Predicción de Series Temporales utilizando Redes de Neuronas Artificiales

Juan Ríos Carrión
Dep. Computación e IA. Facultad de Informática U.P.M
Campus de Montegancedo s/n ,Boadilla del Monte (Madrid)
email: jrios@fi.upm.es

1.- Introducción

Predecir no es una tarea fácil en ningún campo científico. Los Físicos o los Matemáticos nos hablan de que el crecimiento de errores, o caos, impide una predicción con certeza de un sistema dinámico. En el campo de la Meteorología y Economía se asumen los riesgos que conllevan sus predicciones. En el caso de Medicina, la predicción se limita a establecer diagnósticos previos y, a partir de ahí, realizar el tratamiento adecuado para la salud del paciente.

La predicción debe entenderse como un intento permanente de anticipación de un futuro incierto y sobre el que, además, podemos incidir en algunos casos.

La predicción no es un fin en si misma, sino que forma parte de un proceso complejo de toma de decisiones, es por ello aconsejable exponer las técnicas de predicción en el contexto de las situaciones reales en que se aplican, es decir, hay que considerar el entorno. La predicción es una tarea compleja que exige, en ocasiones, la utilización de estadísticas muy complicadas. Si clasificamos las técnicas de predicción según el tipo de información que utilizan, ésta puede ser:

- Información Subjetiva
- Información Histórica
- Información Relacional o Causal

En la primera, la predicción toma como base la propia opinión del experto sobre el futuro de la cuestión en estudio, la segunda utiliza la propia evolución del fenómeno objeto del estudio en periodos anteriores, siendo la característica clave de este enfoque el estudio de un fenómeno en si mismo, a través de su evolución temporal (series temporales) y la tercera toma como base, las reglas internas de funcionamiento del tema cuyo comportamiento se trata de predecir.

Nosotros vamos a utilizar la predicción mediante Redes de Neuronas Artificiales en base al análisis aislado de series, ya que por las características intrínsecas de las redes, es el enfoque más idóneo.

2.- Modelo de Red

El modelo de red que se utilizará será del tipo multicapa alimentado hacia delante y con células analógicas con funciones de activación sigmoidal. Cada capa recibirá señales de la capa inmediatamente anterior y las enviará a la inmediatamente siguiente (Figura 1).

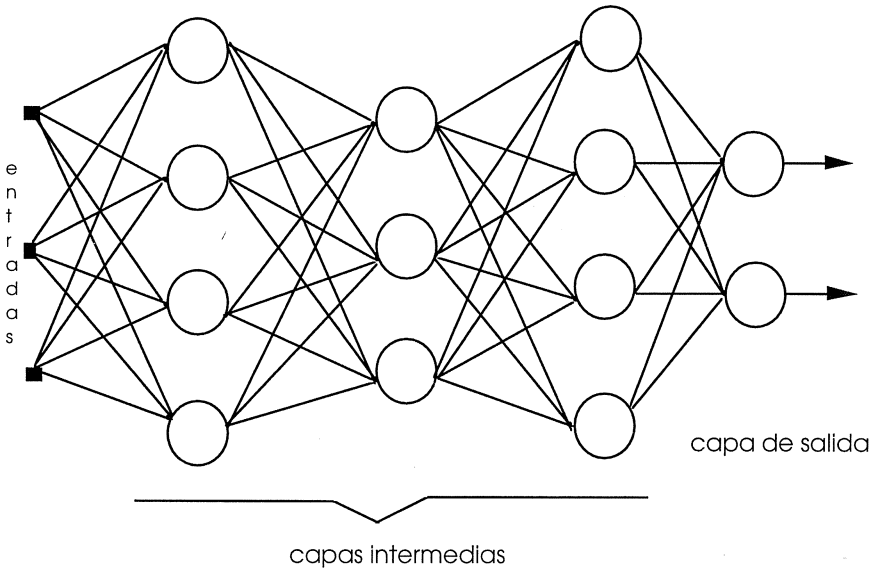

Figura 1.- Modelo de RNA.

