el sistema neuronal de tipo BAM es capaz de pronosticar el nivel de ozono medido real con una eficiencia mayor al 70%, dentro de un margen de error de +/-30 imecas, y limitando el entrenamiento hasta 500 muestras (por el momento). El número de muestras anterior equivale a 2 muestras diarias durante 8 meses ó 3 muestras diarias durante 5 meses. Los resultados de las pruebas hacen suponer que si se entrena un número mayor de muestras la eficiencia del recall tiende a aumentar.

De acuerdo con los resultados obtenidos, la estación del Pedregal presenta influencia directa de las estaciones del norte y del centro. De la misma forma, pero en grado menor, la estación Cerro de la Estrella muestra influencia de las mismas estaciones. Se tiene conocimiento real de que la estación Cerro de la Estrella es efectivamente una de las estaciones que presenta la información más conflictiva debido a que dicha estación se encuentra dentro de un área de turbulencia [13,14]. Los resultados muestran quizá esto.

Los resultados de las pruebas de pronóstico de ozono para estaciones del sur, dados niveles de ozono de estaciones del norte, indican que existe alguna correlación entre estos niveles. La realidad muestra que, efectivamente, existen corrientes de aire que circulan de norte a sur en la mayor parte de las ocasiones. No así de sur a norte, en donde algunas pruebas de recall no incluidas en este trabajo apenas muestran una eficiencia del 35%. A continuación, una tabla resume los resultados obtenidos.

Tabla 2 Resumen de resultados de las pruebas de recall

RETROPROPAGACIÓN (12 variables)

Tipo de muestras de prueba	Muestras entrenamiento	Muestras prueba	% de ruido	Recall
Entrenadas	16	16	0	100
Entrenadas con ruido	16	16	20	88
No entrenadas	16	11	0	73
No entrenadas con ruido	16	11	20	55

BAM (12 variables)

Tipo de muestras de prueba	Muestras entrenamiento	Muestras prueba	% de ruido	Recall
Entrenadas	100	100	0	100
Entrenadas con ruido	100	14	50 / 58	100 / 85
No entrenadas	100	14 / 13	0	42 / 77
No entrenadas con ruido	100	13	10 / 20	61 / 53

BAM (18 variables)

Tipo de muestras de prueba	Muestras entrenamiento	Muestras prueba	% de ruido	Recall
Entrenadas	267	267	0	100
Entrenadas con ruido	180 / 267	10	60	100 / 90
No entrenadas	60 / 180 / 267	21	0	23 / 38 / 86

BAM (46 variables)

Tipo de muestras de prueba	Muestras entrenamiento	Muestras prueba	% de ruido	Recall
Entrenadas	430	430	0	100
No entrenadas	430	51	0	78 / 73

CAPÍTULO 5

5 CONCLUSIONES Y TRABAJO FUTURO

Después de investigar, analizar y probar el modelo de retropropagación y la memoria asociativa bidireccional (BAM), ésta última, la BAM, presenta grandes ventajas como la gran tolerancia al ruido, entre otras. Por medio del sistema BAM desarrollado durante este trabajo, la BAM puede utilizarse también para aplicaciones con grandes volúmenes de información.

El tiempo de aprendizaje de la BAM es mucho más corto que su equivalente en el modelo de retropropagación. Además, dicho tiempo es lineal con respecto al número de muestras entrenadas; no así en el modelo de retropropagación, en donde el tiempo de aprendizaje aumenta considerablemente conforme el número de muestras crece. Lo anterior se midió con respecto de una misma configuración, es decir, un mismo modelo de retropropagación de dimensión fija nxm almacenó un número creciente de muestras a la vez.

Respecto del tiempo de recall es importante mencionar que el modelo de retropropagación le lleva una ventaja a la BAM. El modelo de retropropagación de dos capas, una vez entrenado, realiza el recall de una sóla pasada [10,5,19], lo que significa un corto tiempo de recall. No importa que tan grande sea la dimensión de este modelo, cuánto ruido exista en los patrones de entrada, o cuán grande sea el volumen de patrones asociados, el tiempo de recall es aún corto. Al contrario, el tiempo de recall en la BAM crece linealmente con respecto al volumen de patrones asociados entrenados.

Por otro lado, la BAM es mucho más tolerante al ruido que el modelo de retropropagación. De hecho, algunas pruebas revelaron que, para 400 muestras entrenadas, el recall de muestras no entrenadas fue aceptable (mayor a un 60%) con sólo el 40% de información correcta. El modelo de retropropagación, aunque tolerante al ruido, no es por mucho tan tolerante como lo es la BAM.

Otra ventaja más de la BAM respecto del modelo de retropropagación, es la simplicidad en su algoritmo y la facilidad de su implementación y ejecución [12,17,11]. Además, no requiere ninguna clase de parámetros iniciales [10,7]. El modelo de retropropagación, por el contrario, requiere la inicialización adecuada de los pesos de la matriz y de los umbrales a fin ejecutar el aprendizaje en un tiempo óptimo o razonable [10,18,19].

Una limitante de la BAM es su restringida capacidad de memoria [16] que es de aproximadamente 15 patrones asociados por cada 100 neuronas en la capa de dimensión menor [15], siempre y cuando las muestras se encuentren distribuidas uniformemente.

Este trabajo superó esta limitante por medio del sistema BAM donde se utiliza el producto punto como el criterio de selección de la respuesta correcta. Hasta este momento no se ha encontrado en la literatura una forma similar dentro de un sistema BAM, pero existen otras formas que operan de manera jerárquica (forman un arbol de BAM's).

El modelo de retropropagación de dos capas puede almacenar un número mayor de patrones en una sóla memoria. El problema surge cuando la cantidad de patrones asociados es grande (varias decenas).

Sólo hasta que se ha estudiado y probado el modelo de la memoria asociativa bidireccional (BAM) se detectan las grandes ventajas del proceso bidireccional o recurrente por medio del cual opera. El estudio y análisis de otro par de modelos neuronales recurrentes (al igual que la BAM) como son el modelo de retropropagación recurrente y la memoria holográfica mejorada ya estan llevándose a cabo y son motivo de otros trabajos.

En especial, el paradigma holográfico utiliza formas de codificación que van más allá de las formas tradicionales. La información es codificada por medio de técnicas con analogía holográfica y utiliza las ventajas que las transformaciones complejas brindan para facilitar el método y hacerlo, al mismo tiempo, poderoso. Entre otras ventajas, clama: mayor capacidad de memoria, mayor rapidez en el aprendizaje, aprendizaje de una sola pasada, y vectores de respuesta (salida) con dos fases: magnitud y confiabilidad [20].

REFERENCIAS

[1] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis. Hardvard. Cambridge, MA. Agosto 1974.

[2] Teuvo Kohonen. *Self-Organization and Assocciative Memory*. Vol. 8. Springer Series in Information Sciences. Springer-Verlag, N.Y., 1984.

[3] David Rumelhart, James McClelland, and the PDP Research Group. *Parallel Distributed Processing*. Vol. 1 and 2. MIT Press, Cambridge, MA, 1986.

[4] Kunihiko Fukushima. *A neural network for visual pattern recognition*. Computer. 21(3): 65-75. Marzo 1988.

[5] S. Y. Kung. *Digital Neural Networks*. Prentice Hall. Englewood Cliffs, N.J. 1993.

[6] John J. Hopfield and David W. Tank. *"Neural" computation of decisions in optimization problems*. Biological Cybernetics. 52: 141-152. 1985.

[7] Bart Kosko. *Adaptive bidirectional associative memories*. Applied Optics. 26(23): 4947-4960. Diciembre 1987.

[8] Bart Kosko. *Competitive bidirectional associative memories*. In Proceedings of the IEEE First International Conference on Neural Networks. San Diego, CA, II: 759-766. Junio 1987.

