Universidad del País Vasco
Euskal Herriko Unibertsitatea

ZTF-FCT
Zientzia eta Teknologia Fakultatea
Facultad de Ciencia y Tecnología

Gradu Amaierako Lana / Trabajo Fin de Grado
Ingenieritza Elektronikoko Gradua / Grado en Ingeniería Electrónica

Máquinas de aprendizaje extremo para aplicaciones en ingeniería

Implementaciones de alta velocidad sobre FPGAs

Autor:
Joel Pardo Almanza
Directora:
Inés del Campo Hagelstrom

Leioa, 2015ko ekainaren 25a / Leioa, 25 de junio de 2015

ÍNDICE

INTRODUCCIÓN Y OBJETIVOS

Los últimos avances en la tecnología han hecho que sea posible adquirir gran cantidad de datos en tiempo real mediante diferentes sensores distribuidos y conectados a una red. Unos de los ejemplos, es el desarrollo de la tecnología del internet de las cosas (IoT: Internet of things) [1]. Gracias a esta tecnología, se pueden recibir datos adquiridos por cualquier objeto cotidiano. El problema que surge ahora es el procesado y clasificación de esta gran cantidad de datos que es posible conseguir y almacenar. Los sistemas de los que se adquieren esta gran cantidad de datos normalmente son complejos y es difícil encontrar su función de transferencia para poder clasificar y entender el significado de esos datos dentro del sistema. Una alternativa consiste en utilizar redes neuronales artificiales [2]. Las redes neuronales artificiales son capaces de emular sistemas muy complejos sin tener que saber cómo es la función de transferencia del mismo, solo nos hacen falta datos de entrada y salida del sistema. Además de esto, este tipo de aplicaciones en ocasiones se realizan en lugares remotos y a gran velocidad. Por ello, muchas aplicaciones actuales requieren un hardware específico de alta velocidad. En estos casos pueden ser los dispositivos programables, en concreto las FPGAs (Field Programmable Gate Array), los que mejor se adapten a las necesidades de velocidad, área y consumo para realizar el procesado de los datos después de haber implementado en ellas las redes neuronales artificiales. Estos dispositivos permiten realizar algoritmos complejos que pueden ser modificados una vez ha sido implementado el diseño.

Los principales objetivos del presente trabajo son el estudio de las redes neuronales artificiales, los dispositivos reconfigurables de alta velocidad, como son las FPGAs, y su aplicación a un ejemplo concreto: el reconocimiento en tiempo real de diferentes tipos de suelo en imágenes de satélite. Con este fin se propone el diseño de una red neuronal y su implementación en un dispositivo de lógica programable usando el lenguaje de descripción del hardware VHDL (Very High Description Language). Otro de los objetivos del trabajo es conocer los entornos de desarrollo que los fabricantes de dispositivos de lógica programable ponen a disposición de los diseñadores y manejar herramientas de software matemático como Matlab.

El primer capítulo tiene como objetivo principal el estudio de las redes neuronales. Para ellos se estudiarán las redes neuronales más utilizadas. Además se compararán estas diferentes redes neuronales mediante simulaciones con Matlab en las que el objetivo de las redes será la aproximación de funciones de una y dos variables.
El segundo capítulo se centra en el análisis del hardware necesario en donde poder implementar las redes neuronales. Se estudian en detalle los dispositivos de lógica programable, las tecnologías de dichos dispositivos y se analiza la familia de dispositivos FPGA que se empleará en el trabajo para implementar la red neuronal artificial del ejemplo concreto.
Los objetivos del tercer capítulo son realizar el diseño de la red neuronal para un sistema de reconocimiento de suelos en imágenes de satélite y a continuación implementar ese diseño en un dispositivo de lógica programable. Para ello, se elegirá un tipo concreto de red neuronal de los que se han trabajado en el Capítulo 1. Esta red neuronal deberá cumplir los requisitos que necesita la aplicación. Lo mismo ocurre con el hardware elegido para implementar la red. Por último se darán los resultados de las simulaciones y las conclusiones del trabajo.

Capítulo 1: REDES NEURONALES ARTIFICIALES

1.1 Introducción

Las redes neuronales artificiales son sistemas inspirados en el funcionamiento del cerebro humano [2]. Una red neuronal consiste en un grupo de unidades de procesamiento simples que denominamos neuronas y cuyo objetivo es producir una salida. Los llamados pesos son las conexiones que existen entre las neuronas. Los dos objetivos principales de las redes neuronales son la aproximación de funciones y la clasificación. En la aproximación de funciones su objetivo es aproximar una función estática no lineal, mapeando una función f(x) mediante la red neuronal $f_{NN}(x)$, donde $x \in R^K$. En cambio la clasificación, se basa en el reconocimiento de patrones para clasificar diferentes tipos de datos. Las redes neuronales artificiales están basadas en diferentes arquitecturas, las más importantes son las redes Perceptrón multicapa y las redes de base radial. Además del tipo de arquitectura, también se pueden clasificar dependiendo del método de aprendizaje que se utiliza para determinar los parámetros de la red. Las máquinas de aprendizaje extremo [3] y el método Backpropagation [4], [5] son los que se van a estudiar en este trabajo.

1.1.1 Historia

El estudio del cerebro humano se remonta a miles de años atrás. Con el avance de la tecnología, lo natural es intentar aprovecharse de la misma para comprender mejor el pensamiento humano. El primer paso de las redes neuronales artificiales (RNA) lo dieron en 1943 McCulloch y Pits al escribir una idea de cómo las neuronas podrían trabajar [2]. Ellos modelaron una red neuronal mediante circuitos eléctricos.

Años más tarde, Doald Hebb escribió un libro sobre el funcionamiento de las mismas titulado *Organization of Behavior*. Con el avance de los ordenadores en 1950s se comenzó con el modelado. Durante ese periodo fue cuando Frank Rosenblatt comenzó a trabajar en un *Perceptron*, el cual es la red neuronal más antigua pero no por ello deja de ser útil hoy en día. En 1959, Bernard Widrow y Marcian Hoff hicieron los modelos ADALINE (ADAptative LINear Element) y MADALINE (Multiple ADAptive LINear Elements) [6]. Estos modelos fueron las primeras redes neuronales en ser aplicadas para el problema de eliminar el eco en las líneas telefónicas mediante un filtro adaptativo.

El problema de no tener los medios electrónicos adecuados llevó a que se creyera que estas redes eran limitadas como escribieron Marvin Minsky y Seymour Papert en 1969 en el libro *Perceptrons.* Esto llevó a que se paralizasen varios proyectos durante 15 o 20 años. Varios investigadores continuaron con sus trabajos y desarrollaron fundamentos teóricos para redes neuronales multicapa y descubrieron como implementarlos. En 1982, varios eventos causaron un resurgimiento debido al desarrollo de algoritmos y topologías para las redes, nuevas técnicas de implementación, etc. Hoy en día las discusiones sobre redes neuronales están ocurriendo en todas partes, su futuro parece muy brillante y la clave parece ser el desarrollo de hardware para reducir el periodo de aprendizaje de algunos sistemas.

1.2 Redes Perceptrón multicapa

1.2.1 Fundamentos

Las redes neuronales Perceptrón multicapa (MLP: Multilayer Perceptron) se organizan en una capa de entrada, en una o varias capas ocultas y una única capa de salida [4], [5]. La red consta de nodos o neuronas y cada uno contiene un multiplicador, un sumador y una función de activación g. En la figura 1 vemos un ejemplo para un nodo i. Las entradas x_k, $k=1,\ldots, K$ a la neurona están multiplicadas por un peso w_{ki} y sumadas todas añadiendo además una constante θ_i llamada bias. El resultado de esa operación da la entrada a la función de activación g.
La salida del nodo i es:

$$y_i = g_i = g\left(\sum_{j=1}^{K} w_{ji}x_j + \theta_i\right) \quad (1)$$

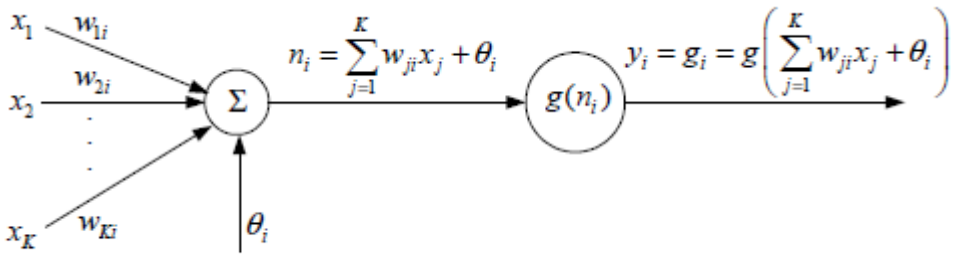


Figura 1: Neurona de una red MLP.

Existen diferentes funciones de activación, las más usadas son:

Función logística o sigmoide:

$$g(x) = \frac{1}{1+e^{-x}} \quad (2)$$

Función tangente hiperbólica:

$$g(x) = \frac{1-e^{-x}}{1+e^{-x}} \quad (3)$$

Función identidad:

$$g(x) = x \quad (4)$$

Conectando varios nodos en paralelo o en serie se crea una típica red MLP como se ve en la figura 2.

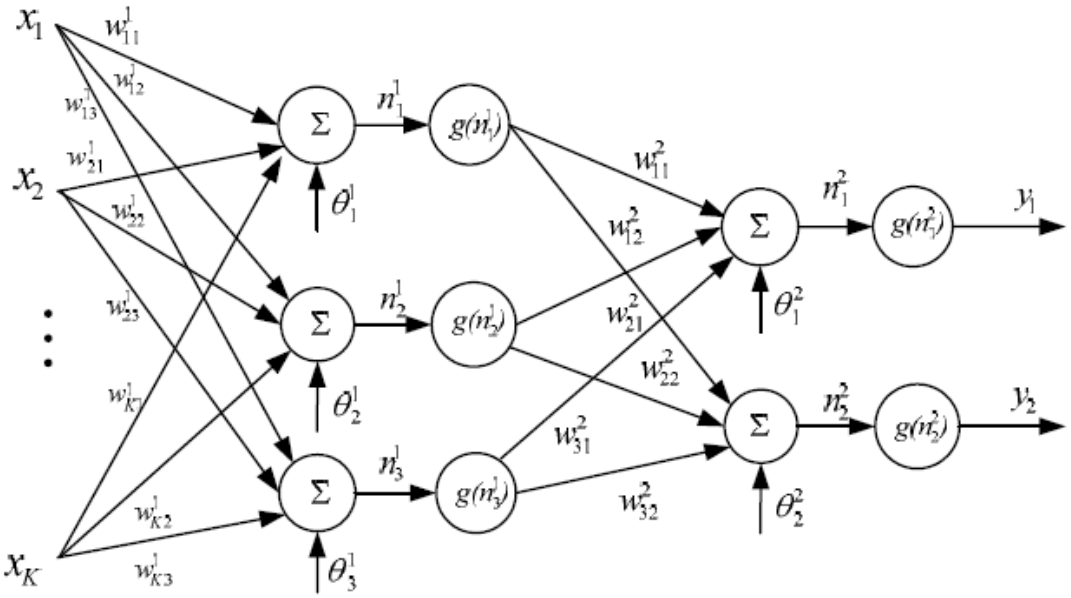


Figura 2: Red neuronal MLP con una única capa oculta y una misma función de activación para todas las neuronas.

