Simulación de Redes Neuronales. Memoria Bidireccional Bivalente Adaptiva: BAM

M.C. Miguel A. Partida Tapia
Subdirector Académico y de Investigación del CINTEC-IPN.
Ing. Rubén Peredo Valderrama e
Ing. Francisco F. Córdova Quiróz
Alumnos de la Maestría del CINTEC-IPN.

El presente artículo tiene como finalidad que el lector se familiarice con los conceptos sobre Redes Neuronales y como ejercicio simular el modelo básico propuesto por Bart Kosko [1988],una memoria bidireccional asociativa (**BAM**) en el lenguaje C++ orientado a objetos de Borland, versión 4.02 para Microsoft Windows 3.11.

Introducción

Tratando de superar los problemas y limitantes de los métodos y técnicas para simular Redes Neuronales, se han abordado y utilizado nuevos enfoques. Entre ellos están:

1. La Teoría de Conjuntos Difusos (Fuzzy Sets), desarrollada por Lofti Zadeh en 1965.

2. Los Autómatas Celulares (descendientes de la Teoría de Autómatas, pero aplicando muchos de los conocimientos de los cultivos celulares y su crecimiento).

3. Los Algoritmos Genéticos (desarrollados por John Holland en 1975, que consisten en una mímica de los procesos naturales de mutación, evolución, y selección natural, para la solución de un problema).

4. La Dinámica Matemática (también llamada *Ciencia del Caos* **[9]**, y que se encarga del estudio de la Dinámica no Lineal).

Sin embargo, el enfoque que ha resultado ser el más poderoso y prometedor, y que posee sólidas bases en la biología, en la fisiología y, particularmente, en las neurociencias, es el conocido como Redes Neuronales. Como su nombre lo indica, se trata de estructuras artificiales que simulan el comportamiento de las neuronas naturales.

Los inicios se dieron con los trabajos de Santiago Ramón y Cajal, realizados a finales del siglo XIX. Estos nos permitieron descubrir los elementos que componen al cerebro: las **neuronas**. Su funcionamiento fue descrito posteriormente por Sherrington, quien también identificó a las **sinapsis**.

En 1943, Warren McCulloch y Walter Pitts publicaron un modelo matemático sobre el comportamiento de las neuronas. El impacto que causó, desencadenó una gran actividad en el estudio de la lógica neuronal. Como resultado de todo esto surgieron los **Perceptrones**, y en él *demostraban* la incapacidad de estos sistemas para la solución de problemas lógicos sencillos (más tarde se comprobó que el problema no había sido estudiado tan a fondo como se pretendía y que los resultados de Minsky y Papert eran solo particulares). Debido a esto, durante mucho tiempo, las investigaciones en redes neuronales permanecieron en el olvido.

A principios de la década de los 80's, y gracias a los avances logrados en el procesamiento paralelo, surge el **Conexionismo**, como antecedente de la actividad actual en redes neuronales. El impulso inicial fue dado por los trabajos de John Hopfield quien retomó los arreglos neuronales demostrando que no habían sido estudiados lo suficiente por Minsky y Papert, y probó que estos sistemas tienden a generar procesos exponenciales que pueden ser fácilmente definidos y explicados por funciones no lineales.

Bastaron sólo 5 años para que se formara una sólida sociedad internacional para el avance de las redes neuronales. En este periodo salieron a la luz muchos trabajos de investigadores que habían trabajado en el anonimato o en la clandestinidad. Igualmente, los avances logrados se vieron enriquecidos por los conocimientos sobre el sistema nervioso central y el proceso de aprendizaje obtenidos por investigadores como Werbs, Widrow y Hebb.

Los conocimientos y técnicas aportados por la recién nacida Dinámica Matemática también han sido un factor decisivo para el análisis y formalización de los sistemas estudiados. Ahí, donde antes estaba el caos, se ha encontrado orden, en donde se veía aleatoriedad, se ha localizado regularidad. Se intuía la existencia de un determinado orden en el cerebro, más las caóticas señales de las neuronas hacían difícil su identificación . La *Ciencia del Caos* nos ha brindado conocimientos sobre los procesos concurrentes y recursivos; comportamiento no lineal, puntos de atracción y procesamiento caótico de señales.

La Inteligencia Artificial y las Redes Neuronales son dos campos diferentes con un mismo propósito: reproducir las habilidades cognoscitivas de humanos y animales mediante máquinas y dispositivos. Con todo y los logros obtenidos en Inteligencia Artificial, en la actualidad las aplicaciones de las redes neuronales son muy diversas y abarcan diferentes campos, usándose exitosamente en: diagnóstico médico, aprobación de crédito y seguros, control e inspección industrial, modelamiento económico, procesamiento de señales y filtrado de ruido, mercadotecnia, predicción bursátil y elaboración de presupuestos, detección de fraudes, reconocimiento de formas, generación de voz, identificación por radar o sonar, corrección ortográfica, desciframiento y recuperación de información.

Redes Neuronales

Descripción de las redes neuronales.

Las Redes Neuronales son modelos matemáticos que procesan en forma paralela información como lo realizan los sistemas biológicos mediante elementos simples llamados neuronas. La simulación de las Redes Neuronales es la experimentación con modelos, típicamente implementados en computadoras. La simulación por computadora de las redes neuronales tiene dos distintas aplicaciones:

1.- Estudio de modelos de sistemas biológicos.

2.- Estudio de redes neuronales artificiales usados para procesos computacionales y de control.

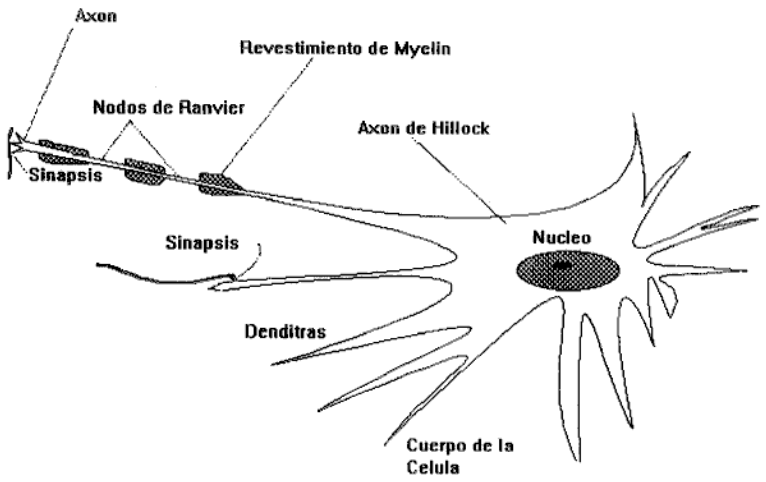


Figura 1

Anatomía de una neurona típica. La mayoría de las dendritas son del orden de 1mm de longitud, mientras que algunos axones encontrados han alcanzado hasta 1m. El axon muestra aquí un recubrimiento de Mylin, el cual previene los intercambios de iones excepto en el Nodo de Ranvier. La propagación del pulso puede alcanzar hasta 120 m/seg .

