CAPÍTULO 3 RECONOCIMIENTO

3.1. RECONOCIMIENTO PARAMÉTRICO

Los modelos de reconocimiento automático paramétrico son los más utilizados en el área del tratamiento de la señal, entre los que se encuentran los algoritmos genéticos, las cadenas de Markov y las redes neuronales. Los modelos de reconocimiento automático paramétrico se caracterizan por la forma automática en la que se extraen las características espectrales de los sonidos. Una vez seleccionados los sonidos representativos del habla, se presentan a los algoritmos paramétricos y, mediante un algoritmo de aprendizaje, el conocimiento de las características espectrales de estos sonidos queda internamente representado en las estructuras internas que cada modelo contiene (estados internos, neuronas, etc.). Esta fase se denomina de **aprendizaje**.

Concluida la fase de aprendizaje se puede pasar a la de **reconocimiento**, en la que los patrones obtenidos del hablante se confrontan con la información de los estados internos (cadenas de Markov), neuronas (redes neuronales) o estructuras de datos (algoritmos genéticos).

3.2 RECONOCIMIENTO MEDIANTE UNA RED NEURONAL

El método de reconocimiento de voz que se va a emplear para la detección de las vocales es un método de reconocimiento paramétrico, concretamente, emplearemos redes neuronales para este

fin. Para abordar tareas como las denominadas del mundo real, donde la información se presenta masiva, imprecisa y distorsionada, los sistemas de procesamiento basados en la arquitectura Von Neumann se muestran ineficaces.

Para tratar este tipo de tareas se han propuesto modelos alternativos, de los cuales las redes neuronales artificiales y los sistemas basados en lógica borrosa son los que cuentan con mayor popularidad y utilización. Estos nuevos modelos de procesamiento y control, junto con algunos otros (como los algoritmos genéticos), se engloban con los términos inteligencia computacional (por oposición a la inteligencia artificial clásica, basada en computadores Von Neumann). El denominador común de estas nuevas técnicas radica en su inspiración en las soluciones que la naturaleza ha encontrado a lo largo de millones de años de evolución para el tratamiento del tipo de información masiva y distorsionada procedente del entorno natural, soluciones que copiadas en sistemas artificiales se espera que contribuyan a resolver importantes problemas tecnológicos (visión, habla, control de sistemas complejos, inteligencia artificial, etc.).

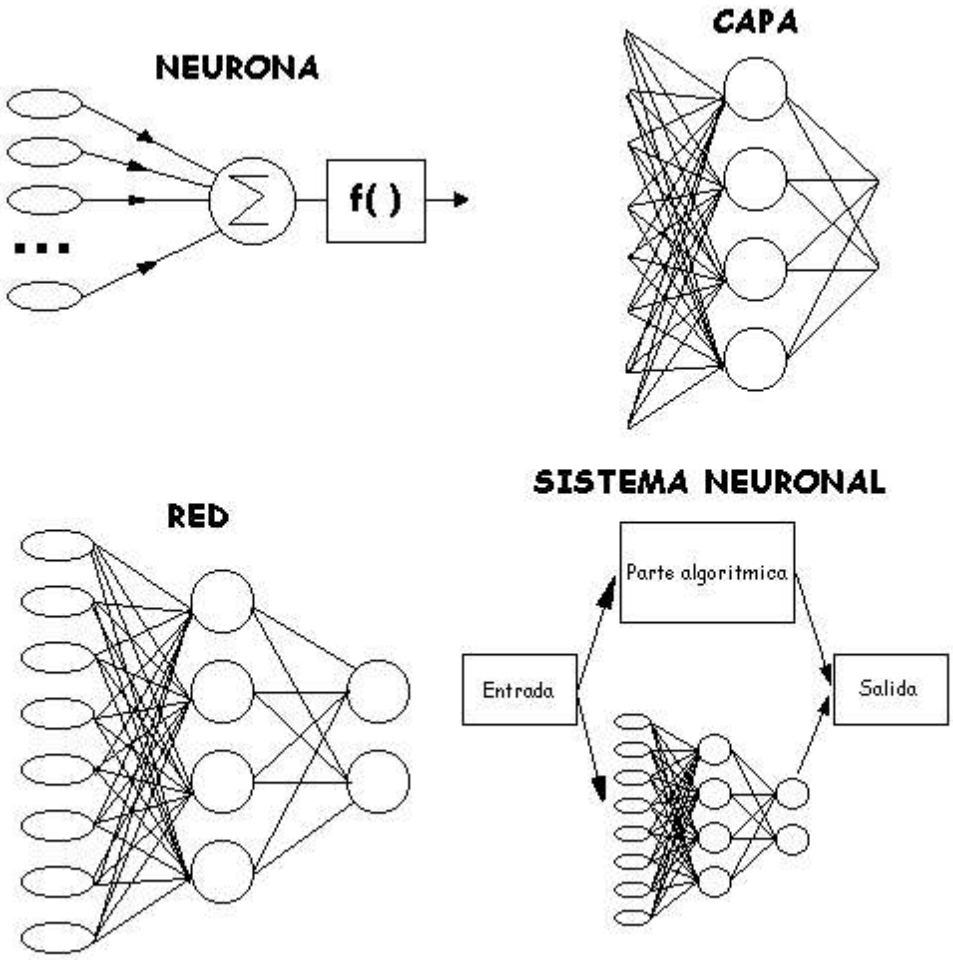
El trabajo en este tipo de materias, hasta hace poco minoritarias, constituye hoy en día algunos de los esfuerzos más dinámicos dentro del I+D. En la actualidad, dos técnicas son las que están causando un mayor impacto, debido a su aplicabilidad práctica: las redes neuronales y los sistemas borrosos. Las redes neuronales artificiales, mediante un estilo de computación paralelo, distribuido y adaptativo, son capaces de aprender a partir de ejemplos. Estos sistemas imitan esquemáticamente la estructura hardware del cerebro para tratar de reproducir algunas de sus capacidades. En la práctica, una red neuronal artificial puede simularse mediante un programa de ordenador, o bien realizarse en circuitos electrónicos específicos.

Las computadoras convencionales tienen un enfoque descendente, en el que elaborados algoritmos manejan complicados símbolos en un procesador muy complejo, el cerebro adopta un enfoque ascendente, en el que el procesamiento es fruto de la autoorganización de millones de procesadores elementales. A causa de estos dos diferentes enfoques, existen problemas que resuelven muy bien los ordenadores y otros en los que el cerebro es mucho más eficiente. Los computadores fueron creados por el hombre específicamente para las tareas que podemos denominar de alto nivel, como el razonamiento o el cálculo, que pueden ser fácilmente resueltos mediante el procesamiento de símbolos; en este tipo de tareas nuestro cerebro no es excesivamente diestro, y actúa en clara desventaja frente a la electrónica. Sin embargo, en tareas de procesamiento de bajo nivel, como las de reconocimiento de patrones, percepción, control, etc., los computadores se desenvuelven todavía torpemente, pues en origen no fueron ideados para ello. Para esta clase de problemas, esenciales para la supervivencia de un organismo vivo, la naturaleza encontró una excelente solución: el procesamiento autoorganizado que emerge de la interacción de numerosos procesadores elementales. Las redes neuronales tratan de emular el hardware del cerebro con el fin de obtener alguna de sus características.

3.2.1 Estructura de un sistema neuronal artificial

Los elementos básicos de un sistema neuronal biológico son las neuronas, que se agrupan en conjuntos compuestos por millones de ellas organizadas en capas, constituyendo un sistema con funcionalidad propia. Un conjunto de estos subsistemas da lugar a un sistema global (el sistema nervioso, en el caso biológico). En la realización de un sistema neuronal artificial puede

establecerse una estructura jerárquica similar. El elemento esencial de partida será la neurona artificial, que se organizará en capas; varias capas constituirán una red neuronal; y, por último, una red neuronal (o un conjunto de ellas), junto con las interfaces de entrada/salida y los módulos convencionales adicionales necesarios, constituirán el sistema global de proceso.



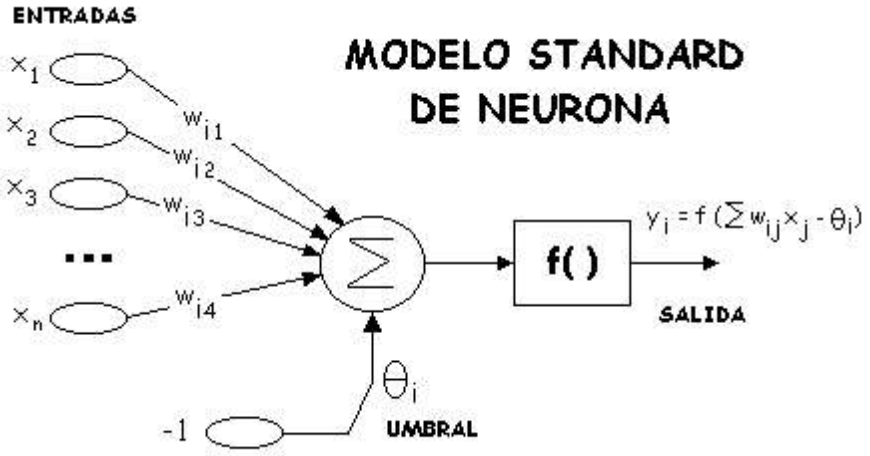
Formalmente, un sistema neuronal o conexionista, está compuesto por los siguientes elementos:

- Un conjunto de procesadores elementales o neuronas artificiales.
- Un patrón de conectividad o arquitectura.
- Una dinámica de activaciones.
- Una regla o dinámica de aprendizaje.
- El entorno donde opera.

3.2.2 Modelo estándar de una neurona artificial

El modelo de neurona expuesto en la sección anterior resulta muy general. En la práctica suele utilizarse un modelo simple de neurona que denominaremos neurona estándar. La neurona estándar consiste en:

- Un conjunto de entradas $x_j(t)$ y pesos sinápticos w_{ij}.
- Una regla de propagación $h(t)=\sigma(w_{ij}, x_j(t))$; $h(t) = \Sigma\ w_{ij}\ x_j$ es la más común.
- Una función de activación $y(t)=f(h(t))$, que representa simultáneamente la salida de la neurona y su estado de activación.



Con frecuencia se añade al conjunto de pesos de la neurona un parámetro adicional θ, que denominaremos umbral, que se resta del potencial postsináptico, por lo que el argumento de la función de activación queda:

$$\sum w_{ij} x_j - \theta_i$$

Esto representa añadir un grado de libertad adicional a la neurona. En conclusión, el modelo de neurona que denominaremos estándar queda:

$$y_i = f_i\left(\sum w_{ij} x_j - \theta_i\right)$$

3.2.3 Arquitectura de redes neuronales.

En general, las neuronas se suelen agrupar en unidades estructurales que denominaremos capas. Las neuronas de una capa pueden agruparse, a su vez formando grupos neuronales (clusters). Dentro de un grupo, o de una capa si no existe este tipo de agrupación, las neuronas suelen ser del mismo tipo. Finalmente, el conjunto de una o más capas constituye la red neuronal.

Se distinguen tres tipos de capas: de entrada, de salida y ocultas. Una capa de entrada o sensorial está compuesta por neuronas que reciben datos o señales procedentes del entorno (por ejemplo, proporcionados por sensores). Una capa de salida es aquella cuyas neuronas proporcionan la respuesta de la red neuronal (sus neuronas pueden estar conectadas a efectores). Una capa oculta es aquella que no tiene una conexión directa con el entorno, es decir, que no se conecta directamente ni a órganos sensores ni a efectores. Este tipo de capa proporciona a la red neuronal grados de libertad adicionales, gracias a los cuales puede encontrar representaciones internas correspondientes a determinados rasgos del entorno, proporcionando una mayor riqueza computacional.

Las conexiones entre las neuronas pueden ser excitatorias o inhibitorias: un peso sináptico negativo define una conexión inhibitoria, mientras que uno positivo determina una conexión excitatoria. Habitualmente, no se suele definir una conexión como de un tipo o de otro, sino que por medio del aprendizaje se obtiene un valor para el peso, que incluye signo y magnitud.