La salida y_i, i=1,2 de la red MLP es:

$$y_i = g\left(\sum_{j=1}^{3} w_{ji}^2 g\left(\sum_{1}^{K} w_{kj}^1 x_k + \theta_j^1\right) + \theta_j^2\right) \qquad (5)$$

Observando (5) se ve que una red MLP es una función no lineal parametrizada con un espacio de entrada $x \in R^K$ y salida $y \in R^M$ (en este caso M=3).

Dados los datos de entrada y salida (x_i, y_i), $i=1,\ldots, N$, el objetivo es determinar los parámetros (w_{ji}^k, θ_j^k) para poder modelar correctamente la función objetivo.

Método de aprendizaje Backpropagation para las redes MLP

En primer lugar se define el error cuadrático para el *p*-ésimo dato de entrada-salida.

$$E_p = \sum_k (d_k - x_k)^2 \qquad (6)$$

Donde d_k es la salida deseada para el nodo k y x_k es la salida actual cuando se tiene el par de datos de entrada-salida p. Para encontrar el vector gradiente, se define el término de error $\overline{\epsilon}_i$ para el nodo i como:

$$\overline{\epsilon}_i = \frac{\partial^+ E_p}{\partial \overline{x}_i} \tag{7}$$

Mediante la regla de la cadena se puede escribir la formula recursiva para $\overline{\epsilon}_i$ como:

$$\overline{\epsilon}_i = \begin{cases} -2(d_i - x_i)\frac{\partial x_i}{\partial \overline{x}_i} = -2(d_i - x_i)x_i(1 - x_i) \\ \frac{\partial x_i}{\partial \overline{x}_i} = \sum_{j,i<j} \frac{\partial^+ E_p}{\partial \overline{x}_i}\frac{\partial \overline{x}_i}{\partial x_i} = x_i(1 - x_i)\sum_{j,i<j} \overline{\epsilon}_j w_{ij} \end{cases} \tag{8}$$

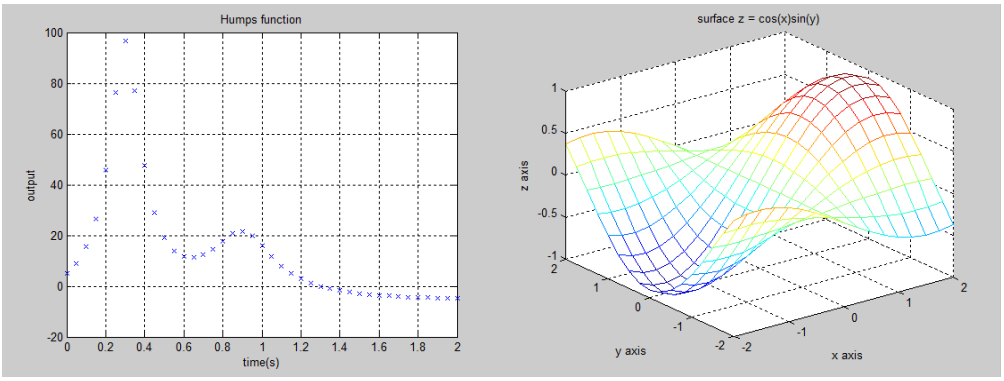
El primer caso si el nodo *i* es un nodo de salida y el segundo caso para el resto. La variable w_{ij} es el peso que conecta el nodo *i* y *j*. En el caso de que no haya conexión directa sería 0.
Para el aprendizaje "offline", el peso w_{ki} solo es actualizado después de cada iteración o después de la presentación del conjunto completo de datos. En forma vectorial se puede expresar del siguiente modo.

$$\Delta \boldsymbol{w} = -\eta \boldsymbol{\nabla}_{\boldsymbol{w}} E \tag{9}$$

Donde η es un coeficiente de aprendizaje que afecta a la convergencia y estabilidad del peso durante el entrenamiento y $E = \sum_p E_p$.

1.2.2 Aproximación de funciones mediante MLP

Aproximar funciones es uno de los objetivos principales de las redes neuronales, por ello en este apartado y en los apartados 1.3.2 y 1.3.3 se van a utilizar diferentes arquitecturas de redes neuronales artificiales para modelar funciones [7]. Las funciones serán de una función de una variable y otra dos variables como se muestran en la figura 3.
En este apartado se va a trabajar con las redes MLP, en los otros dos apartados citados anteriormente se seguirá el mismo proceso que aquí, pero en cada uno de ellos se utilizará una arquitectura diferente.



(a) (b)

Figura 3: Funciones test: (a) Función "Humps" dromedario (b) f(x)=sin(x)·cos(x).

La estructura de este apartado es la siguiente: en primer lugar se explicará cuáles son los parámetros asignados a la red que se ha creado mediante Matlab. En segundo lugar se reflejarán los datos del entrenamiento. En tercer lugar se observará la diferencia entre la aproximación y la función referencia. En cuarto lugar se verá el error cometido y para finalizar, se analizará en función del número de neuronas en la capa oculta, como ha sido el entrenamiento o proceso de aprendizaje.

Caso 1: Función de una variable.

Los parámetros usados son los siguientes: la red tiene dos capas ocultas, una de 40 neuronas y otra de 1. Además 1 nodo en la capa de salida y entrada. La función de activación utilizada es la sigmoide para la 1ª capa oculta y lineal para la 2ª. Se ha realizado un entrenamiento de 1000 iteraciones con 1e-3 como error objetivo. Y se ha fijado el "learning rate" o η coeficiente de aprendizaje en 0,05. En la figura 4 se ve el interfaz gráfico con el que se trabaja en Matlab durante el entrenamiento y en el que se puede seguir la evolución de diferentes parámetros durante el mismo.

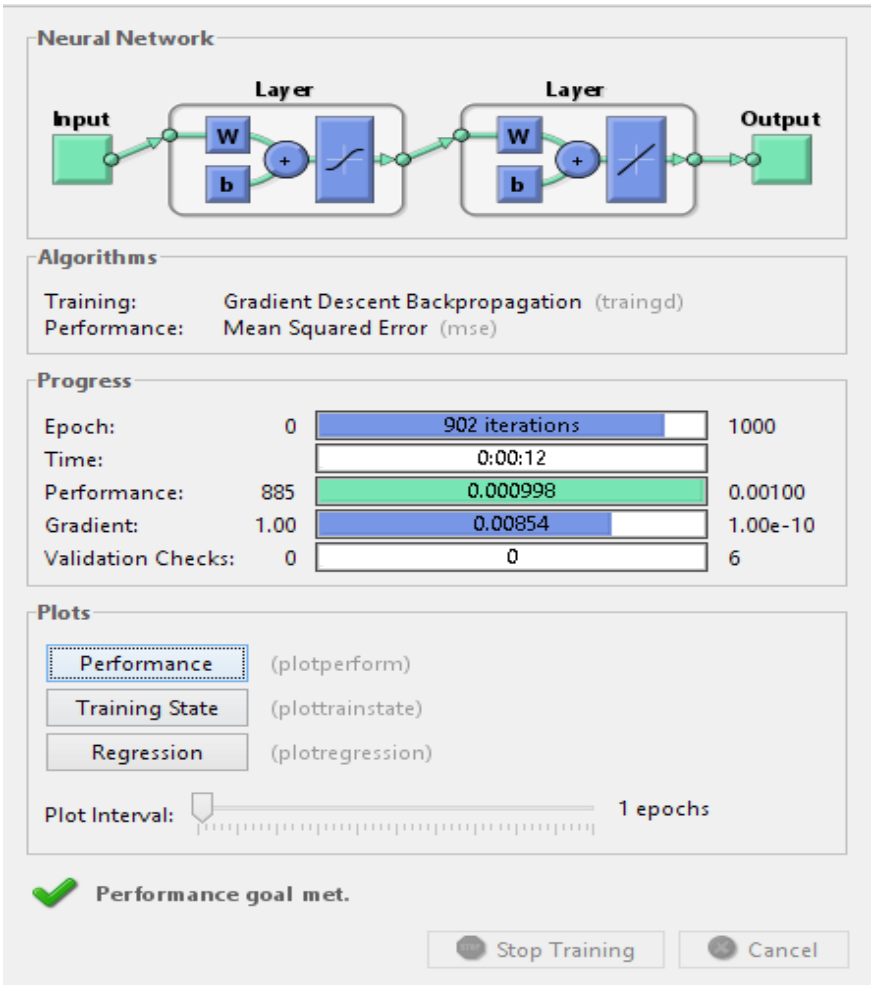


Figura 4: Interfaz gráfico (Matlab) durante el proceso de aprendizaje de una red neuronal.

Una vez realizado el entrenamiento en la figura 5 se puede observar la función aproximada y la función referencia.

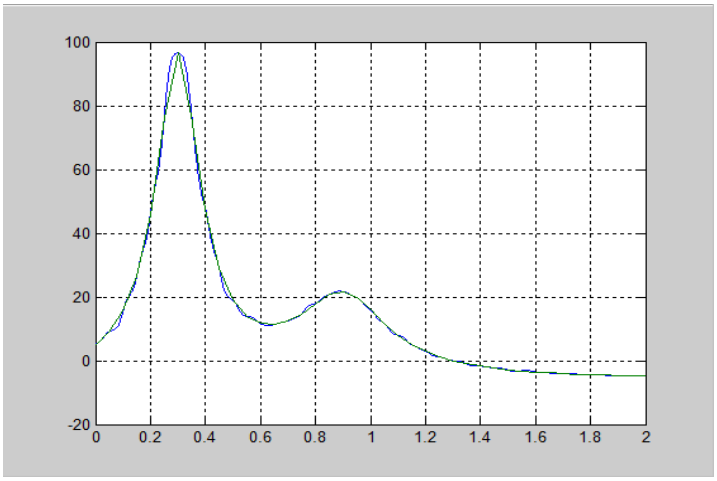


Figura 5: Aproximación (Azul) VS Referencia (Verde).

Es difícil apreciar el error con la figura 5 por ello, restando las dos funciones se puede conseguir directamente el error. Este último proceso se puede ver en la figura 6.