[9] Gail A. Carpenter and Stephen Grossberg. *The ART of adaptive pattern recognition by a self-organizing neural network*. Computer. 21(3): 77-88. Marzo 1988.

[10] James A. Freeman, David M. Skapura. *Neural Networks: algorithms, applications, and programming techniques*. 1a edición. Addison-Wesley. Reading, MA, 1991.

[11] Bart Kosko. *Bidirectional Associative memories*. IEEE Transactions on Systems, Man, and Cybernetics. 18(1): 49-60. Enero-febrero 1988.

[12] Edgar Sinencio. Sánchez, C. Lau. *Artificial Neural Networks*. IEEE PRESS, Piscataway, N.Y., 1992.

[13] Comunicación privada con el Dr. Héctor Rivero. Instituto de Física de la Universidad Nacional Autónoma de México.

[14] Comunicación privada con el Dr. Carlos Ruiz. Instituto Tecnológico y de Estudios Superiores de Monterrey.

[15] Patrick K. Simpson. *"Higher-Ordered and Intraconnected Bidirectional Associative Memories"*. IEEE Transactions on Systems, Man, and Cybernetics. Vol. 20. No. 3, pp. 637-653. May/June 1990.

[16] Anthony Kuh and Bradley W. Dickinson. *Information capacity of associative memories.* IEEE Transactions on Information Theory. 35(1): 59-68. Enero 1989.

[17] Adam Blum. *Bidirectional Associative Memory Systems in C++.* Dr. Dobb´s Journal. Abril 1990.

[18] Roland Köberle. *Neural Networks as Content Addressable Memories And Learning Machines.* Computer Physics Communications 56: 43-50. 1989.

[19] R. Paul Gorman and Terrence J. Sejnowski. *Analysis of hidden units in a layered network trained to classify sonar targets.* Neural Networks. 1(1):76-90. 1988.

[20] Branko Soucek and the IRIS group. *Fuzzy, Holographic, and Parallel Intelligence.* John Wiley & Sons. 1992.

[21] Bart Kosko. *Constructing an associative memory.* Byte. 12(10): 137-144. Septiembre 1987.

[22] Ruiz-Suárez J.C., Ruiz-Suárez L.G., Castro T., Gay C., Eidels-Dubovoi S. Photolysis of nitrogen dioxide and ozone in the atmosphere of Mexico City, in Air Pollution, Ed. by Comp. Mech. Publications, Elsevier Applied Science, Southampton, 1993.

APÉNDICE A: ARTÍCULO INTERNACIONAL PRODUCTO DE ESTA TESIS

Aceptado por el Wessex Institute of Technology, University of Portsmouth, UK., para participar en la IX Conferencia Internacional sobre Aplicaciones de Inteligencia Artificial para la Ingenieria, en Pensylvania, U.S.A. (AIENG/94). Este artículo aparecerá en el libro titulado IX International Conference on Applications of Artificial Intelligence in Engineering.

Short-term Forecasting of Ozone by Means of a Bidirectional Associative Memory
México City: Case Study

J. C. Ruiz-Suárez[a], R. Smith-Pérez[a], J. Torres-Jiménez[a] & L. G. Ruiz-Suárez[b]

[a]Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus-Morelos.
Apdo. Postal 99-C, Cuernavaca, Morelos 62050, México.
[b]Centro de Ciencias de la Atmósfera, Universidad Nacional A. de México, Circuito Exterior, Cd. Universitaria, México D.F. 04510, México.

In this work we report preliminary results of a study aiming to set up an intelligent tool to perform ozone forecasting in the very polluted atmosphere of México City. This tool is based in the paradigm of neural nets. Although our project involves several neural net models, this work presents results obtained by using only one of them, the Bidirectional Associative Memory (BAM), which in turn behaves as a heteroassociative content addressable memory (CAM).

We analyse and preprocess daily patterns of meteorological variables and concentrations of pollutants as measured by five monitoring stations in México City. These patterns are used to train a system of BAM's and then we use it to forecast ozone at two different points in the city.

INTRODUCTION

In the last several years, air pollution phenomena have become a main concern in the scientific community. Around the globe, scientists have created deterministic and stochastic computational models to gain insight of the real world, its dynamics and behaviour under possible perturbations. On one hand, deterministic models, such as meteorological hydrostatic and nonhydrostatic pronostic models, gaussian, Euler, Lagrange or chemical models, are used to make pollution forecasts and evaluate ¿what if? scenarios. These models, which numerically solve the complete set of time-dependant equations, have been used to study urban barrier effects on polluted urban boundary layers [1], mesoescale dispersion of pollutants [2,3], long range transport [4], etc.

Results show that, if properly validated, deterministic models are excellent tools to provide an objective assesment in source receptors relationships [5].

On the other hand, stochastic models are based on quasi-empirical rules to analyze trends and periodical changes of air quality parameters as a function of meteorological variables and concentrations of secondary pollutants. Several techniques are used for this purpose, being the most important, frequency distribution analysis, time series, and spectral analysis. Due to the fact that they do not establish deterministic cause-effect relationships, stochastic models are limited in their nature. However, these models may be used in many situations to carry on short term forecasting in real time. Specially when local meteorological conditions change rapidly or in complex wind and terrain. Very recently, a model based in neural networks able to forecast SO_2 around a thermal power plant has been reported [6].

In this work, we report preliminary results of a study aiming to set up an inteligent tool to perform ozone forecasting in the very polluted atmosphere of Mexico City. This tool is based in the paradigm of neural nets.

Ozone, one of the best indicators of air pollution, reaches in México City unhealthful levels most of the days each year. Its high concentration indirectly depends on on-road and off-road vehicles as well as on industrial emissions and local and regional climatological conditions. High altitude complicates the problem some more.

Although our project involves several neural network models, this work presents results obtained by using only one of them, the Bidirectional Associative Memory (BAM) [7], which in turn behaves as a heteroassociative content addressable memory (CAM). We set up a system of BAM's and succeed in storing as many patterns as we want. No **ad hoc** parameters are needed during training and computational time is minimum.

Data from the main monitoring stations in México City, from January to May of 1992, are used to train the system of BAM's. We then use the BAM system to forecast ozone at two different points in the city. In the next sections we present a short description of the BAM model, the procedure used in this work, the results and conclusions.

The bidirectional associative memory is a two-layer dynamical system able to recall stored patterns by bidirectional feedback searching. Information passes from one neuron layer to the other by operating the connection matrix M. Information then passes backwards through the matrix transpose M^T. When an input enters the BAM, the neurons evolve to a stable state of two-pattern reverberation. This state corresponds to a system energy minimum. If the back-and-forth operations

$$\begin{array}{l} A \dashrightarrow M \dashrightarrow B \\ A' \dashleftarrow M^T \dashleftarrow B \\ \quad\quad\quad\quad \vdots \\ A_i \dashrightarrow M \dashrightarrow B_i \\ A_i \dashleftarrow M^T \dashleftarrow B_i \end{array}$$

are of finite number, the bidirectional process is a stable heteroassociative CAM. In fact, by identifying the Lyapunov function

$$E = -AMB^T \qquad (1)$$

it can be shown that the BAM is stable for any matrix M [7]. A and B in equation (1) are the stored temporal patterns.