3.2.4 Modos de operación: Recuerdo y Aprendizaje.

Clásicamente se distinguen dos modos de operación en los sistemas neuronales: el modo recuerdo o ejecución, y el modo aprendizaje o entrenamiento. Este último es de particular interés, pues una característica fundamental de las redes neuronales es que se tratan de sistemas entrenables, capaces de realizar un determinado tipo de procesamiento o cómputo aprendiéndolo a partir de un conjunto de patrones de aprendizaje o ejemplos.

3.2.4.1 Fase de aprendizaje. Convergencia.

En el contexto de las redes neuronales puede definirse el aprendizaje como el proceso por el que se produce el ajuste de los parámetros libres de la red a partir de un proceso de estimulación por el entorno que rodea la red. El tipo de aprendizaje vendrá determinado por la forma en la que dichos parámetros son adaptados. En la mayor parte de las ocasiones el aprendizaje consiste simplemente en determinar un conjunto de pesos sinápticos que permita a la red realizar correctamente el tipo de procesamiento deseado.

Cuando se construye un sistema neuronal, se parte de un cierto modelo de neurona y de una determinada arquitectura de red, estableciéndose los pesos sinápticos iniciales como nulos o aleatorios. Para que la red resulte operativa es necesario entrenarla, lo que constituye el modo aprendizaje. El entrenamiento o aprendizaje se puede llevar a cabo a dos niveles. El más convencional es el de modelado de las sinapsis, que consiste en modificar los pesos sinápticos siguiendo una cierta regla de aprendizaje, construida normalmente a partir de la optimización de una función de error o coste, que mide la eficacia actual en la operación de la red. Si denominamos $w_{ji}(t)$ al peso que conecta la neurona presináptica j con la postsináptica i en la iteración t, el algoritmo de aprendizaje, en función de las señales que en el instante t llegan procedentes del entorno, proporcionará el valor $\Delta w_{ji}(t)$ que da la modificación que se debe incorporar en dicho peso, el cual quedará actualizado de la forma:

$$\Delta w_{ji}(t+1)=w_{ji}(t)+\Delta w_{ji}(t)$$

El proceso de aprendizaje es usualmente iterativo, actualizándose los pesos de la manera anterior, una y otra vez, hasta que la red neuronal alcanza el rendimiento deseado.

Algunos modelos neuronales incluyen otro nivel en el aprendizaje, la creación o destrucción de neuronas, en el cual se modifica la propia arquitectura de la red. En cualquier caso, en un proceso de aprendizaje la información contenida en los datos de entrada queda incorporada en la propia estructura de la red neuronal, la cual almacena la representación de una cierta imagen de su entorno.

Los dos tipos básicos de aprendizaje son el supervisado y el no supervisado, cuya distinción proviene en origen del campo del reconocimiento de patrones. Ambas modalidades pretenden estimar funciones entrada/salida multivariable o densidades de probabilidad, pero mientras que en el aprendizaje supervisado se proporciona cierta información sobre estas funciones (como la distribución de las clases, etiquetas de los patrones de entrada o salidas asociadas a cada patrón), en el autoorganizado no se proporciona información alguna. Las reglas de aprendizaje supervisadas suelen ser computacionalmente más complejas, pero también más exactos sus resultados.

En el aprendizaje supervisado se presenta a la red un conjunto de patrones, junto con la salida deseada u objetivo, e iterativamente ésta ajusta sus pesos hasta que su salida tiende a ser la deseada, utilizando para ello información detallada del error que comete en cada paso.

En el aprendizaje no supervisado o autoorganizado se presentan a la red multitud de patrones sin adjuntar la respuesta que deseamos. La red, por medio de la regla de aprendizaje, reconoce regularidades en el conjunto de entradas, extrae rasgos, o agrupa patrones según su similitud (clustering). Un ejemplo típico de modelo que emplea este tipo de aprendizaje es el de los mapas autoorganizados.

3.2.4.2 Fase de recuerdo o ejecución.

Generalmente (aunque no en todos los modelos), una vez que el sistema ha sido entrenado, el aprendizaje "se desconecta", por lo que los pesos y la estructura quedan fijos, estando la red neuronal ya dispuesta para procesar datos. Este modo de operación se denomina modo recuerdo o de ejecución.

3.3 MAPAS AUTOORGANIZADOS

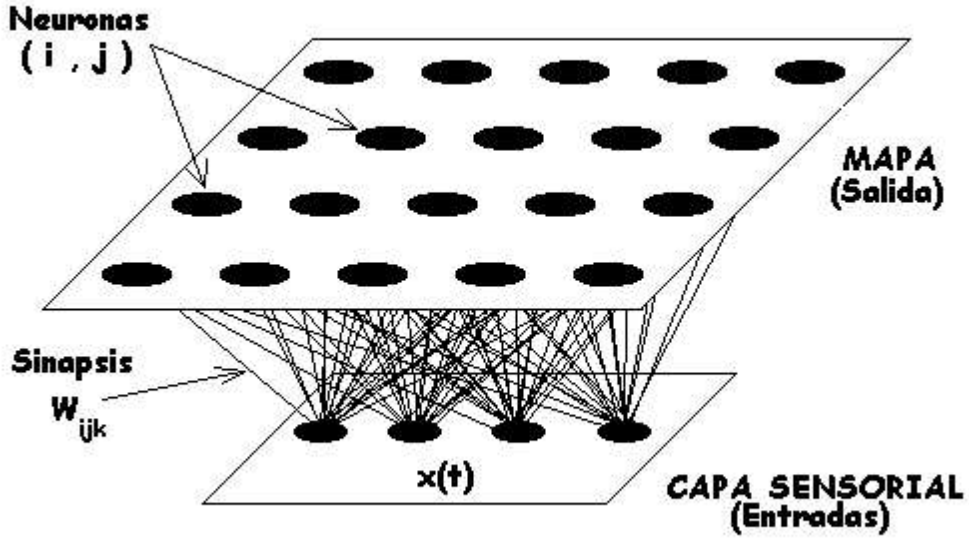
La base del reconocedor automático de vocales son las redes neuronales. En el apartado anterior se describieron las redes neuronales y se puso de manera patente su utilidad en las funciones de reconocimiento de patrones procedentes del mundo real. Para la realización del reconocedor automático de vocales se va a utilizar un sistema autónomo de reconocimiento basado en redes neuronales: los mapas autoorganizados de Kohonen. Se debe contar con un número lo suficientemente amplio de vocales en diversas pronunciaciones para proporcionar la mayor generalidad posible al sistema. Como ya se explicó, las redes neuronales necesitan de un conjunto suficientemente elevado de parámetros a partir de los cuales ella misma pueda proporcionar en la fase de aprendizaje la mayor generalidad posible.

3.3.1 Modelo de mapas autoorganizados (Kohonen, 1982).

Se observa que en muchas regiones del córtex de los animales superiores aparecen zonas donde las neuronas detectoras de rasgos (o características) se distribuyen en una ordenación topológica. Por lo que respecta al sentido del oído, existen en el cerebro áreas que representan mapas tonotópicos, donde los detectores de determinados rasgos relacionados con el tono de un sonido se encuentran ordenados en dos dimensiones. En resumen, la representación de la información en la corteza cerebral aparece con frecuencia organizada espacialmente, circunstancia que en el modelo neuronal de mapas autoorganizados se trata de reproducir.

Los mapas autoorganizados, aparte de su interés como una sencilla modelización de redes neuronales naturales, poseen un gran potencial de aplicabilidad práctica en la clasificación de patrones. En este modelo, las neuronas se organizan en una arquitectura unidireccional de dos capas. La primera es la capa de entrada o sensorial, que consiste en m neuronas, una por cada variable de entrada, que se comportan como buffers, distribuyendo la información procedente del espacio de entrada a las neuronas de la segunda capa. Las entradas son muestras estadísticas $\mathbf{x}(t) \in R^m$ del espacio sensorial. El procesamiento se realiza en la segunda capa, que forma el mapa de rasgos, consistente habitualmente en una estructura rectangular de nx*ny neuronas, que operan en paralelo. Aunque la arquitectura rectangular es la más corriente, a veces también se utilizan capas de una sola dimensión (cadena lineal de neuronas) o de tres dimensiones (paralelepípedo). Etiquetaremos las m neuronas de entrada con el índice k ($1 \le k \le m$), y las nx*ny neuronas del

mapa con un par de índices (i,j) ($1 \le i \le nx$, $1 \le j \le ny$) que determinan su localización espacial. Cada neurona de entrada *k* está conectada a todas las neuronas (i,j) del mapa mediante un peso sináptico w_{ijk}.



En resumen, el mapa puede describirse como una matriz de procesadores elementales (i,j) ordenados en dos dimensiones, que almacenan un vector de pesos sinápticos o vector de referencia (codebook) $\mathbf{w_{ij}}(t)$, con $\{\mathbf{w_{ij}}(t);\ \mathbf{w_{ij}} \in R^m,\ 1 \le i \le nx,\ 1 \le j \le ny\}$.

En **fase de ejecución** (operación normal de la red), los pesos permanecen fijos. En primer lugar, cada neurona (i,j) calcula la similitud entre el vector de entradas **x,** $\{x_k \mid 1 \le m\}$ y su propio vector de pesos sinápticos $\mathbf{w_{ij}}$, según una cierta medida de distancia o criterio de similitud establecido. A continuación, se declara vencedora la neurona **g** = (g1,g2), cuyo vector de pesos $\mathbf{w_g}$ es más similar al de entradas. De esta manera, cada neurona actúa como un detector de rasgos específicos, y la neurona ganadora nos indica el tipo de rasgo o patrón detectado en el vector de entradas.

$$d(\mathbf{w_g},\mathbf{x}) = \min_{ij}\{d(\mathbf{w_{ij}},\mathbf{x})\}$$

En la **fase de aprendizaje** cada neurona del mapa sintoniza con diferentes rasgos del espacio de entrada. El proceso es el siguiente. Tras la presentación y procesamiento de un vector de entradas **x**(t), la neurona vencedora modifica sus pesos de manera que se parezcan un poco más a **x**(t). De este modo, ante el mismo patrón de entrada, dicha neurona responderá en el futuro todavía con más intensidad. El proceso se repite para numerosos patrones de entrada, de forma que al final los diferentes vectores de referencia sintonizan con dominios específicos de las variables de

entrada, y tienden a representar la función densidad de probabilidad $p(\mathbf{x})$ (o función de distribución) del espacio sensorial. Si dicho espacio está dividido en grupos, cada neurona se especializará en uno de ellos, y la operación esencial de la red se podrá interpretar entonces como un análisis cluster.

Lo descrito hasta el momento responde a un esquema competitivo clásico de relativa sencillez, en el que cada neurona actúa en solitario. Sin embargo, el modelo de mapa de Kohonen aporta una importante novedad, pues incorpora a este esquema relaciones entre las neuronas próximas en el mapa. Para ello introduce una **función de vecindad**, que define un entorno alrededor de la neurona ganadora actual (vecindad); su efecto es que durante el aprendizaje se actualizan tanto los pesos de la vencedora como los de las neuronas pertenecientes a su entorno. De esta manera, en el modelo de mapas autoorganizados se logra que neuronas próximas sintonicen con patrones similares, quedando de esta manera reflejada sobre el mapa una cierta imagen del orden topológico presente en el espacio de entrada (ordenación de los detectores de rasgos).

En esencia, por medio del proceso descrito, los mapas autoorganizados realizan la proyección no lineal de un espacio multidimensional de entrada R^m sobre un espacio discreto de salida, representado por la capa de neuronas. El mapa representa una imagen del espacio sensorial, pero de menor número de dimensiones, reflejando con mayor fidelidad aquellas dimensiones del espacio de entrada de mayor varianza (que suelen coincidir con los rasgos más importantes de las entradas). La proyección se realiza de manera óptima, en el sentido de que la topología del espacio de entrada preserva en lo posible sobre la superficie de la red (entradas $\mathbf{x}$ similares se corresponden con neuronas próximas). Así, la distribución de las neuronas sobre el mapa resulta ser un reflejo de la función densidad de probabilidad $p(\mathbf{x})$: regiones en el espacio sensorial cuyos representantes $\mathbf{x}$ aparecen con más frecuencia ($p(\mathbf{x})$ mayor) serán proyectadas sobre un número mayor de neuronas en el mapa.