Las C capas de células serán distinguidas por un índice C = 1, 2,..., C. El valor c=0 se reservará para designar el conjunto de entradas de la red, sin tomarlas como una capa de la RNA. Dentro de cada capa tenemos n_c células reales , que serán designadas mediante un índice que tomará valores de 1 a n_c.

Usaremos, además, las siguientes notaciones:

- $S_{ip}^{(c)}$: Salida de la i-ésima célula de la capa c-ésima cuando se presenta el patrón de entrada p. Para c=0

representa la i-ésima componente del patrón de entrada p.

- $O_{ip}^{(c)}$: Es otra notación para $S_{ip}^{(c)}$.
- $net_{ip}^{(c)}$: Entrada neta de la i-ésima célula de la capa c .
- $w_{ij}^{(c)}$: Peso de la sinapsis entre la célula j-ésima de la capa c-1 y la i-ésima de la capa c. Sólo está definido para $1 \leq c \leq C$ y $1 \leq i \leq n_c$.
- t_{ip}: Salida deseada de la i-ésima célula de la capa de salida al presentarse el patrón p de entrada.

El índice p representa el patrón de entrada, y en el caso de que el conjunto de posibles patrones sea discreto, p tomará valores enteros entre 1 y P, donde P representa el número total de posibles patrones. En tal caso, los elementos arriba definidos pueden interpretarse como elementos de sendas matrices. En otros casos el conjunto de patrones será continuo, y p será entonces un parámetro que varía continuamente.

- Modelo analógico con función de activación sigmoidal ordinaria:

La función de activación será la sigmoide

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

y el conjunto de valores de activación será el intervalo abierto (O,1).

- Modelo analógico con función de activación sigmoidal simétrica:

Su función de activación es la tangente hiperbólica

$$th^{(x)} = (e^{x} - e^{-x})/ (e^{x} + e^{-x})$$

y su conjunto de valores de activación es el intervalo abierto (-1, +1).

Una vez definido el modelo, su modo de funcionamiento se definirá así: por cada patrón "p" de entrada que se presente, los

siguientes cómputos son realizados por las células de cada capa c de la red:

$$net_{ip}(c) = \sum_{j=1}^{nc-1} w_{ij}\ s_{jp}(c-1)$$

$$s_{ip}(c) = F\ (net_{ip}(c))$$

Los citados cómputos se inician en la primera capa de células, y se propagan hacia delante a través de las sucesivas capas c = 1, 2, ..., C.

3.- Método iterativo para el ajuste de la Red

El ajuste de una red de neuronas artificial consiste en la determinación de los valores de sus pesos sinápticos que mejor aproximan las salidas reales de la red a las salidas deseadas para cada patrón de entrada. El modo más habitual de realizar dicha determinación consiste en someter la red a un entrenamiento, durante el cual se le presentan los distintos patrones de entrada, se comparan las salidas proporcionadas por la red con las deseadas, y se modifican los pesos sinápticos de acuerdo con algún algoritmo que asegure una progresiva aproximación entre las respuestas de la red y las que se desean.

Desde un punto de vista matemático, el proceso de entrenamiento se puede asimilar a un problema de optimización. En primer lugar, hay que definir una función "distancia" entre las salidas reales de la red y las deseadas. Una manera de hacer esto, consiste en considerar las salidas como componentes de un vector, y usar la distancia euclidea. Si la red tiene n células de salida, y para el patrón p genera las salidas $\underline{s}_p = (s_{1p}, s_{2p},...,s_{np})$, siendo $\underline{t}_p = (t_{1p}, t_{2p}, ...,t_{np})$ las salidas deseadas, entonces la siguiente función (llamada "función error") sirve como medida de la diferencia entre unas y otras:

$$E_p = \frac{1}{2} \sum_{i=1}^{n} (t_{ip} - s_{ip})^2$$

De hecho la distancia euclídea entre $\underline{s}_p$ y $\underline{t}_p$ es $\sqrt{2E_p}$

El problema ahora puede plantearse como minimización de E_p, o más bien minimización del promedio de E_p sobre todos los patrones p.