The network dynamics for the ith processing element in layer B is given by

$$b_i(t+1) = \begin{cases} 1 & \text{if } b_i^*(t+1) > 0 \\ b_i(t) & \text{if } b_i^*(t+1) = 0 \\ -1 & \text{if } b_i^*(t+1) < 0 \end{cases} \quad (2)$$

where

$$b_i^*(t+1) = \Sigma^n_j M_{ij} a_j$$

For the ith processing element in layer A

$$a_i(t+1) = \begin{cases} 1 & \text{if } a_i^*(t+1) > 0 \\ a_i(t) & \text{if } a_i^*(t+1) = 0 \\ -1 & \text{if } a_i^*(t+1) < 0 \end{cases} \quad (3)$$

where

$$a_i^*(t+1) = \Sigma^m_j b_j M_{ji}$$

The update can be synchronously when all the processing element updates occur at a clock cycle, or asynchronously when only some subset is updated each time. The Hebbian outer-product rule can be used to encode q (A_i,B_i) pairs in the BAM:

$$M = \Sigma^n_i B_i^T A_i \quad (4)$$

For discrete implementations, multiplying and adding bipolar quantities produce excitatory and inhibitory connections (instead of only excitatory and zero-weight connections with binary data). Thus, the BAM works out better if the input and output patterns are encoded as bipolar strings (i.e., A_i E $\{-1,+1\}^n$ and B_i E $\{-1,+1\}^m$).

PROCEDURE

The México City automatic monitoring network (RAMA) has 25 stations. Five of these stations, Tlanepantla (northwest), Xalostoc (northeast), C. de la Estrella (southeast), Pedregal (southwest), and Merced (center) are the most complete for they measure up to 9 parameters every minute 24 hours a day (data are hourly averaged). The parameters measured in these stations are the following: wind direction (WD) and wind velocity (WV), temperature (T), relative humidity (RH), sulphur dioxide (SO_2), carbon monoxide (CO), ozone (O_3), nitrogen dioxide (NO_2), and nitrogen oxides (NO_x). Inmediate ozone production depends principally on two factors, the concentration of NO_2 in the atmosphere and the solar actinic flux (in the 280-420 nanometers window) on the surface boundary layer.

The first parameter is measured by the monitoring network, but the second is not. To circunvent this problem, we calculate, for every day at noon from january to may, the surface actinic flux by using a model experimentally validated in the past at Mexico City conditions[8]. Ozone dispersion is a nonlinear function of WD, WV, T and NO_x/NO_2.

RH is important to consider because this variable may play a role in the overall reactivity of the system, either by affecting important chain termination reactions or for the production of wet aerosols (which in turn affects the uv actinic flux). The NO_x/NO_2 ratio represents an indicator of the state of the precursor mixture.

CO is the least reactive pollutant measured by the RAMA. It may be transported by the atmosphere, long distances, without reacting with other species. It certainly participates very little in the O_3 chemistry. However, it is because this natural property that can be used as an indirect quantification of wind drift. Finally, in the presence of H_2O_2 and wet aerosols, SO_2 does partcipate in the chemistry of ozone. However, during the dry season (which is the season considered in this work) SO_2 plays the same role as CO.

BAM's are globally stable and able to efficiently recall stored patterns. However, they face the strong limitation of low memory capacity. Indeed, storage capacity is less than the least of the input or output number of neurons ($q<\min(n,p)$). Thus, to store a large number of associative pairs in a BAM would be hard, if not impossible, with a moderate number of neurons.

To solve this problem we set up a system of k BAM's. Each individual BAM is trained by a small number of q pairs A_i, B_i (between 3 and 10, depending on the Hamming distance between them). Thereafter, when a new input A enters the BAM system, each individual BAM gives an output pair A_i, B_i. The right-answering BAM is the one who gives the inner product $A \cdot A_i$ closest or equal to n (remember that the inner product of two identical bipolar vectors is equal to the vector dimension). The corresponding B_i is then the output.

Two different cases are considered in this work. Firstly, a BAM system is trained with 100 patterns from data taken at noon at the Pedregal station (as mentioned above, the solar actinic flux is also considered in the input patterns). Thus, when a vector representing the above variables enters the input layer, the output layer exits (once equilibrium is reached) an ozone concentration value in IMECA (IMECA stands for mexican air quality index and the equivalence with ppm is: 100 IMECA = 0.11 ppm).

Secondly, pollution generated at one city spot will affect pollution measured somewhere else. In our case, secondary pollutants produced in northern parts of the city travel slowly to the south. Thus, in order to account for the correlation between different monitoring stations, a BAM system is trained with 650 patterns formed by data taken from 8 A.M. to 16 P.M. in each one of the above five stations.

Since each station measures up to 9 variables (ozone included), each input pattern is a vector representing 47 real values (the solar actinic flux is not considered here, but instead, month and hour are taken as variables). As output patterns we have the corresponding 650 vectors formed only by hourly ozone concentrations at the Pedregal and Cerro de la Estrella stations.

The BAM code was written in C and runs on a 386 PC.

RESULTS

Two different tests were carried out to evaluate the performance of the neural network used in this work. Firstly, we input to the BAM system each one of the A_i vectors used during the training stage. The BAM system efficiently recalls the expected vectors B_i (i.e., the patterns in memory are efficiently addressable with no error at all). This may be an advantage over other models, such as the multilayer perceptron, where even the training patterns are not always completely addressable.

Secondly, several patterns which were left out during training (specifically for this test) were input to the BAM system.

The first and larger BAM system was tested by 50 new patterns (these patterns were chosen to have up to 4 input variables not reported by the RAMA, i.e., zero valued). Figure 1 and 2 show the obtained results. Figure 1 shows the predicted ozone concentration at the Pedregal station whereas Figure 2 shows the ozone concentration at the Cerro de la Estrella station. Results obtained for the Pedregal are clearly better than Cerro de la Estrella's. This may indicate there is a larger influence from the northern and center variables in the city to the southwest concentration of ozone.

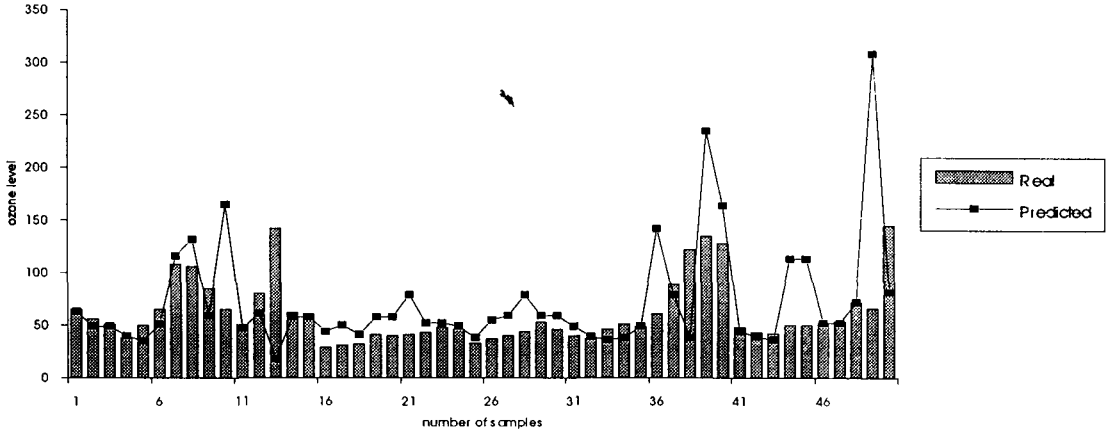


Figure 1 Recall Test for no trained samples of the Predregal station. Almost 450 samples were trained.