La función vecindad representa matemáticamente de una forma sencilla el efecto global de las interacciones laterales existentes entre las neuronas en el cerebro, pues en vez de considerar en detalle que una neurona trata de activar a sus vecinas y de inhibir a las alejadas (como sucede en el córtex), modelamos esta situación mediante una sencilla función que define el tamaño de la vecindad en torno a la vencedora, dentro de la cual todas las neuronas son premiadas actualizando sus pesos, y fuera de ella son castigadas al no actualizar sus pesos o al hacerlo en sentido contrario.

3.3.2 Algoritmo de aprendizaje.

En los modelos de redes neuronales basados en mapas autoorganizados la modificación de los pesos no se aplica solamente a una neurona específica (la ganadora), sino también a su vecindad. Al comienzo del entrenamiento la vecindad comprende una amplia región del mapa, lo que permite una ordenación global de los pesos sinápticos. Con el transcurso de las iteraciones, el tamaño de la vecindad se reduce, y finalmente solamente se modifican los pesos de la neurona ganadora. Así, el proceso de aprendizaje comprende dos fases fundamentales: una ordenación global, en la que se produce el despliegue del mapa; y un ajuste fino, en el que las neuronas se especializan. Un algoritmo autoorganizado bastante habitual es el siguiente:

1) **Inicialización de los pesos** sinápticos w_{ijk}. Se puede partir en t=0 de diferentes configuraciones: pesos nulos, aleatorios de pequeño valor absoluto (lo más habitual), o con un valor de partida predeterminado.

2) En cada iteración, presentación de un patrón **x**(t) tomado de acuerdo con la función de distribución $p(\mathbf{x})$ del espacio sensorial de entrada (en la muy habitual situación de disponer solamente de un conjunto finito de patrones de entrenamiento, basta con tomar al azar uno de ellos y presentarlo a la red).

3) Cada neurona (i,j) en paralelo del mapa calcula la similitud entre su vector de pesos sinápticos $\mathbf{w_{ij}}$ y el actual vector de entradas **x**. Un criterio de medida de similitud muy utilizado es la distancia euclídea:

$$d^2(\mathbf{w_{ij}},\mathbf{x}) = \Sigma_{k=1}^{n}(w_{ijk}-x_k)^2$$

4) Determinación de la **neurona ganadora g** = (g1,g2), cuya distancia sea la menor de todas.

5) **Actualización de los pesos** sinápticos de la neurona ganadora **g** = (g1,g2), y los de sus neuronas vecinas. La regla empleada es:

$$w_{ijk}(t+1) = w_{ijk}(t) + \alpha(t) * h(|\mathbf{i}-\mathbf{g}|,t) * (x_k(t) - w_{ijk}(t))$$

Donde $\alpha(t)$ es un parámetro denominado **ritmo de aprendizaje**. La función h(|i-g|,t) se denomina **función de vecindad**, puesto que establece qué neuronas son las vecinas a la actualmente ganadora. Esta función depende de la distancia entre la neurona **i** y la ganadora **g**, valiendo cero cuando **i** no pertenece a la vecindad de **g** (con lo que sus pesos no son actualizados), y un número positivo cuando sí pertenece (sus pesos sí son modificados). La vecindad es un conjunto de neuronas centrado en la ganadora. Tanto α como el radio de vecindad disminuyen monótonamente con t.

6) Si se ha alcanzado el número máximo de iteraciones establecido, entonces el proceso de aprendizaje finaliza. En caso contrario, se vuelve al paso 2.

Se puede realizar a continuación una segunda fase en el aprendizaje, en la que se produce el ajuste fino del mapa, de modo que la distribución de los pesos sinápticos se ajuste más a la de las entradas. El proceso es similar al anterior, tomando $\alpha(t)$ constante e igual a un pequeño valor (por ejemplo, 0.01), y radio de vecindad constante e igual a uno.

En el aprendizaje, el número de iteraciones debe ser suficientemente grande por requerimientos estadísticos, así como proporcional al número de neuronas del mapa (a más neuronas, son necesarias más iteraciones), e independiente del número de componentes de **x**. Aunque 500 iteraciones por neurona constituyen una cifra adecuada, de 50 a 100 suelen ser suficientes para la mayor parte de problemas. Entre 20.000 y 100.000 iteraciones representan cifras habituales en la simulación por ordenador del entrenamiento de un mapa autoorganizado.

3.3.2.1 Consideraciones prácticas: ritmo de aprendizaje y función de vecindad.

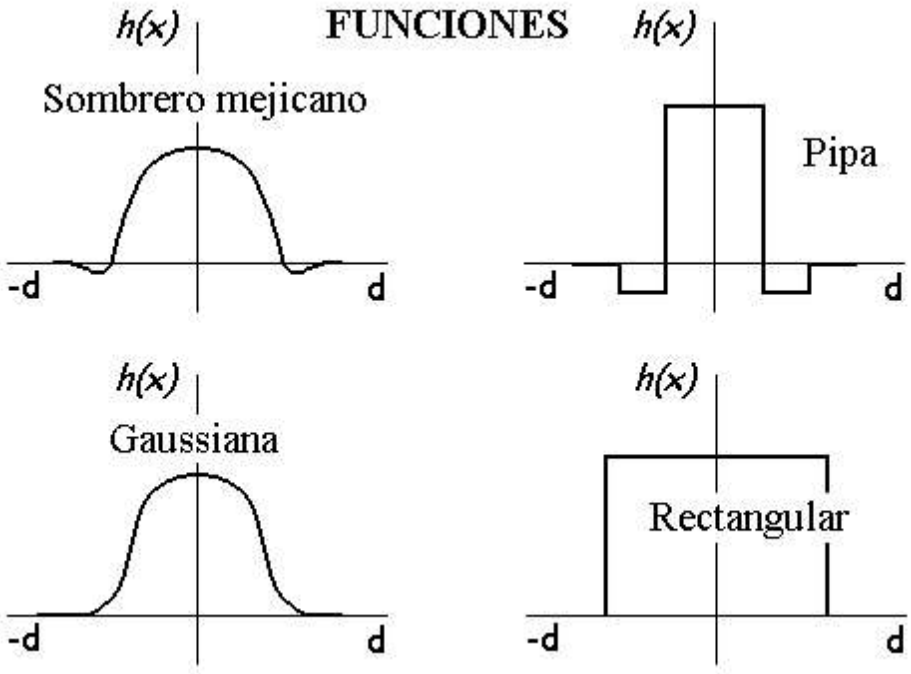
El **ritmo de aprendizaje** $\alpha(t)$ es una función monótonamente decreciente con el tiempo, siendo habitual su actualización mediante una función lineal.

$$\alpha(t) = \alpha_0 + (\alpha_f - \alpha_0)\, t / t_f$$

α_0 es el ritmo de aprendizaje inicial (<1), α_f el final (≅0.01) y t_f el máximo número de iteraciones hasta llegar a α_f.

La función de vecindad $h(|\mathbf{i}-\mathbf{g}|,t)$, define en cada iteración t si una neurona **i** pertenece o no a la vecindad de la vencedora **g**. La vecindad es simétrica y centrada en **g**, de ahí que representemos como uno de sus argumentos la distancia entre la neurona genérica **i**=(i,j) y la vencedora **g**= (g1,g2), pues:

$$|i-g| = \sqrt{(i-g1)^2 + (j-g2)^2}$$



En general, *h(x)* decrece con la distancia a la vencedora, y depende de un parámetro denominado radio de vecindad R(t), que representa el tamaño de la vecindad actual. En realidad, bajo la forma funcional de *h(x)* se encapsula el complejo sistema de interacciones laterales existente entre las

neuronas del mapa, que sí aparecerían explícitamente en una formulación del modelo más orientada a la biología, estableciendo la forma y la intensidad de la interacción entre neuronas.

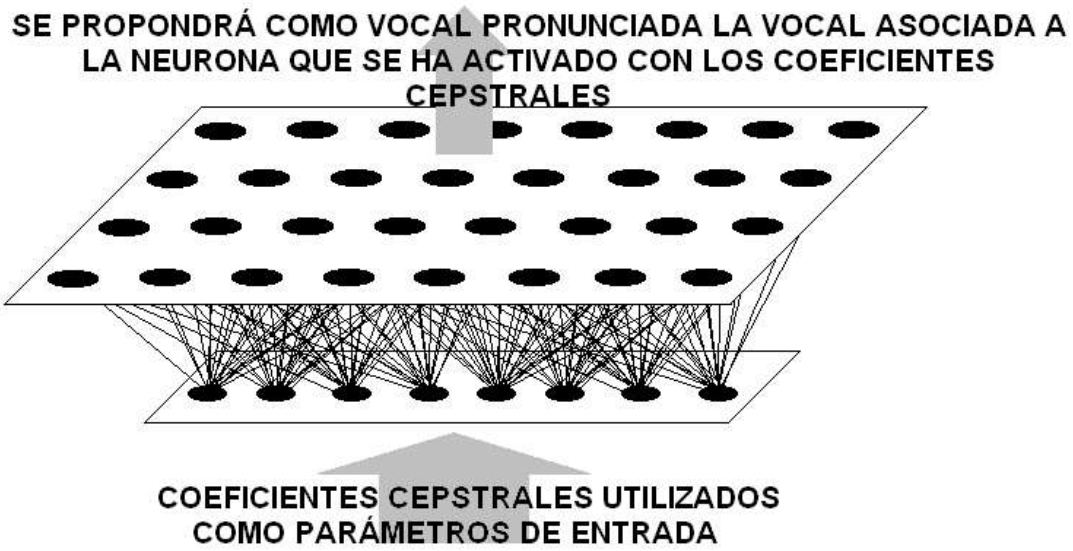
La función de vecindad posee una forma definida, pero su radio R(t) varía con el tiempo. Se parte de un valor inicial R_0 grande, que determina vecindades amplias, con el fin de lograr la ordenación global del mapa. R(t) disminuye monótonamente con el tiempo, hasta alcanzar un valor final de R_f=1, por el que solamente se actualizan los pesos de la neurona vencedora y las adyacentes. Una posible función de actualización de R(t) es la siguiente, linealmente decreciente, donde t es la iteración y t_f el número de iteraciones para alcanzar R_f:

$$R(t) = R_0 + (R_f - R_0)\, t / t_f$$

3.4 SIMULACIÓN MATLAB®.

Antes de la implementación física del reconocedor basado en la red neuronal de mapas autoorganizados, se hace necesaria la ejecución externa del algoritmo de aprendizaje de la red neuronal para obtener los pesos sinápticos de las neuronas y el mapeado neurona-vocal que permita establecer el modelo de procesamiento del dispositivo. En primer lugar, se debe establecer la topología de la red neuronal que se desplegará en la FPGA como sistema de reconocimiento.

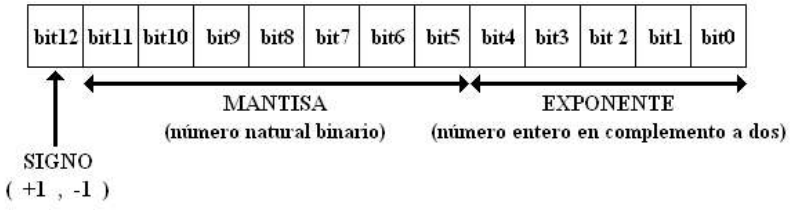
<u>SOLUCIÓN ADOPTADA:</u> La topología seleccionada es un mapa autoorganizado de 32 neuronas distribuidas en cuatro filas y ocho columnas. La capa de entrada estará formada por ocho neuronas, conformando el buffer donde se almacenarán los ocho parámetros que se utilizarán para activar las neuronas. Estos ocho parámetros son los ocho primeros coeficientes cepstrales, representativos de las características vocálicas de la señal de audio y obtenidos a partir de las muestras captadas por el CODEC TLC320AD77C de Texas Instruments presente en la tarjeta XSA. Existe un mapeado que asocia cada neurona a una vocal, de tal forma que cuando los parámetros cepstrales activen una neurona, se propondrá como vocal pronunciada la vocal asociada a la neurona.



Una vez establecida la solución adoptada, debe ponerse en marcha el algoritmo de aprendizaje aplicando como patrones los coeficientes cepstrales de diversas pronunciaciones vocálicas aleatorias. Para obtener la máxima coherencia en la solución adoptada conviene tomar los coeficientes cepstrales que servirán como patrones del procesamiento final realizado en el diseño VHDL sobre la FPGA de las muestras captadas por el CODEC TLC320AD77C de Texas Instruments. Esto se consigue mediante una modificación del diseño final en VHDL del reconocedor que muestre a través del display de leds los coeficientes cepstrales de la vocal pronunciada. Invento un protocolo para leer los coeficientes cepstrales captados con el diseño modificado en la tarjeta XSA.

