Redes de Neuronas Artificiales: Reglas de Aprendizaje

Antonino Santos del Riego
Servicios Informáticos de Apoyo a la Investigación
Universidade da Coruña
email: nino@udc.es

M. Virtudes Castro Villar
Laboratorio RNASA
Departamento de Computación
Universidade da Coruña
email: virtu@udc.es

Se puede ver el aprendizaje como la relación entre una entrada y una salida acompañada por cambios en algunos de los elementos involucrados. Las reglas de aprendizaje son las encargadas de modificar los pesos sinápticos de las Redes de Neuronas Artificiales (RNA en lo sucesivo) para que su respuesta sea "correcta". Entre reglas de aprendizaje más conocidas están las siguientes:

1.- Regla de Hebb

Para Hebb, el principio que regía la autoorganización neuronal era el siguiente:

"Una sinapsis aumenta en eficacia (peso sináptico) si las dos neuronas conectadas por ella tienden a estar activas o inactivas simultáneamente. En caso contrario, la fuerza de conexión (peso sináptico) se atenuará."

Siguiendo este principio, se puede utilizar el siguiente procedimiento como esquema de aprendizaje no supervisado:

$$\Delta W_{ij} = \mu * A_i(t) * O_j(t)$$

donde,

ΔW_{ij} - incremento que se ha de sumar o restar al peso sináptico.

μ - tasa de aprendizaje.

$A_i(t)$ - activación de la neurona *i* en el tiempo *t*.

$O_j(t)$ - señal que emite la neurona *j* en el tiempo *t*.

Esta regla refuerza los pesos sinápticos de las sinapsis habitualmente activas; se trata de un procedimiento de aprendizaje no supervisado.

1.1.- Ejemplo de aprendizaje utilizando la Regla de Hebb

Para este ejemplo se utilizará la Red de Neuronas Artificial de la figura 1.

Los pesos sinápticos iniciales son:

$W_{31} = 2$ $W_{32} = -1$

$W_{41} = -1$ $W_{42} = 1$

Se utilizará una función de activación del tipo umbral:

$A_i = 0$ si $W_{ij} * O_j < 1$

$A_i = 1$ si $W_{ij} * O_j >= 1$

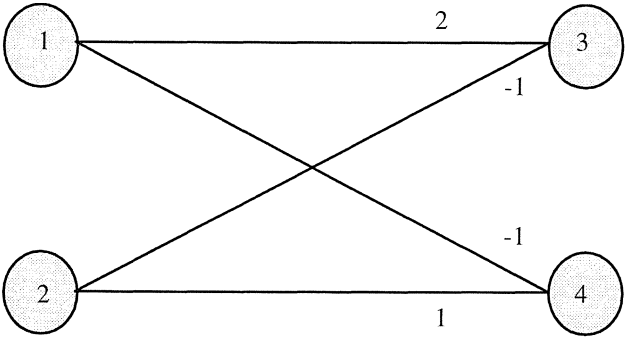

Figura 1.- RNA ejemplo Regla de Hebb.

Supóngase que en un momento inicial las respuestas de las neuronas 1 y 2 son $O_1=1$ y $O_2=0$. Por tanto, la activación de las neuronas 3 y 4 será:

$$A_3 = S_j\, O_j * W_{3j} = 1 * 2 + 0 * -1 = 2$$

Dado que 2 es mayor que el umbral, la neurona 3 se activará, por lo que $A_3 = 1$.

$$A_4 = S_j\, O_j * W_{4j} = 1 * -1 + 0 * 1 = -1$$

Como -1 no supera el umbral, la neurona 4 permanece desactivada, por tanto $A_4=0$.

Los incrementos de los pesos sinápticos serán (con $\mu = 0.1$)

$$\Delta W_{ij} = \mu * A_i(t) * O_j(t)$$

$$\Delta W_{31} = 0.1 * 1 * 1 = 0.1$$

$$\Delta W_{32} = 0.1 * 1 * 0 = 0$$

$$\Delta W_{41} = 0.1 * 0 * 1 = 0$$

$$\Delta W_{42} = 0.1 * 0 * 0 = 0$$

Por tanto, los nuevos pesos sinápticos son:

$$W_{31} = 2 + 0.1 = 2.1$$

$$W_{32} = -1$$

$$W_{41} = -1$$

$$W_{42} = 1$$

Si a continuación las respuestas de las neuronas 1 y 2 son 0 y 1, es decir $O_1=0$ y $O_2=1$. Los estados de activación de las neuronas 3 y 4 serán:

$A_3 = S_j\, O_j * W_{3j} = 0 * 2.1 + 1 * -1 = -1 < 1$ por tanto $A_3 = 0$

$A_4 = S_j\, O_j * W_{4j} = 0 * -1 + 1 * 1 = 1 >= 1$ por tanto $A_4 = 1$

Los incrementos de pesos sinápticos serán:

$$\Delta W_{31} = 0.1 * 0 * 0 = 0$$

$\Delta W_{32} = 0.2 * 0 * 1 = 0$

$\Delta W_{41} = 0.1 * 1 * 0 = 0$

$\Delta W_{42} = 0.1 * 1 * 1 = 0.1$

Por tanto, los nuevos pesos sinápticos son:

$W_{31} = 2.1$

$W_{32} = -1$

$W_{41} = -1$

$W_{42} = 1 + 0.1 = 1.1$

Sucesivos patrones de estímulos que lleguen desde las neuronas 1 y 2 irán produciendo sucesivos cambios en los pesos sinápticos que unen a estas neuronas con las neuronas 3 y 4. Se puede observar que los pesos de las sinapsis que unen a la neurona 1 con la 3 y a la 2 con la 4 serán los que irán aumentando dada la configuración de pesos inicial de la RNA de este ejemplo.

2.- Regla del Perceptrón

El perceptrón es un procedimiento de convergencia desarrollado por Frank Rosenblant, introduciendo una mejora en la Regla de Hebb, en la que las variaciones de los pesos sinápticos son proporcionales al producto de las actividades sinápticas de las neuronas emisoras y receptoras.

El Perceptrón utiliza una función de activación lineal y una función de transferencia umbral, precisándose que la suma de los impulsos que llegan a una neurona sea mayor o igual que el umbral para que se produzca el disparo de la misma.

El Perceptrón está formado por un conjunto de neuronas de entrada, las cuales se conectan al azar con una primera capa de neuronas asociativas, que se conectan a una segunda capa de neuronas de respuesta.

Así, el Perceptrón puede considerarse como una Red de Neuronas Artificial, cuyas neuronas proporcionan salidas 0 y 1, y poseen una función de activación umbral.

$F(x) = 0 \quad si \quad x <= 0$

$F(x) = 1 \quad si \quad x > 0$

Las conexiones entre las neuronas de entrada y la capa de neuronas asociativa son fijas y no se modifican durante el período de entrenamiento. En cambio, los pesos sinápticos entre las neuronas asociativas y las de respuesta se alteran según una regla específica, de la siguiente manera: Cada vez que se presenta un patrón *p*, cada una de las neuronas de respuesta, r_i, proporciona una salida O_i, (0, 1) que puede ser correcta o no. De ello depende la forma como se alteran los pesos sinápticos, W_{ij}, de las conexiones entre cada unidad asociativa, a_j, con las neuronas de respuesta, r_i.

Si la salida O_i es correcta, los pesos no se alteran, pero si no lo es, entonces cada W_{ij} se incrementa en una cantidad $\mu(P_i - O_i)$, para cada neurona asociadora a_j activa, es decir, cuya salida sea 1, siendo P_i la salida deseada de la i-ésima neurona de respuesta, y μ la tasa de aprendizaje (un parámetro fijo, positivo). Simultáneamente, el umbral se decrementa en la misma cantidad.

Entonces, para cada patrón *p* presentado a las neuronas de entrada, se tiene que:

$$\Delta W_{ij} = \mu (P_{ip} - O_{ip}) O'_{jp}$$

donde,

P_{ip} - salida deseada de la neurona de respuesta *i* ante el patrón *p*.

O_{ip} - salida de la neurona de respuesta *i* ante el patrón *p*.

O'_{jp} - salida de la neurona asociadora *j* ante el patrón *p*.

y el nuevo peso sináptico será el anterior más el incremento, ΔW_{ij}, es decir:

$$W_{ij} \text{ (nuevo)} = W_{ij} \text{ (anterior)} + \Delta W_{ij}$$

Por ser el Perceptrón simple un modelo de dos capas, su capacidad de aprendizaje se ve seriamente limitada, no siendo capaz, por ejemplo, de implementar la función EXOR.