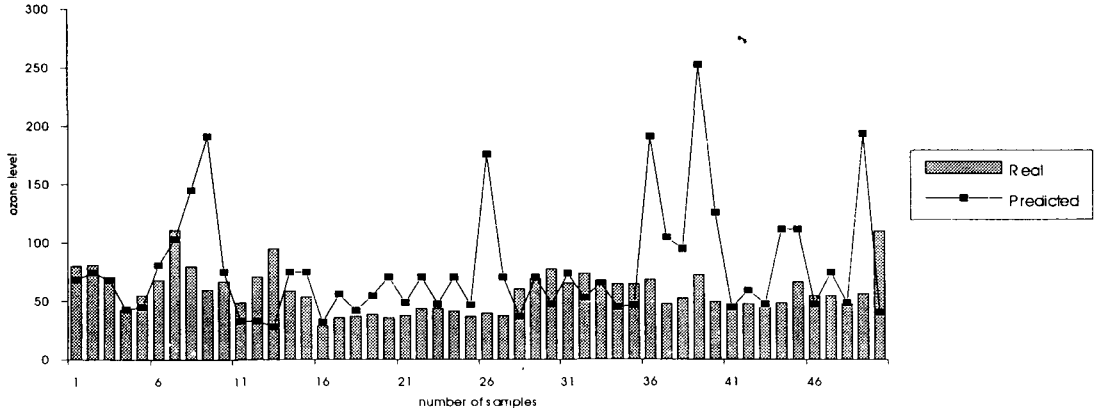


Figure 2 Recall test for no-trained samples of Cerro de la Estrella station. Almost 450 samples were trained.

CONCLUSIONS

A Bidirectional Associative Memory was used to efficiently associate daily multivariable patterns to ozone concentrations in México City.

The poor storage capacity of the BAM was improved by a BAM classifying system. In the largest case (the BAM system with 650 vector pairs) learning takes no more than 6 minutes and does not need any **ad hoc** parameters. The BAM system is able to recall with zero error all the patterns used to train the network and also, to reasonably predict ozone concentration during days and hours whose patterns were not in the memory.

Compared to feedforward networks with gradient descent learning, the BAM model, which is fedback and uses a simple outer-product learning rule, has far more noise tolerance. This makes it suitable to numerical applications of the kind studied in this article.

Results presented in this article are preliminary. Future work will deal with a more sophisticated analysis. From the point of view of the neural network, we would study other models such as the feedback backpropagation and the holographic neural method. From the point of view of the physics and chemestry of ozone, we would add variables not used in this work, such as mesoscale and monitoring station interconnective variables.

ACKNOWLEDGEMENTS

We would like to thank Héctor Riveros from the Institute of Physics of the National University of México for providing the data used in this work.

REFERENCES

1. Bornstein R., Thunis P and Schayes G. Simulation of urban barrier effects on polluted urban boundary layers using the three-dimensional URBME T/TVM model with urban topography-new results from New York City, in Air Pollution, Ed. by Comp. Mech. Publications, Elsevier Applied Science, Southampton, 1993.

2. Lyons W.A., Pielke R.A., Cotton W.R., Tremback, C.J. Uliasz M., Walko R.L., Thorson W., Klitch M. and Struck D. K. An interactive operational emergency response dispersion system using a mesoscale numerical prognostic model, 9th Intl. Conf. on Interactive Information and Processing Systems for Meteor., Oceanog. and Hydrol., AMS, Anaheim, 6 pp., 1993.

3. Pielke R.A., Lyons W.A., McNider, R.T., Moran M.D., Stocker R.A.,Walko R.L and Uliasz M. Regional and mesoscale meteorological modeling as applied to air quality studies, Air Pollution Modeling & Its Applications VIII, H. Van Dop and D.G. Steyn, Eds., Plenum Press, New York, pp. 259-290, 1991.

4. Kotamarthi V. and Carmichael G. The Long Range Transport of Pollutants in the Pacific Rim Region. Atmospheric Environment, 1990, 24, 1521-1534.

5. Zannetti P. Air Pollution Modeling-Theories, Computational Methods and Available Software. Southampton: Computational Mechanics and New York: Van Nostrand Reinhold, 1990.

6. Boznar M., Lesjak M. and Mlakar P. A neural network-based method for short-term predictions of ambient SO_2 concentrations in highly polluted industrial areas of complex terrain. Atmospheric Environment, 1993, 27B, 221-230.

7. Kosko B. Adaptive bidirectional associative memory, Appl. Opt., 1987, 26, 4947-4960.

8. Ruiz-Suárez J.C., Ruiz-Suárez L.G., Castro T., Gay C., Eidels-Dubovoi S. Photolysis of nitrogen dioxide and ozone in the atmosphere of México City, in Air Pollution, Ed. by Comp. Mech. Publications, Elsevier Applied Science, Southampton, 1993.

Keyword index: ozone forecasting, México City, neural networks, bidirectional associative memory.