1. En un determinado instante, el display de leds avisará al usuario para que comience la pronunciación de una vocal.

2. Posteriormente, cuando se considere que la señal de voz es estable, el CODEC TLC320AD77C de Texas Instruments tomará 512 muestras que la FPGA procesará.



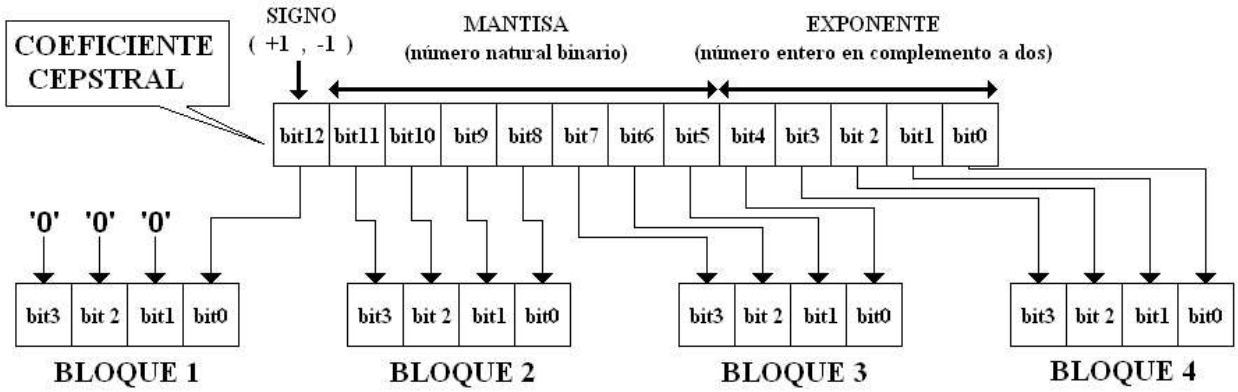
$$\text{SIGNO} \times \text{MANTISA} \times 2^{\text{EXPONENTE}}$$

3. Con las 512 muestras, la FPGA calculará los coeficientes cepstrales correspondientes. Estos son los datos que se utilizarán en el proceso de aprendizaje. Los ocho coeficientes cepstrales

se encuentran en el formato flotante de 13 bits utilizado por la unidad aritmética del reconocedor en el diseño en VHDL, donde el bit más significativo se corresponde con el signo de la mantisa ('0' para el signo positivo y '1' para el signo negativo), los 7 bits siguientes se corresponden con el módulo de la mantisa, y los últimos 5 bits se corresponden con el exponente entero en complemento a dos de 5 bits de la potencia en base dos por la que se multiplicará la mantisa para conformar el número flotante.

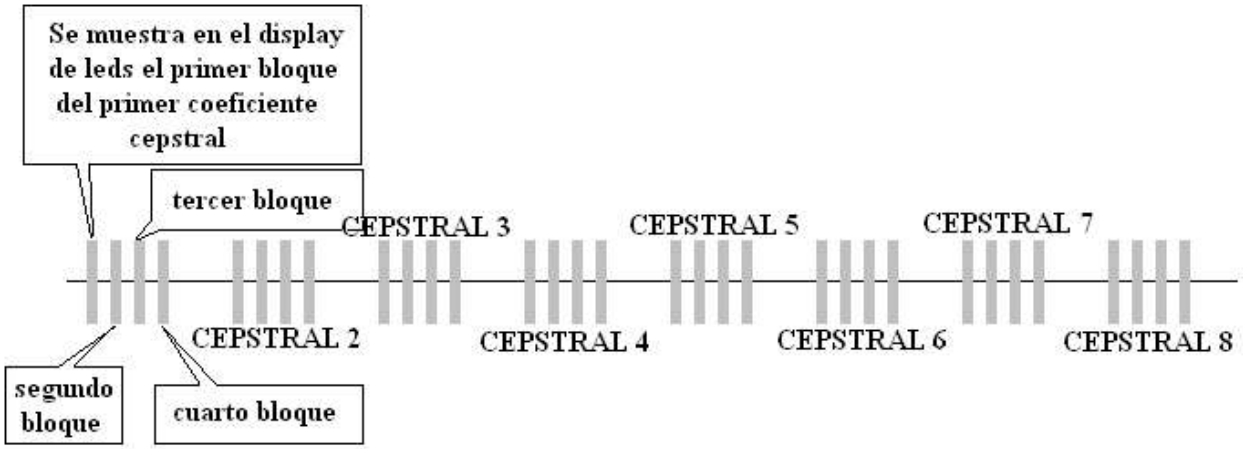
4. Los coeficientes cepstrales serán almacenados para que puedan ser posteriormente leídos por el usuario a través del display de leds.

5. Se inicia el proceso de transmisión de los coeficientes cepstrales a través del display de leds. Cada coeficiente cepstral se dividirá en cuatro bloques de cuatro bits cada uno. El primer bloque tendrá un '0' lógico en los 3 bits más significativos y el bit número 12 del coeficiente cepstral en el bit menos significativo. El segundo bloque estará formado por los bits 11, 10, 9 y 8 del coeficiente cepstral. El tercer bloque estará formado por los bits 7, 6, 5 y 4 del coeficiente cepstral. Y el cuarto bloque estará formado por los bits 3, 2, 1 y 0 del coeficiente cepstral.



6. El display de leds mostrará el código hexadecimal de los bits del primer bloque y mantendrá el dato el tiempo suficiente como para permitir al usuario su lectura. Se apagará el display de leds para indicar una transición. Posteriormente, el display de leds mostrará el código hexadecimal de los bits del segundo bloque y mantendrá el dato el tiempo suficiente como para permitir al usuario su lectura. Se apagará el display de leds para indicar una transición y continuará el proceso para los bloques segundo y primero. Terminada la lectura, el usuario dispondrá del código hexadecimal del primer coeficiente hexadecimal.

7. Se apagarán los leds del display un tiempo considerable para indicar la transición entre coeficientes cepstrales de forma que el usuario esté preparado para la lectura de los cuatro siguientes bloques correspondientes al siguiente coeficiente cepstral. Volverá al paso 6 hasta que se hayan leído todos los coeficientes cepstrales.

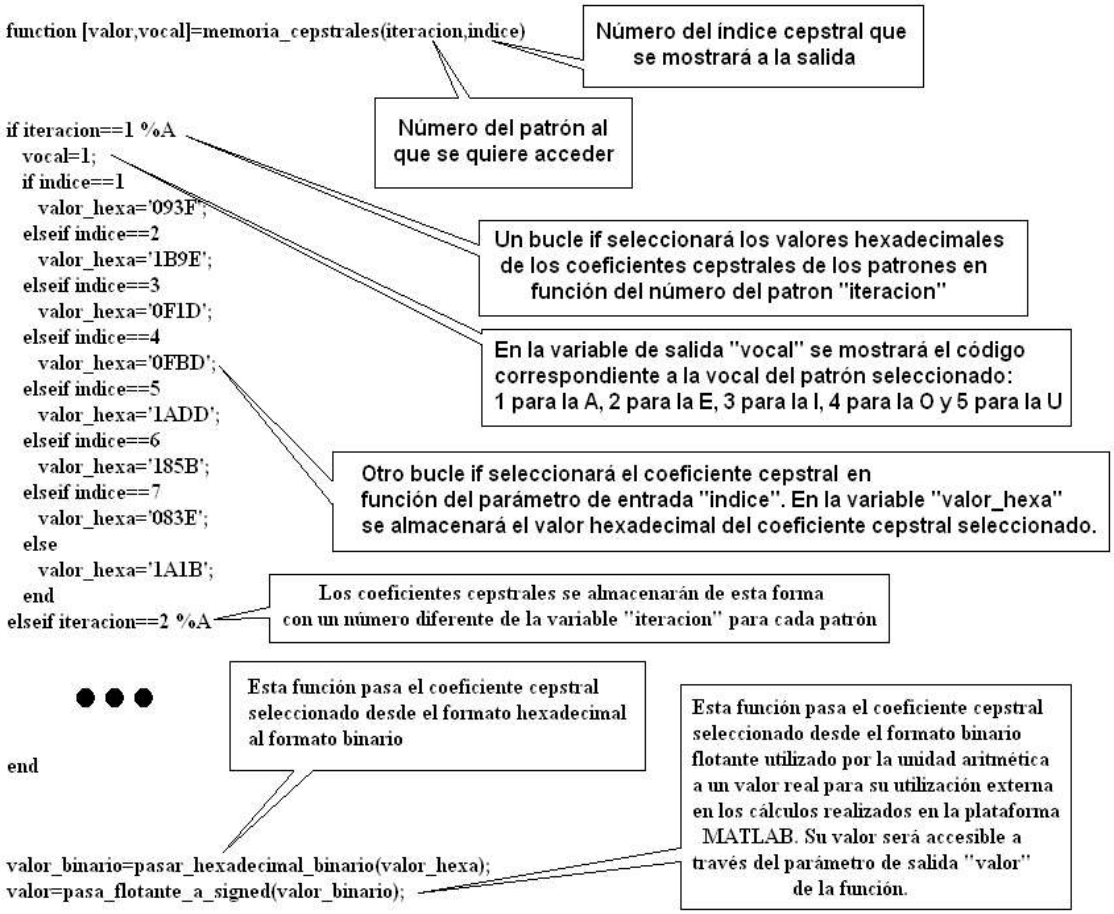


8. Volverá al paso 1 para iniciar de nuevo una nueva pronunciación vocálica.

El protocolo anterior puede implementarse con una modificación sencilla del diseño final en VHDL que calcula los coeficientes cepstrales en la FPGA. Tras poner en marcha el sistema pueden obtenerse tantos patrones vocálicos como se deseen. Estos serán los patrones que se utilizarán en el algoritmo de aprendizaje de la red neuronal. Como el algoritmo se ejecutará sobre la plataforma MATLAB®, es conveniente almacenar el código hexadecimal de los coeficientes cepstrales de las muestras tomadas en un formato adecuado para que los patrones sean accesibles por la función que desarrolla el algoritmo de aprendizaje. La opción adoptada es almacenar los coeficientes en la estructura de una función ("memoria_cepstrales") que proporcionará el valor del coeficiente cepstral indicado por el número del patrón utilizado en ese momento (valor del índice de entrada "iteracion") y por el índice actual del coeficiente cepstral (de 1 a 8) indicado en el parámetro de entrada "indice".

La función "memoria_cepstrales" almacena los coeficientes cepstrales de los patrones de aprendizaje de la red neuronal y proporciona los valores de los mismos al realizar una llamada sobre la misma con el número del patrón que se quiere leer en ese momento y el número del índice cepstral. El código MATLAB® de dicha función es el siguiente:

memoria_cepstrales.m



La función tiene dos parámetros de entrada: “iteracion”, que se corresponde con el número del patrón vocálico al que se quiere acceder; “indice”, que se corresponde con el número del índice cepstral del patrón correspondiente que se quiere leer en ese momento.

Un bucle “if” seleccionará los valores hexadecimales de los coeficientes cepstrales de los patrones en función del número del patrón “iteracion”. Para cada valor del parámetro “iteraccion” se mostrará en la variable de salida “vocal” el código correspondiente a la vocal del patrón seleccionado: 1 para la A, 2 para la E, 3 para la I, 4 para la O y 5 para la U. Para cada valor del parámetro “iteraccion”, otro bucle “if” seleccionará el coeficiente cepstral en función del parámetro de entrada “indice”. En la variable “valor_hexa” se almacenará el valor hexadecimal del coeficiente cepstral seleccionado. Cada patrón se añadirá mediante la inclusión de un nuevo

valor en la selección del bucle "if" incrementando el valor asociado a la igualdad con la variable "iteracion". De esta forma, se consigue almacenar los coeficientes cepstrales de los patrones que se deseen, a la vez que la función facilita el acceso a los mismos.