El Perceptrón de un único nivel, que fue el inicialmente propuesto por Rosenblatt puede usarse para clasificar sus posibles patrones de entrada en dos grupos: aquellos que permiten que la neurona supere su umbral y por tanto se active y aquellos que no logran activarla.

Pero esos dos grupos de patrones de entrada han de ser linealmente separables, es decir, si se realiza una representación gráfica, las respuestas de la RNA de valor 1 se deben poder separar, mediante una línea recta, de las respuestas de valor 0.

2.1.- Algoritmo de entrenamiento del Perceptrón (Pseudocódigo)

Se puede utilizar el siguiente pseudocódigo para entrenar el Perceptrón:

```
Para cada patrón en el conjunto de entrenamiento
    (
    Aplicar el siguiente patrón al Perceptrón
    Grabar la respuesta del Perceptrón
    Si la respuesta del Perceptrón es correcta
        (
        y la respuesta fue +1, entonces
            el nuevo vector de pesos = vector de pesos antiguo +
                                       vector de patrones de entrada
        y la respuesta fue -1, entonces
            el nuevo vector de pesos = vector de pesos antiguo -
                                       vector de patrones de entrada
        )
    Si la respuesta del Perceptrón es incorrecta
        (
        y la respuesta fue +1, entonces
            el nuevo vector de pesos = vector de pesos antiguo -
                                       vector de patrones de entrada
        y la respuesta fue -1, entonces
            el nuevo vector de pesos = vector de pesos antiguo +
                                       vector de patrones de entrada
        )
    )      /* fin para cada patrón en el conjunto de entrenamiento */
```

En otras palabras:

```
Para cada patrón
{Aplicar patrón
 Si (O_i(t) - T_i(t)) = 0
    {Si O_i = +1 → W_ij = W_ij + O_j
     Si O_i = -1 → W_ij = W_ij - O_j
    }
 Si (O_i(t) - T_i(t)) ≠ 0
    {Si O_i = +1 → W_ij = W_ij - O_j
     Si O_i = -1 → W_ij = W_ij + O_j
    }
}
```

2.2.- Ejemplo de Perceptrón

Se tienen dos puntos con salida +1 (A) y otros dos con -1 (B). Hay que distinguir entre los puntos A y B.

El vector inicial de pesos es W_0 (-0.6,0.8). El umbral es 0.

Los pasos para resolver el problema son:

1) Aplicar A_1 al Perceptrón y computar el nuevo *I*. Esto es, tomar la suma ponderada de los elementos del patrón de entrada. Cada elemento del patrón se multiplica por el correspondiente peso del Perceptrón y el producto resultante se añade a *I*. El patrón de entrada A_1 tiene $x_1 = 0.3$ y $x_2 = 0.7$.

Entonces la computación para *I* es:

$$I = w_1 x_1 + w_2 x_2$$

$$I = (-0.6) * (0.3) + (0.8) * (0.7).$$

$$I = -0.18 + 0.56 = 0.38$$

Como I>0 la salida es +1. La salida deseada para el patrón es +1; entonces, el nuevo vector de pesos es el viejo vector de pesos más el vector de entrada $W_1 = W_0 + A_1$. Esta operación consisten en la suma de sus respectivos componentes.

Los pesos sinápticos serán:

$$W_{1x} = -0.6 + 0.3 = -0.3$$

$$W_{1y} = 0.8 + 0.7 = 1.5$$

2) Aplicar B_1 al Perceptrón y computar la nueva entrada *I*:

$$I = (-0.3) * (-0.6) + (1.5) * (0.3)$$

$$I = 0.18 + 0.45 = 0.63$$

Como I>0 la salida es 1, debiendo ser -1. Por lo tanto, B_1 debe ser restado del vector de pesos actual para conseguir W_2.

$$W_2 = W_1 - B_1$$

$$W_{2x} = (-0.3) - (-0.6) = 0.3$$

$W_{2y} = (1.5) - (0.3) = 1.2$

3) Aplicar A_2.

$I = (0.3) * (0.7) + (1.2) * (0.3) = (0.21) + (0.36) = (0.57)$.

Como I>0, la salida es +1 y, por tanto, se deben añadir los pesos del vector de entrada al antiguo vector de pesos para conseguir el nuevo.