Circuitos neurales y su representación

Las estructura de las Redes Neuronales se define como una colección de *procesadores paralelos* conectados entre sí organizados como gráficas dirigidas, conformando una RED. Cada nodo representa un Elemento de Procesamiento (PE), conectado por medio de arcos. En la **figura 2** se muestran algunos tipos de redes.

Los elementos en las redes que quedan entre los PE de entrada y la salida se le llaman *niveles escondidos* (**hidden-layer**) y las conexiones de entrada y salida son definidas por *conexiones pesadas* (**weighted connections**).

En la **figura 2** las gráficas (a) y (b), ilustran los principios de divergencia y convergencia en un circuito neural. Cada neurona envía impulsos a muchas otras neuronas (*divergencia*), y recibe impulsos de muchas otras (*convergencia*). Las gráficas (b), (c), y (d) contienen retroalimentaciones. Las conexiones sinápticas pueden ser exitatorias e inhibitorias, estos circuitos facilitan sistemas de control ya que se pueden realizar retroalimentaciones positivas y negativas.

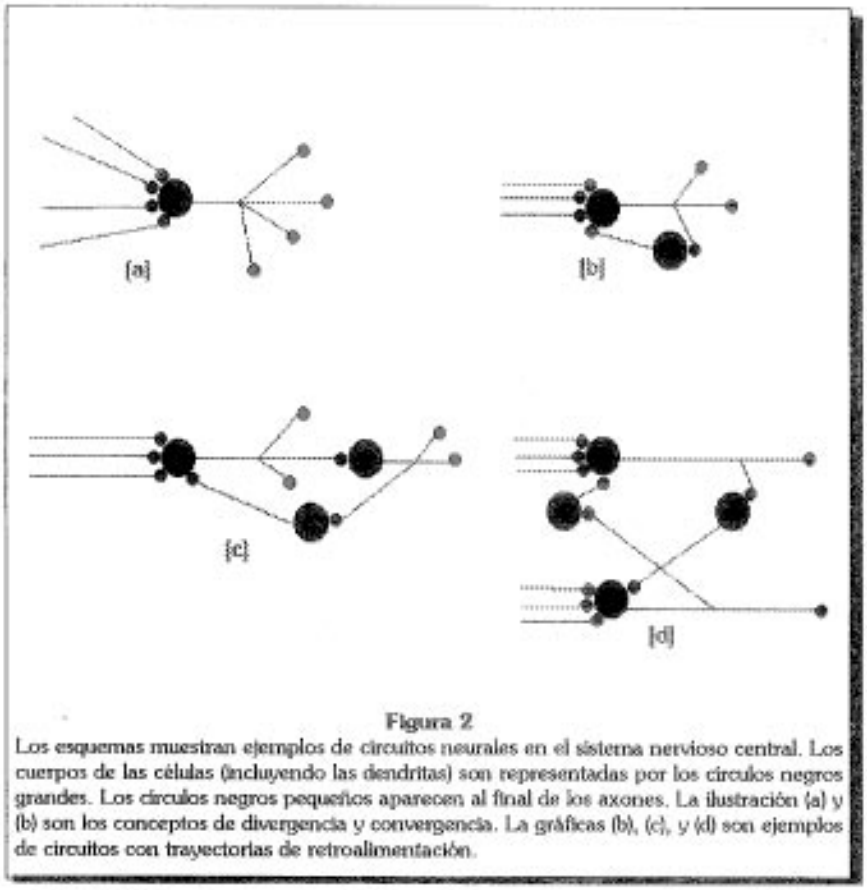


Figura 2

Los esquemas muestran ejemplos de circuitos neurales en el sistema nervioso central. Los cuerpos de las células (incluyendo las dendritas) son representadas por los círculos negros grandes. Los círculos negros pequeños aparecen al final de los axones. La ilustración (a) y (b) son los conceptos de divergencia y convergencia. La gráficas (b), (c), y (d) son ejemplos de circuitos con trayectorias de retroalimentación.

En 1943, los trabajos de McCulloch y Pitts establecieron cinco reglas fundamentales para el estudio de las redes neuronales como son:

1. La actividad de una neurona siempre está disponible en el proceso.

2. Un cierto número fijo de sinapsis (> 1) podrán ser exitadas en un período de latencia adicional para una neurona dada.

3. El único retardo significativo en un sistema nervioso es el retardo sináptico.

4. La actividad de cualquier sinapsis inhibitorio absolutamente previene excitación de una neurona en un tiempo dado.

5. La estructura de la interconexión de la red no cambia con el tiempo.

Asumiendo que un 1 identifica la neurona como un principio binario: Esto implica una razón *on/off*. Para definir la notación, indicaremos como $N_i(t)$, lo cual denota que la neurona i_{esimo} se dispara en un tiempo t. la notación, $!N_i(t)$, denota que la neurona i_{esimo} no se disparará en el tiempo t. Usando esta notación, nosotros podemos describir la acción de ciertas redes neuronales usando proposiciones lógicas.

En la **figura 3** (siguiente página) se muestran cinco redes simples. A continuación se describe la notación para cada uno de los ejemplos, en la figura (a) describe a dos neuronas, donde la neurona 2 se dispara después de la neurona 1.

Las expresiones quedan como sigue:

$N2(t) = N1(t-1)$ (a)

$N3(t) = N1(t-1) \vee N2(t-1)$ (b)
(Disyunción)

$N3(t) = N1(t-1) \,\&\, N2(t-1)$ (c)
(Conjunción)

$N3(t) = N1(t-1) \,\&\, !N2(t-1)$ (d)
(Conjunción negada)

Una prueba palpable de esta teoría fue que cualquier red que no contenga conexiones de retroalimentación puede describirse en términos de combinación de estas simples expresiones, y viceversa.