Las salidas obtenidas s_{ip} dependen de los pesos sinápticos de la red, y por tanto E_p puede considerarse como una función de dichos pesos. El problema de minimizar una función de varias variables es bien conocido, y se dispone de muy diversos métodos para resolverlo .

Los métodos de optimización admiten una primera división en deterministas y aleatorios. Por otro lado, el método de optimización se dice de orden p cuando el algoritmo de optimización posee determinadas propiedades de convergencia que se verán a continuación. En una primera aproximación, podemos considerar de orden p aquellos algoritmos en los que intervienen las derivadas parciales de la función objetivo hasta el orden p, aunque este concepto requiere algunas precisiones.

Los métodos iterativos deterministas para minimizar una función f: $R^n \rightarrow R$ se basan fundamentalmente en algoritmos de la siguiente forma:

1. Elegir un punto $x_0 \in R$
2. Dado $\underline{x}_k$, seleccionar una dirección de búsqueda $\underline{d}_k$.
3. Seleccionar un número real positivo adecuado a_k
   (generalmente para minimizar $f(\underline{x}_k + a_k \underline{d}_k)$).
4. Calcular $\underline{x}_{k+1} = \underline{x}_k + a_k \underline{d}_k$.
5. Sustituir $\underline{x}_k$ por $\underline{x}_{k+1}$ y regresar al paso 2.

Los métodos deterministas de primer orden para la optimización de funciones hacen uso de las derivadas parciales de la función objetivo hasta el primer orden para la determinación de la dirección de búsqueda. Uno de los más conocidos es el denominado "método de descenso de gradiente" (o de "ascenso" si se trata de un problema de maximización). Se basa en el uso del gradiente (cambiado de signo) de la función objetivo como dirección de búsqueda, y su fundamento es el siguiente:

Sea $f(\underline{x})$ la función a minimizar, donde $\underline{x} = (x_1, x_2, ..., x_n)$. Su desarrollo en serie de Taylor hasta el primer orden es el siguiente:

$$f(\underline{x}+\underline{h}) = \underline{f}(x) + \nabla f(\underline{x}) \cdot h = f(x_1,x_2,...,x_n) + \sum_{i=1}^{n} h_i \frac{\partial}{\partial x_i} f(x_1,x_2,...,x_n)$$

donde $h = (h_1,h_2,...,h_n)$ y $\nabla f(\underline{x})$ = gradiente de $f(\underline{x})$, es decir, un vector cuyas componentes son:

$$\frac{\partial}{\partial x_i} \quad f \quad (x_1, x_2, ..., x_n)$$

Si tomamos $h = -h\nabla f(x)$, donde h es una constante positiva lo bastante pequeña como para justificar la aproximación utilizada, obtenemos

$$f(\underline{x} - h\nabla f(\underline{x})) = f(x) - h(\nabla f(\underline{x}))^2$$

lo que nos lleva a que el valor de la función $f(x)$ tiende a descender al pasar de

$$\underline{x} \text{ a } \underline{x} - h \nabla f(\underline{x})$$

este resultado sirve como fundamento del siguiente algoritmo: tómese un punto arbitrario de partida $\underline{x}_0$ y aplíquese reiteradamente la siguiente fórmula:

$$Æ \underline{x}_k = - h \nabla f(\underline{x}_k)$$

$$\underline{x}_{k+1} = \underline{x}_k + \Delta \underline{x}_k$$

donde h es un pequeño parámetro positivo fijo. Esto proporciona una sucesión $\{x_k\}$ de puntos para los que es de esperar que los valores de la función $f(x_k)$ vayan descendiendo progresivamente.

Este algoritmo tiene varios inconvenientes:

1. No hay criterio "a priori" para la elección del punto de partida $\underline{x}_0$ y del parámetro h. Sin embargo, la elección de dichos parámetros puede ser fundamental para el buen éxito del método. Por ejemplo, un valor excesivamente elevado de h puede conducir a oscilaciones

indeseadas, pero un valor demasiado pequeño haría muy lento al algoritmo.

2. Puede conducir a un punto de mínimo relativo (donde $\nabla f(\underline{x})=\underline{0}$) que no sea mínimo absoluto.