APÉNDICE B: CÓDIGO FUENTE DEL SISTEMA BAM

```
                    /* SYSTEMA BAM  PARA EL PRONOSTICO DE OZONO */

#include "stdio.h"

int LenN=166, LenM=27, tamBam=3, numVar=22, nsalidas=3;          /* valor default */

int entra[53];
int buenos[10]={0,0,0,0,0,0,0,0,0,0}, ceros[10]={0,0,0,0,0,0,0,0,0,0}, media[10]={0,0,0,0,0,0,0,0,0,0};
int p2[12]={2048,1024,512,256,128,64,32,16,8,4,2,1};
int dataIn[530], dataIn2[530], dataOut[50], resp[50], bits[53];
int Mat [50] [530], margen, numEjem;
char fileIn[12]="", fileOut[12]="", fileArch[12]="";
FILE *fpI, *fpO, *fpA;

main()                                  /* rutina de menu principal  */
{
 int op=1;
 do {
        clrscr();
        printf("\n      SISTEMA BAM \n\n");
        printf("      Información.....1\n");
        printf("      Configuración.....2\n");
        printf("      Cambio a binario....3\n");
        printf("      Aprendizaje.....4\n");
        printf("      Recall.....5\n");
        printf("      Finalizar.....6\n\n");
        printf("          Op > ");
        scanf("%d",&op);
        clrscr();
        switch(op) {
                case 1: info(); break;
                case 2: config(); break;
                case 3: binario(); break;
                case 4: train(); break;
                case 5: mainrecall(); break;
                default: exit();
        }
 } while(op!=6);
}

info()
{   /* en esta parte se incluye un archivo texto de
       información general sobre el programa*/
}

config()                                /* rutina del menu de configuración */
{
        int op;
        printf("\n    MENU DE CONFIGURACIÓN\n\n");
        printf("      Conf X archivo.....1\n");
        printf("      Conf Automatica...2\n");
        printf("      Finalizar.....3\n\n");
        printf("          Op > ");
        scanf("%d",&op);
        clrscr();
        switch(op) {
                case 1: configArch(); break;
```

```
                case 2: configAuto(); break;
        }
}

configAuto()        /* ejecuta la configuración automatica a partir del archivo de entrenamiento */
{
        FILE *ft;
        int i,cuenta,dato;
        long total;
        char ins[535]="";
        printf("\nArchivo de datos reales: ");
        scanf("%s",&fileArch);
        if( (fpA=fopen(fileArch,"r"))==NULL) {
          printf("No se encuentra el archivo %s",fileArch);
          scanf("%c");
        }
        else {
          fgets(ins,500,fpA);
          ft=fopen("temp.tmp","w");
          fputs(ins,ft);
          fclose(ft);
          ft=fopen("temp.tmp","r");
          for(numVar=0;fscanf(ft,"%d")!=EOF;numVar++);
          fclose(ft);
          for(cuenta=0;cuenta<numVar;cuenta++)
                bits[cuenta]=0;
          total=0;
          cuenta=0;
          rewind(fpA);
          while ( fscanf(fpA,"%d ",&dato)!=EOF ) {
                total++;
                if(dato>bits[cuenta])
                        bits[cuenta]=dato;
                if((cuenta+1)%numVar==0)
                        cuenta=-1;
                cuenta++;
          }
          if(total%numVar!=0) {
                clrscr();
                printf("\n\nEl archivo de datos esta incorrecto. Faltan o sobran datos");
                scanf("%c%c");
          }
          else {
                numEjem=total/numVar;
                for(total=0;total<numVar;total++)
                        for(cuenta=11;cuenta>-1;cuenta--)
                          if(p2[cuenta]>bits[total]) {
                                bits[total]=11-cuenta;
                                break;
                          }
                printf("\n");
                i=0;
                LenN=0;
                LenM=0;
                printf("\nEntra el numero de variables de salida: ");
                scanf("%d",&nsalidas);
                for(i=numVar-nsalidas;i<numVar;i++)
                        bits[i]=9;
                for(i=0;i<(numVar-nsalidas);i++)
                        LenN+=bits[i];
```

```
                for(i=numVar-nsalidas;i<numVar;i++)
                        LenM+=bits[i];
                letcon();
            }
        }
}

configArch()    /* rutina de configuración por archivo: los datos se leen de un archivo de usuario */
{
        int i,j;
        char op;
        printf("\nArchivo de configuración: ");
        scanf("%s",&fileArch);
        if( (fpA=fopen(fileArch,"r"))==NULL) {
          printf("No se encuentra el archivo %s",fileArch);
          scanf("%c%c");
        }
        else {
          fscanf(fpA,"%d",&numVar);
          fscanf(fpA,"%d",&nsalidas);
          i=0;
          LenN=0;
          LenM=0;
          while( fscanf(fpA,"%d",&bits[i]) != EOF ) {
                if(i>=numVar) break;
                if(i<(numVar-nsalidas)) LenN+=bits[i];
                else LenM+=bits[i];
                i++;
          }
          for(j=numVar-nsalidas;j<numVar;j++)
                bits[j]=9;      /* obligo a que las salidas sean de 9 bits */
          if(i>numVar) {
                printf("\n\nEl vector BITS es mayor al numero de variables");
                scanf("%c%c");
          }
          if(i<numVar-1) {
                printf("\n\nEl vector BITS es menor al numero de variables. Se abortara");
                scanf("%c%c");
          }
          else letcon();
        }
}

letcon()                /*  despliega en pantalla la configuración actual del sistema */
{
        char op;
        scanf("%c"); clrscr();
        printf("  CONFIGURACION LEIDA DE DISCO\n\n");
        printf("  # de variables totales: %d\n",numVar);
        printf("  # de ozonos de salida: %d\n",nsalidas);
        printf("  # de neuronas de entrada: %d\n",LenN);
        printf("  # de neuronas de salida: %d\n",LenM);
        printf("  # de ejemplos en archivo: %d\n",numEjem);
         printf("\n\nEs correcta la configuración [s/n]?: ");
        scanf("%c",&op);
        if(op=='n' || op=='N')    /* la configuración actual no es correcta. Termina */
          exit();
        fclose(fpA);
}
```

```
binario()               /*  rutina de conversión entero-binaria del archivo de entrenamiento */
{
        int Vec[53],len,i,j,if1;
        char aux[12],num[12],línea[535],línea2[105];
        printf("\nArchivo de datos reales: ");  scanf("%s",&fileIn);
        if( (fpI=fopen(fileIn,"r"))==NULL) {
          printf("No se encuentra el archivo %s",fileIn);
          scanf("%c%c");
        }
        else  {
          printf("\nArchivo binario de salida: ");
          scanf("%s",&fileOut);
          printf("\n\n\n");
          printf("       ...CONVIRTIENDO A BINARIO...");
          fpO=fopen(fileOut,"w");
          j=0;
          if1=numVar-nsalidas;
          while( fscanf(fpI,"%d",&Vec[j]) != EOF )  {
                  if(j==0)  {
                          strcpy(línea,"");
                          strcpy(línea2,"");
                  }
                  itoa(Vec[j],num,2);         /* convierte a binario */
                  len=bits[j]-strlen(num);    /* completa a los bits adecuados */
                  strcpy(aux,"");
                  strncat(aux,"00000000000",len);
                  strcat(aux,num);
                  if(j<if1)
                          strcat(línea,aux);          /* va generando la línea binaria */
                  else
                          strcat(línea2,aux);
                  if(j==numVar-1)  {
                          fprintf(fpO,"%s\n",línea);
                          fprintf(fpO,"%s\n",línea2);
                  }
                  if(j==(numVar-1))
                          j=-1;
                  j++;
           }
        }
        fclose(fpI); fclose(fpO);
}

nombre(fileTem,nbams)       /* construye el nombre físico de los archivos de cada subBAM */
char *fileTem;
int nbams;
{
          char cnum[3],sLenN[3];
          strcpy(fileTem,"");
          strcat(fileTem,"BAM");
          itoa(nbams,cnum,10);
          strcat(fileTem,cnum);
          strcat(fileTem,".");
          itoa(LenN,sLenN,10);
          strcat(fileTem,sLenN);
          return(*fileTem);
}

train()                                 /* rutina del menu de entrenamiento */
{
```

```
        int op;
        printf("\n     MENU DE APRENDIZAJE\n\n");
        printf("      Archivo completo...1\n");
        printf("      Solo N ejemplos.....2\n");
        printf("      Finalizar ..............3\n\n");
        printf("           Op > ");
        scanf("%d",&op);
        clrscr();
        switch(op) {
                case 1: train2(1); break;
                case 2: train2(2); break;