Elemento de procesamiento general

Los elementos computacionales simples que representan a un sistema neural son llamados *Neuronas Artificiales*, estos están referidos a nodos, unidades, o elementos de procesamiento (PE). En la **figura 4** (siguiente página) se muestra un modelo de PE. Cada uno de los PE es numerado, del 1 hasta i_{esimo}

De igual manera que en las neuronas biológicas, los PE tienen muchas entrada y una sola salida. Cada entrada es representada como x_j, y a cada una de estas se le asocia un ***peso*** o fuerza de ***conexión***. Los pesos o conexiones definen de donde provienen j_{esimo} nodo hasta el nodo i_{esimo} y es representado como w_{ij}. La salida PE corresponde a la frecuencia de disparo de la neurona, y los pesos corresponden a la fuerza de las conexiones sinápticas entre neuronas. En este modelo, estas cantidades pueden ser representadas como números reales.

Un PE puede aceptar varios tipos de entradas, dado que cada una de ellas realizan diversos efectos. Una conexión de entrada puede ser exitatoria o inhibitoria, por ejemplo. Para nuestro propósito de estudio, definiremos a las conexiones exitatorias con ***pesos positivos*** y a las conexiones inhibitorias con ***pesos negativos***. Otros tipos pueden ser posibles. Cada PE determina una

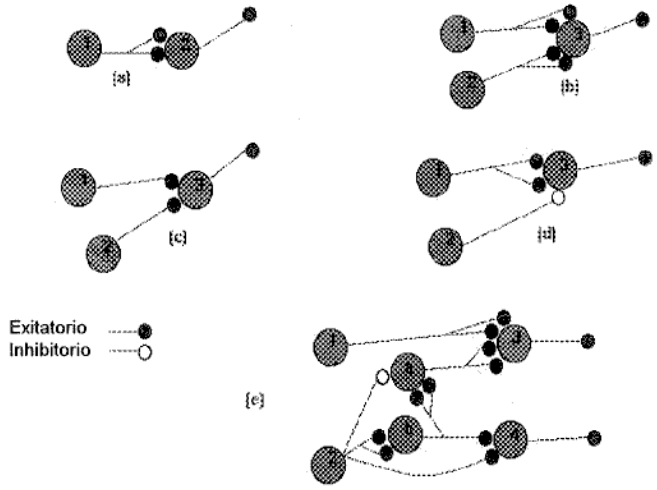


Figura 3

En la figura se muestran ejemplos de redes simples expuestas por McCulloch-Pitts que pueden ser definidas en términos de la notación proposicional lógica. Los círculos grandes indican células nerviosas, los círculos pequeños llenos representan conexiones exitatorias, los vacíos conexiones inhibitorias. La red (a) ilustra precedencia, (b) disyunción, (c) conjunción, y (d) conjunción negada. La red (e) es una combinación de las anteriores redes (a) - (d).

red de entradas en base a todas las conexiones de entrada. En ausencia de conexiones especiales, nosotros calculamos típicamente la red de entradas por suma de todos los valores de entrada, multiplicado cada uno de ellos por su correspondiente peso. En otras palabras, la red de entradas de i_{esima} describe la siguiente expresión:

$$net_i = \sum_j x_j w_{ij} \qquad (1)$$

donde el índice, *j*, recorre todas las conexiones del PE.

Note que la excitación y la inhibición son limitadas automáticamente por el signo de los pesos. El cálculo de la suma de productos juega una regla importante en la simulación de las redes neuronales. Por el gran número de interconexiones en las redes, la velocidad del cálculo determinará el rendimiento de la simulación de una red neuronal dada.

Cada una de las redes de entrada es calculada, y convertida a un *valor de activación* o simplemente *activación* para PE. Podemos, entonces, escribir el valor de activación como:

lo cual denota que la activación está implícita en la función de la red de entrada.

Modelos Aditivos Bivalentes

La activación de modelos aditivos discretos corresponden a neuronas con funciones de señales de umbral. Las neuronas pueden asumir únicamente dos valores, **ON** y **OFF**. **ON** representa al valor de la señal +1. **OFF** representa 0 ó -1. Estos valores, **bivalentes**, corresponden a modelos de redes neuronas contenidas en el modelo clásico de McCulloch y Pitts [1943].

Los modelos bivalentes pueden representar comportamientos asíncronos y estocásticos. Estas propiedades ofrecen valores prácticamente pequeños en simulaciones a pequeña escala. Existen dos tipos de redes neuronales bivalentes aditivos: la no adaptiva, la memoria asociativa adaptiva bivalente bidireccional (**BAM**) y la autoasociativa o modelo de **Hopfield**.

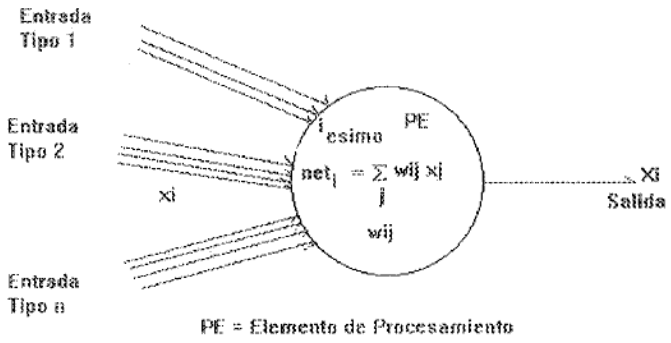


Figura 4

Esta estructura representa un elemento simple de procesamiento (PE) en una red. Las conexiones son representadas por flechas desde otros elementos de procesamiento. Cada una de las conexiones de entrada están asociadas con una cantidad, $\mathbf{w_{ij}}$, llamada **pesos**. Estos elementos tienen una única salida, la cual puede ser conectada en **abanico** con otras unidades.

Modelos Existentes

Existen varios tipos de redes neuronales; en la actualidad su número llega a sobrepasar los cuarenta modelos. Algunos de estos son sencillos y muy limitados, mientras que otros son sumamente flexibles y poderosos. Para cada rama en la clasificación mostrada existe un modelo que la caracteriza. Como se describen a continuación.