A pesar de sus inconvenientes, el método de descenso de gradiente ha dado origen a uno de los métodos más empleados para el ajuste iterativo de redes neuronales: el método de retropropagación.

Algoritmo

Dada la función error:

$$E_p = \frac{1}{2} \sum_{i=1}^{n} (t_{ip} - S_{ip})^2$$

la aplicación del método de descenso de gradiente conduce a la siguiente fórmula de actualización de pesos:

$$\Delta_p W_{ij} = -h\, \partial E_p / \partial w_{ij})$$

El problema ahora es el cálculo de $\partial E_p / \partial w_{ij}$. Si w_{ij} es el peso sináptico de una neurona de salida, el cálculo de la derivada parcial no presenta dificultad:

$$\partial E_p / \partial w_{ij} = -(t_{ip} - s_{ip})\ (\partial S_{ip} / \partial w_{ij}) = -(t_{ip} - s_{ip})\, s_{jp}\, F'\ (net_i)$$

donde $net_i = \sum_j w_{ij}\, s_{jp}$

La derivada de la función de activación admite una forma simple en el caso de que sea una sigmoide:

$$F(x) = 1/(1+e^{-x}) \rightarrow F'(x) = F(x) \cdot (1-F(x))$$

o la tangente hiperbólica:

$$F(x) = th(x) \rightarrow F'(x) = (1-(F(x))^2)$$

Para neuronas situadas en capas anteriores, el cálculo $\partial E_p / \partial w_{ij}$ es algo más difícil, pero puede simplificarse poniendo:

$$\delta_{ip} = -\partial E_p / \partial net_{ip}$$

Aplicando la regla de la cadena, tenemos:

$$-\partial E_p / \partial w_{ij} = -(\partial E_p / \partial net_{ip})(\partial net_{ip} / \partial w_{ij}) = \delta_{ip} s_{jp}$$

Basta, por tanto, calcular δ_{ip} para todas las células de la red. Para ello desarrollaremos una fórmula recursiva.

En general se tiene:

$$\delta_{ip} = -(\partial E_p / \partial s_{ip})(\partial s_{ip} / \partial net_{ip})$$

$$\partial s_{ip} / \partial net_{ip} = F'(net_i)$$

Para las neuronas de salida se verifica:

$$\partial E_p / \partial s_{ip} = -(t_{ip} - s_{ip})$$

luego

$$\delta_{ip} = (t_{ip} - s_{ip}) F'(net_i)$$

si la i-'esima célula es de salida.

Para las células de capas posteriores se tiene:

$$\partial E_p / \partial s_{ip} = \sum_k (\partial E_p / \partial net_{kp})(\partial net_k / \partial s_{ip}) = -\sum_k \delta_{kp} w_{ki}$$

donde k recorre las células con las que se halla conectada la i-ésima neurona. De aquí se obtiene la siguiente fórmula recursiva:

$$\delta_{ip} = F'(net_i) \sum_k \delta_{kp} w_{ki}$$

De este modo, los cálculos pueden disponerse así:

1.Alimentación hacia delante. Comenzando por las entradas de la red y progresando hacia delante, calcular las salidas:

$$net_{ip} = \sum_j w_{ij}\, s_{jp}$$

$$s_{ip} = F(net_{ip})$$

2. Calcular δ_{ip} para la capa de salida:

$$\delta_{ip} = (t_{ip} - s_{ip})\, F'(net_{ip})$$

3. Calcular δ_{ip} para las capas anteriores a la salida mediante la siguiente fórmula recursiva ("retropropagación"):

$$\delta_{ip} = F'(net_{ip}) \sum_k \delta_{kp}\, w_{ki}$$

donde el índice k recorre todas las células a las que se halla conectada la i-ésima neurona.