$W_3 = W_2 + A_2$

$W_{3x} = (0.3) + (0.7) = 1$

$W_{3y} = (1.2) + (0.3) = 1.5$

Aplicar B_2.

$I = (1) * (-0.2) + (1.5) * (-0.8) = (-0.2) + (-1.2) = -1.4$

Como I<0 entonces la salida es - 1 y correcta, por tanto, se deben restar los pesos del vector de entrada al antiguo vector de pesos para conseguir el nuevo.

$W_4 = W_3 - B_2$

$W_{4x} = (1) - (-0.2) = (1.2)$

$W_{4y} = (1.5) - (0.8) = (2.3)$

El vector final de pesos después del entrenamiento es W_4 (1.2, 2.3)

4) Utilizar este vector de pesos para comprobar que el Perceptrón clasifica correctamente cada uno de los patrones de entrenamiento. Esto se realiza computando la salida del Perceptrón para cada patrón y comprobando si generan la respuesta correcta (+1, -1).

5) Seleccionar más patrones A y B demostrando que el Perceptrón los clasifica correctamente.

3.- Regla "Backpropagation" o Retropropagación del Error

El algoritmo de backpropagation es un método sistemático para el entrenamiento supervisado de Redes de Neuronas Artificiales. Este

algoritmo ha sido propuesto por Rumelhart (1983) y modelizado matemáticamente por Le Cun (1985) y Parker (1985).

El descubrimiento del algoritmo backpropagation ha jugado un importante papel en el resurgir de las RNA (abandonadas desde años atrás ante la inexistencia de algoritmos de entrenamiento de RNA multinivel y la escasa aplicabilidad de las RNA mononivel), extendiéndose el rango de problemas a los cuales pueden ser aplicadas.

La figura 2 representa un modelo de RNA Backpropagation con tres capas de neuronas: una de entrada, la oculta y la de salida, relacionadas por medio de dos matrices de pesos.

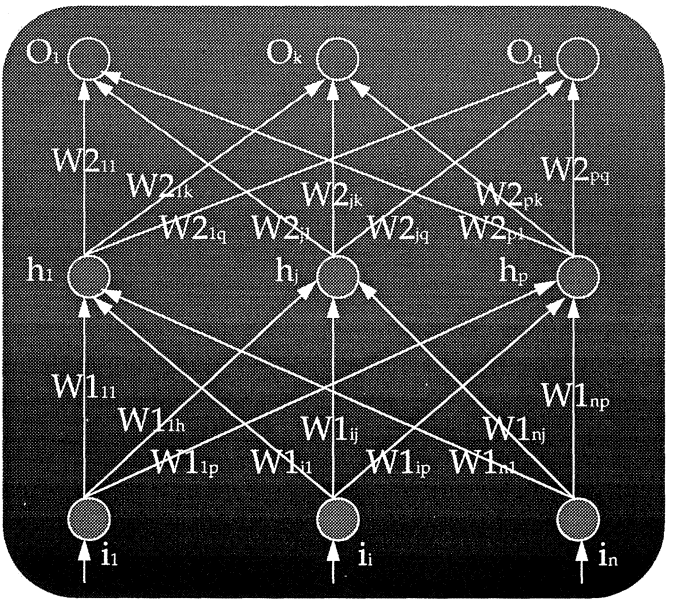

Figura 2.- Modelo de RNA Backpropagation.

Los parámetros que intervienen en el algoritmo son:

- La tasa de aprendizaje μ (suele tomar valores entre 0.01 y 1), que indica cómo han de cambiar los pesos sinápticos durante el proceso de aprendizaje.

- El momento, Θ, como se verá posteriormente, indica cómo influye el cambio de los pesos sinápticos previo sobre la modificación de pesos actual.
- Los valores iniciales de los pesos influyen en la convergencia del algoritmo y han de tomarse aleatoriamente (valores grandes pueden producir la saturación o parálisis de la RNA).
- El número de neuronas ocultas influyen en relación directa en la eficacia de aprendizaje y de generalización del algoritmo (un número pequeño precisa de más casos de entrenamiento y un número grande incrementa los requerimientos de potencia y tiempo de la máquina; además, un número adecuado de neuronas internas evita los problemas de mínimos locales).