        }
}

train2(int caso)                /* rutina de entrenamiento: genera las subBAM´s */
{
  int ndata,nul,borra,i,j,k,insnum,outnum,numbams;
  char ins[535]="", outs[535]="", cnum[3];
  if(banArchBin==0)  {
        printf("\nArchivo de datos binario: ");
        scanf("%s",&fileOut);
  }
  if( (fpO=fopen(fileOut,"r"))==NULL) {
                printf("No se encuentra el archivo %s",fileOut);
                scanf("%c%c");
  }

  else {
        if(caso==2) {
                printf("\n# de datos a entrenar: ");
                scanf("%d",&ndata);
        }
        else ndata=10000;
        gotoxy(32,10);
        printf("...ENTRENANDO...");
        gotoxy(37,11);
        printf("BAM");
        nul=0;
        borra=0;
        numbams=0;
        while ( nul!=1 && ndata>0 )  { /* lee los archivos */
                numbams+=1;
                gotoxy(40,11);
                printf("%d",numbams);
                k=0;
                for(i=0;i<LenM;i++)             /* inicializa la matriz  */
                  for(j=0;j<LenN;j++)
                        Mat[i][j]=0;
                while (k!=tamBam) {
                  if (fgets(ins,LenN+5,fpO)!=NULL)  {
                        ndata-=1;
                        fgets(outs,LenN+5,fpO);
                        for(i=0;i<LenM;i++)  {
                                outnum=(outs[i]=='0')?-1:1;
                                for(j=0;j<LenN;j++)  {
                                  insnum=(ins[j]=='0')?-1:1;
                                  Mat[i][j]=Mat[i][j]+insnum*outnum;
                                }
                        }
                  }
```

```
                    else {
                            nul=1;
                            if(k==0) borra=1;
                    }
                    k++;
                  }
                  if(borra==0)  {
                    nombre(&fileIn,numbams);
                    fpI=fopen(fileIn,"w");
                    for(i=0;i<LenM;i++)  {      /* guarda la matriz en archivo de salida */
                            for(j=0;j<LenN;j++)
                                    fprintf(fpI,"%d ",Mat[i][j]);
                            fprintf(fpI,"\n");
                    }
                  }
                  fclose(fpI);
          }
          fclose(fpO);
  }
}

recall(char car)                                /* rutina de recall */
{
  int max=-1000,num,i=0,j=0,valor=-1000,numbams;
  numbams=0;
  while ( valor!=LenN ) {
          numbams+=1;
          if(valor>max)  {
            max=valor;
            copia();
          }
          nombre(&fileOut,numbams);
          gotoxy(37,12);
          printf("BAM%d ",numbams);
          if( (fpO=fopen(fileOut,"r"))==NULL )
            break;
          for(i=0;i<LenM;i++)
            for(j=0;j<LenN;j++)
                    Mat[i][j]=0;
          i=0;
          j=0;
          while( fscanf(fpO,"%d",&Mat[j][i]) != EOF )  {

          if(i%LenN==LenN-1) {
                    j++;
                    i=-1;
            }
            i++;
          }
          fclose(fpO);
          entrada();
          runBam();
          valor=0;
          for(i=0;i<LenN;i++)
            valor+=dataIn[i]*dataIn2[i];
   }
   if(valor>max) max=valor;
   if(max==LenN) display(dataOut,car);
   else display(resp,car);
  }
```

```
recallXejem()                   /* rutina  pre-recall: opera sobre una entrada interactiva */
{
  printf("\n\n\n\n");
  printf("       ...EJECUTANDO RECALL X EJEMPLO...");
  recall('E');
  scanf("%c%c");
}

recallXarch()                   /* rutina pre-recall: opera sobre el archivo de pruebas */
{
  int i,j,z;
  printf("\nArchivo de prueba: ");
  scanf("%s",&fileIn);
  printf("\nArchivo de salida: ");
  scanf("%s",&fileArch);
  printf("\nDesviación: ");
  scanf("%d",&margen);
  if( (fpI=fopen(fileIn,"r"))==NULL || (fpA=fopen(fileArch,"w"))==NULL ) {
            printf("No se encuentra(n) el(los) archivo(s)");
            scanf("%c%c");
  }
  else {
            for(i=0;i<nsalidas;i++) {
                      ceros[i]=0;
                      buenos[i]=0;
            }
            gotoxy(23,10);
            printf("...EJECUTANDO RECALL X ARCHIVO...");
            gotoxy(36,11);
            printf("EJEM");
            i=0;
            z=0;
            while( fscanf(fpI,"%d ",&entra[i]) != EOF ) {
                      if(i%(numVar-nsalidas)==(numVar-nsalidas)-1) {
                        for(j=numVar-nsalidas;j<numVar;j++)
                                  fscanf(fpI,"%d ",&entra[j]);
                        z++;
                        gotoxy(41,11);
                        printf("%d ",z);
                        recall('A');
                        fprintf(fpA,"\n");
                        i=-1;
                      }
                      i++;
            }
            fclose(fpI);
            fprintf(fpA,"\n#: %d, CEROS:",z);
            for(i=0;i<nsalidas;i++)
                      fprintf(fpA," %d",ceros[i]);
            fprintf(fpA,", ACIERTOS:");
            for(i=0;i<nsalidas;i++)
                      fprintf(fpA," %d",buenos[i]);
            fprintf(fpA,", PORCENTAJE:");
            for(i=0;i<nsalidas;i++)
                      fprintf(fpA," %d",100*buenos[i]/z);
            fclose(fpA);
  }
}

mainrecall()                    /* rutina del menu de recall */
```

```
{
        int op;
        printf("\n      MENU DE RECALL\n\n");
        printf("      Recall X ejemplo....1\n");
        printf("      Recall X archivo....2\n");
        printf("      Finalizar.....3\n\n");
        printf("          Op > ");
        scanf("%d",&op);
        clrscr();
        switch(op) {
                case 1: recallXejem(); break;
                case 2: recallXarch(); break;
        }
}

copia()  /* esta rutina actualiza la respuesta correcta del sistema BAM a medida que se ejecuta el recall*/
{
 int i;
 for(i=0;i<LenM;i++)
        resp[i]=dataOut[i];
}

igual(list1,list2)              /* verifica la condición de paro del sistema en el recall */
int list1[530], list2[530];
{
 int i=-1,flag=1;
 do {
        i++;
        flag=list1[i]==list2[i];
 } while(flag!=0 && i<LenN-1);
 return(flag);
}

runBam()                                /* rutina de ejecución de recall */
{
        int i,j, roll=0, ant[530];
        for(i=0;i<LenN;dataIn2[i]=dataIn[i],i++);
        do {
          for(i=0;i<LenN;ant[i]=dataIn2[i],i++);
          for(i=0;i<LenM;i++) {
                dataOut[i]=0;
                for(j=0;j<LenN;dataOut[i]=dataOut[i]+Mat[i][j]*dataIn2[j],j++);
                dataOut[i]=(dataOut[i]>0)?1:-1;
          }
          for(i=0;i<LenN;i++) {
                dataIn2[i]=0;
                for(j=0;j<LenM;dataIn2[i]=dataIn2[i]+Mat[j][i]*dataOut[j],j++);
                dataIn2[i]=(dataIn2[i]>0)?1:(dataIn2[i]==0)?ant[i]:-1;
          }
          roll++;
        } while(!igual(ant,dataIn2));
}

display(arreglo,car)        /* esta rutina escribe el resultado en pantalla ó a un archivo */
int arreglo[50];
char car;
{
  int i,index=numVar-nsalidas-1,total=0;
  if(car=='E') {
        clrscr();
```

```
        printf("RESULTADO: ");
    }
    for(i=0;i<LenM;i++)  {
        total+=p2[(i%9)+3]*((arreglo[i]==1)?1:0);
        if(i%9==8) {
            index++;
            if(car=='E')
              printf("%d ",total);
            else {
              if(entra[index]!=0)
                  if(total<=(entra[index]+margen) && total>=(entra[index]-margen)) {
                      buenos[index-(numVar-nsalidas)]++;
                      media[index-(numVar-nsalidas)]+=100*(total-entra[index])/entra[index];
                  }
              else {
                      ceros[index-(numVar-nsalidas)]++;
                      buenos[index-(numVar-nsalidas)]++;
              }
              fprintf(fpA,"%d\t%d\t",entra[index],total);
            }
            total=0;
        }
    }
}

entrada()       /* rutina de conversión entero-binaria de cada muestra del recall por archivo */
{
        char aux[12],num[12],línea[535];
        int i,len;
        strcpy(línea,"");
        for(i=0;i<(numVar-nsalidas);i++) {
            itoa(entra[i],num,2);       /* convierte a binario */
            len=bits[i]-strlen(num);    /* completa a los bits adecuados */
            strcpy(aux,"");
            strncat(aux,"00000000000",len);
            strcat(aux,num);
            strcat(línea,aux);      /* va generando la línea binaria */
        }
        for(i=0;i<LenN;i++)
            dataIn[i]=(línea[i]=='1')?1:-1;
}
```