4. Calcular los incrementos de los pesos:

$$\Delta_p w_{ij} = \mu\, \delta_{ip}\, s_{jp}$$

En el modelo de red mostrado en la sección anterior, cada neurona se conecta únicamente a células de la capa siguiente, lo cual permite expresar el algoritmo haciendo referencia explícita a las capas de células. Usamos el índice $c = 0,1,2,...,C$ para ello. Con las notaciones de la sección, el algoritmo de retropropagación sería así:

1. Para $c = 1$ hasta C, hacer lo siguiente:

1.1. $$net^{(c)}_{ip} = \sum_j w^{(c)}_{ij}\, s_{jp}^{(c-1)}$$

1.2. $$s_{ip}(c) = F(net^{(c)}_{ip})$$

2. Calcular $\delta^{(c)}_{ip} = (t_{ip} - s^{(c)}_{ip})\, F'(net^{(c)}_{ip})$

3. Para $c = C-1$ descendiendo hasta 1, hacer lo siguiente:

3.1. $$\delta^{(c)}_{ip} = F'(net^{(c)}_{ip}) \sum_k \delta_{kp}^{(c+1)}\, w_{ki}^{(c+1)}$$

4. Calcular $\Delta_p w_{ij} = \mu \delta_{ip} s_{jp}$

Una vez calculados los incrementos de pesos $\Delta_p w_{ij}$, caben dos posibilidades:

a) Actualización "incremental" de pesos. Los pesos se actualizan inmediatamente tras la presentación de cada patrón:

$$W_{ij} \text{ (nuevo)} = W_{ij} \text{ (antiguo)} + \Delta_p w_{ij}$$

b) Actualización de pesos "en lote" (batch): los incrementos se acumulan para todos los posibles patrones, y los pesos sólo se actualizan después de haber pasado todos los patrones:

$$W_{ij} \text{ (nuevo)} = W_{ij} \text{ (antiguo)} + \sum_p \Delta_p w_{ij}$$

4.- Arquitectura

En este apartado se van a revisar las arquitecturas de redes no recurrentes que se pueden utilizar para el tratamiento de señales que varían en el tiempo.

4.1.- Reconocimiento de Secuencias: Redes de Neuronas con Retraso Temporal Fijo (Time-Delay Neural Network)

Cuando se usan redes multicapa para el tratamiento de secuencias, se suele aplicar una idea muy simple que consiste en que la entrada de la red se componga no sólo del valor de la secuencia en un determinado instante, sino en instantes anteriores. Es como alimentar la red con una ventana temporal.

La idea de introducir el estado de una variable en diversos instantes en la red no sólo se puede aplicar a la entrada, sino también a las activaciones de las neuronas. Una red donde las activaciones de algunas neuronas son simplemente una copia de las activaciones de otra en instantes anteriores de la denominada Red de Neuronas con Retraso Temporal o Time-Delay Neural Network [TDNN] [HAWA 90] [LAHI 88].

Las neuronas con las que se trabajan en redes multicapa con retrasos temporales responden a la ecuación.

$$x_i = f_i \left(\sum_j w_{ij} \cdot x_j \right)$$

Como se observa no existe una dependencia temporal, y la propagación o cálculo de las activaciones se realiza desde la capa superior a la inferior como en cualquier red multicapa.

En estas redes un paso de tiempo hay que entenderlo como iteración. La conexión entre la neurona *j* y la *i*, introduciendo retrasos temporales, se realizará como:

$$x_i' = f_i \left(\sum_l w_{il} \cdot x_l \right)$$

$$x_l = x_j \quad (t - t_l)$$

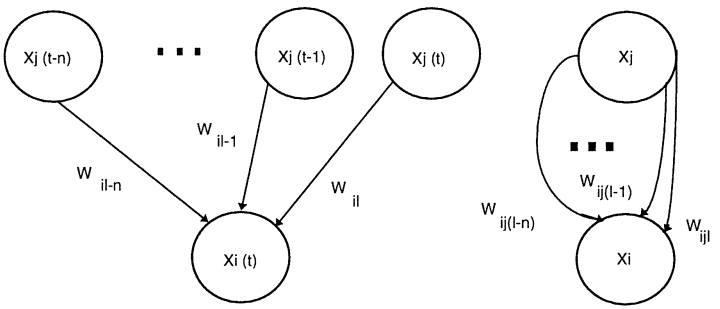

Figura 2.- Dos neuronas conectadas con retrasos temporales.