- **Modelo de Hopfield.** Con base a los trabajos de Shunichi Amari, de 1977, Jonh Hopfield presentó en 1982 un modelo sumamente sencillo de una red recurrente de tipo discreto, y que se ha constituido como modelo canónico de las de las redes neuronales con retroalimentación. Este modelo asocia el patrón presentado consigo mismo, y permite su posterior recuperación al presentar un patrón completo o impreciso. En estudios subsecuentes de Hopfield y otros investigadores se han presentado variantes continuas, estadísticas y generalizadas. Las principales características de este modelo son: generalización, estabilidad, convergencia de estados y autoorganización.
- **Memoria Asociativa Bidireccional.** Este modelo, identificado comúnmente como **BAM**, por sus siglas en inglés, constituye una extensión generalizada de las redes de Hopfield. Fue presentada por Bart Kosko en 1988. El modelo permite la asociación de patrones diferentes, a manera de una memoria hetereoasociativa. Para la **BAM**, las redes de Hopfield constituyen un caso especial dentro del conjunto de datos con los que se trabaja, por lo que ésta exhibe características de este modelo y otras adicionales.
- **Modelo ART.** (Adaptive Resonance Theory). Este modelo es muy complejo y se emplea más como objeto de investigación. Surgió como resultado de los trabajos de Gail Carpenter Stephen Grossberg realizados entre 1986 y 1987 sobre modelación de los procesos macroscópicos de la actividad cerebral. Su funcionamiento parte de la convergencia de las señales hacia patrones estables por la reverberación de las señales en los elementos de procesamiento. El modelo es capaz de proporcionar abstracción y generalización.
- **Asociador Lineal.** Utiliza la regla de aprendizaje de Hebb. El modelo sólo funciona si los patrones son ortogonales y su capacidad es de apenas un 10% a 20% con respecto al número de neuronas presentes.
- **Propagación Inversa.** (Backpropagation). El modelo parte del algoritmo publicado por Rumelhart, Hilton y Williams en 1985, aunque mucho de su trabajo ya había sido anticipado por Parker en 1982 y anteriormente por Werbos en 1974. Las redes neuronales de propagación inversa son netamente multiestrato; éstas se modifican a sí mismas en el mismo instante en que generan patrones de salida, por lo que resultan sumamente flexibles y de gran poder de generalización. Su entrenamiento es cíclico y consume demasiado tiempo y recursos de cómputo.
- **Neocognitron.** Esta es la versión de 1980 del Cognitron, ambos desarrollados por Kunihiko Fukushima, quien también publicó una versión supervisada en 1983. Este modelo es el resultado de las investigaciones de Fukushima sobre los órganos visuales en gatos y macacos. El Neocognitron es una red neuronal cuyo funcionamiento radica en su algoritmo de aprendizaje, estructura de las capas y los diversos tipos de conexiones manejados. En conjunto, la organización del modelo se asemeja mucho a la del sistema visual de pequeños mamíferos. El Neocognitron provee las capacidades de su antecesor y muchas más, puede reconocer patrones eliminando translación, rotación, distorsión, cambios de escala, y puede seleccionar entre dos patrones presentados simultáneamente, proporcionando una gran capacidad de generación. Computacionalmente hablando, el modelo resulta excesivamente costoso en tiempo y espacio.
- **Modelo de Kohonen.** Teuvo Kohonen ha propuesto a través de su obra un modelo no supervisado de memoria asociativa. En este caso el grupo de neuronas que conforman la red se comportan en la forma de "El ganador toma todo", en una arquitectura de un solo estrato.

Estructura del la BAM

La estructura de la **BAM** se compone de 2 estratos, identificados como los grupos de neuronas $\{a_1,a_2,\ldots,a_n\}$ y $\{b_1,b_2,\ldots,b_n\}$. En cada estrato, cada neurona **a** se encuentra conectada con cada neurona **b** y viceversa. Las **n x p** conexiones generadas conforman una matriz denominada ***matriz de correlación***, representada por $\boldsymbol{W}$. Por el método utilizado en la construcción de $\boldsymbol{W}$ todas las conexiones sinápticas son consideradas de la misma longitud y de tipo exitatorio. Estas conexiones satisfacen nuestras necesidades de proceso y de almacenamiento de información siendo suficientes para explotar las características de estabilidad y bidireccionalidad de la matriz de conexiones. Por no ser necesarias, se

omiten otro tipo de conexiones (de retroalimentación inhibitorias y de retardo) y de funciones para el procesamiento de las señales.

Por la sencillez del modelo, el comportamiento de la neurona de la **BAM** es síncrono. Este no corresponde del todo al comportamiento de las neuronas biológicas, asíncrono por naturaleza. Aunque, si se desea hacer una comparación, puede llegar a equiparse con el comportamiento de las neuronas cuando, en grupos pequeños, son capaces de sincronizar sus impulsos.

Funcionamiento y construcción de la BAM

La agrupación de los valores que asumen en un mismo instante todas las neuronas en cada estrato se puede tratar como una ***cadena de bits*** representando un ***patrón***, o como un ***vector*** indicando el ***estado*** del sistema. Las combinaciones posibles de vectores o cadenas forman dos conjuntos; $\{A_1, A_2, ..., A_n\}$, para las neuronas **a**, y $\{B_1, B_2, ..., B_n\}$, para las neuronas **b**. Así, los elementos A_i y B_i provienen de los espacios $\{0,1\}^n$ y $\{0,1\}^p$, respectivamente.

La ***asociación*** de dos vectores se establece durante su almacenamiento, y se representa como el par vectores, $A_i^T B_i$, lo cual dará por resultado una matriz que será agregada a W mediante una operación de adición que se hará para todos y cada uno de los pares a asociar.

$$W = \sum_k^m A_k^T B_k \qquad (2)$$

El algoritmo de almacenamiento por correlación constituye una aproximación discreta del aprendizaje Hebbiano. Podemos ver que cada neurona constituye una memoria a corto plazo, vigente en el momento de presentación o recuperación de la información. La memoria a largo plazo es formada por las conexiones, que son reforzadas o debilitadas por las neuronas involucradas.