La RNA sobre la que se aplicará el algoritmo estará formada por la interconexión de neuronas del tipo representado en la figura 3. Cada neurona estará formada por una serie de entradas *xi*. Cada conexión que va de la neurona *i* a la neurona *j* estará ponderada con el peso *wji*. La función de activación de la neurona será la suma de los productos de cada entrada a esa neurona por su peso correspondiente.

Normalmente se empleará una función sigmoide (continua, monótona, no decreciente y derivable).

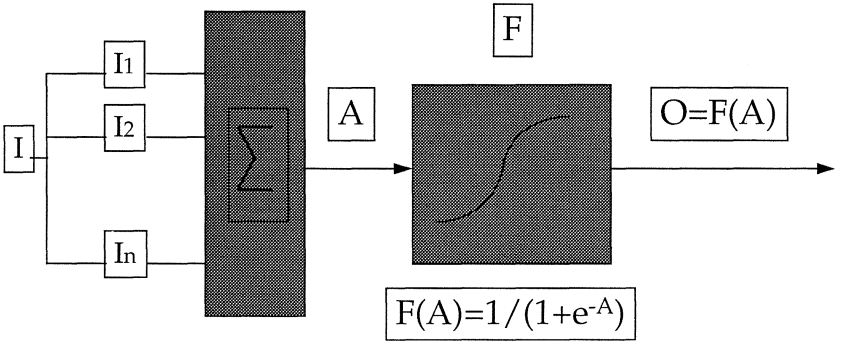

Figura 3.- Modelo de Elemento de Proceso.

La RNA cumplirá los siguientes requisitos:

- Ser multicapa.
- Ser "feed-forward" (de alimentación hacia adelante) de modo que cada neurona recibe entradas de la capa inmediatamente anterior.
- Ser completamente conectada entre capas (cada neurona recibe una entrada de cada neurona de la capa anterior y emite una salida a cada neurona de la capa posterior).

La RNA tomará un vector n-dimensional como entrada y emitirá un vector m-dimensional como salida.

Se tendrá además un juego de ensayo formado por un conjunto de hechos (pares formados por un vector de entrada -*n*- y un vector de salida objetivo -*m*-, que es la salida deseada para esa entrada en la RNA). Para un patrón de entrada *i* se define la salida producida por la RNA como "o_s" y la salida esperada por "d_s".

3.1.- Entrenamiento de la RNA

Para el entrenamiento de una RNA backpropagation se deben seguir los siguientes pasos:

0) Asignar valores aleatorios (entre -1 y +1) a los pesos entre la capa de entrada-oculta y oculta-salida, y a los umbrales de las neuronas de la capa oculta y de salida.

Pasos progresivos:

1) Computar la activación de las neuronas de la capa oculta.

$O_h = F(i * W_1)$

O_h - vector de las neuronas de la capa oculta.

i - vector de neuronas de la capa de entrada.

W_1 - matriz de pesos entre las capas de entrada y salida.

F() - función de activación (sigmoidal).

$$F(x) = \frac{1}{1+e^{-x}}$$

2) Computar las activaciones de las neuronas de la capa de salida.

$O_s = F(O_h * W_2)$

O_s - capa de salida.

O_h - capa oculta.

W2 - matriz de pesos que conecta la capa oculta con la de salida.

Pasos regresivos:

3) Computar el error de la capa de salida (diferencia entre la salida deseada y la obtenida).

$e_s = O_s * (1-O_s) * (d_s-O_s)$

e_s - vector dé errores en la capa de salida.

O_s - vector de salida obtenido.

d_s - activación deseada en la capa de salida.

4) Computar el error de la capa oculta:

$e_h = O_h * (1-O_h) * W_2 * e_s^T$

e_h - vector de errores de las neuronas de la capa oculta.

O_h - vector de salida de la capa oculta.

e_s^T - trasnpuesto del vector e_s.

5) Ajustar los pesos entre la capa oculta y de salida.

$W2_t = W2_{t-1} + \Delta W2_t$

$W2_t$ - matriz de pesos ajustados en el ciclo *t*.

$W2_{t-1}$ - matriz de pesos en el ciclo *t-1*.

$\Delta W2_t$ - matriz que representa los incrementos de los pesos en el ciclo *t*.

$\Delta W2_t = \mu * O_h * e_s + \Theta * \Delta W2_{t-1}$

μ - coeficiente de aprendizaje.

Θ - factor momento que permite que las modificaciones al, *t*.