donde *t* significa interacción y t_l es el retraso temporal. Las neuronas *xl* son simplemente copias de la activación de x_j en instantes o iteraciones anteriores. Se puede dar otra interpretación que consiste en asignar a los pesos distintas velocidades de conexión, siendo unos más lentos que

otros, con lo cual en vez de tener una capa con varias neuronas conteniendo copias de las activaciones de la j tendríamos sólo la neurona j pero conectada a la i con varios pesos de distinta velocidad. La ecuación anterior se transformaría en:

$$x_i = f_i \left(\sum_j \sum_k w_{ijk} \cdot x_j \right)$$

donde w_{ijk} correspondería al peso que conecta la neurona j con la i con retraso o velocidad k. En la figura 2 se visualiza la conexión entre dos neuronas para las dos interpretaciones. Esta última interpretación tiene una gran importancia ya que es bien sabido que existen retrasos temporales significativos en los axones y sinápsis de las redes de neuronas biológicas.

El algoritmo de aprendizaje a utilizar puede ser perfectamente el Backpropagation [RUHW 86] aunque también es posible utilizar otro tipo de algoritmos que son variaciones del anterior, teniendo en cuenta la existencia de retrasos temporales.

4.2.- Reconocimiento de Secuencias: Red de Neuronas con Retraso Temporal Modificable

Las redes con retraso temporal tienen la dificultad de exigir una definición previa de los retrasos de cada peso. Sería adecuado poder entrenar estos retrasos de modo que cada peso adecuara su velocidad de transmisión en función del problema. Se han desarrollado algoritmos que permiten esta características [PEAR 90]. En este apartado se va a describir el algoritmo de Retropropagación Temporal [DADA 91]. Las neuronas de una red con retrasos modificables obedecen a la siguiente ecuación:

$$x_i(t) = f_i \left(\sum_j \sum_k w_{ijk}(t) \cdot x_j(t - t_{ijk}) \right)$$

donde t_{ijk} es el retraso de la conexión k-ésima entre la neurona j y la i, $w_{ijk}(t)$ es el valor del peso en el instante t de dicha conexión, y t significa tiempo y es una variable continua. En este caso el cálculo de la activación de una neurona no sigue el orden establecido por las capas sino

que sigue el orden temporal. En general se va a exigir que algunas neuronas de la red, las de salida, sigan una trayectoria a lo largo del tiempo. El algoritmo de aprendizaje se basa en el descenso del gradiente.

4.3.- Redes Parcialmente Recurrentes

Estas redes son redes multicapa donde existe una capa con neuronas especiales, llamadas neuronas de contexto, que guardan la activación de una capa de neuronas normales del instante anterior. En la figura 3 se muestran cuatro arquitecturas típicas, [JORD 86], [MOZE 89]. Las recurrencias aparecen, si consideramos que la copia de la activación de la capa de neuronas normales, se realiza a través de unas conexiones con pesos fijos con valor 1. La manera en que estas neuronas de contexto influyen en el comportamiento de la red queda clara, si seguimos la operación de la red en forma algorítmica.