Si los valores en la matriz de conexiones representa la resistencia sináptica entre las neuronas, entonces es necesario tener en cuenta los valores inhibitorios de cada neurona para la correspondiente generación de conexiones inhibitorias. Por el carácter aditivo de las operaciones de almacenamiento de los 0's en los patrones binarios son ignorados; sin embargo, si empleáramos un código ***(bivalente)*** los estados inhibitorios sí serían tomados en cuenta. La razón principal reside en el hecho de que 1+0=1, pero 1+(-1)=0. Para esto deberemos transformar los vectores binarios a

$$Y_i = fc_B(B_i) \qquad (4)$$

$$fc_B = \left.\begin{matrix} y_k = 1, \rightarrow b_k = 1 \\ y_k = -1, \rightarrow b_k = 0 \end{matrix}\right\} \forall k = 1,2,....,p$$

y los principios de creación W continúan siendo válidos al utilizar X_i y Y_i, por lo tanto la ecuación de matriz de pesos queda como sigue:

$$W = \sum_k^m X_k^T Y_k \qquad (5)$$

El carácter bipolar de X_i y Y_i agrega características especiales al almacenamiento de las asociaciones. Primero, tenemos que para cada vector X_i existe un ***complemento*** X_i^c, tal que $X_i^c = -X_i$, y similarmente, $Y_i^c = -Y_i$. Si consideramos que $X_i^T Y_i = (-X_i^T)(Y_i) = (X_i^c)^T (Y_i^c)$, veremos que tanto los vectores como sus complementos crean la misma matriz de conexiones. Entonces el codificar (A_i, B_i), también guarda en la memoria a (A_i^c, B_i^c) y viceversa. Nótese que (A_i, B_i) puede ***borrarse*** de W almacenando $X_i^T Y_i^c = -X_i^T Y_i$, dado el carácter aditivo de W.

Una **BAM** se comportará como una red de Hopfield cuando $A = B$ y, siendo la **BAM** una generalización de este modelo, la presentación y recuperación deberá igualmente hacerse mediante procesos recu-

$$\frac{\Delta E}{\Delta b_k} = -AW^k \qquad (8)$$

$\Delta \mathbf{b}_k > \mathbf{0}$ sólo es posible si la entrada en la neurona $\mathbf{b}_k > \mathbf{0}$ ($\mathbf{AW}^k > 0$), y $\Delta \mathbf{b}_k < \mathbf{0}$ si y sólo si $\mathbf{AW}^k < \mathbf{0}$; así, en cualquier caso $\Delta \mathbf{E} = -\Delta \mathbf{b}_k (\mathbf{AW}^k) < 0$, cuando $\Delta \mathbf{a}_k = \mathbf{0}$, $\Delta \mathbf{b}_k = \mathbf{0}$, $\Delta \mathbf{E}_k = \mathbf{0}$.

Resumiendo, cualquier cambio en el vector **A** o en el vector **B**, en la recuperación, resulta en un decremento de **E** dada la concordancia de signos del producto ***vector-matriz*** y el cambio de estado en las neuronas y, todo cambio en **E** es por una cantidad finita, a través de trayectorias discretas en $\{0,1\}^n$ **X** $\{0,1\}^p$. Entonces la convergencia a una solución, y que consiste en determinar el límite de los decrementos en **E**. La máxima variación estará dada por un cambio de estado en todas las neuronas del estrato, dado en cantidades discretas de 1 ó -1, donde:

$$E(A,B) \geq -\sum_i^n \sum_j^p \left| w_{ij} \right| \quad \forall \mathbf{A},\mathbf{B} \qquad (9)$$

Dado que ***W*** se mantiene invariante en la recuperación, **E** se encuentra acotada, lo que nos asegura que eventualmente los vectores **A** y **B** dejarán de cambiar alcanzando un punto fijo ($\mathbf{A}_f$, $\mathbf{B}_f$) que es un ***mínimo local*** o un ***punto de atracción***, esto demostrará que el sistema es ***estable***.

Capacidad. Teorema de Saturación

Este teorema establece que los modelos aditivos se saturan mientras que esto no ocurre en los modelos multiplicativos. De acuerdo con esto, y al ser un modelo aditivo, la **BAM** debe presentar un límite en su almacenamiento al saturar los valores de sus conexiones.

La saturación y los límites en la capacidad de la **BAM** dependen de su dimensionalidad, y aquélla sólo es perceptible hasta que se efectúe la recuperación de uno de los patrones almacenados, ya que durante el almacenamiento no tenemos forma de saber si la adición de una asociación altera significativamente o borra otra. La recuperación de un vector está determinado por el respectivo producto ***vector-matriz*** (7). Dicho producto puede expanderse para recuperar a un $\mathbf{B}_i$, como:

$$A_i W = (A_i X_i^T) Y_i + \sum_{j \neq i}^{m} (A_i X_j^T) Y_j \qquad (10)$$

Así tomando (2,3 y 4), tendremos:

$$X_i W = (X_i X_i^T) Y_i + \sum_{j \neq i}^{m} (X_i X_j^T) Y_j$$

$$= nY_i + \sum_{j \neq i}^{m} (X_i X_j^T) Y_j$$

$$= \sum_j c_{ij} Y_j \qquad (11)$$

El factor ***n*** es el valor máximo que se puede obtener para amplificar las características de $\mathbf{Y}_i$. Los coeficientes de ***ruido***, dados por $\mathbf{X}_i \mathbf{X}_i^T$, pueden ayudar a corregir las distorsiones producidas por $\mathbf{Y}_i$ en grado proporcional en la similitud de $\mathbf{X}_i$ con $\mathbf{X}_j$. De manera similar podemos concluir que el factor de ***corrección de ruido*** en el caso de la recuperación de $\mathbf{A}_i$ es **p**. El factor de ruido definido en (11) está determinado por la cantidad de ***m*** patrones almacenados en ***W***, por lo que debemos esperar recuperaciones fallidas si $\mathbf{m} > \mathbf{n}$ o $\mathbf{m} > \mathbf{p}$. De ocurrir esto, la suma de los factores de ruido excedería al factor de corrección, de aquí que la capacidad de la **BAM** sea estimada en :

$$m < min(n, p)$$

Bibliografía

1.- García-Pelayo y Gross, Ramón. Diccionario Pequeño Larousse en Color, 1981.

2.- Standart Dictionary of the English Languaje. Funk & Wagnalls, 1967.

3.- Wienderhold, Gio. File Organization for Data Base Design; McGraw-Hill, 1967.

4.- Shaft, Adam; Historia y Verdad. Teoría y Praxis.

5.- Rodríguez René. 3EBAM: Un Sistema de Memorias Asociativas Bidireccionales, Tesis Maestría 1994.

6.- Villee, Claude A. Biología, 1986.

7.- Van Gigch, Jonh P. Teoría General de Sistemas, 1990.

8.- Hofstadter, Douglas R. Godel, Escher, Bach: Una Eterna Trenza Dorada; CONACYT, 1982.