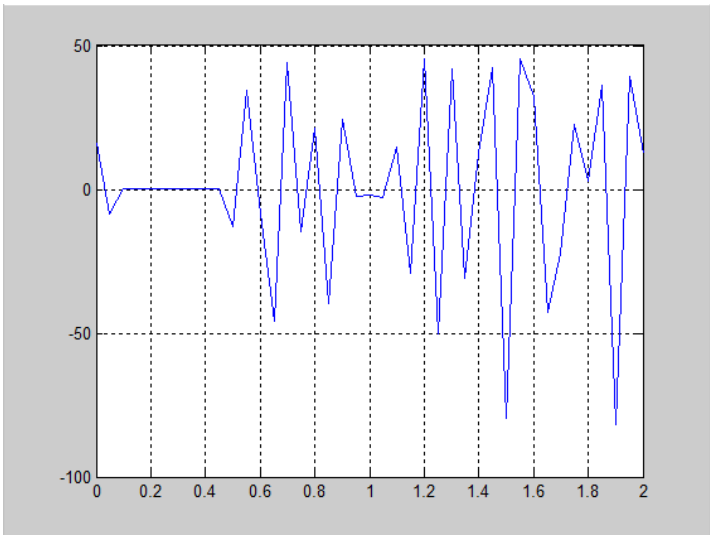


Figura 6: Error (x1000) entre la aproximación y la función referencia de una variable para la red MLP.

Para finalizar se va a ver como es el proceso de aprendizaje para diferentes números de neuronas en la 1ª capa oculta. Para ello, se va a representar el error cuadrático medio en función de las iteraciones para 10, 20, 30 y 40 neuronas en la capa oculta (ver figura 7).

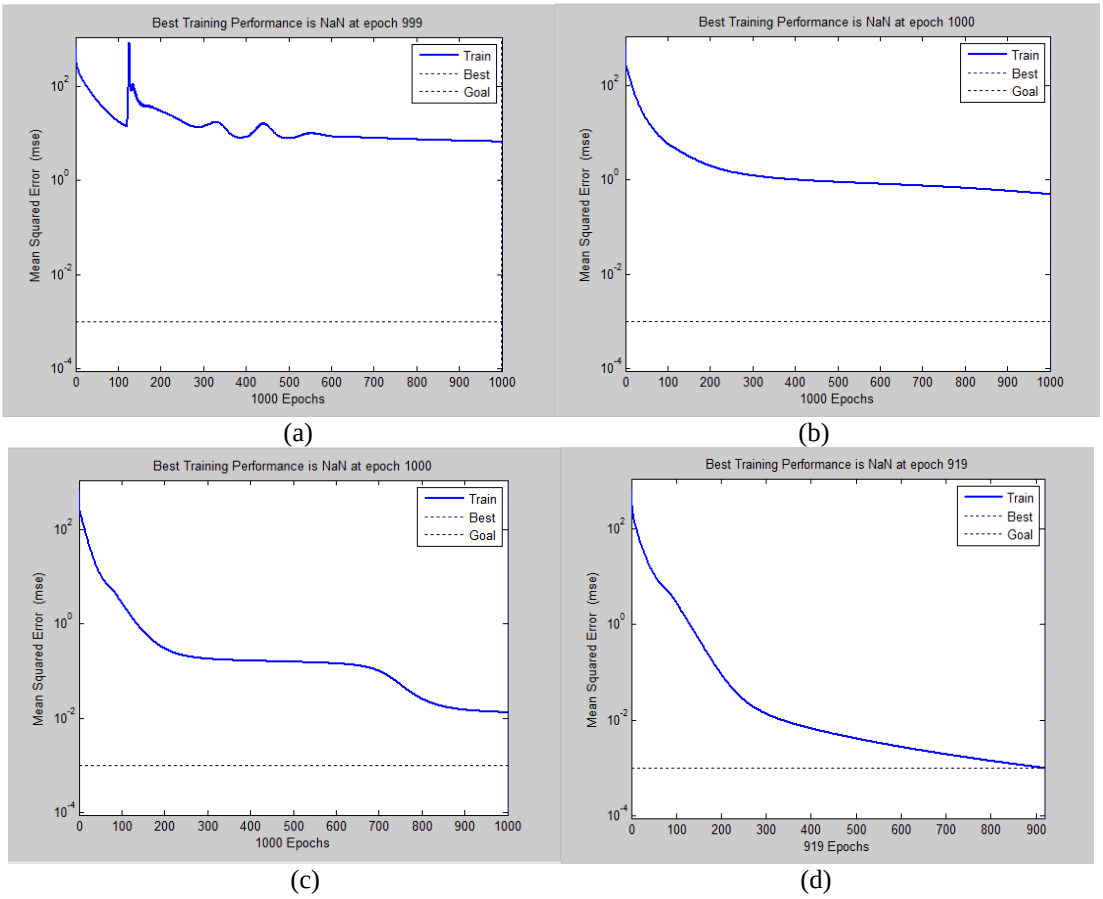


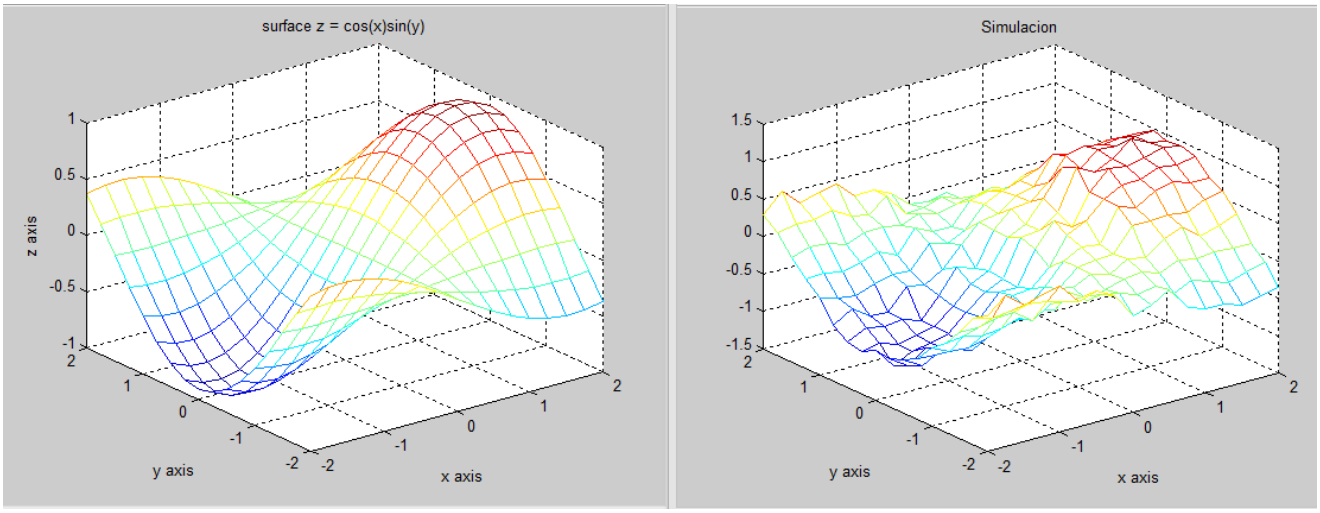
Figura 7: Proceso de aprendizaje en función del número de neuronas en la 1ª capa oculta. Error cuadrático medio frente a iteraciones. (a) 10 neuronas (b) 20 neuronas (c) 30 neuronas (d) 40 neuronas.

Como puede verse, en el caso de 10, 20 y 30 neuronas no se llega al error objetivo después de las 1000 iteraciones. Además, en el caso de 10 y 20 neuronas el error cuadrático medio se satura y deja de disminuir aunque haya más iteraciones. Para el caso de las 40 neuronas se observa que hasta las 300 iteraciones el error cuadrático medio disminuye muy rápidamente. En las 600 iteraciones siguientes el error decrece mucho más despacio. Finalmente se llega al error objetivo.

Caso 2: Función de dos variables.

Los parámetros usados son los siguientes, la red tiene dos capas ocultas, una de 50 neuronas y otra de 17. Además 1 nodo en la capa de salida y 2 en la de entrada. La función de activación utilizada es la sigmoide para la 1ª capa oculta y lineal para la 2ª. Se ha realizado un entrenamiento de 5000 iteraciones con 10e-5 como error objetivo. Y se ha fijado el "learning rate" o η coeficiente de aprendizaje en 0,05.

Una vez realizado el entrenamiento se van a comparar mediante Matlab las dos funciones, es decir, en primer lugar se va a representar la función referencia citada anteriormente y en segundo lugar la función aproximada por la red neuronal que se ha creado en este apartado con los parámetros citados. La representación de estas dos funciones se puede ver en la figura 8.



(a) (b)

Figura 8: (a) Función referencia (b) Función aproximada.

Es difícil apreciar el error con la figura 8, por ello, restando las dos funciones se puede conseguir directamente el error. Este último proceso se puede ver en la figura 9.

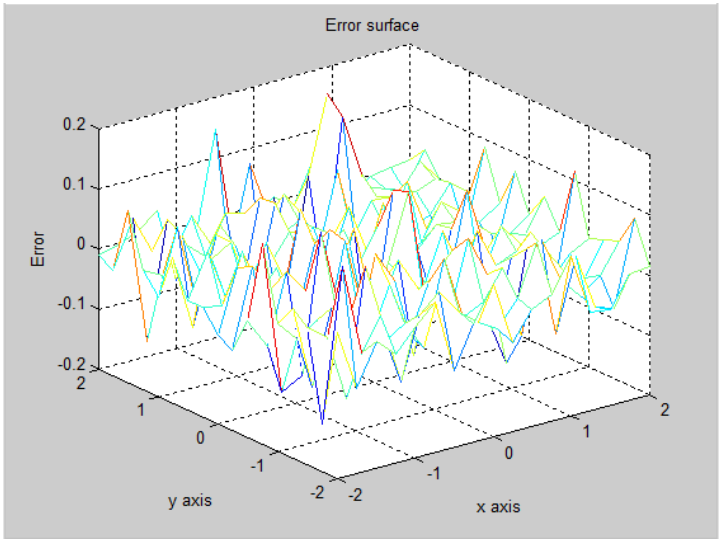


Figura 9: Error entre la aproximación y la función referencia de dos variables para la red MLP.

Para finalizar, se va a realizar el mismo proceso que en el anterior caso con la función de una variable, es decir, se va a analizar como es el proceso de aprendizaje para diferente número de neuronas en la 1ª capa oculta. Para ello, se va a expresar el error cuadrático en función de las iteraciones para diferente número de neuronas. En la figura 10 se aprecian los diferentes casos.