$\Delta W2_{t-1}$ - incremento de pesos realizado sobre la matriz W2 en el ciclo anterior *t-1*.

6) Ajustar los pesos entre la capa de entrada y la oculta.

$W1_t = W1_{t-1} + \Delta W_{1t}$

W_{1t} - matriz de pesos ajustados en el ciclo *t*.

$W1_{t-1}$ - matriz de pesos en el ciclo *t-1*.

ΔW_{1t} - matriz que representa los incrementos de los pesos en el ciclo *t*.

$\Delta W1_t = \mu * i * e_h + \Theta * \Delta W1_{t-1}$

$\Delta W1_{t-1}$ - incremento de pesos realizado sobre la matriz W1 en el ciclo anterior *t-1*.

Repetir los pasos desde 1 a 6 para todos los patrones de entrenamiento hasta que el error de la capa de salida (vector e_s) tenga la tolerancia deseada para cada patrón y para cada neurona. Es el límite de convergencia.

3.2.- Funcionamiento de la RNA

La fase de funcionamiento presenta los siguientes pasos:

1) Computar la activación de la capa oculta.

$O_h = F(W1 * i)$

2) Computar la salida de la capa de salida.

$O_s = F(W2 * O_h)$

O_s - vector resultado.

3.3.- Modificaciones al algoritmo

Se pueden utilizar 'umbrales' en las neuronas de la capa oculta y de salida, quedando de la siguiente manera:

$O_h = F(i * W1 + umbral1)$

$O_s = F(W2 * O_h + umbral2)$

Durante el entrenamiento, los vectores umbral se ajustan de la siguiente forma:

$$umbral1 = umbral1 + \mu * e_s$$

Además, el coeficiente de aprendizaje μ puede irse decrementando a lo largo del proceso de entrenamiento para evitar saltar de un lado a otro de los niveles de convergencia y prologar el proceso de entrenamiento.

Típicamente, el momento Θ está comprendido entre 0 y 1.

Los pesos pueden modificarse después de presentar un conjunto de vectores de entrada.

El tamaño de la capa de entrada y de salida está en función de la aplicación y, el de la capa oculta, suele ser función del tamaño de entrada y de salida y de la complejidad del problema a resolver.

4.- Memorias asociativas bidireccionales (BAM)

Las Memorias Asociativas Bidireccionales (en lo sucesivo BAM) están orientadas al reconocimiento de patrones en tiempo real (reconocimiento de voz, clasificadores, etc.). Las BAM son Redes de Neuronas Artificiales construidas, no entrenadas.

4.1.- Topología

Bart Kosho desarrolló las RNA "BAM" basándose en las memorias direccionables por el contenido. Las BAM se utilizan para almacenar *´m´* pares de patrones (A_i, B_i), con $A_i \in \{-1, +1\}n$ y $B_i \in \{-1, +1\}p$ en una matriz M de $n*p$ elementos. Las BAM presentan algunas limitaciones; para un *´n´* muy grande, los requerimientos de almacenamiento se incrementan considerablemente. Su capacidad de almacenamiento viene dada por $m<\min(n,p)$. La figura 4 representa la topología de una BAM.

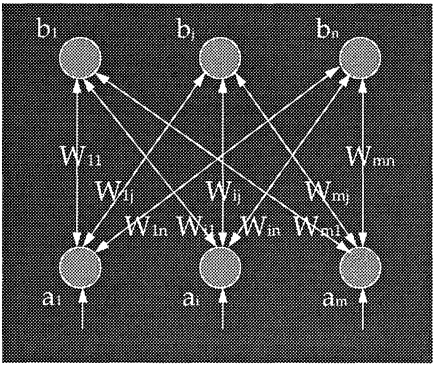

Figura 4.- Topología de una BAM.

4.2.- Construcción

La construcción de una BAM consiste en la suma de las matrices de correlación de cada uno de los pares de patrones. Esto es; para codificar los *´m´* pares de patrones, la matriz M se calculará como:

$$M = \sum_{i=1}^{m} A_i^T * B_i$$

Para realizar los cálculos de M, los valores utilizados serán {-1, +1}.

4.3.- Ejemplo de BAM

Si se quiere almacenar los siguiente pares de patrones:

$A_1 = (1\ 0\ 1\ 0\ 1\ 0)$ $B_1 = (1\ 1\ 0\ 0)$

$A_2 = (1\ 1\ 1\ 0\ 0\ 0)$ $B_2 = (1\ 0\ 1\ 0)$

Se seguirán los siguientes pasos:

1) Convertir los ceros en unos.