Como el parámetro cepstral se encuentra en el formato hexadecimal obtenido con la ejecución del diseño modificado de la parte del reconocedor que capta los coeficientes cepstrales, es preciso trasladar esa información a un valor apropiado en la simulación MATLAB® Por eso, en primer lugar, se realiza una conversión de la cadena de caracteres hexadecimal a una cadena de caracteres que muestre el coeficiente cepstral seleccionado en formato binario. La función "pasar_hexadecimal_binario" utilizada al final de la función "memoria_cepstrales" realiza esta operación.

pasar_hexadecimal_binario.m

```
function [valor_binario]=pasar_hexadecimal_binario(valor_hexadecimal)
%Convierte la cadena de caracteres de entrada, que se corresponde con un
%valor hexadecimal, a una cadena de caracteres que será mostrada a la salida
%con el valor binario correspondiente

n=size(valor_hexadecimal);
N=max(n);

hexadecimal=valor_hexadecimal(1);
switch hexadecimal
case 'F',
  binario='1111';
case 'E',
  binario='1110';
case 'D',
  binario='1101';
case 'C',
  binario='1100';
case 'B',
  binario='1011';
case 'A',
  binario='1010';
case '9',
  binario='1001';
case '8',
  binario='1000';
case '7',
  binario='0111';
case '6',
  binario='0110';
case '5',
  binario='0101';
case '4',
  binario='0100';
case '3',
  binario='0011';
```

pasar_hexadecimal_binario.m

```
case '2',
  binario='0010';
case '1',
  binario='0001';
case '0',
  binario='0000';
otherwise,
  error('error');
end
valor_binario=binario;
if N>=2
 for i=2:1:N
   hexadecimal=valor_hexadecimal(i);
   switch hexadecimal
   case 'F',
     binario='1111';
   case 'E',
     binario='1110';
   case 'D',
     binario='1101';
   case 'C',
     binario='1100';
   case 'B',
     binario='1011';
   case 'A',
     binario='1010';
   case '9',
     binario='1001';
   case '8',
     binario='1000';
   case '7',
     binario='0111';
   case '6',
     binario='0110';
   case '5',
     binario='0101';
   case '4',
     binario='0100';
   case '3',
     binario='0011';
   case '2',
     binario='0010';
   case '1',
     binario='0001';
   case '0',
     binario='0000';
   otherwise,
     error('error');
   end
   valor_binario=strcat(valor_binario,binario);
 end
end
```

El coeficiente cepstral obtenido se encuentra en el formato binario flotante utilizado por la unidad aritmética del reconocedor en el diseño VHDL. Se realiza una llamada a la función "pasa_flotante_a_signed" para convertir el coeficiente cepstral binario a un número real con el que poder operar. Este valor real será el valor del coeficiente cepstral seleccionado por el número del patrón y el número del índice cepstral (1-8, ya que son 8 coeficientes cepstrales por cada patrón), que se mostrará a la salida como valor resultado de la llamada a la función. De esta manera se accede a los coeficientes cepstrales de los patrones utilizados en el algoritmo de aprendizaje de la red neuronal.

pasa_flotante_a_signed.m

```
function [salida]=pasa_flotante_a_signed(entrada)
%La función "pasa_flotante_a_signed" calcula el valor real correspondiente a la variable binaria
%introducida en la entrada de la función. Este número binario representa a un número flotante en
%base dos. La variable de entrada será una cadena de 16 bits, donde los 3 bits más significativos
%no se utilizan, el cuarto bit más significativo indica el signo del número en cuestión ('0' si es
%positivo y '1' si es negativo), los siguientes 7 bits conforman el módulo de la mantisa y los
%últimos 5 bits el exponente entero en base 2 en complemento a dos. Así se conforma el número
%real que se mostrará a la salida, que se corresponde con el número binario de la entrada
%en el formato flotante utilizado por la unidad aritmética del diseño en VHDL .
%salida=  + ó - (según el signo aportado en la entrada) MÓDULO * 2^(EXPONENTE)

signo_mantisa=entrada(4);
for i=1:1:7
  modulo_mantisa(i)=entrada(i+4);
end
for i=1:1:5
  exponente_binario(i)=entrada(i+11);
end

if signo_mantisa=='1'
  signo=-1;
else
  signo=1;
end

%El módulo de la mantisa se encuentra en formato binario positivo, por lo que se realiza
%una llamada a la función "pasar_binario_modulo" para obtener su valor decimal positivo
modulo=pasar_binario_modulo(modulo_mantisa,7);
%El exponente de la base 2 del número flotante se encuentra en formato binario en complemento
%a dos, por lo que se realiza una llamada a la función "pasar_binario_entero" para obtener
%el valor del entero decimal correspondiente
exponente=pasar_binario_entero(exponente_binario,5);

salida=signo*modulo*(2^exponente);
```

En la función "pasa_flotante_a_signed" se ha realizado una llamada a la función "pasar_binario_modulo" para convertir un número binario positivo, asociado al módulo de la mantisa del coeficiente cepstral en formato binario flotante, al formato decimal.

pasar_binario_modulo.m

```
function [modulo]=pasar_binario_modulo(num_binario,dimension)
%Realiza la conversión del número binario introducido como parámetro de entrada
%desde el formato binario al formato decimal
%INTERFAZ:
%NUM_BINARIO: cadena de caracteres que contiene el número positivo en binario
%            que se pasará a formato decimal
%DIMENSION: número de bits del número en formato binario
%MODULO: formato decimal del número binario introducido a la entrada
%Ejemplo:
%La ejecución de la función: pasar_binario_modulo('00110110101',11)
%proporciona el número decimal: 437 que se corresponde con la representación decimal
%del número positivo binario de 11 bits '00110110101'

suma=0;
for i=1:1:dimension
  if num_binario(i)=='1'
    suma=suma+(2^(dimension-i));
  end
end
modulo=suma;
```

En la función "pasa_flotante_a_signed" se ha realizado una llamada a la función "pasar_binario_entero" para convertir un número binario en complemento a dos, asociado al exponente en base dos del coeficiente cepstral en formato binario flotante, al formato decimal.

pasar_binario_entero.m

```
function [numero]=pasar_binario_entero(num_binario,dimension)
%Realiza la conversión del número binario entero introducido como parámetro de entrada
%desde el formato binario al formato decimal.
%Si el bit más significativo es '0', el número es positivo y la representación binaria
%es el correspondiente valor del número
%Si el bit más significativo es '1', el número es negativo y la representación binaria
%se corresponde con el complemento a dos del módulo del número
%INTERFAZ:
%NUM_BINARIO: cadena de caracteres que contiene el número entero en binario
%            que se pasará a formato decimal
```

pasar_binario_entero.m

```
%DIMENSION: número de bits del número en formato binario
%NUMERO: formato decimal del número binario introducido a la entrada
%Ejemplo:
%La ejecución de la función: pasar_binario_entero('00110110101',11)
%Proporciona el número decimal: 437 que se corresponde con la representación decimal
%del número entero binario de 11 bits '00110110101'
%La ejecución de la función: pasar_binario_entero('10110110101',11)
%Proporciona el número decimal: -587 que se corresponde con el negado del número
%que resulta de sumarle una unidad al número que se obtiene de la correspondencia
%decimal del inverso (se invierte el valor de los bits) del valor binario de
%entrada: '01001001010'=486

%Si el número es negativo, se activa el flag y se invierte el valor de los bits
if num_binario(1)=='1'
  flag='1';
  for i=1:1:dimension
    if num_binario(i)=='1'
      num_binario(i)='0';
    else
      num_binario(i)='1';
    end
  end
else
  flag='0';
end

%Se calcula el valor decimal positivo correspondiente al número binario
suma=0;
for i=1:1:dimension
  if num_binario(i)=='1'
    suma=suma+(2^(dimension-i));
  end
end

%Si el número es negativo, se incrementa en uno el valor decimal positivo obtenido para
%luego cambiar el signo y obtener así el valor negativo correspondiente, si no, el número
%será el valor decimal positivo obtenido
if flag=='0'
  numero=suma;
else
  numero=suma+1;
  numero=-numero;
end
```

3.4.1 APRENDIZAJE DE LA RED NEURONAL EN MATLAB®

Una vez almacenados los parámetros cepstrales de los patrones que se utilizarán en el algoritmo de aprendizaje en el fichero "memoria_cepstrales.m", se procederá a la ejecución del algoritmo

de aprendizaje del mapa autoorganizado sobre la plataforma MATLAB® mediante la llamada al procedimiento "aprende_mapa".

```
%Llamada a la función "aprende_mapa" con la que se obtendrán los pesos sinápticos de las
%neuronas sintonizados para provocar una activación en las neuronas del mapa que permitan
%el reconocimiento de las vocales.
aprende_mapa
```

El procedimiento "aprende_mapa" realiza en primer lugar una lectura de todos los patrones de aprendizaje. Luego se inicia la primera fase del algoritmo de aprendizaje descrito en los apartados anteriores. En él, tras 100.000 iteraciones se actualizarán de forma progresiva los pesos sinápticos de las neuronas para así sintonizar de forma progresiva con los patrones vocálicos. El algoritmo de aprendizaje queda representado en los siguientes pasos:

1. En cada iteración, se presenta uno de los patrones tomados de forma aleatoria.

2. Cada neurona (i,j) en paralelo del mapa calcula la similitud entre su vector de pesos sinápticos $\mathbf{w_{ij}}$ y el vector de coeficientes cepstrales del patrón. El criterio de medida de similitud es la distancia euclídea:

$$d^2(\mathbf{w_{ij}},\mathbf{x}) = \Sigma_{k=1}^{n}(w_{ijk}-x_k)^2$$

3. Se determina la **neurona ganadora g** = (g1,g2), cuya distancia sea la menor de todas.

4. Se actualizan los pesos sinápticos de la neurona ganadora **g** = (g1,g2), y los de sus neuronas vecinas. La regla empleada es:

$$w_{ijk}(t+1) = w_{ijk}(t) + \alpha(t) * h(|i-g|,t) * (x_k(t) - w_{ijk}(t))$$

   Donde $\alpha(t)$ es un parámetro denominado **ritmo de aprendizaje**. La función $h(|i-g|,t)$ se denomina **función de vecindad**, puesto que establece qué neuronas son las vecinas a la actualmente ganadora. Esta función depende de la distancia entre la neurona **i** y la ganadora **g**, valiendo cero cuando **i** no pertenece a la vecindad de **g** (con lo que sus pesos no son actualizados), y uno cuando sí pertenece (sus pesos sí son modificados). La vecindad es un conjunto de neuronas centrado en la ganadora. Tanto α como el radio de vecindad disminuyen monótonamente con t. El ritmo de aprendizaje disminuye monótonamente con el número de iteración:

$$\alpha(t) = \alpha_0 + (\alpha_f - \alpha_0)\, t / t_f$$

α_0: 0.618 α_f: 0.00618 t: iteración actual t_f: 100.000

El radio de vecindad disminuye también monótonamente con el número de iteración:

$$R(t) = R_0 + (R_f - R_0)\, t / t_f$$

R_0: 6 R_f: 1 t: iteración actual t_f: 100.000

5. Si se ha alcanzado el número máximo de iteraciones establecido, entonces el proceso de aprendizaje finaliza. En caso contrario, se vuelve al paso 1.

Luego se inicializará la fase segunda del algoritmo de aprendizaje, con la ejecución de los mismos pasos en el algoritmo, pero donde el ritmo de aprendizaje es constante e igual a 0.00618 y el radio de vecindad es constante e igual a 1. Tras 100.000 iteraciones se obtendrán unos pesos sinápticos que serán almacenados para su uso posterior en los siguientes ficheros:

- **"peso1_8x4.dav"**: donde se almacenarán los pesos sinápticos de las 32 neuronas asociados con el primer coeficiente cepstral.
- **"peso2_8x4.dav"**: donde se almacenarán los pesos sinápticos de las 32 neuronas asociados con el segundo coeficiente cepstral.
- **"peso3_8x4.dav"**: donde se almacenarán los pesos sinápticos de las 32 neuronas asociados con el tercer coeficiente cepstral.
- **"peso4_8x4.dav**": donde se almacenarán los pesos sinápticos de las 32 neuronas asociados con el cuarto coeficiente cepstral.
- **"peso5_8x4.dav"**: donde se almacenarán los pesos sinápticos de las 32 neuronas asociados con el quinto coeficiente cepstral.
- **"peso6_8x4.dav"**: donde se almacenarán los pesos sinápticos de las 32 neuronas asociados con el sexto coeficiente cepstral.
- **"peso7_8x4.dav"**: donde se almacenarán los pesos sinápticos de las 32 neuronas asociados con el séptimo coeficiente cepstral.
- **"peso8_8x4.dav**": donde se almacenarán los pesos sinápticos de las 32 neuronas asociados con el octavo coeficiente cepstral.