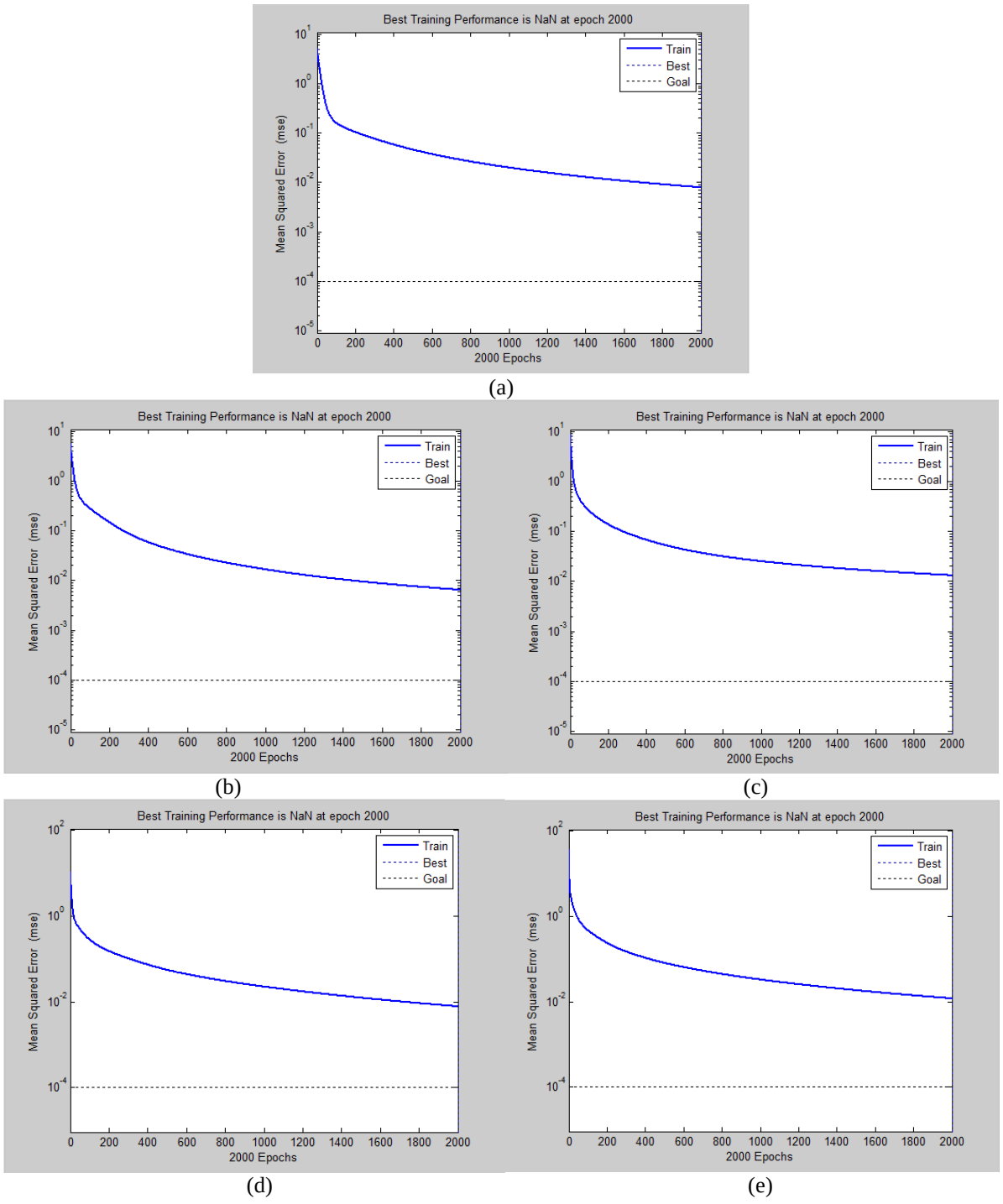


Figura 10: Proceso de aprendizaje en función del número de neuronas en la 1ª capa oculta. Error cuadrático frente a iteraciones. (a) 10 neuronas, (b) 20 neuronas, (c) 30 neuronas, (d) 40 neuronas, (e) 100 neuronas.

En este caso, en ninguno de los cinco casos se llega al error objetivo. Se puede ver que hasta las 200 iteraciones el error decrece rápidamente pero a partir de ahí se satura en todos los casos y no llega a bajar de 10^{-2}.

1.3 Redes de base radial

1.3.1 Fundamentos

Las redes de base radial (RBF: Radial basis functions) son muy similares a las MLP si nos referimos a arquitectura general, su mayor diferencia se encuentra las funciones llamadas niveles de activación o funciones de base radial [4], [5]. El nivel de activación para la unidad de campo receptiva i se define como:

$$w_i = R_i(x) = R_i(\|\boldsymbol{x} - \boldsymbol{u}_i\|/\sigma_i),\ i = 1,2,\dots,H \tag{10}$$

Donde *x* es un vector de entrada multidimensional, u_i tiene su misma dimensión. *H* es el número de funciones de base radial y $R_i(\cdot)$ es la función de base radial con un único máximo en el origen. Las funciones de base radial más usadas son las Gaussianas y las sigmoides o logísticas. A continuación, en la ecuación (11) y (12), vemos cada una de ellas.

$$R_i(x) = exp\left[-\frac{\|\boldsymbol{x}-\boldsymbol{u}_i\|^2}{2\sigma_i^2}\right] \tag{11}$$

$$R_i(x) = \frac{1}{1+exp[\|\boldsymbol{x}-\boldsymbol{u}_i\|^2/\sigma_i]} \tag{12}$$

Como se ve, el nivel de activación será máximo cuando el vector de entrada esté en el centro u_i de esa unidad.
La salida puede darse de dos formas, la suma de cada unidad, o la media de esa suma.

$$d(x) = \sum_{i=1}^{H} c_i w_i = \sum_{i=1}^{H} c_i R_i(x) \tag{13}$$

$$d(x) == \frac{\sum_{i=1}^{H} c_i w_i}{\sum_{i=1}^{H} w_i} = \frac{\sum_{i=1}^{H} c_i R_i(x)}{\sum_{i=1}^{H} R_i(x)} \tag{14}$$

Donde c_i es el valor de salida asociado a esa unidad receptiva *i* o también se puede ver como el peso que conecta esa unidad de conexión con la unidad de salida.

Para poder entender mejor como es la estructura de las redes RBF, se puede observar la figura 11 en la que se encuentran diferentes redes con la arquitectura típica de las redes de base radial. Además hay diferentes ejemplos en los que se diferencia las dos salidas tipo que pueden tener estas redes y las cuales se han expresado anteriormente con las ecuaciones (13) y (14).

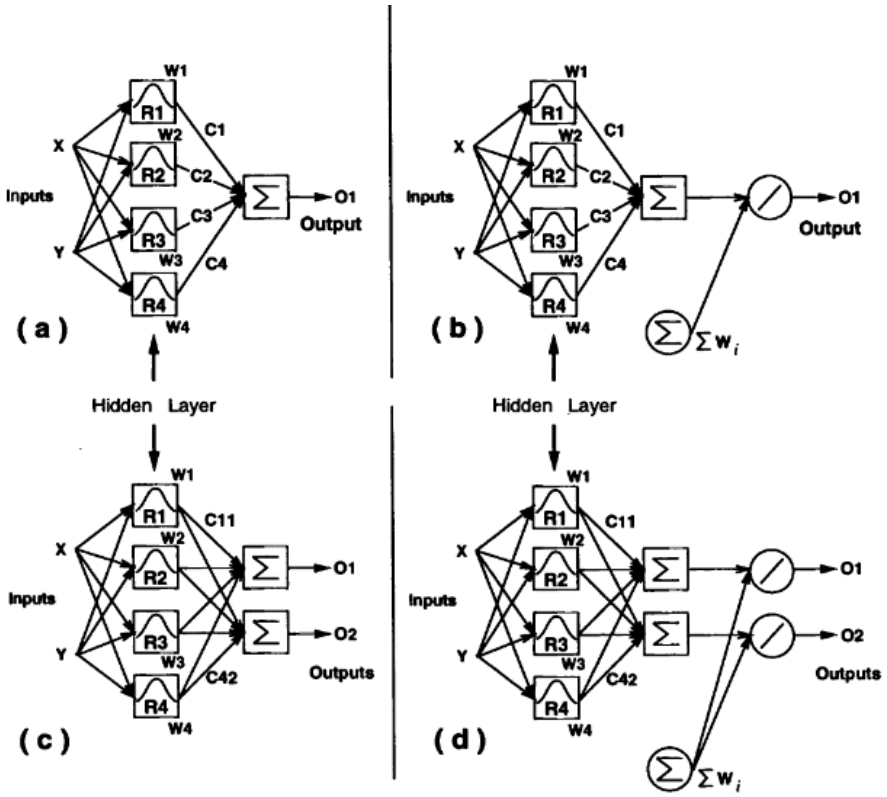


Figura 11: Cuatro redes de arquitectura de base radial. (a) Única salida y sin hacer la media. (b) Una salida haciendo la media. (c) Dos salidas sin hacer la media.(d) Dos salidas haciendo la media.

Método de aprendizaje Backpropagation para las redes RBF

Si se asume que no hay ningún ruido en nuestros datos para el entrenamiento, el objetivo es obtener o estimar una función $d(\cdot)$ que nos proporcione las salidas exactas deseadas para todo los datos del entrenamiento. Considerando una función Gaussiana como función de base radial centrada en u_i y con parámetro de anchura σ_i:

$$w_i = R_i(\|x - u_i\|) = exp\left[\frac{(x-u_i)^2}{2\sigma_i^2}\right] \qquad (15)$$

Cada dato de entrada de entrenamiento x_i nos sirve como centro de nuestra función de base radial. Por lo tanto utilizando la ecuación (13), la salida será:

$$d(x) = \sum_{i=1}^{H} c_i exp\left[\frac{(x-x_i)^2}{2\sigma_i^2}\right] \qquad (16)$$

Para cada σ_i, $i=1,\ldots,$ n dado, obtenemos n ecuaciones lineales para los n coeficientes desconocidos c_i.

$$\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} exp\left[\frac{(x_1-x_1)^2}{2\sigma_1^2}\right]+\cdots+exp\left[\frac{(x_1-x_n)^2}{2\sigma_n^2}\right] \\ exp\left[\frac{(x_2-x_1)^2}{2\sigma_1^2}\right]+\cdots+exp\left[\frac{(x_2-x_n)^2}{2\sigma_n^2}\right] \\ \vdots \\ exp\left[\frac{(x_n-x_1)^2}{2\sigma_1^2}\right]+\cdots+exp\left[\frac{(x_n-x_n)^2}{2\sigma_n^2}\right] \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \quad (17)$$

Si lo reescribimos de forma compacta:

$$\mathbf{D=GC} \quad (18)$$

Cuando la matriz G no es singular, tenemos solución única:

$$\mathbf{C=G^{-1}D} \quad (19)$$

1.3.2 Aproximación de funciones mediante RBF

Caso 1: Función de una variable.

Los parámetros usados son los siguientes, la red tiene dos capas ocultas de 24 neuronas. Además 1 nodo en la capa de salida y entrada. La función de base radial utilizada es la gaussiana. Se ha fijado el radio o parámetro de anchura σ=0.1 y el error objetivo en 0.02.

Una vez realizado el entrenamiento, en la figura 12 se puede observar la función aproximada y la función referencia.