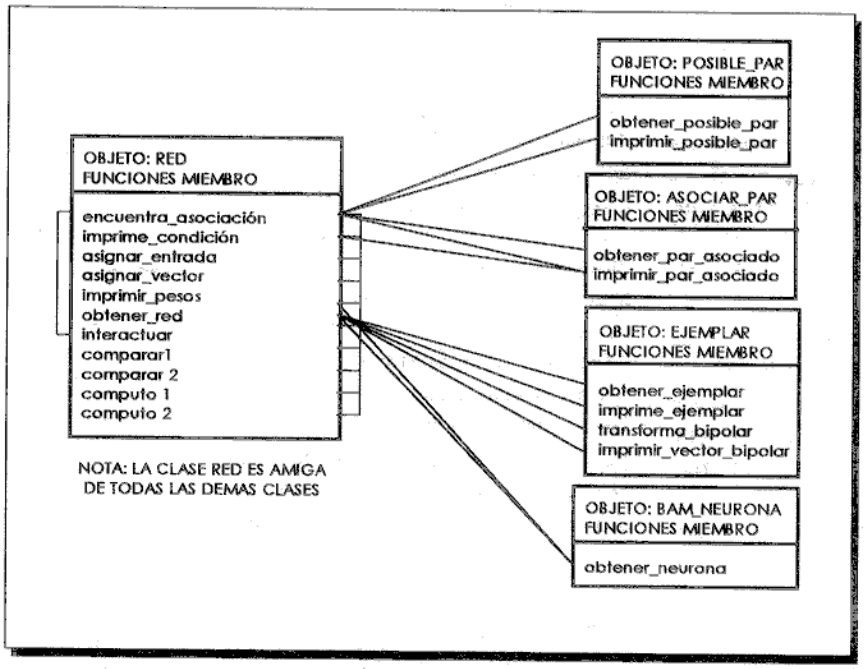
9.- Flores, Edmundo. La Creacion de la Nueva Ciencia del Caos y el Ocaso de la Metereología y de la Econometría; El BUHO, No. 253, EXCELSIOR, 1990.

10.- Freeman, James. Neural Networks, Algorithms, Applications, and Programming Techniques, Addison Wesley, 1991.

11.- Blum, Adam. Neural Networks in C++, Wiley, 1992.

12.- Kosko Bart. Neural Networks and Fuzzy Systems, Prentice Hall 1992.