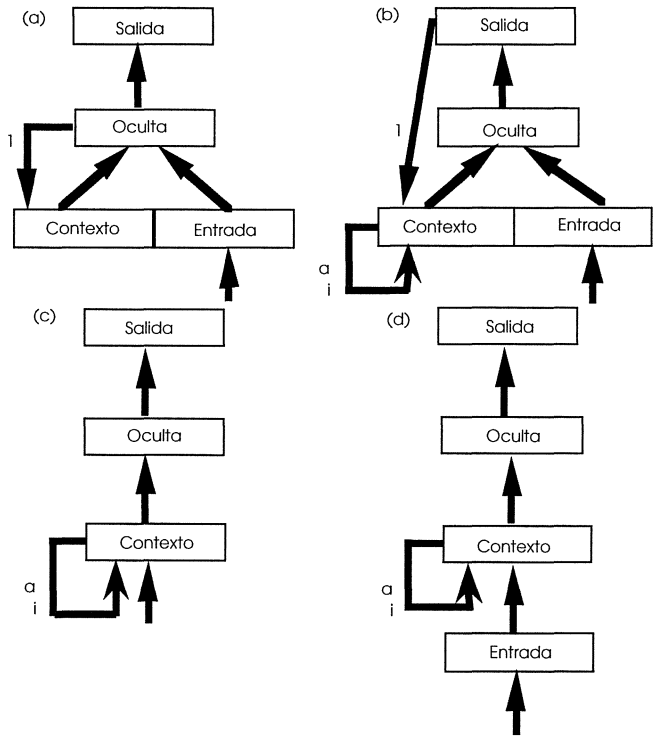

Figura 3.- Arquitecturas típicas de Redes Parcialmente Recurrentes.

1. Para cada patrón repetir:

1.1 Desde t_0 hasta t_1 (época o duración del patrón) hacer en cada t:

a) Introducir el dato del instante t en la red

b) Propagar las activaciones como en una red multicapa, es decir, desde la capa de entrada a la de salida. El cálculo de las activaciones se realiza con la fórmula:

$$x_i = f_i \left(\sum_j w_{ij} \cdot x_j \right)$$

si la capa no tiene conexiones con la capa de contexto, y con:

$$x_i = f_i \left(\sum_j w_{ij} \cdot x_j + \sum_k w_{ik} \cdot c_k \right)$$

donde c_k es la activación de la neurona de contexto k, si tiene conexiones desde la capa de contexto.

c) Calcular el error cometido en el instante t.

d) Retropropagar el error y calcular los $\mathrm{W}w_{ij}$ con el algoritmo de retropropagación del grandiente para redes multicapa.

e) Sumar el $\mathrm{W}w_{ij}$ al calculado en el instante anterior.

f) Copiar las activaciones de la capa de neuronas a la de contexto. Es decir:

$$c_k = 1 \cdot x_k$$

1.2 Actualizar los pesos con $w_{ij} = w_{ij} + \mu \cdot \Delta w_{ij}$

Obviamente la actualización de los pesos se puede realizar para cada patrón (como se ha

expuesto) o en modo por lotes, después de calcular los incrementos para todos los patrones.

5.- Entorno

Por último y como consecuencia de la necesidad de diseñar y entrenar distintas redes para luego activarlas según el estado del entorno, se ha creado una única red capaz de variar los pesos de las conexiones en función del entorno . De este modo tendríamos una red, pero que respondería al entorno cambiando su configuración interna según fuese éste. Tendríamos entonces la potencia de varias redes en solo una que sería capaz de autoconfigurarse según la situación en que se encuentre. Al diseño de una red de este tipo se le ha denominado Diseño de Arquitecturas Modulares.

El fundamento básico de esta arquitectura modular es el siguiente:

Sea una conexión del tipo de la figura 4a, y una como la de figura 4b.

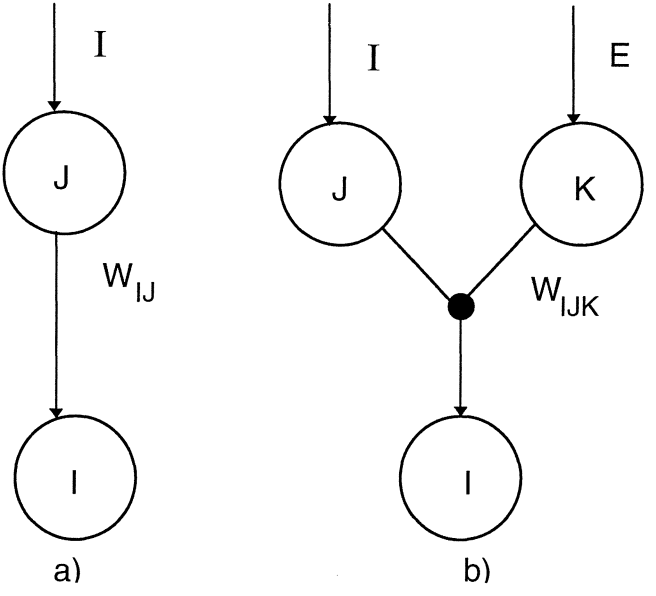

Figura 4.- Conexión Σy conexión ΣJ.