Estos pesos sinápticos se utilizarán en el proceso de ejecución de la red neuronal, para generar el mapeado neurona-vocal según la neurona activada al presentarse los patrones de las distintas vocales.

El código MATLAB® del procedimiento "aprende_mapa" constituye el núcleo del proceso de aprendizaje de la red neuronal.

aprende_mapa.m

```
function aprende_mapa()
%función de aprendizaje de un mapa autoorganizado

%definimos los parámetros a utilizar
num_tomas=160; %número de tomas realizadas, cada toma se corresponde con
                    %un patrón de aprendizaje
num_par=8; %sólo utilizamos 8 coeficientes cepstrales
nx=8; %dimensión x del mapa autoorganizado
ny=4; %dimensión y del mapa autoorganizado

%LEEMOS LOS COEFICIENTES CEPSTRALES
operacion='LECTURA DE DATOS';
%recorremos las tomas de las vocales
%e inicializamos una matriz que guarda los
%valores cepstrales de cada uno de los patrones
matriz_cepstrales=0;
cuenta=1;
[t_inicio,itera]=iniciotiempo(operacion);
for toma=1:1:num_tomas
  for i=1:1:num_par
    [valor,num_vocal]=memoria_cepstrales(toma,i);
    matriz_cepstrales(cuenta,i)=valor;
  end
  matriz_cepstrales(cuenta,num_par+1)=num_vocal;
  cuenta=cuenta+1;
  itera=fasetiempo(t_inicio,itera,num_tomas,operacion);
end

%*****************************************************************
%************** ALGORITMO DE APRENDIZAJE ********
%*****************************************************************

%POSEE 2 FASES
%  FASE 1/2
%  FASE 2/2


%Entre 20.000 y 100.000 iteraciones representan cifras habituales
%en la simulación por ordenador del entrenamiento de un mapa
%autoorganizado.
cuenta_final=100000;
```

aprende_mapa.m

```
%pasos es el número de iteraciones que pasan hasta que se muestra
%el tiempo que queda
pasos=1000;

%1->Inicialización de pesos
%aleatorios de pequeño valor absoluto
disp('Inicialización de los pesos');
for i=1:1:nx
  for j=1:1:ny
    peso1(i,j)=0.0001*rand(1);
    peso2(i,j)=0.0001*rand(1);
    peso3(i,j)=0.0001*rand(1);
    peso4(i,j)=0.0001*rand(1);
    peso5(i,j)=0.0001*rand(1);
    peso6(i,j)=0.0001*rand(1);
    peso7(i,j)=0.0001*rand(1);
    peso8(i,j)=0.0001*rand(1);
  end
end

operacion='FASE 1/2 DEL APRENDIZAJE EN EL MAPA AUTOORGANIZADO';
%Se realiza a continuación la primera fase en el aprendizaje,
%en la que se produce el ajuste global del mapa, de modo que la
%distribución de los pesos sinápticos se distribuye entre neuronas
%cercanas. El ritmo de aprendizaje a(t) es alto al principio
%disminuyendo con el tiempo, al igual que el radio de vecindad

%variables para mostrar el tiempo que me queda
iteratotal=ceil(cuenta_final/pasos);
[t_inicio,itera]=iniciotiempo(operacion);

for cuenta=1:1:cuenta_final
  %2->En cada iteración, presentación de un patrón x(t) tomado
  %de acuerdo con la función de distribución p(x) del espacio
  %sensorial de entrada (en la muy habitual situación de disponer
  %solamente de un conjunto finito de patrones de entrenamiento,
  %basta con tomar al azar uno de ellos y presentarlo a la red)
  indice=ceil(num_tomas*rand(1));
  while indice~=11
    indice=ceil(num_tomas*rand(1));
  end
  vector=0;
  for k=1:1:num_par
    vector(k)=matriz_cepstrales(indice,k);
  end
  vocal=matriz_cepstrales(indice,num_par+1);
  %3->Cada neurona (i,j) en paralelo del mapa calcula
  %la similitud entre su vector de pesos sinápticos w ij
  %y el actual vector de entradas x. Un criterio de medida de
  %similitud muy utilizado es la distancia euclídea:
```

aprende_mapa.m

```
%d2 (w ij ,x) = suma para todo k de (w ijk -x k )2
for i=1:1:nx
  for j=1:1:ny
    suma=0;
    suma=suma+((peso1(i,j)-vector(1))^2);
    suma=suma+((peso2(i,j)-vector(2))^2);
    suma=suma+((peso3(i,j)-vector(3))^2);
    suma=suma+((peso4(i,j)-vector(4))^2);
    suma=suma+((peso5(i,j)-vector(5))^2);
    suma=suma+((peso6(i,j)-vector(6))^2);
    suma=suma+((peso7(i,j)-vector(7))^2);
    suma=suma+((peso8(i,j)-vector(8))^2);
    dist(i,j)=suma;
  end
end
%4->Determinación de la neurona ganadora (g1,g2),
%cuya distancia sea la menor de todas.
minimo=inf;
for i=1:1:nx
  for j=1:1:ny
    if dist(i,j)<minimo
      minimo=dist(i,j);
      g1=i;
      g2=j;
    end
  end
end
%5->Actualización de los pesos sinápticos de la neurona ganadora
%(g1,g2), y los de sus neuronas vecinas. La regla empleada es:
%w ijk (t+1) = w ijk (t) + a(t) * h(|i-g|,t) * (x k (t) - w ijk (t))
%Donde a(t) es un parámetro denominado ritmo de aprendizaje.
%La función h(x) se denomina función de vecindad, puesto que
%establece qué neuronas son las vecinas a la actualmente ganadora.
%Esta función depende de la distancia entre la neurona i y la
%ganadora g, valiendo cero cuando i no pertenece a la vecindad de
%g (con lo que sus pesos no son actualizados), y un número
%positivo cuando sí pertenece (sus pesos sí son modificados).
%La vecindad es un conjunto de neuronas centrado en la ganadora.
%Asumimos que pertenece a la vecindad cuando h(|i-g|,t)=1, lo
%que ocurrirá cuando la distancia a la neurona ganadora sea menor
%que el radio
%Tanto a(t) como el radio de vecindad disminuyen monótonamente con t.
%a(t) es una función monótonamente decreciente con el tiempo,
%siendo habitual su actualización mediante una función lineal.
%a(t) = a o  + (a f  - a o ) t / tf
alfa=0.618+((0.00618-0.618)*cuenta/cuenta_final);
%Una posible función de actualización de R(t) es la siguiente,
%linealmente decreciente, donde t es la iteración y t f  el
%número de iteraciones para alcanzar R f :
%R(t) = R o  + (R f  - R o ) t / tf
radio=6+((1-6)*cuenta/cuenta_final);
for i=1:1:nx
```

aprende_mapa.m

```
    for j=1:1:ny
      modulo=sqrt((i-g1)^2+(j-g2)^2);
      if modulo<=radio
         peso1(i,j)=peso1(i,j)+(alfa*(vector(1)-peso1(i,j)));
         peso2(i,j)=peso2(i,j)+(alfa*(vector(2)-peso2(i,j)));
         peso3(i,j)=peso3(i,j)+(alfa*(vector(3)-peso3(i,j)));
         peso4(i,j)=peso4(i,j)+(alfa*(vector(4)-peso4(i,j)));
         peso5(i,j)=peso5(i,j)+(alfa*(vector(5)-peso5(i,j)));
         peso6(i,j)=peso6(i,j)+(alfa*(vector(6)-peso6(i,j)));
         peso7(i,j)=peso7(i,j)+(alfa*(vector(7)-peso7(i,j)));
         peso8(i,j)=peso8(i,j)+(alfa*(vector(8)-peso8(i,j)));
      end
    end
  end
  if cuenta==itera*pasos;
    itera=fasetiempo(t_inicio,itera,iteratotal,operacion);
  end
end

operacion='FASE 2/2 DEL APRENDIZAJE EN EL MAPA AUTOORGANIZADO';
%Se realiza a continuación una segunda fase en el aprendizaje,
%en la que se produce el ajuste fino del mapa, de modo que la
%distribución de los pesos sinápticos se ajuste más a la de las
%entradas. El proceso es similar al anterior, tomando  a(t)
%constante e igual a un pequeño valor 0.00618, y
%radio de vecindad constante e igual a uno.
alfa=0.00618;
radio=1;

%variables para mostrar el tiempo que me queda
iteratotal=ceil(cuenta_final/pasos);
[t_inicio,itera]=iniciotiempo(operacion);

for cuenta=1:1:cuenta_final
  %2->En cada iteración, presentación de un patrón x(t) tomado
  %de acuerdo con la función de distribución p(x) del espacio
  %sensorial de entrada (en la muy habitual situación de disponer
  %solamente de un conjunto finito de patrones de entrenamiento,
  %basta con tomar al azar uno de ellos y presentarlo a la red
  indice=ceil(num_tomas*rand(1));
  for k=1:1:num_par
    vector(k)=matriz_cepstrales(indice,k);
  end
  vocal=matriz_cepstrales(indice,num_par+1);
  %3->Cada neurona (i,j) en paralelo del mapa calcula
  %la similitud entre su vector de pesos sinápticos w ij
  %y el actual vector de entradas x. Un criterio de medida de
  %similitud muy utilizado es la distancia euclídea:
  %d2 (w ij ,x) = suma para todo k de (w ijk -x k )2
  for i=1:1:nx
    for j=1:1:ny
```

aprende_mapa.m

```
    suma=0;
    suma=suma+((peso1(i,j)-vector(1))^2);
    suma=suma+((peso2(i,j)-vector(2))^2);
    suma=suma+((peso3(i,j)-vector(3))^2);
    suma=suma+((peso4(i,j)-vector(4))^2);
    suma=suma+((peso5(i,j)-vector(5))^2);
    suma=suma+((peso6(i,j)-vector(6))^2);
    suma=suma+((peso7(i,j)-vector(7))^2);
    suma=suma+((peso8(i,j)-vector(8))^2);
    dist(i,j)=suma;
  end
end
%4->Determinación de la neurona ganadora (g1,g2),
%cuya distancia sea la menor de todas.
minimo=inf;
for i=1:1:nx
  for j=1:1:ny
    if dist(i,j)<minimo
      minimo=dist(i,j);
      g1=i;
      g2=j;
    end
  end
end
%5->Actualización de los pesos sinápticos de la neurona ganadora
%(g1,g2), y los de sus neuronas vecinas. La regla empleada es:
%w ijk (t+1) = w ijk (t) + a(t) * h(|i-g|,t) * (x k (t) - w ijk (t))
%Donde a(t) es un parámetro denominado ritmo de aprendizaje.
%tomando  a(t)
%constante e igual a un pequeño valor 0.00618, y
%La función h(x) se denomina función de vecindad, puesto que
%establece qué neuronas son las vecinas a la actualmente ganadora.
%Esta función depende de la distancia entre la neurona i y la
%ganadora g, valiendo cero cuando i no pertenece a la vecindad de
%g (con lo que sus pesos no son actualizados), y un número
%positivo cuando sí pertenece (sus pesos sí son modificados).
%La vecindad es un conjunto de neuronas centrado en la ganadora.
%radio de vecindad constante e igual a uno.
for i=1:1:nx
  for j=1:1:ny
    modulo=sqrt((i-g1)^2+(j-g2)^2);
    if modulo<=radio
      peso1(i,j)=peso1(i,j)+(alfa*(vector(1)-peso1(i,j)));
      peso2(i,j)=peso2(i,j)+(alfa*(vector(2)-peso2(i,j)));
      peso3(i,j)=peso3(i,j)+(alfa*(vector(3)-peso3(i,j)));
      peso4(i,j)=peso4(i,j)+(alfa*(vector(4)-peso4(i,j)));
      peso5(i,j)=peso5(i,j)+(alfa*(vector(5)-peso5(i,j)));
      peso6(i,j)=peso6(i,j)+(alfa*(vector(6)-peso6(i,j)));
      peso7(i,j)=peso7(i,j)+(alfa*(vector(7)-peso7(i,j)));
      peso8(i,j)=peso8(i,j)+(alfa*(vector(8)-peso8(i,j)));
```

aprende_mapa.m

```
      end
    end
  end
  if cuenta==itera*pasos;
    itera=fasetiempo(t_inicio,itera,iteratotal,operacion);
  end
end

disp('Escribiendo en ficheros los resultados...')
fichero=sprintf('peso1_%dx%d',nx,ny);
escribe(fichero,peso1);
fichero=sprintf('peso2_%dx%d',nx,ny);
escribe(fichero,peso2);
fichero=sprintf('peso3_%dx%d',nx,ny);
escribe(fichero,peso3);
fichero=sprintf('peso4_%dx%d',nx,ny);
escribe(fichero,peso4);
fichero=sprintf('peso5_%dx%d',nx,ny);
escribe(fichero,peso5);
fichero=sprintf('peso6_%dx%d',nx,ny);
escribe(fichero,peso6);
fichero=sprintf('peso7_%dx%d',nx,ny);
escribe(fichero,peso7);
fichero=sprintf('peso8_%dx%d',nx,ny);
escribe(fichero,peso8);
```