APÉNDICE C: CÓDIGO FUENTE DEL SISTEMA DE RETROPROPAGACIÓN

```
/* SYSTEMA DE RETROPROPAGACIÓN PARA EL PRONOSTICO DE OZONO */

#include "stdio.h"
#include "stdlib.h"

#define CoefAc 0.3

int LenN=166, LenM=27, tamBam=3, numVar=22, nsalidas=3;          /* valor default */

int entra[33];
int buenos[10]={0,0,0,0,0,0,0,0,0,0},ceros[10]={0,0,0,0,0,0,0,0,0,0}, media[10]={0,0,0,0,0,0,0,0,0,0};
int p2[12]={2048,1024,512,256,128,64,32,16,8.4,2,1};
int VecEntrada[300], VecInt[30], VecSalida[30], bits[33];
int margen, numEjem, status;
char fileIn[12]="", fileOut[12]="", fileArch[12]="";
char ins[305]="", outs[305]="";
float W [30] [300], VecUmbral[30], Net;
FILE *fpI, *fpO, *fpA;

main()                                  /* rutina del menu principal */
{
  int op=1;
  do {
          clrscr();
          printf("\n SISTEMA RETROPROPAGACIÓN\n\n");
          printf("     Información.....1\n");
          printf("     Configuración.....2\n");
          printf("     Cambio a binario....3\n");
          printf("     Aprendizaje.....4\n");
          printf("     Recall..... .....5\n");
          printf("     Finalizar.....6\n\n");
          printf("          Op > ");
          scanf("%d",&op);
          clrscr();
          switch(op) {
                  case 1: info(); break;
                  case 2: config(); break;
                  case 3: binario(); break;
                  case 4: train(); break;
                  case 5: mainrecall(); break;
                  default: exit(status);
          }
  } while(op!=6);
}

info()
{   /* es esta parte se incluye un archivo texto de
       información general sobre el sistema*/
}
config()                                /* rutina del menu de configuración */
{
          int op;
          printf("\n    MENU DE CONFIGURACION\n\n");
          printf("     Conf X archivo.....1\n");
          printf("     Conf Automatica...2\n");
          printf("     Finalizar.....3\n\n");
          printf("          Op > ");
```

```
        scanf("%d",&op);
        clrscr();
        switch(op) {
                case 1: configArch(); break;
                case 2: configAuto(); break;
        }
}

configAuto()    /* rutina de configuración automática a partir del archivo de entrenamiento */
{
        FILE *ft;
        int i,cuenta,dato;
        long total;
        strcpy(ins,"");
        printf("\nArchivo de datos reales: ");
        scanf("%s",&fileArch);
        if( (fpA=fopen(fileArch,"r"))==NULL) {
          printf("No se encuentra el archivo %s",fileArch);
          scanf("%c");
        }
        else {
          fgets(ins,500,fpA);
          ft=fopen("temp.tmp","w");
          fputs(ins,ft);
          fclose(ft);
          ft=fopen("temp.tmp","r");
          for(numVar=0;fscanf(ft,"%d")!=EOF;numVar++);
          fclose(ft);
          for(cuenta=0;cuenta<numVar;cuenta++)
                bits[cuenta]=0;
          total=0;
          cuenta=0;
          rewind(fpA);
          while ( fscanf(fpA,"%d ",&dato)!=EOF ) {
                total++;
                if(dato>bits[cuenta])
                        bits[cuenta]=dato;
                if((cuenta+1)%numVar==0)
                        cuenta=-1;
                cuenta++;
          }
          if(total%numVar!=0) {
                clrscr();
                printf("\n\nEl archivo de datos esta incorrecto. Faltan o sobran datos");
                scanf("%c%c");
          }
          else {
        numEjem=total/numVar;
        for(total=0;total<numVar;total++)
                        for(cuenta=11;cuenta>-1;cuenta--)
                          if(p2[cuenta]>bits[total]) {
                                bits[total]=11-cuenta;
                                break;
                          }
                printf("\n");
                i=0;
                LenN=0;
                LenM=0;
                printf("\nEntra el numero de variables de salida: ");
                scanf("%d",&nsalidas);
```

```
                for(i=numVar-nsalidas;i<numVar;i++)
                        bits[i]=9;
                for(i=0;i<(numVar-nsalidas);i++)
                        LenN+=bits[i];
                for(i=numVar-nsalidas;i<numVar;i++)
                        LenM+=bits[i];
                letcon();
        }
       }
}

configArch()    /* rutina de configuración por archivo: lee los datos de un archivo de usuario */
{
        int i,j;
        char op;
        printf("\nArchivo de configuración: ");
        scanf("%s",&fileArch);
        if( (fpA=fopen(fileArch,"r"))==NULL) {
          printf("No se encuentra el archivo %s",fileArch);
          scanf("%c%c");
        }
        else {
          fscanf(fpA,"%d",&numVar);
          fscanf(fpA,"%d",&nsalidas);
          i=0;
          LenN=0;
          LenM=0;
          while( fscanf(fpA,"%d",&bits[i]) != EOF ) {
                if(i>=numVar) break;
                if(i<(numVar-nsalidas)) LenN+=bits[i];
                else LenM+=bits[i];
                i++;
          }
          for(j=numVar-nsalidas;j<numVar;j++)
                bits[j]=9;      /* obligo a que las salidas sean de 9 bits */
          if(i>numVar) {
                printf("\n\nEl vector BITS es mayor al numero de variables");
                scanf("%c%c");
          }
          if(i<numVar-1) {
                printf("\n\nEl vector BITS es menor al numero de variables. Se abortara");
                scanf("%c%c");
          }
          else letcon();
        }
}

letcon()                        /* despliega la configuración actual del sistema  */
{
        char op;
        scanf("%c");
        clrscr();
        printf("  CONFIGURACION LEIDA DE DISCO\n\n");
        printf("  # de  variables  totales: %d\n",numVar);
        printf("  # de  ozonos  de  salida: %d\n",nsalidas);
        printf("  # de neuronas de entrada: %d\n",LenN);
        printf("  # de neuronas de  salida: %d\n",LenM);
        printf("  # de ejemplos en archivo: %d\n",numEjem);
        printf("\n\nEs correcta la configuración [s/n]?: ");
        scanf("%c",&op);
```

```
        if(op=='n' || op=='N')   /* la configuración leida no es correcta. Termina */
          exit(status);
        fclose(fpA);
}

binario()                       /* conversión entero-binaria del archivo de entrenamiento */
{
        int Vec[33],len,i,j,if1;
        char aux[12],num[12],línea[305],línea2[105];
        printf("\nArchivo de datos reales: ");
        scanf("%s",&fileIn);
        if( (fpI=fopen(fileIn,"r"))==NULL) {
          printf("No se encuentra el archivo %s",fileIn);
          scanf("%c%c");
        }
        else {
          printf("\nArchivo binario de salida: ");
          scanf("%s",&fileOut);
          printf("\n\n\n");
          printf("      ...CONVIRTIENDO A BINARIO...");
          fpO=fopen(fileOut,"w");
          j=0;
          if1=numVar-nsalidas;
          while( fscanf(fpI,"%d",&Vec[j]) != EOF )  {
                  if(j==0) {
                          strcpy(línea,"");
                          strcpy(línea2,"");
                  }
                  itoa(Vec[j],num,2);        /* convierte a binario */
                  len=bits[j]-strlen(num);   /* completa a los bits adecuados */
                  strcpy(aux,"");
                  strncat(aux,"00000000000",len);
                  strcat(aux,num);
                  if(j<if1)
                          strcat(línea,aux);        /* va generando la línea binaria */
                  else
                          strcat(línea2,aux);
                  if(j==numVar-1) {
                          fprintf(fpO,"%s\n",línea);
                          fprintf(fpO,"%s\n",línea2);
                  }
                  if(j==(numVar-1))
                          j=-1;
                  j++;
           }
          }
          fclose(fpI);
          fclose(fpO);
}
IniciaPesos()                   /* rutina de incialización de conexiónes y umbrales */
{
  int i,j;
  srand(13);
  for(i=0;i<LenM;i++) {
          VecUmbral[i]=(rand()%10)*0.1;
          for(j=0;j<LenN;j++)
                  W[i][j]=(rand()%20)*0.05;
  }
}
```