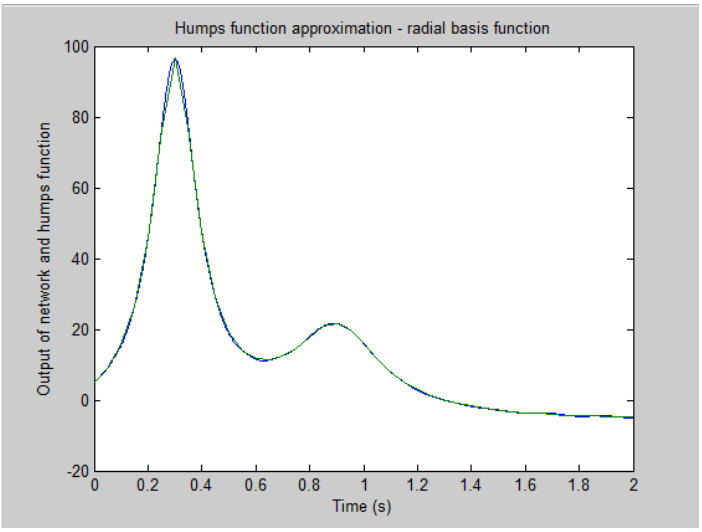


Figura 12: Aproximación (Azul) VS Referencia (Verde).

Es difícil apreciar el error con la figura 12 por ello, restando las dos funciones se puede conseguir directamente el error. Este último proceso se puede ver en la figura 13.

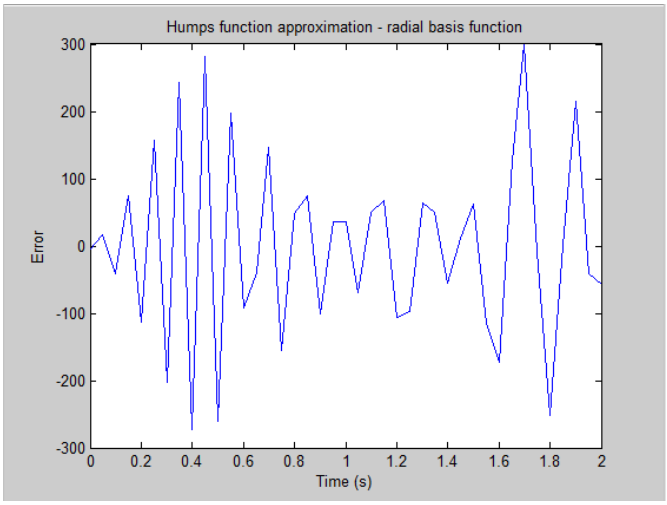
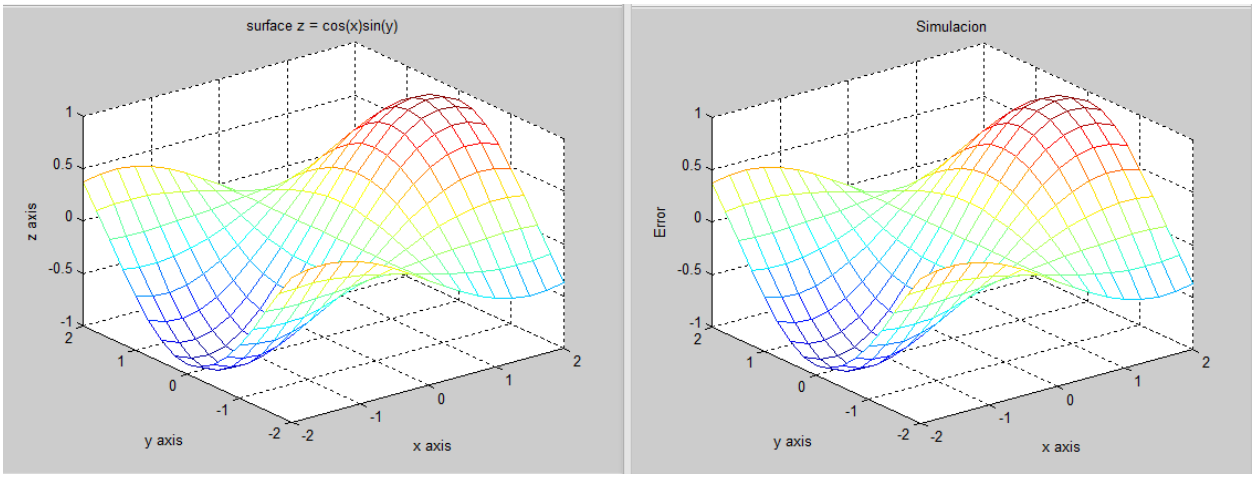


Figura 13: Error(x1000) entre la aproximación y la función referencia de una variable para la red RBF.

Se puede ver que el error es mayor que con la red MLP en la que no superaba los ±70 (error x 1000). Aun así, se han utilizado menos número de neuronas y el proceso de aprendizaje es mucho más rápido debido a que no hay iteraciones.

Caso 2: Función de dos variables.

Los parámetros usados son los siguientes, la red tiene dos capas ocultas de 17 neuronas. Además 1 nodo en la capa de salida y 2 en la de entrada. La función de base radial utilizada es la gaussiana. En este caso no se ha fijado ningún error específico ni el parámetro σ.
Una vez realizado el entrenamiento en la figura 14 se puede observar la función aproximada y la función referencia.



(a) (b)

Figura 14: (a) Función referencia (b) Función aproximada.

En la figura 15 se ha representado la diferencia entre la función de referencia y la función aproximada.

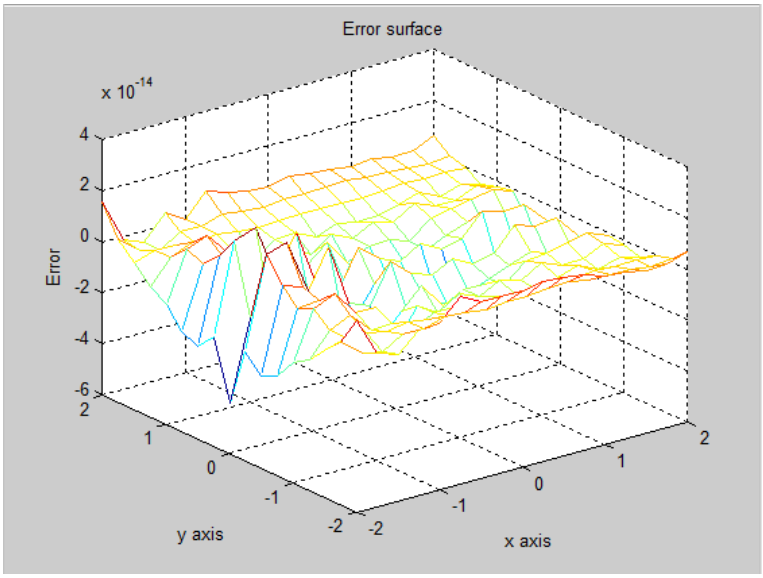


Figura 15: Error entre la aproximación y la función referencia de dos variables para la red RBF.

En este caso el error es casi nulo. Por lo tanto se consigue un error menor que en el caso de las redes MLP, en las que además se utilizaban más neuronas. Esto es debido a lo que se ha explicado de la figura 10. El error conseguido en la red MLP se satura aunque aumentemos en número de neuronas. Por último, el proceso de aprendizaje es también más rápido debido a que no hay iteraciones.

1.3 Máquinas de aprendizaje extremo

1.3.1 Fundamentos

La estructura de las máquinas de aprendizaje extremo (ELM: Extreme learning machines) [3] es la misma que se explica en el caso de MLP, aunque en las redes ELM se dispone únicamente de una capa oculta, y las neuronas de salida no tienen función de activación ni bias, por lo tanto, la función de salida de la red es:

$$f_L(x) = \sum_{i=1}^{L} \beta_i G(a_i, b_i, x) \qquad (20)$$

Estas redes tienen dos ventajas principales respecto al de método de entrenamiento frente al BP:
-Se pueden fijar los pesos y los bias antes de tener los datos de entrenamiento ya que se calculan aleatoriamente.

-Lo único que hay que obtener durante el aprendizaje son los pesos de salida, los cuales se calculan de la siguiente manera (sin iteraciones):

$$f_L(x) = \sum_{i=1}^{L} \beta_i G(a_i, b_i, x) = t_j \tag{21}$$

es equivalente a:

$$H\boldsymbol{\beta} = T \tag{22}$$

donde

$$H = \begin{bmatrix} G(a_1,b_1,x_1)\dots G(a_L,b_L,x_1) \\ G(a_1,b_1,x_2)\dots G(a_L,b_L,x_2) \\ \vdots \\ G(a_1,b_1,x_N)\dots G(a_L,b_L,x_N) \end{bmatrix}_{NxL} \tag{23}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \beta_2^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{LxM} \tag{24}$$

$$T = \begin{bmatrix} t_1^T \\ t_2^T \\ \vdots \\ t_L^T \end{bmatrix}_{NxM} \tag{25}$$

H es llamada la matriz de salida de la capa oculta.
Los tres pasos de entrenamiento son los siguientes:

-Generar los parámetros de la capa oculta aleatoriamente (pesos, bias)
-Calcular la matriz H.
-Calcular la matriz de pesos de salida β:

$$\beta = H^{\dagger}T \tag{26}$$

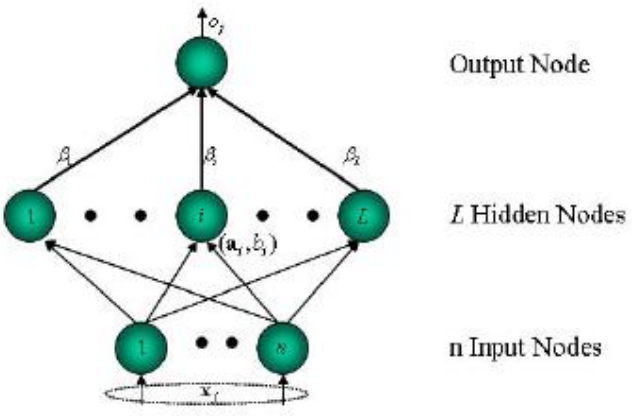


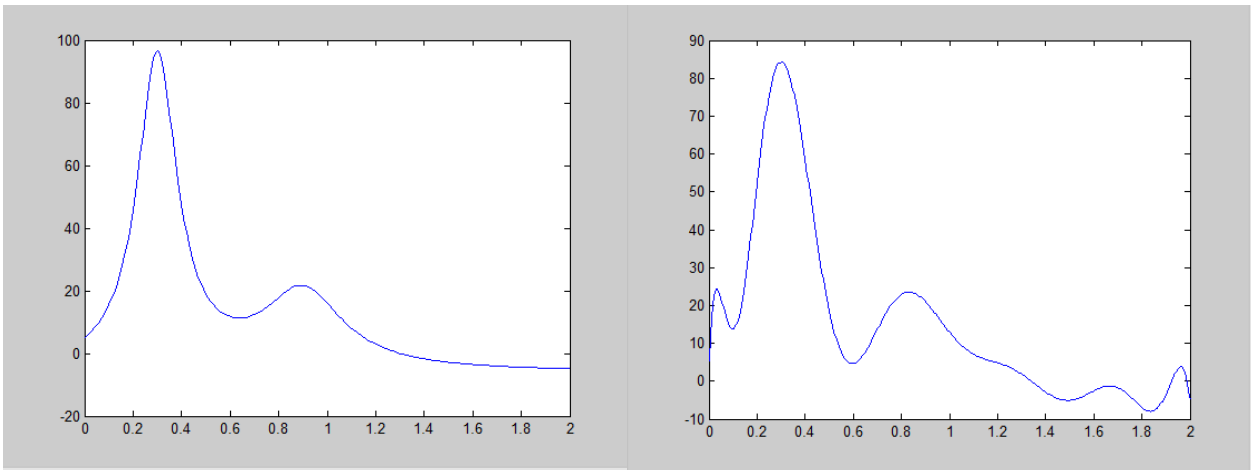
Figura 16: Estructura de una red a la que se le aplica el método de entrenamiento ELM.