Resulta útil informar del tiempo restante en el proceso para evitar que el programador desespere en la espera de la conclusión de los bucles del algoritmo. Dos funciones se utilizan a tal fin: La función "inicio_tiempo", ejecutada al inicio de cada bucle para activar la cuenta temporal que se utilizará en la medida de los tiempos restantes; y la función "fasetiempo", que calcula de forma aproximada el tiempo restante hasta la conclusión del bucle actual.

inicio_tiempo.m

```
function [t_inicio,itera]=iniciotiempo(texto)
% función destinada a mostrar el tiempo de proceso que me queda
% con esta función inicializamos el reloj que cuenta el tiempo empleado

disp(texto)
t_inicio=cputime;
itera=1;
```

fasetiempo.m

```
function [itera]=fasetiempo(t_inicio,itera,iteratotal,operacion)
% función destinada a mostrar el tiempo de proceso que me queda
% contamos el tiempo que ha transcurrido desde el inicio del
% proceso y estimamos el tiempo que nos queda

texto=sprintf('%s',operacion);
disp(texto)
texto=sprintf('   Acabada la fase %d de %d',itera,iteratotal);
disp(texto)
disp('   Tiempo estimado para terminar:')

taux=(cputime-t_inicio)/itera;
taux=taux*(iteratotal-itera);
dias=floor(taux/(24*60*60));
taux=taux-(dias*24*60*60);
horas=floor(taux/(60*60));
taux=taux-(horas*60*60);
minutos=floor(taux/60);
taux=taux-(minutos*60);
segundos=floor(taux);

texto_1=sprintf('%d dias, %d horas',dias,horas);
texto_2=sprintf('%d minutos, %d segundos',minutos,segundos);
texto=sprintf('      %s, %s',texto_1,texto_2);
disp(texto)
disp(' ')
disp(' ')

itera=itera+1;
```

La ejecución de estas funciones mostraría en pantalla un texto similar al siguiente:

```
FASE 1/2 DEL APRENDIZAJE EN EL MAPA AUTOORGANIZADO
  Acabada la fase 6 de 100
  Tiempo estimado para terminar:
    0 dias, 0 horas, 3 minutos, 0 segundos
```

Como los pesos sinápticos calculados en el algoritmo de aprendizaje serán utilizados en la fase de ejecución de la red neuronal, es necesario un almacenamiento a largo plazo de los mismos y la forma más sencilla de lograrlo es guardar los valores de los pesos sinápticos en archivos que

podrán ser leídos en el momento en el que se desee. La función "escribe" creará los archivos donde se almacenarán los valores de los pesos sinápticos de la red neuronal calculados por el algoritmo de aprendizaje. Esta función es utilizada en el procedimiento de aprendizaje ejecutado sobre la plataforma MATLAB® para almacenar los pesos finales obtenidos. En la llamada a la función se proporciona como parámetros de entrada el nombre del fichero donde se almacenarán los pesos y un vector con los pesos de todas las neuronas asociadas a un coeficiente cepstral (en cada fichero se almacenarán los pesos de todas las neuronas asociadas a cada uno de los coeficientes cepstrales, por lo que se crearán ocho ficheros para almacenar todos los pesos de la red neuronal). Todos los ficheros tendrán la extensión ".dav".

escribe.m

```
function escribe(fichero,datos)
%escribe(fichero,datos)
%función que escribe datos en un fichero de extensión .dav
%fichero es el nombre del fichero sin la extensión

preci='float32';
fichero=strcat(fichero,'.dav');

fid=fopen(fichero,'w');
count=fwrite(fid,datos,preci);
fclose(fid);
```

3.4.2 RECUERDO POR LA RED NEURONAL EN MATLAB®

Ya han sido establecidos los pesos sinápticos de la red neuronal, sus valores almacenados en unos ficheros para permitir su acceso en el momento en el que se soliciten, ahora puede ponerse en funcionamiento la red neuronal y aprovechar sus características de sintonización con los patrones para reconocer la vocal pronunciada. Se utiliza un emulador de la red neuronal ejecutado en MATLAB® para ver la bondad de la solución encontrada. Mediante la llamada a la función "recuerda_mapa", que toma como entrada un vector de coeficientes cepstrales, calcula la neurona que se activará por sintonizar mejor con los coeficientes cepstrales presentados. La función devolverá el índice de la fila y de la columna de la neurona activada. Los parámetros de la red neuronal son los pesos almacenados en los ficheros creados en la fase de aprendizaje, por lo que se realizará una lectura de los mismos para la puesta a punto de la red neuronal. Durante la ejecución del algoritmo de recuerdo, cada neurona (i,j) calcula la similitud entre el vector de coeficientes cepstrales de entrada y su propio vector de pesos sinápticos $\mathbf{w_{ij}}$, según el criterio de distancia euclídea.

$$d^2(\mathbf{w_{ij}},\mathbf{cepstrales}) = \Sigma_{k=1}^{n}(w_{ijk}\text{-}cepstral_k)^2$$

A continuación, se declara vencedora la neurona **g** = (g1,g2), cuyo vector de pesos $\mathbf{w}_g$ es más similar al de entradas. De esta manera, cada neurona actúa como un detector de rasgos específicos, y la neurona ganadora nos indica el tipo de rasgo o patrón detectado en el vector de entradas.

En el apartado anterior de simulación MATLAB® se calcularon los ocho primeros coeficientes cepstrales como parámetros significativos de la señal de audio pronunciada. Estos coeficientes serán los que se utilicen en el proceso de reconocimiento para detectar la vocal pronunciada. La red neuronal en la fase de recuerdo se activará con la ejecución sobre la plataforma MATLAB® del siguiente código:

```
%La función "recuerda_mapa" inicia el proceso de recuerdo de la red neuronal donde los
%estímulos son el vector de coeficientes cepstrales pasado a la entrada de la función
%como parámetro. La función devuelve el número de la fila y de la columna de la neurona
%que se activará tras el proceso de recuerdo
[num_fila,num_columna]=recuerda_mapa(cepstrales);
```

Se ha realizado una llamada a la función “recuerda_mapa” cuya ejecución representa al proceso de recuerdo realizado en la red neuronal. Los estímulos de la red neuronal son los coeficientes cepstrales de la señal de audio captada que excitarán a las neuronas para activar aquella que sintonice mejor con el patrón detectado.

recuerda_mapa.m

```
function [num_fila,num_columna]=recuerda_mapa(cepstrales)
%La función “recuerda_mapa” toma como entrada un vector de coeficientes cepstrales,
%calcula la neurona que se activará por sintonizar mejor con los coeficientes cepstrales
%presentados. La función devolverá el índice de la fila y de la columna de la neurona activada.

close all;

%definimos los parámetros a utilizar
num_par=8; %solo utilizamos 8 coeficientes cepstrales
nx=8; %dimensión x del mapa autoorganizado
ny=4; %dimensión y del mapa autoorganizado

%LEEMOS LOS PESOS
disp('Leemos los pesos de la red neuronal...')
fichero=sprintf('peso1_%dx%d',nx,ny);
dato1=lee(fichero,nx*ny);
fichero=sprintf('peso2_%dx%d',nx,ny);
dato2=lee(fichero,nx*ny);
fichero=sprintf('peso3_%dx%d',nx,ny);
dato3=lee(fichero,nx*ny);
fichero=sprintf('peso4_%dx%d',nx,ny);
```

recuerda_mapa.m

```
dato4=lee(fichero,nx*ny);
fichero=sprintf('peso5_%dx%d',nx,ny);
dato5=lee(fichero,nx*ny);
fichero=sprintf('peso6_%dx%d',nx,ny);
dato6=lee(fichero,nx*ny);
fichero=sprintf('peso7_%dx%d',nx,ny);
dato7=lee(fichero,nx*ny);
fichero=sprintf('peso8_%dx%d',nx,ny);
dato8=lee(fichero,nx*ny);
for i=1:1:nx
  for j=1:1:ny
    peso1(i,j)=dato1((i-1)*ny+j);
    peso2(i,j)=dato2((i-1)*ny+j);
    peso3(i,j)=dato3((i-1)*ny+j);
    peso4(i,j)=dato4((i-1)*ny+j);
    peso5(i,j)=dato5((i-1)*ny+j);
    peso6(i,j)=dato6((i-1)*ny+j);
    peso7(i,j)=dato7((i-1)*ny+j);
    peso8(i,j)=dato8((i-1)*ny+j);
  end
end


%*****************************************************************
%********************** RECORDAMOS ***********************
%*****************************************************************

disp('Recordando...')
for k=1:1:num_par
  vector(k)=cepstrales(k);
end
%para cada neurona (i,j) en paralelo del mapa calculo
%la similitud entre su vector de pesos sinápticos w ij
%y el actual vector de entradas x. Un criterio de medida de
%similitud muy utilizado es la distancia euclídea:
%d2 (w ij ,x) = suma para todo k de (w ijk -x k )2
for i=1:1:nx
  for j=1:1:ny
    suma=0;
    suma=suma+((peso1(i,j)-vector(1))^2);
    suma=suma+((peso2(i,j)-vector(2))^2);
    suma=suma+((peso3(i,j)-vector(3))^2);
    suma=suma+((peso4(i,j)-vector(4))^2);
    suma=suma+((peso5(i,j)-vector(5))^2);
    suma=suma+((peso6(i,j)-vector(6))^2);
    suma=suma+((peso7(i,j)-vector(7))^2);
    suma=suma+((peso8(i,j)-vector(8))^2);
    dist(i,j)=suma;
  end
end
```

recuerda_mapa.m

```
%Determino la neurona ganadora (g1,g2),
%cuya distancia sea la menor de todas.
minimo=inf;
for i=1:1:nx
  for j=1:1:ny
    if dist(i,j)<minimo
      minimo=dist(i,j);
      g1=i;
      g2=j;
    end
  end
end

num_fila=g1;
num_columna=g2;
```

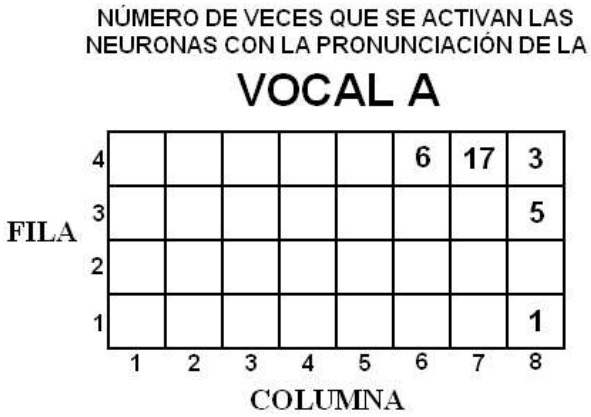
Para realizar la lectura de los pesos sinápticos de la red neuronal almacenados en los ficheros con extensión ".dav" se han realizado llamadas a la función "lee". Esta función extrae los 32 pesos sinápticos correspondientes a uno de los coeficientes cepstrales que fueron almacenados en la fase de aprendizaje. Para cada coeficiente cepstral existe un fichero donde se encuentran almacenados los pesos que relacionan a ese coeficiente cepstral con las 32 neuronas. El nombre del fichero es proporcionado como parámetro de entrada en la llamada a la función.