Supongamos que nuestra arquitectura genérica estuviera compuesta de conexiones Σ entre las neuronas *i* y *j*. Para que el valor de la conexión entre las dos neuronas varíe, en respuesta a lo que fuere, es necesario que el valor del mismo peso w_{ij} varíe. Supóngase que se cambia el tipo de conexión a una ΣJ añadiendo una neurona *k*, externa a la red original, en la conexión entre *i* y *j*. En este caso, para que la conexión varíe, pueden ocurrir dos cosas, o bien se cambia el valor de w_{ij}, o bien se cambia el valor de la activación de la neurona *k*. Supóngase que la neurona *k* responde al entorno. Si se precisa un cambio en el valor de la conexión, éste puede estar perfectamente producido por la neurona *k* externa a la red, pero influenciada por el entorno. En efecto:

en 1.a: $x_i = f(w_{ij}\ \ x_j(I))$

en 1.b: $x_i = f(w_{ijk}\ x_j(I)\ x_k(E)) = f(w_{ijk}\ x_k(E)\ x_j(I)) = f(w_{ij}(E)\ x_j(I))$

donde $wij(E) = wijk\ x_k(E)$ es variable ya que depende de $x_k(E)$.

Visto lo anterior, un nivel donde se necesiten varias redes de idéntica arquitectura para responder según sea el entorno se puede ocupar con un sólo módulo como el de la figura 5.

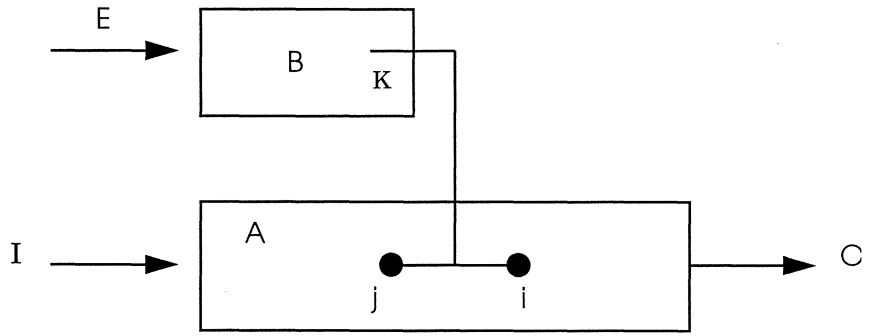

Figura 5.- Arquitectura Modular.

Donde *A* es la red normal y *B* es la red cuya salida controla las conexiones de la normal.

6.- Bibliografía

- HABA 90. Hampshire, J.B. & Waibel, A.H.: "A Novel Objetive Function for Improved Phoneme Recognition Using Time-Delay Neural Networks". IEEE Transactions on Neural Networks, 1, 216-228. 1990.
- JORD 86. Jordan, M.I.: "Attractor Dynamics and Parallelism in a Connectionist Sequential Machine". En Proceedings of the International Joint Conference on Neural Networks, I, 643-644. 1986.
- LAHI 88. Lang, K. & Hinton, G.: "A Time-Delay Neural Network Arquitecture for Speech Recognition". Carnegie Mellon University Tech. Reprt. CMU-CS-88-152. 1988.
- MOZE 89. Mozer, M.C.: "A Focused Back-Propagation Algorithm for Temporal Pattern Recognition". Complex Systems, 3, 349-381. 1989.
- PEAR 90. Pearlmutter, B.A.: "Dynamic Recurrent Neural Networks". Tech. Report CMU-CS 88-191, School of Computer Science, Carnegie Mellon Univesity. 1990.
- RUCL 86. Rumelhart, D.E. & McClelland, J.L.: "Parallel Distributed Processing, vol. 1: Foundations". Cambridge, MA: The MIT Press. 1986.
- RUHW 86. Rumelhart, D.E., Hinton, G.E & Williams, R.J.: "Learning Internal Representations by Error Propagation". En Parallel Distributed Processing, vol. 1, Cambridge, MA. The MIT Press. 1986.