1.3.2 Aproximación de funciones mediante ELM

Caso 1: Función de una variable.

Los parámetros usados son los siguientes, la red tiene una capa oculta, ésta es de 40 neuronas. Además tiene 1 nodo en la capa de salida y entrada. La función de activación que se utiliza para las neuronas de la capa oculta es la sigmoide.

En la figura 17 se puede observar la función aproximada y la función referencia, mientras que en la figura 18 se ha representado el error.



(a) (b)

Figura 17: (a) Función referencia (b) Función aproximada.

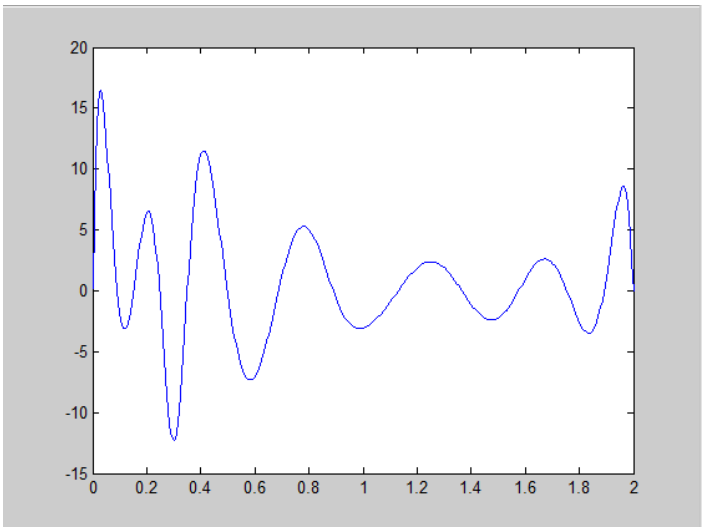


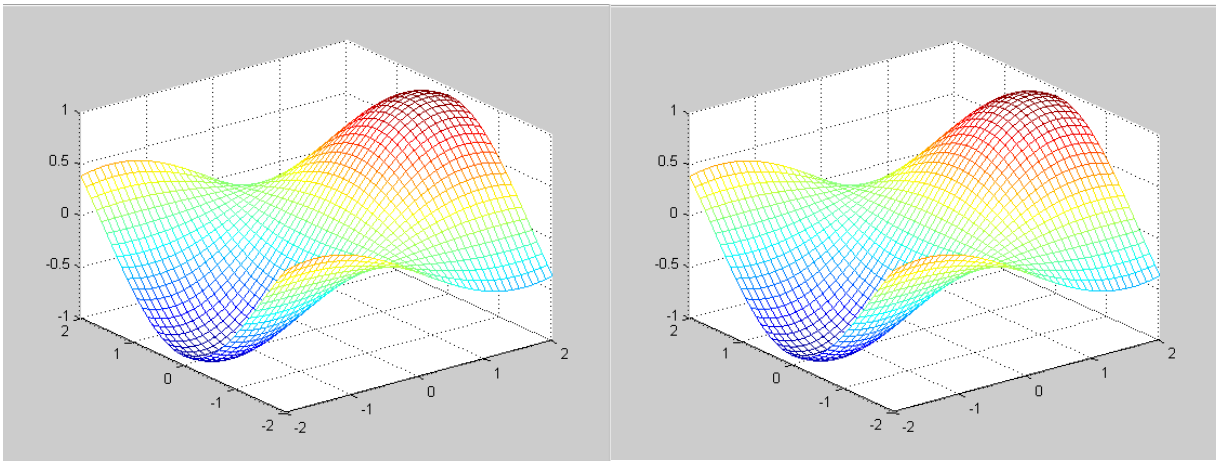
Figura 18: Error entre la aproximación y la función referencia de una variable para la red ELM.

Se puede observar que el error en este caso es mucho mayor que en el caso de las redes MLP y RBF. Por lo tanto habría que diseñar una red con más neuronas poder llegar al mismo error. En el caso de que no haya que llegar a un error como el obtenido en las redes MLP o RBF, esta red es la más adecuada debido a su proceso de aprendizaje rápido y sencillo. No utiliza ni iteraciones y solamente hay que obtener durante el aprendizaje los pesos de salida.

Caso 2: Función de dos variables.

Los parámetros usados son los siguientes, la red tiene una capa oculta de 50 neuronas. Además 1 nodo en la capa de salida y 2 en la de entrada. La función de activación usada para las neuronas de la capa oculta es la sigmoide.

Finalmente en la figura 19 se puede observar la función aproximada junto con la función referencia. Y en la figura 20 se ha representado la diferencia entre ellas.



(a) (b)

Figura 19: (a) Función referencia (b) Función aproximada.

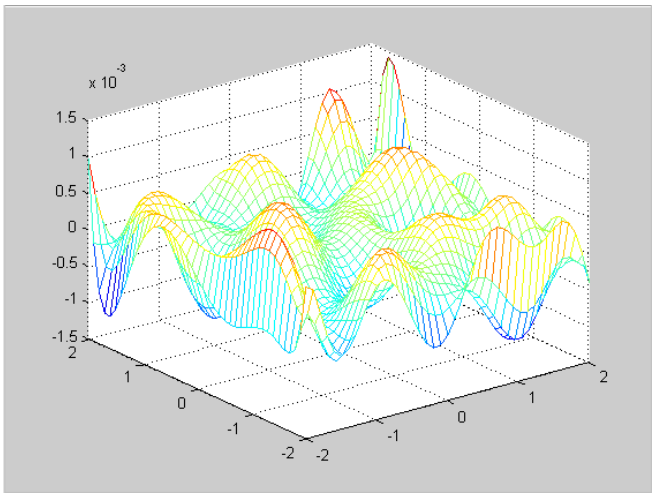


Figura 20: Error entre la aproximación y la función referencia de dos variables para la red ELM.

En resumen, se puede decir que para un número determinado de neuronas, el error obtenido con la red ELM suele ser mayor en general que el cometido con las demás redes, esto se puede observar en la aproximación de funciones de una variable que se ha realizado. Aun así, el proceso de aprendizaje de las redes ELM hace que este tipo de redes sean muy interesantes para aplicaciones en las que la rapidez y sencillez sean los objetivos primordiales.

Capítulo 2: LÓGICA PROGRAMABLE

2.1 Introducción

Los circuitos integrados se pueden clasificar dependiendo de su metodología de diseño y fabricación [8]. Existen tres grandes grupos como se observa en la figura 22 [9]: circuitos integrados estándar, circuitos integrados de aplicación específica (ASICs:Application Specific Integrated Circuit) y dispositivos de lógica programable (PLDs:Programmable Logic Device).
Los circuitos integrados estándar poseen características lógicas y eléctricas perfectamente definidas. Este tipo de circuitos presenta la ventaja de su reducido coste y su gran fiabilidad debido a que se fabrican en grandes series. Sin embargo, para poder realizar circuitos un tanto complejos se necesitarán un gran número de dispositivos de este tipo. Además es muy común que pueda haber copias no autorizadas.
Los ASICs son circuitos que se diseñan para realizar una función específica en una aplicación concreta. Existen varios tipos de ASICs, siendo los más utilizados los de tecnología full-custom y semi-custom. En los circuitos full-custom, el diseñador tiene control total en el diseño de los dispositivos. Son los más rápidos y eficientes, pero son los que requieren más tiempo de diseño y tienen mayores costes. Los semi-custom, dan cierta libertad de diseño, pudiendo, por ejemplo, especificar las conexiones que deben ser realizadas entre los transistores [10]. Sin embargo, tanto los full-custom como los semi-custom requieren de la intervención del fabricante en la fase final de producción.

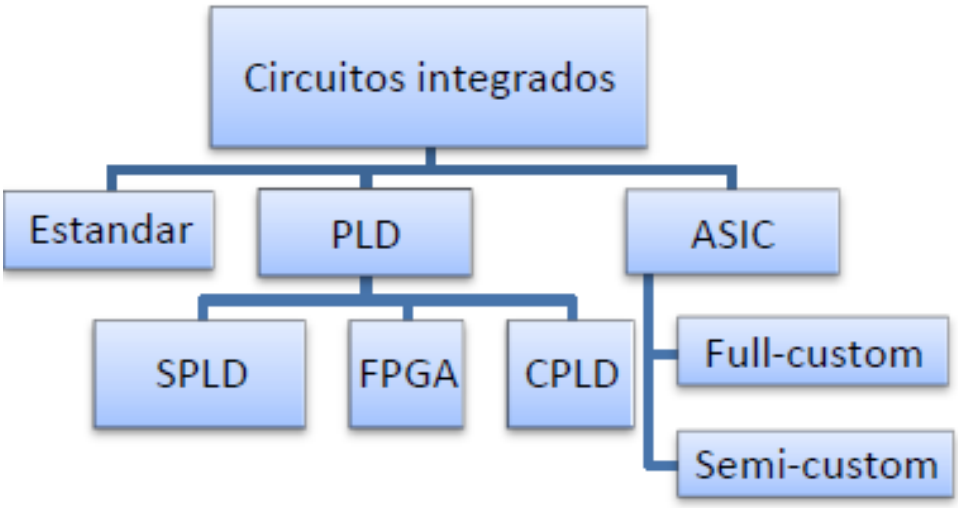


Figura 21: Clasificación de los circuitos integrados.

Por último, en los dispositivos de lógica programable el fabricante dispone de una gran cantidad de circuitos básicos en un sustrato de silicio. Estos circuitos básicos pueden utilizarse para diferentes funciones cambiando sus conexiones internas, lo cual puede hacer el diseñador desde el exterior del dispositivo. Los primeros dispositivos de lógica programable fueron los SPLDs (Simple Programmable Logic Device). Su estructura interna está formada por un conjunto de matrices de puertas AND y puertas OR. El avance en el desarrollo de estos dispositivos ha dado lugar a dos grupos: CPLD (Complex Programmable Logic Device) y FPGA (Field Programmable Gate Array).