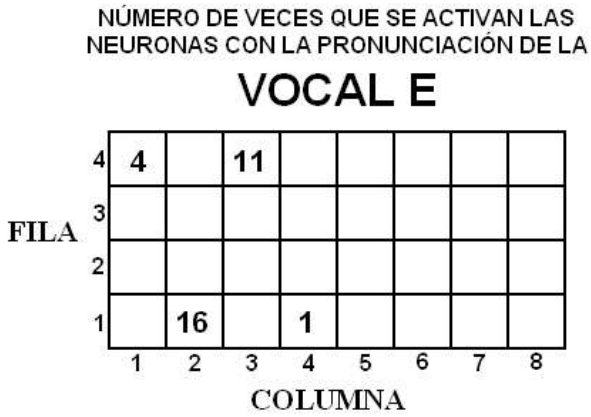
lee.m

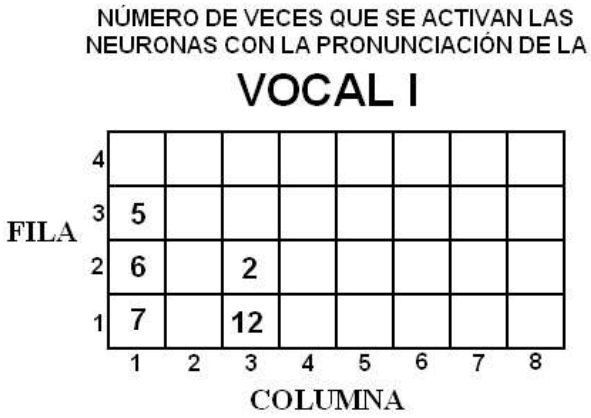
```
function datos=lee(fichero,numd)
%datos=lee(fichero,numd)
%este fichero lee datos de un fichero de extensión .dav
%fichero es el nombre del fichero sin la extensión
%numd es el número de datos a leer


preci='float32';
fichero=strcat(fichero,'.dav');

fid=fopen(fichero,'r');
x=fread(fid,numd,preci);
for i=1:1:numd
  datos(1,i)=x(i);
end

fclose(fid);
```

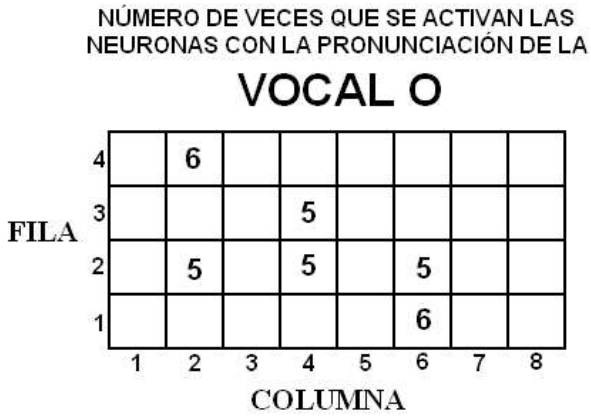
3.4.3 MAPEADO FINAL

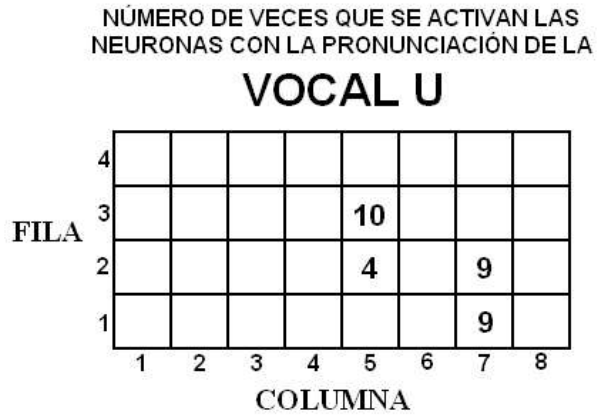
Aplicando el proceso de recuerdo a todos los patrones utilizados en el algoritmo de aprendizaje, podemos saber cuales son las neuronas que se activan con la pronunciación de las distintas vocales. Así, tras ejecutar sobre la plataforma MATLAB® la función de recuerdo de la red neuronal "recuerda_mapa" con los coeficientes cepstrales de todos los patrones utilizados en el aprendizaje de la red neuronal, puedo llevar la cuenta del número de veces que se activan las neuronas con la pronunciación de las distintas vocales. Las siguientes gráficas resumen, como resultado de la ejecución de la fase de recuerdo de la red neuronal, el número de activaciones producidas por neurona y por vocal.



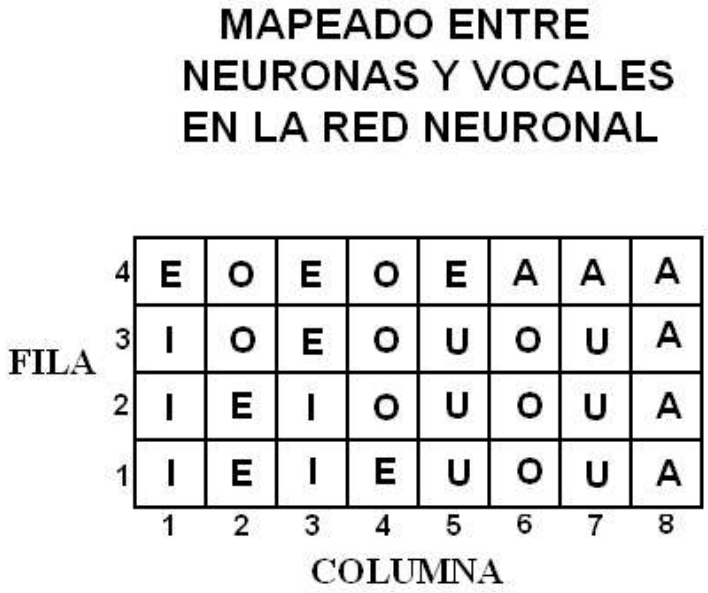








Se observa una distribución topológica de los rasgos característicos de las distintas vocales en el mapa autoorganizado. A cada neurona se le asignará la vocal que más activaciones tenga provocadas. En el caso de que una neurona no se active con ninguno de los patrones, se asignará la vocal correspondiente al patrón que más sintonice con la neurona. De esta forma se realizará un mapeado entre la posición de la neurona y la vocal que se detectará en el caso de que se active la neurona. De acuerdo a la regla anterior, la siguiente gráfica resume la correspondencia entre neuronas y vocales. Si quisiera simular el reconocimiento de la vocal en la plataforma MATLAB® realizaría una llamada a la función "reconoce_mapa", que devuelve el código de la vocal correspondiente a la neurona activada en el proceso de recuerdo de la red neuronal.



%La función “reconoce_mapa” detecta la vocal pronunciada en función de la neurona activada en
%el proceso de recuerdo de la red neuronal
[num_vocal]=reconoce_mapa(num_fila,num_columna)

Con la llamada a la función “reconoce_mapa” se propone como vocal reconocida la vocal asignada a la neurona que se ha activado en el proceso de recuerdo de la red neuronal. Esta función recibe como parámetros de entrada el número de la fila y de la columna correspondientes a la neurona que ha sintonizado mejor con los coeficientes cepstrales de la señal de audio captada. Devolverá el código de la vocal reconocida según la asignación neurona-vocal establecida en el mapeado. Devolverá un 1 cuando se reconozca la vocal 'A', un 2 cuando se reconozca la vocal 'E', un 3 cuando se reconozca la vocal 'I', un 4 cuando se reconozca la vocal 'O' y un 5 cuando se reconozca la vocal 'U'. El código de la función MATLAB® que realiza el mapeado entre las neuronas y las vocales es el siguiente:

reconoce_mapa.m

```
function [num_vocal]=reconoce_mapa(num_fila,num_columna)
%Esta función reconoce la vocal pronunciada según la neurona activada
%La asignación entre neurona-vocal se ha realizado de tal forma que
%se minimicen los errores. La función devolverá el código de la vocal
%reconocida: 1 para la 'A', 2 para la 'E', 3 para la 'I', 4 para la 'O'
%y 5 para la 'U'.

if num_fila==1
  if num_columna==1
    num_vocal=3;
  elseif num_columna==2
    num_vocal=4;
  elseif num_columna==3
    num_vocal=3;
  elseif num_columna==4
    num_vocal=4;
  elseif num_columna==5
    num_vocal=5;
  elseif num_columna==6
    num_vocal=4;
  elseif num_columna==7
    num_vocal=5;
  else
    num_vocal=2;
  end
elseif num_fila==2
  if num_columna==1
    num_vocal=3;
  elseif num_columna==2
    num_vocal=1;
```

reconoce_mapa.m

```
  elseif num_columna==3
    num_vocal=3;
  elseif num_columna==4
    num_vocal=4;
  elseif num_columna==5
    num_vocal=3;
  elseif num_columna==6
    num_vocal=4;
  elseif num_columna==7
    num_vocal=5;
  else
    num_vocal=2;
  end
elseif num_fila==3
  if num_columna==1
    num_vocal=3;
  elseif num_columna==2
    num_vocal=1;
  elseif num_columna==3
    num_vocal=3;
  elseif num_columna==4
    num_vocal=1;
  elseif num_columna==5
    num_vocal=5;
  elseif num_columna==6
    num_vocal=4;
  elseif num_columna==7
    num_vocal=5;
  else
    num_vocal=2;
  end
else
  if num_columna==1
    num_vocal=4;
  elseif num_columna==2
    num_vocal=1;
  elseif num_columna==3
    num_vocal=5;
  elseif num_columna==4
    num_vocal=1;
  elseif num_columna==5
    num_vocal=5;
  elseif num_columna==6
    num_vocal=1;
  elseif num_columna==7
    num_vocal=5;
  else
    num_vocal=2;
  end
end
```

La función "reconoce_mapa" realiza el proceso final de reconocimiento basado en la aplicación de una red neuronal sobre los coeficientes cepstrales de 512 muestras de una señal de audio. Así, es hora ya de poner a prueba el sistema global de reconocimiento con la presentación al conjunto de diversas pronunciaciones vocálicas para ponderar la bondad de la solución adoptada. Tómense los siguientes datos como los resultados finales en el procedimiento de simulación en la plataforma MATLAB®. En primer lugar se muestra el porcentaje de error obtenido en el proceso de reconocimiento de los 160 patrones vocálicos diferentes utilizados en el algoritmo de aprendizaje de la red neuronal. En segundo lugar se muestra el porcentaje de error obtenido en el proceso de reconocimiento de 32 pronunciaciones vocálicas diferentes a las utilizadas como patrones en el algoritmo de aprendizaje de la red neuronal. Los resultados obtenidos confirman como viable la solución adoptada en el mecanismo de reconocimiento vocálico que se implementará en el diseño VHDL del prototipo de ratón PS/2 controlado por la voz.

Porcentaje de aciertos del sistema reconocedor basado en la red neuronal implementada en el proyecto tras la presentación al mismo de los 160 patrones de pronunciaciones vocálicas diferentes utilizados en el proceso de aprendizaje de la red neuronal.

PRONUNCIACIÓN DE LA VOCAL A	**100 % de aciertos**
PRONUNCIACIÓN DE LA VOCAL E	**100 % de aciertos**
PRONUNCIACIÓN DE LA VOCAL I	**100 % de aciertos**
PRONUNCIACIÓN DE LA VOCAL O	**100 % de aciertos**
PRONUNCIACIÓN DE LA VOCAL U	**100 % de aciertos**

Porcentaje de aciertos del sistema reconocedor basado en la red neuronal implementada en el proyecto tras la presentación al mismo de 32 pronunciaciones vocálicas diferentes a los patrones utilizados en el proceso de aprendizaje de la red neuronal.

PRONUNCIACIÓN DE LA VOCAL A	**100 % de aciertos**
PRONUNCIACIÓN DE LA VOCAL E	**100 % de aciertos**
PRONUNCIACIÓN DE LA VOCAL I	**100 % de aciertos**
PRONUNCIACIÓN DE LA VOCAL O	**100 % de aciertos**
PRONUNCIACIÓN DE LA VOCAL U	**100 % de aciertos**