$X_1 = (1\ -1\ 1\ -1\ 1\ -1)$ $Y_1 = (1\ 1\ -1\ -1)$

$X_2 = (1\ 1\ 1\ -1\ -1\ -1)$ $Y_2 = (1\ -1\ 1\ -1)$

2) Calcular la matriz M.

$$X1^T * Y1 = \begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix}$$

$$X2^T * Y2 = \begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} 2 & 0 & 0 & -2 \\ 0 & -2 & 2 & 0 \\ 2 & 0 & 0 & -2 \\ -2 & 0 & 0 & 2 \\ 0 & 2 & -2 & 0 \\ -2 & 0 & 0 & 2 \end{bmatrix}$$

Se puede eliminar un par (A_i, B_i) haciendo M-X_i^T*Y_i.

4.4.- Funcionamiento

Se desea obtener B_i cada vez que se presenta A_i a la RNA y obtener A_i cada vez que se presente B_i.

Cada neurona B_j (Fb) recibe una entrada de todas las neuronas en "Fa" con una función no lineal del tipo umbral. Una función típica puede ser.

$f(x,y) = 1$ si $x>0$

$f(x,y) = 0$ si $x<=0$

Ahora, se tiene un patrón B1. La salida del patrón B se alimenta hacia atrás con la matriz M traspuesta, para producir el patrón A1. Cada neurona A_i en A recibe entradas de cada neurona B_j en B y aplica la misma función umbral. A1 producirá entonces B2.

$F(A_i*M) = B_i$

$F(B_i*M^T) = A_i$

4.5.- Codificación

La codificación de una BAM implica los siguientes pasos:

1) Realizar la matriz M1 (suma de las matrices de correlación de cada uno de los pares de patrones {-1, +1}).

2) Si almacena todos los patrones (proceso de test), terminar el proceso.

3) En caso contrario, borrar los pares de patrones no almacenados. Estos patrones se almacenan en otra matriz M2. Se repite el proceso hasta conseguir almacenar todos los pares de patrones.

4.6.- Funcionamiento

Se siguen los siguientes pasos:

1) Presentar un patrón A a la BAM, utilizando cada una de las matrices M_i. Se obtienen "i" patrones respuesta.

2) Calcular la energía de cada par (A,Yi).

$$E = -A * M_i * Y_i^T$$

3) Tomar el Y_i con energía más próxima a la de la Matriz M que almacena los patrones ortogonales.

$$E = -n * m$$

5.- Redes de Neuronas Artificiales Hopfield

Una RNA Hopfield está formada por una única capa con interconexión total entre todos sus elementos de proceso (figura 5). Se utilizan para el reconocimiento de aquellos patrones que estén almacenados en sus pesos sinápticos. Las Hopfield son redes retroalimentadas que proporcionan respuestas dinámicas, progresando hasta llegar a un estado estable o solución. El número de patrones que se pueden almacenar viene dado por la siguiente expresión:

$$N^{\circ}\ Patrones = \frac{N}{4 * \log(N)}$$

N - Número de Elementos de Proceso.

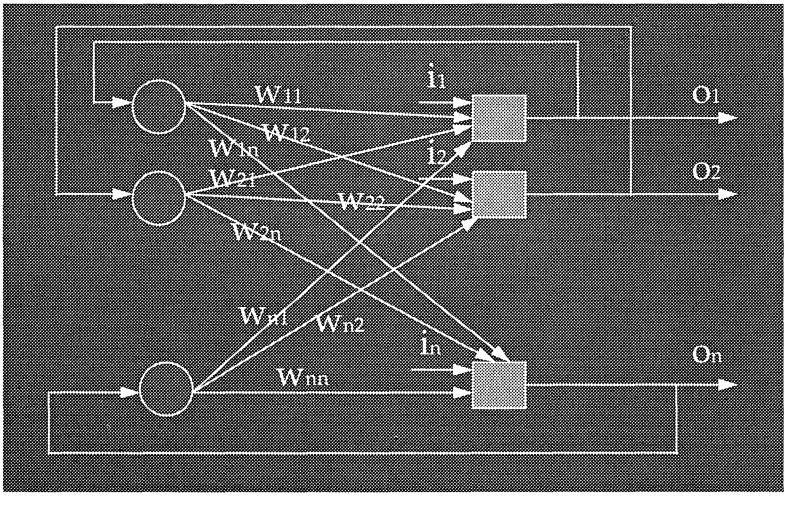

Figura 5.- Topología de una RNA Hopfield.