2.2 Dispositivos de lógica programable

2.2.1 Dispositivos SPLDs

Los primeros dispositivos que se desarrollaron fueron los SPLDs, estos dipositivos estaban compuestos de una matriz de interconexiones de puertas AND y OR. Dependiendo de la programabilidad de las matrices AND y OR existen diferentes tipos de SPLDs. La figura 22 refleja los 3 tipos básicos dependiendo de su programabilidad. En primer lugar están los de tipo PROM en la que solo se puede programar la matriz OR. En cambio en las PAL solamente se puede programar la matriz AND. Por último en el caso de las PLA se puede programar tanto la matriz AND como la matriz OR.

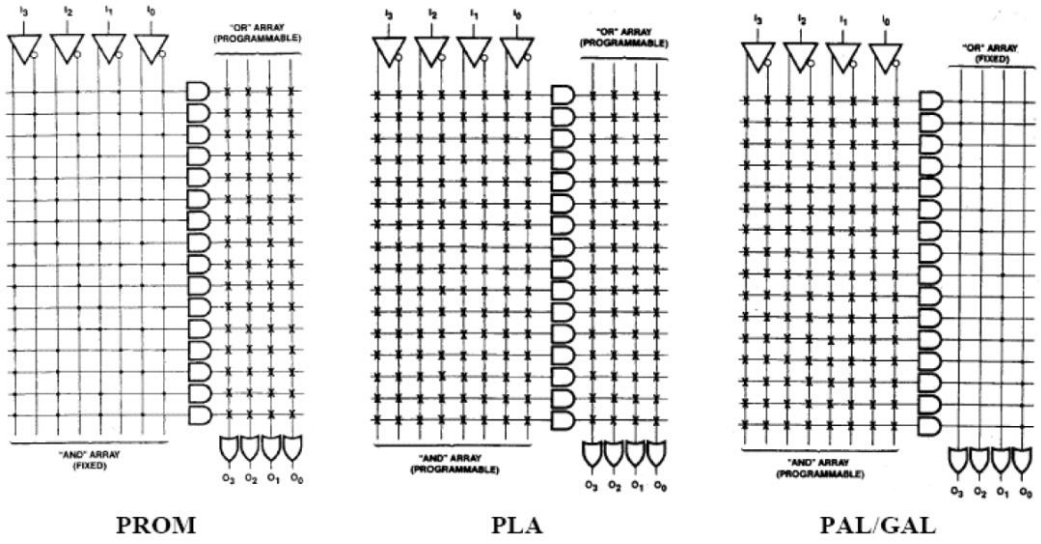


Figura 22: Diferentes formas de programabilidad de las matrices AND y OR.

Además de ello podían contener dispositivos secuenciales como registros, por ello tenían salidas secuenciales y combinacionales. En la figura 23 se puede ver la arquitectura interna de un SPLD en la que se ven la matriz AND, la matriz OR, una salida combinacional, un pin configurable como entrada o salida y una salida secuencial debido a que tiene un registro conectado.

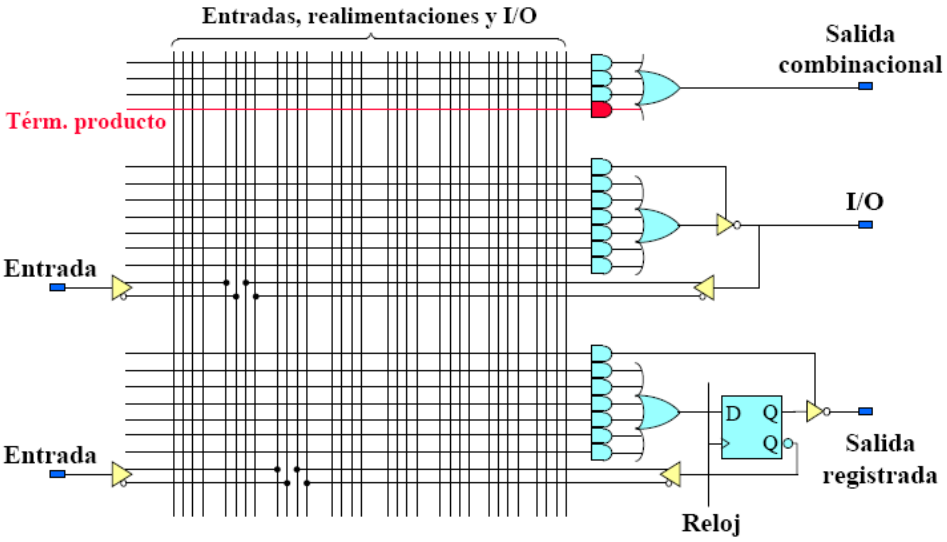


Figura 23: Arquitectura interna de un SPLD de tipo PAL.

2.2.2 Dispositivos CPLDs

Los siguientes dispositivos que se desarrollaron fueron los CPLDs. Estos dipositivos están compuestos por múltiples bloques lógicos, cada uno de ellos similar a la arquitectura de un SPLD. Es decir, los bloques lógicos conservaban los elementos de un SPLD pero además también tenían elementos como multiplexores. Estos bloques están interconectados entre sí mediante la matriz de interconexiones programable (matriz PI). Constan además de bloques de entrada/salida (I/O). En la figura 24 se puede ver la arquitectura típica de un CPLD.

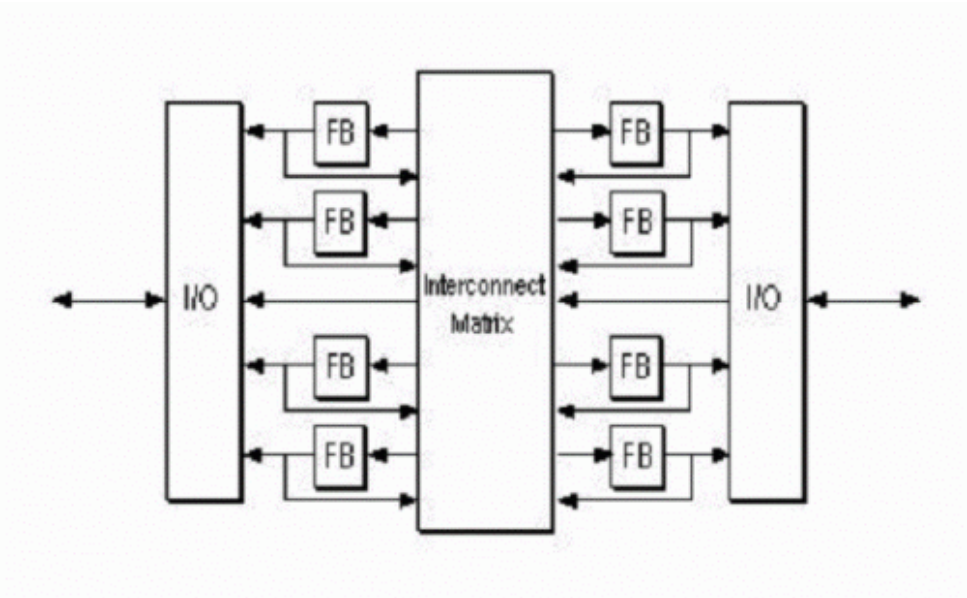


Figura 24: Arquitectura típica de un CPLD.

2.2.3 Dispositivos FPGAs

Por último, se desarrollaron las FPGAs [8]. Estos dispositivos surgen en la década de los 80 y se han desarrollado exitosamente hasta la actualidad. Estos dispositivos son capaces de reproducir desde funciones sencillas hasta complejos sistemas digitales complejos.

Una FPGA se divide principalmente en:

CLBs (Configurable logick block): Estos bloques se dividen a su vez en slices. Puede haber varios tipos de slices y cada uno tiene una función diferente como pueden ser de memoria o funciones lógicas. Cada slice está compuesto de bloques lógicos sencillos.

Interconexionado: Es la red de conexiones que conecta los CLBs entre sí. La tecnología de programación típica suele ser SRAM.

Bloques I/O: Los bloques de entrada salida o "Input/Output Blocks, IOB" proporcionan una interconexión unidireccional o bidireccional, entre los pines del empaquetado y la lógica interna del FPGA.

Bloques extra: Pueden contener bloques de memoria SRAM y multiplicadores totalmente optimizados para hacer más eficiente el funcionamiento de nuestro dispositivo programable.

DCM (Digital clock manager): Muchas FPGAs actuales contiene un módulo llamado DCM que proporcionan un control flexible sobre la frecuencia y la fase del Clock.

En las siguiente figura 25 se ve la arquitectura estándar de una FPGA.

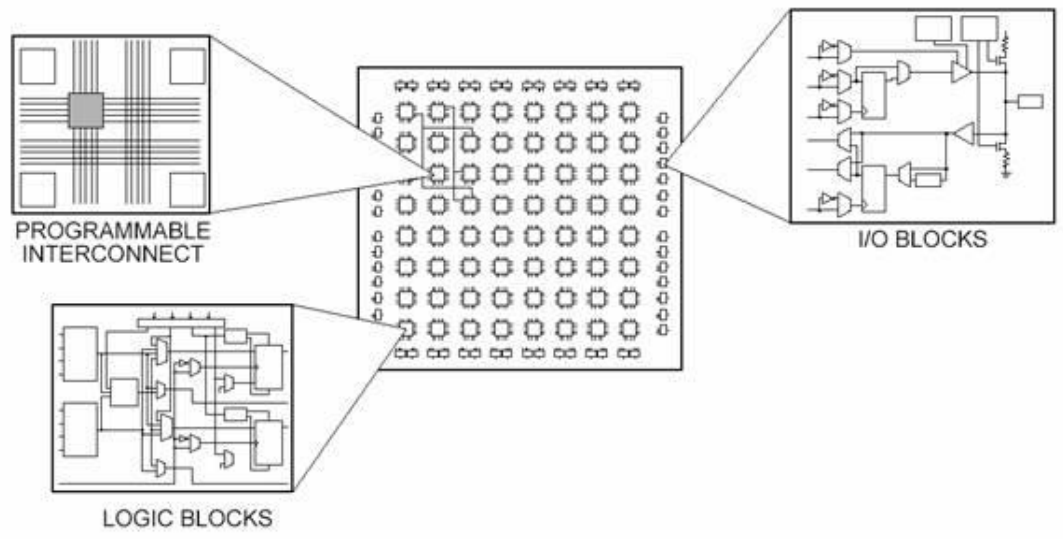


Figura 25: Arquitectura estándar de una FPGA.

2.3 Tecnologías de dispositivos lógicos programables

Existen diferentes tecnologías que permiten configurar los PLDs. Seguidamente se introducen los más utilizados.