```
PTruena()          /*  rutina del criterio de paro para el aprendizaje del modelo de retropropagación */
{
 int i,j,Truena=0;
 fclose(fpO);
 fpO=fopen(fileOut,"r");
 while ( fgets(ins,LenN+5,fpO)!=NULL && !Truena)  {
          fgets(outs,LenN+5,fpO);
          for(i=0;i<LenN;i++)
                   VecEntrada[i]=(ins[i]=='1')?1:0;
          for(i=0;i<LenM;i++)
                   VecSalida[i]=(outs[i]=='1')?1:0;
          for(i=0;i<LenM;i++)  {
                   Net=0;
                   for(j=0;j<LenN;j++)
                     Net+=VecEntrada[j]*W[i][j];
                   Net-=VecUmbral[i];
                   VecInt[i]=(Net<0)?0:1;
          }
          i=0;
          do  {
                   Truena=VecSalida[i]!=VecInt[i];
                   i++;
          }  while (i!=LenM && !Truena);
 }
 return((Truena==0)?1:0);
}

train()                                   /* rutina del menu de entrenamiento */
{
          int op;
          printf("\n      MENU DE APRENDIZAJE\n\n");
          printf("       Archivo completo.....1\n");
          printf("       Solo N ejemplos.....2\n");
          printf("       Finalizar.....3\n\n");
          printf("            Op > ");
          scanf("%d",&op);
          clrscr();
          switch(op) {
                   case 1: train2(1); break;
                   case 2: train2(2); break;
          }
}

train2(int caso)                          /* rutina de entrenamiento */
{
  int num=0,Continua=0,ndata,i,j;
  if(banArchBin==0)  {
           printf("\nArchivo de datos binario: ");
           scanf("%s",&fileOut);
  }
  if( (fpO=fopen(fileOut,"r"))==NULL) {
                   printf("No se encuentra el archivo %s",fileOut);
                   scanf("%c%c");
  }
  else  {
           if(caso==2) {
                   printf("\n# de datos a entrenar: ");
                   scanf("%d",&ndata);
           }
           else ndata=numEjem;
```

```
        IniciaPesos();
        gotoxy(32,10);
        printf("...ENTRENANDO...");
        gotoxy(37,11);
        printf("EJEM");
        while (!Continua)  {          /* lee los archivos */
                num++;
                gotoxy(41,11);
                printf("%d ",num);
                fgets(ins,LenN+5,fpO);
                fgets(outs,LenN+5,fpO);
                for(i=0;i<LenN;i++)
                        VecEntrada[i]=(ins[i]=='0')?0:1;
                for(i=0;i<LenM;i++)
                        VecSalida[i]=(outs[i]=='0')?0:1;
                for(i=0;i<LenM;i++)  {          /* entrena */
                        Net=0;
                        for(j=0;j<LenN;j++)
                                Net+=VecEntrada[j]*W[i][j];
                        Net-=VecUmbral[i];
                        VecInt[i]=(Net<0)?0:1;
                }
                for(i=0;i<LenM;i++)
                        for(j=0;j<LenN;j++)
                                W[i][j]+=CoefAc*(VecSalida[i]-VecInt[i])*VecEntrada[j];
                if(PTruena())
                        Continua=1;
                else  {
                        fclose(fpO);
                        fpO=fopen(fileOut,"r");
                        if(num<ndata)
                                for(i=1;i<=num;i++) {
                                  fgets(ins,LenN+5,fpO);
                                  fgets(outs,LenN+5,fpO);
                                }
                        else num=0;
                }
        }
  }
  fclose(fpO);
}


recall(char car)                                /* rutina pre-recall */
{
        entrada();
        runPercep();
        display(VecInt,car);
}

recallXejem()                   /* rutina de pre-recall: opera con una muestra interactiva */
{
  printf("\n\n\n\n");
  printf("       ...EJECUTANDO RECALL X EJEMPLO.. ");
  recall('E');
  scanf("%c%c");
}

recallXarch()                   /* rutina de pre-recall: opera con el archivo de prueba */
{
```

```
int i,j,z;
printf("\nArchivo de prueba: ");
scanf("%s",&fileIn);
printf("\nArchivo de salida: ");
scanf("%s",&fileArch);
printf("\nDesviación: ");
scanf("%d",&margen);
if( (fpI=fopen(fileIn,"r"))==NULL || (fpA=fopen(fileArch,"w"))==NULL ) {
        printf("No se encuentra(n) el(los) archivo(s)");
        scanf("%c%c");
}
else {
        for(i=0;i<nsalidas;i++) {
                ceros[i]=0;
                buenos[i]=0;
        }
        gotoxy(23,10);
        printf("...EJECUTANDO RECALL X ARCHIVO...");
        gotoxy(36,11);
        printf("EJEM");
        i=0;
        z=0;
        while( fscanf(fpI,"%d ",&entra[i]) != EOF ) {
                if(i%(numVar-nsalidas)==(numVar-nsalidas)-1) {
                  for(j=numVar-nsalidas;j<numVar;j++)
                        fscanf(fpI,"%d ",&entra[j]);
                  z++;
                  gotoxy(41,11);
                  printf("%d ",z);
                  recall('A');
                  fprintf(fpA,"\n");
                  i=-1;
                }
                i++;
        }
        fclose(fpI);
        fprintf(fpA,"\n#: %d, CEROS:",z);
        for(i=0;i<nsalidas;i++)
                fprintf(fpA," %d",ceros[i]);
        fprintf(fpA,", ACIERTOS:");
        for(i=0;i<nsalidas;i++)
                fprintf(fpA," %d",buenos[i]);
        fprintf(fpA,", PORCENTAJE:");
        for(i=0;i<nsalidas;i++)
                fprintf(fpA," %d",100*buenos[i]/z);
        fclose(fpA);
 }
}

mainrecall()                                    /* rutina del menu de recall */
{
        int op;
        printf("\n       MENU DE RECALL\n\n");
        printf("      Recall X ejemplo..1\n");
        printf("      Recall X archivo..2\n");
        printf("      Finalizar.....3\n\n");
        printf("           Op > ");
        scanf("%d",&op);
        clrscr();
        switch(op) {
```

```
                case 1: recallXejem(); break;
                case 2: recallXarch(); break;
        }
}

runPercep()             /* rutina del recall del modelo de retropropagación */
{
        int i,j;
        for(i=0;i<LenM;i++) {
          Net=0;
          for(j=0;j<LenN;j++)
                Net+=VecEntrada[j]*W[i][j];
          Net-=VecUmbral[i];
          VecInt[i]=(Net<0)?0:1;
        }
}

display(arreglo,car)    /* despliega el resultado en pantalla ó a un archivo de salida */
int arreglo[30];
char car;
{
  int i,index=numVar-nsalidas-1,total=0;
  char sino;
  if(car=='E') {
        clrscr();
        printf("RESULTADO: ");
  }
  for(i=0;i<LenM;i++) {
        total+=p2[(i%9)+3]*((arreglo[i]==1)?1:0);
        if(i%9==8) {
                index++;
                if(car=='E')
                  printf("%d ",total);
                else {
                  if(entra[index]!=0)
                        if(total<=(entra[index]+margen) && total>=(entra[index]-margen)) {
                                sino='+';
                                buenos[index-(numVar-nsalidas)]++;
                                media[index-(numVar-nsalidas)]+=100*(total-entra[index])/entra[index];
                        }
                        else sino=' ';
                  else {
                                sino='+';
                                ceros[index-(numVar-nsalidas)]++;
                                buenos[index-(numVar-nsalidas)]++;
                  }
                  sino=sino;
                  fprintf(fpA,"%d\t%d\t",entra[index],total);
                }
                total=0;
        }
  }
}

entrada()               /* rutina de conversión entero-binaria de cada muestra del archivo de prueba */
{
        char aux[12],num[12],línea[305];
        int i,len;
          strcpy(línea,"");
          for(i=0;i<(numVar-nsalidas);i++) {
```

```
                itoa(entra[i],num,2);          /* convierte a binario */
                len=bits[i]-strlen(num);    /* completa a los bits adecuados */
                strcpy(aux,"");
                strncat(aux,"00000000000",len);
                strcat(aux,num);
                strcat(línea,aux);          /* va generando la línea binaria */
        }
        for(i=0;i<LenN;i++)
                VecEntrada[i]=(línea[i]=='1')?1:0;
}
}
```