5.1.- Teorema de Estabilidad

Una RNA Recurrente Hopfield es estable si la matriz que representa sus pesos es simétrica con ceros en la diagonal principal. ($w_{ij}=w_{ji}$ si i<>j , $w_{ii}=0$). Esto es condición suficiente pero no necesaria.

5.2.- Matriz de Pesos para Clasificación de Patrones

La forma de asignar los pesos a las conexiones de una RNA Hopfield se define de la siguiente forma:

$$w_{ij} = \begin{cases} \sum_{0}^{M-1} X_i^S \cdot X_j^S & (i \neq j) \\ 0 & (i = j) \end{cases}$$

$$x_i^S = \begin{cases} -1 \\ +1 \end{cases}$$

x_i^S - Elemento i del patrón S.

M - Número de patrones.

5.3.- Funcionamiento de la RNA Hopfield para Clasificación

Las RNA Hopfield utilizan una función de transferencia tipo umbral retropropagando su salida dinámicamente hasta obtener un estado estable:

$$A_i(t) = \sum_j O_j(t-1) \cdot w_{ij}$$

$$O_i(t) = 1 \quad si \quad A_i(t) > Umbral$$

$$O_i(t) = 0 \quad si \quad A_i(t) < Umbral$$

$$O_i(t) = O_i(t-1) \quad si \quad A_i(t) = Umbral$$

6.- Dominios de aplicación de las RNA

En la actualidad, las RNA se están aplicando en tareas de:

- ◆PREDICCIÓN.
  - ◆dbd: "Delta Bar Delta"
  - ◆drs: "Directed Random Search"
  - ◆edbd: "Extendel Delta Bar Delta"
  - ◆bkpfuncs, bkprcir: "Backpropagation"
  - ◆dnna: "Digital Neural Network"
  - ◆som: "Self-Organizing Map"
  - ◆adaline: "Adaline (Adaptive Linear Network)"
  - ◆madaline: "Multiple Adaline Network"
- ◆CLASIFICACIÓN.
  - ◆catlrnn: "Categorical Learning"
  - ◆cntrprop: "Counterpropagation"
  - ◆lvq: "Learning Vector Quantization"
  - ◆pnn: "Probabilistic Neural Network"
  - ◆som_cat: "Self-Organizing-Map into Categorization"
- ◆ASOCIACIÓN.
  - ◆bam: "Bidirectional Associative Memory"
  - ◆boltzcmp, bolzio: "Boltzmann Pattern"
  - ◆hamming, hamlit: "Hamming Network"
  - ◆hopfield: "Hopfield Network"
  - ◆spr: "Spatio-Temporal Pattern Recognition"
  - ◆art: "Adaptive Resonance Theory"
  - ◆som: "Self-Organizain Map"
- ◆FILTRADO.
  - ◆recirc: "Recirculation"

Bibliografía

- Blum A., (1992), "Neural Works in C++: An Object-Oriented Framework for Building Connectionist Systems", Wiley Professional Computing.
- Caudill M. & Butler C., (1992), "Understanding Neural Networks. Computer Explorations. Volume I: Basic Networks", MIT Press.
- Caudill M. & Butler C., (1992), "Understanding Neural Networks. Computer Explorations. Volume II: Advanced Networks", MIT Press.
- NeuralWare, (1991), "System Guide", NeuralWare INC.
- NeuralWare, (1991), "Using Nworks: NeuralWorks Professional II/Plus and NeuralWorks Explorer", NeuralWare INC.
- NeuralWare, (1991), "Reference Guide: NeuralWorks Professional II/Plus and NeuralWorks Explorer", NeuralWare INC.
- NeuralWare, (1991), "Neural Computing: NeuralWorks Professional II/Plus and NeuralWorks Explorer", NeuralWare INC.
- Ríos J., Pazos A., Brisaboa N. & Serafín C., (1991), "Estructura, Dinámica y Aplicaciones de las Redes de Neuronas Artificiales". Centro de Estudios Ramón Areces, S.A.