13.- Freeman & Skapura. Neural Networks, Addison Wesley 1992.




```
//Programa para una BAM (Memoria asociativa
bidireccional)
//Elaborado por:

//M. en C. Miguel Angel Partida Tapia
//Alumno de la Maestría: Rubén Peredo
Valderrama
//Alumno de la Maestría: Francisco Flávio Córdova
Q.

//Fila de encabezado para el programa de la BAM
//CINTEC 1995

#include <iostream.h>
#include <math.h>
#include <stdlib.h>
#define MAX_TAMANO 10

class bam_neurona
{
//Declaración de la clase bam_neurona
//Sección protegida y pública
 protected:
   int nnbr;
   int entrada_neurona,salida_neurona;
   int salida;
   int activacion;
   int peso_salida[MAX_TAMANO];
   char *nombre;
   friend class red;
   //Declaración de clase red como amiga

public:
   bam_neurona() { }; //Constructor
   void obtener_neurona(int,int,int,char *);
};

class ejemplar
{
//Declaración de la clase ejemplar
//Sección protegida y pública

protected:
   int dimension_x,dimension_y;
   int v1[MAX_TAMANO], v2[MAX_ TAMANO];
   int u1[MAX_TAMANO], u2[MAX_TAMANO];
   friend class red;
   //Declaración de la clase red como amiga
   friend class matriz;
   //Declaración de la clase matriz como amiga

public:
   ejemplar() { }; //Constructor
   void obtener_ejemplar(int,int,int *,int *);
   void imprime_ejemplar();
   void transforma_bipolar();
   void imprimir_vector_bipolar();
};

class asociar_par
{
//Declaración de la clase asociar_par
//Sección protegida y pública

protected:
   int dimension_x,dimension_y,idn,o,p;
   int v1[MAX_TAMANO],v2[MAX_TAMANO];
   friend class red;
//Declaración de la clase red como amiga

public:
   asociar_par() { }; //Constructor
   void obtener_par_asociado(int,int,int);
   void imprimir_par_asociado();
};
class posible_par
{
//Declaración de la clase posible_par
//Sección protegida y pública

protected:
   int dimension_x,dimension_y;
   int v1[MAX_TAMANO],v2[MAX_TAMANO];
   friend class red;
   //Declaración de la clase red como amiga

public:
   posible_par() { }; //Constructor
   void obtener_posible_par(int,int);
   void imprimir_posible_par();
};

class red
{
//Declaración de la clase red
//Sección pública
public:
   int   anmbr, bnmbr, bandera, nexmplr, nasspr,
         nentra;
   bam_neurona   (anrn) [MAX_TAMANO],
                 (bnrn)[MAX_TAMANO];
   ejemplar (e)[MAX_TAMANO];
   asociar_par (as)[MAX_TAMANO];
   posible_par (pp)[MAX_TAMANO];
   int   salidas1[MAX_TAMANO],
         salidas2[MAX_TAMANO];
   int  matriz1[MAX_TAMANO][MAX_TAMANO],
         matriz2
MAX_TAMANO][MAX_TAMANO];

   red() { }; //Constructor
   void obtener_red(int,int,int,int [][6],int [][5]);
   void comparar1(int,int);
   void comparar2(int,int);
   void imprimir_pesos();
   void interactuar();
   void encuentra_asociacion(int *);
   void asignar_entrada(int *);
   void asignar_vector(int,int *,int *);
   void computo1();
   void computo2();
   void imprime_condicion();
```

```
//Programa para una BAM (Memoria asociativa
bidireccional)
//Elaborado por:

//M. en C. Miguel Angel Partida Tapia
//Alumno de la Maestría: Rubén Peredo
Valderrama
//Alumno de la Maestría: Francisco Flávio Córdova
Q.

//Fila fuente para el programa de la BAM
//CINTEC 1995

#include <c:\bc4\bin\bamwin.h>
#include <conio.h>


void bam_neurona::obtener_neurona(int m1,int
m2,int m3,char *y)
{
int i;
nombre = y;
nnbr = m1;
salida_neurona = m2;
entrada_neurona  = m3;

for(i=0;i<salida_neurona;++i){
   peso_salida[i] = 0 ;
   }

salida = 0;
activacion = 0;
}

void  ejemplar::obtener_ejemplar(int  k,int  l,int
*b1,int *b2)
{
// Función miembro ejemplar::obtener_ejemplar
// Método para obtener el vector

int i2;
dimension_x = k;
dimension_y = l;

for(i2=0;i2<dimension_x;++i2){
   v1[i2] = b1[i2]; }

for(i2=0;i2<dimension_y;++i2){
   v2[i2] = b2[i2]; }
}

void ejemplar::imprime_ejemplar()
{
// Función miembro ejemplar::imprime_ejemplar
// Método para imprimir el vector

int i;
cout<<"\nEl vector X es:"<<"                El vector
Y es:\n";

for(i=0;i<dimension_x;++i){
   cout<<v1[i]<<"  ";}
   cout<<"                ";
for(i=0;i<dimension_y;++i){
   cout<<v2[i]<<"  ";}

cout<<"\n";
}

void ejemplar::transforma_bipolar()
{
// Función miembro ejemplar::transforma_bipolar
// Método para transformar el vector a su forma
bipolar
```

```
int i;

for(i=0;i<dimension_x;++i){
   u1[i] = 2*v1[i] -1;}

for(i=0;i<dimension_y;++i){
   u2[i] = 2*v2[i] - 1;}

}

void ejemplar::imprimir_vector_bipolar()
{
// Función miembro ejemplar:: imprimir_vector_
   bipolar
// Método para imprimir el vector bipolar

int i;
cout<<"\nForma bipolar de los vectores\n";

cout<<"El vector X bipolar"<<"              El vector
Y bipolar\n";
for(i=0;i<dimension_x;++i){
   cout<<u1[i]<<"  ";}

cout<<"                ";
for(i=0;i<dimension_y;++i){
   cout<<u2[i]<<"  ";}

cout<<"\n";
}


void asociar_par::obtener_par_asociado(int i, int
j, int k)
{
// Función miembro asociar_par :: obtener_par_
   asociado
// Método para obtener el par asociado

idn = i;
dimension_x = j;
dimension_y = k;
}

void asociar_par::imprimir_par_asociado()
{
// Función miembro asociar_par::imprimir_par_
   asociado
// Método para imprimir el par asociado

int i;

cout<<"\nEl vector X par asociado # "<<idn<<"
es: \n";

for(i=0;i<dimension_x;++i){
   cout<<v1[i]<<"    ";}
   cout<<"\nPresione una tecla para continuar\n";
   getch();

cout<<"\nEl vector Y par asociado # "<<idn<<"
es: \n";

for(i=0;i<dimension_y;++i){
   cout<<v2[i]<<"    ";}
   cout<<"\nPresione una tecla para continuar";
   getch();


cout<<"\n";
}
```

```
void posible_par::obtener_posible_par(int k,int j)
{
// Función miembro posible_par:: obtener_
   posible_ par
// Método para obtener el posible par

dimension_x = k;
dimension_y = j;

}

void posible_par::imprimir_posible_par()
{
// Función miembro posible_par::imprimir_
   posible_par
// Método para imprimir el posible par

int i;
cout<<"\n";
cout<<"\nEl vector X en posible asociación par
es: \n";

for(i=0;i<dimension_x;++i){
   cout<<v1[i]<<"    ";}

cout<<"\nEl  vector Y en posible asociación par
es: \n";

for(i=0;i<dimension_y;++i){
   cout<<v2[i]<<"    ";}

cout<<"\n";
}

void red::obtener_red(int k,int l,int k1,int b1[][6],int
b2[][5])
{
// Función miembro red::obtener_red
// Método para obtener la red de trabajo

anmbr = k;
bnmbr = l;
nexmplr = k1;
nasspr = 0;
nentra = 0;
int i,j,i2,d;
bandera =0;
char *y1="ANEURON", *y2="BNEURON" ;

for(i=0;i<nexmplr;++i){

e[i].obtener_ejemplar(anmbr,bnmbr,b1[i],b2[i]);
   e[i].imprime_ejemplar();
   e[i].transforma_bipolar();
   e[i].imprimir_vector_bipolar();
   }
   cout<<"\nPulse una tecla para continuar\n";
   getch();

for(i=0;i<anmbr;++i){

anm[i].bam_neurona::obtener_neurona(i,bnmbr,0,y1);}

for(i=0;i<bnmbr;++i){

bnm[i].bam_neurona::obtener_neurona(i,0,anmbr,y2);}

for(i=0;i<anmbr;++i){

   for(j=0;j<bnmbr;++j){
      matriz1[i][j]  = 0;

      for(i2=0;i2<nexmplr;++i2){
         matriz1[i][j]  += e[i2].u1[i]*e[i2].u2[j];}
```

```
   matriz2[j][i] = matriz1[i][j];
   anrn[i].peso_salida[j] = matriz1[i][j];
   bnrn[j].peso_salida[i] = matriz2[j][i];
}


primir_pesos();
ut<<"\n";


id red::asignar_entrada(int *b)

unción miembro red::asignar_entrada
étodo para asignar entrada

 i;
ut<<"\n";

(i=0;i<anmbr;++i){
anrn[i].salida = b[i];
salidas1[i] = b[i];
}



id red::comparar1(int j,int k)

unción miembro red::comparar1
étodo de comparación 1

 i;

(i=0;i<anmbr;++i){

if(pp[j].v1[i] != pp[k].v1[i]) bandera = 1;
break;
}



id red::comparar2(int j,int k)

unción miembro red::comparar2
étodo de comparación 2

 i;

(i=0;i<anmbr;++i){

if(pp[j].v2[i] != pp[k].v2[i]) bandera = 1;
break;}



id red::computo1()

unción miembro red::computo1
étodo para el primer cálculo

 j;

(j=0;j<bnmbr;++j){
int ii1;
int c1 =0,d1;
cout<<"\n";

for(ii1=0;ii1<anmbr;++ii1){
  d1 = salidas1[ii1] * matriz1[ii1][j];
  c1 += d1;
  }

bnrn[j].activacion = c1;
cout<<"\nsalida nivel neurona "<<j<<" la acti-
ción es "<<c1<<"\n";
if(bnrn[j].activacion <0) {
```

```
    bnrn[j].salida = 0;
    salidas2[j] = 0;}

  else

    if(bnrn[j].activacion>0) {

      bnrn[j].salida = 1;
      salidas2[j] = 1;}

      else

      {cout<<"\nA 0 es obtenido usando previo
valor de salida \n";

      if(nentra<=nexmplr){

        bnrn[j].salida = e[nentra-1].v2[j];
        cout<<"\nPresione una tecla para
continuar\n";
        getch();
        cout<<"\n";
        }

      else

        { bnrn[j].salida = pp[0].v2[j];}
        salidas2[j] = bnrn[j].salida; }

    cout<<"\nsalida nivel neurona "<<j<<" la sa-
lida es "<<bnrn[j].salida<<"\n";
    cout<<"\nPresione una tecla para
continuar\n";
    getch();
    }
}


void red::computo2()
{
// Función miembro red::computo1
// Método para el segundo cálculo

int i;
cout<<"\n";

for(i=0;i<anmbr;++i){
  int ii1;
  int c1=0;

  for(ii1=0;ii1<bnmbr;++ii1){
    c1 += salidas2[ii1] * matriz2[ii1][i]; }

  anrn[i].activacion = c1;
  cout<<"\nla entrada nivel neurona "<<i<<" la
activación es "<<c1<<"\n";

  if(anrn[i].activacion <0 ){anrn[i].salida = 0;
          salidas1[i] = 0;}


  else

    if(anrn[i].activacion >0 ) {

      anrn[i].salida = 1;
      salidas1[i] = 1;
      }

    else

    { cout<<"\nA 0 es obtenida, usando valor
previo si es posible\n";
```

```
    if(nentra<=nexmplr) {anrn[i].salida = e[nentra-
1].v1[i];
          cout<<"\nPresione una tecla para
continuar\n";
          getch();
          cout<<"\n";
          }

    else {anrn[i].salida = pp[0].v1[i];}

    salidas1[i] = anrn[i].salida;}
    cout<<"\nla entrada nivel neurona "<<i<<" la
salida es "<<anrn[i].salida<<"\n";
    cout<<"\nPresione una tecla para
continuar\n";
    getch();
    cout<<"\n";
    }
}

void red::asignar_vector(int j1,int *b1,int *b2)
{
// Función miembro red::asignar_vector
// Método para la asignación del vector

int  j2;

for(j2=0;j2<j1;++j2){
  b2[j2] = b1[j2];}

}

void red::imprimir_pesos()
{
int i3,i4;
clrscr();
cout<<"\nMatriz de pesos--Nivel de entrada a
Nivel de salida: \n\n";

for(i3=0;i3<anmbr;++i3){

  for(i4=0;i4<bnmbr;++i4){
    cout<<anrn[i3].peso_salida[i4]<<"     ";}

  cout<<"\n"; }

cout<<"\n";
cout<<"\Presione una tecla para continuar\n";
getch();
clrscr();

cout<<"\nMatriz de pesos--Nivel de salida a Nivel
de entrada: \n\n";

for(i3=0;i3<bnmbr;++i3){

  for(i4=0;i4<anmbr;++i4){
    cout<<bnrn[i3].peso_salida[i4]<<"     ";}

  cout<<"\n"; }

cout<<"\n";
cout<<"\nPresione una tecla para continuar\n";
getch();
clrscr();
}

void red::interactuar()
{
int i1;

for(i1=0;i1<nexmplr;++i1){
  encuentra_asociacion(e[i1].v1);
  }
```

```
}

void red::encuentra_asociacion(int *b)
{
// Función miembro red::encuentra_asociacion
// Método para la busqueda de asociación

int j;
bandera = 0;
   asignar_entrada(b);
   nentra ++;
   cout<<"\nVector de entrada:\n" ;

   for(j=0;j<6;++j){
      cout<<b[j]<<"     ";}
   cout<<"\nPresione una tecla para continuar\n";
   getch();
   cout<<"\n";
   pp[0].obtener_posible_par(anmbr,bnmbr);
   asignar_vector(anmbr,salidas1,pp[0].v1);

   computo1();

   if(bandera>=0){
      asignar_vector(bnmbr,salidas2,pp[0].v2);

      cout<<"\n";
      pp[0].imprimir_posible_par();
      cout<<"\n";

      computo2(); }

   for(j=1;j<MAX_TAMANO;++j){
      pp[j].obtener_posible_par(anmbr,bnmbr);
      asignar_vector(anmbr,salidas1,pp[j].v1);

      computo1();

      asignar_vector(bnmbr,salidas2,pp[j].v2);

      pp[j].imprimir_posible_par();
      cout<<"\n";

      comparar1(j,j-1);
      comparar2(j,j-1);

      if(bandera == 0) {

         int j2;
         nasspr += 1;
         j2 = nasspr;


as[j2].obtener_par_asociado(j2,anmbr,bnmbr);
         asignar_vector(anmbr,pp[j].v1,as[j2].v1);
         asignar_vector(bnmbr,pp[j].v2,as[j2].v2);

         cout<<"\nPATRONES ASOCIADOS:\n";
         as[j2].imprimir_par_asociado();
         j = MAX_TAMANO ;
      }


      else

         if(bandera == 1)

            {
            bandera = 0;
            computo1();
            }

   }
```

```
}

void red::imprime_condicion()
{
// Función miembro red::imprime_condicion
// Método para la impresión de los resultados
finales
// Encontrados por la BAM

int j;
cout<<"\n";
cout<<"\nLOS PARES ASOCIADOS SIGUIEN-
TES FUERON ENCONTRADOS POR LA
BAM\n\n";

for(j=1;j<=nasspr;++j){
   as[j].imprimir_par_asociado();
   cout<<"\n";}

}


void main()
{
//Función principal

int ar = 6, br = 5, nex = 3,d;
int vector_entradas[][6] = {1, 0, 1, 0, 1, 0, 1, 1, 1,
0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
1};
int vector_salidas[][5] = {1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
1, 0, 0, 1, 0};

clrscr();
cout<<"\n\nESTE PROGRAMA ES UNA RED
DE MEMORIA ASOCIATIVA BIDIRECIONAL
(BAM).\n\n";
cout<<"\n\nLA RED ES ACTUALIZADA POR
ILUSTRACION CON:\n\n";
cout<<ar<<" NEURONAS DE ENTRADA, Y\n";
cout<<br<<" NEURONAS DE SALIDA.\n ";
cout<<"\n\n\n";
cout<<nex<<" VECTORES USADOS AL
DECODIFICAR \n ";
cout<<"\n\nPresione una tecla para continuar";
getch();
clrscr();

static red bamn;
bamn.obtener_red(ar,br,nex,vector_entradas,
vector_salidas) ;
bamn.interactuar();
bamn.encuentra_asociacion(vector_entradas[3]);
bamn.encuentra_asociacion(vector_entradas[4]);
bamn.imprime_condicion();
}
```