2.3.1 Tecnología PROM

La primera tecnología que se utilizó para programar las interconexiones de las matrices AND y OR está formada por fusibles (NiCr o Titanio- Tungsteno) y diodos unidos en serie como se observa en la figura 27 [11], [12]. La principal desventaja de esta tecnología es su carácter destructivo ya que no admiten reconfiguración.

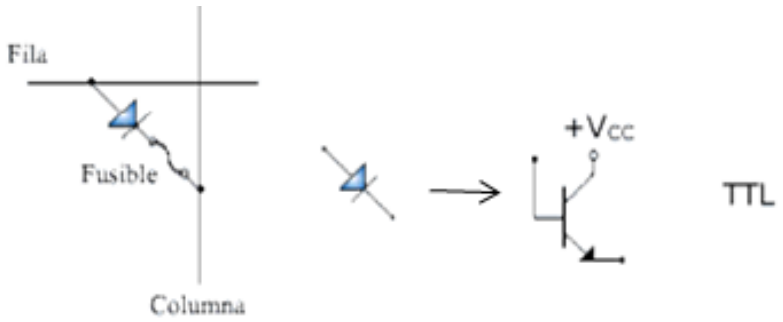


Figura 26: Elementos programables de tipo fusible.

2.3.2 Tecnología EPROM

La tecnología que permitió el gran desarrollo de los SPLD hasta el punto de ser denominados CPLDs fue la tecnología EPROM. Los dispositivos de tecnología EPROM (Erasable Programmable Read Only Memory) son dispositivos no volátiles reprogramables [11], [12]. La configuración del dispositivo se almacena en una celda EPROM. La celda consiste en un único transistor NMOS con una puerta flotante, también conocido como FAMOS (Floating-gate Avalanche-injection MOS), ver figura 27 y figura 28. La puerta flotante es una capa de polisilicio rodeada de óxido aislante, que se coloca entre la puerta y el sustrato. Cuando se almacenan electrones en la capa flotante, se incrementa la tensión umbral del transistor, lo que provoca que quede polarizado en corte y el transistor queda programado.

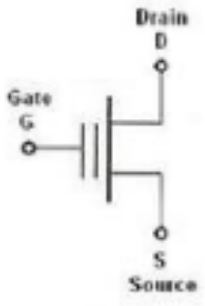


Figura 27: elemento de matriz EPROM.

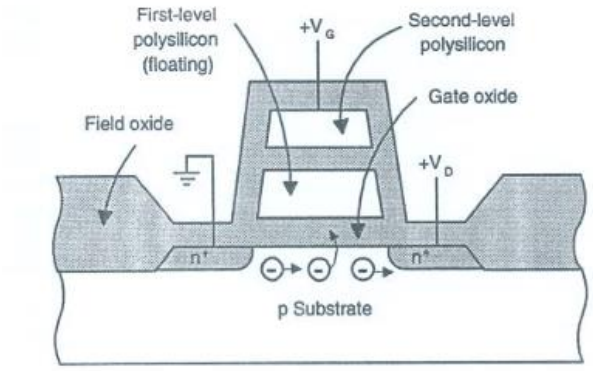


Figura 28: Transistor FAMOS (EPROM).

El gran inconveniente de esta tecnología es el proceso de borrado ya que se realiza mediante radiación ultravioleta. Lo que requiere la extracción del circuito del sistema para aplicar raciación UV. Con el fin de eliminar este problema se desarrolló la tecnología EEPROM.

2.3.3 Tecnología EEPROM

Una celda EEPROM (Electrically Erasable Programmable Read Only Memory) consta de dos transistores: un MOSFET de acumulación y un FLOTOX (Floating Gate Tunnel Oxide). El FLOTOX es un transistor de puerta flotante con una fina capa de óxido sobre la zona del drenador que puede programarse y borrarse eléctricamente por Efecto Tunel [12]. En la figura 29, puede observarse el símbolo de la celda básica. En figura 30, se observan las tensiones necesarias para que el transistor sea programado. La ventaja de esta tecnología es que puede configurarse en el propio sistema, sin necesidad de extraer el chip.

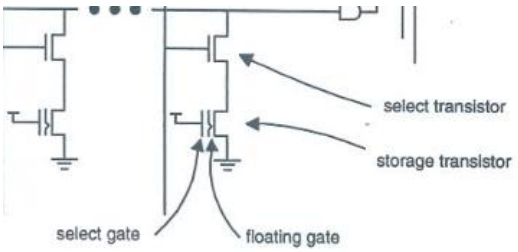


Figura 29: Celda EEPROM.

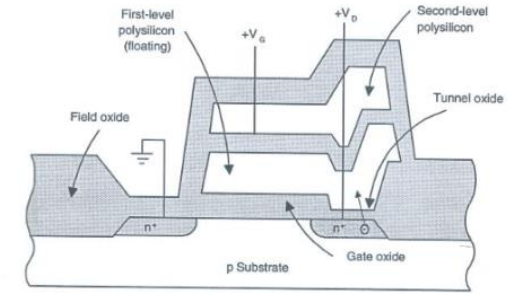


Figura 30: Transistor FLOTOX (EEPROM).

2.3.4 Tecnología Antifusible

Una de las tecnologías empleadas en las FPGAs es la tecnología antifusible. Los antifusibles, figura 32, están compuestos por dos capas conductoras separadas por un aislante de alta impedancia [13]. Al aplicar una tensión elevada entre las dos zonas conductoras el aislante se convierte en un conductor, creando así un "link" o zona de paso. De esta forma se reduce la impedancia. Esta situación es irreversible y por tanto no son reprogramables.

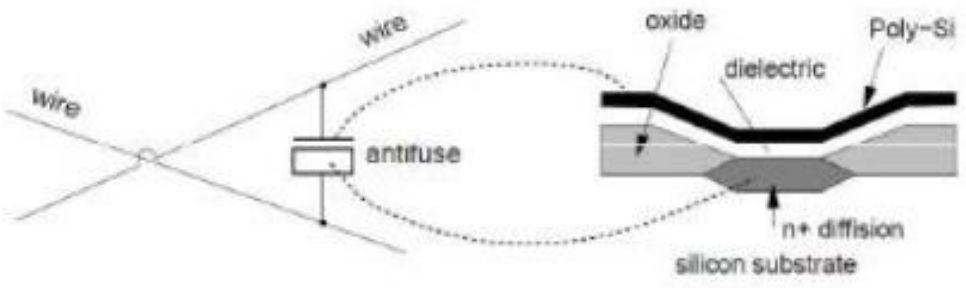


Figura 31: Símbolo electrónico y sección lateral de un antifusible ONO.

2.3.5 Tecnología SRAM

Está tecnología es la que típicamente se usa en los dispositivos FPGAs que hemos analizado en el apartado 2.2.3. La celda básica de la tecnología SRAM se compone de cuatro transistores, formando un biestable [12], [13]. Un biestable es un circuito que almacena un bit de memoria, puede estar en uno de los dos estados lógicos. Además se utilizan otros dos transistores adicionales, como puede verse en la figura 33, para controlar el acceso al biestable durante las operaciones de lectura y escritura. Se trata, por tanto, de una celda reprogramable volátil, ya que la información se pierde al desconectar la alimentación del dispositivo.

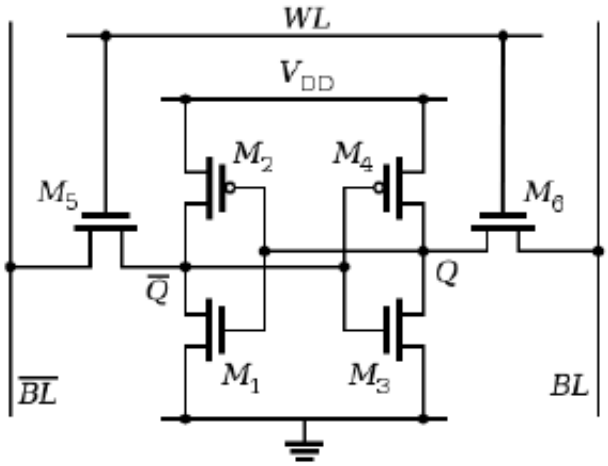


Figura 32: celda de la memoria SRAM compuesta por seis transistores.

2.4 Resumen

Para resumir las tecnologías presentadas en los apartados anteriores se presenta en la Tabla 1 cada dispositivo programable y sus características típicas, como por ejemplo, la tecnología de los elementos programables.

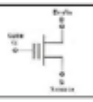
Dispositivo (PLD)	Tecnología del elemento programable	Elemento programable	Símbolo	Método de programación	Reprogramable	Método de borrado	Volátil	Fabricantes
SPLD	Bipolar	Fusible en serie con diodo		Eléctrica en corriente	No	-	No	ATMEL, RHOM semiconductor
CPLD	EPROM	Transistor FAMOS		Eléctrica en tensión	Si	Luz ultravioleta	No	ATMEL, Dinies
CPLD	EEPROM	Transistor FLOTOX		Eléctrica en tensión	Si	Eléctrico	No	Microchip, ATMEL, RHOM semiconductor
FPGA	Antifusible	Antifusible		Eléctrica en tensión	No	-	No	Actel, Quicklogic
FPGA	SRAM	Celda SRAM		Eléctrica en tensión	Si	Eléctrico	Si	Altera, Xilinx

Tabla 1: Características principales de las tecnologías de dispositivos de lógica programable.

2.5 Familia Virtex-5

Una de las familias de FPGAs más utilizadas por los diseñadores en los últimos años ha sido la familia Virtex-5 de Xilinx [14], [15], [16]. La familia de FPGA Virtex-5 necesita una alimentación entre 1.2V y 3.3V dependiendo del dispositivo y la frecuencia máxima de operación puede alcanzar 550MHz. Las Virtex-5 se basan en la arquitectura ASMBL (Advanced Silicon Modular Block) con una interconexión diagonal que proporciona rutas más rápidas para la propagación de las señales internas. Los elementos que contiene una Virtex-5 son:

-Bloques lógicos configurables, CLBs (Configurable Logic Block): son los principales recursos de la lógica para la implementación de circuitos secuenciales y combinacionales. Un elemento CLB está divido en dos segmentos, SLICE, figura 33. Cada SLICE contiene 4 LUTs (Look-up table), 4 flip-flops, multiplexores, un elemento para lógica de acarreo, memoria RAM y registros de desplazamiento. Los dos últimos sólo se encuentran en uno de los dos SLICEs.

-Entradas/salidas (I/O): hasta 1,200 pines de I/O programables.

-Memoria RAM: las estructuras de bloques de RAM tienen una capacidad de 36 Kbit y operan a 550 MHz.

-Reloj: con bloques para la gestión del reloj (CMT - Clock Management Tiles), compuestos por 2 bloques DCM (Digital Clock Manager) y un bucle de enganche de fase (PLL -Phase-Locked Loop). Máxima frecuencia de operación de 550MHz.

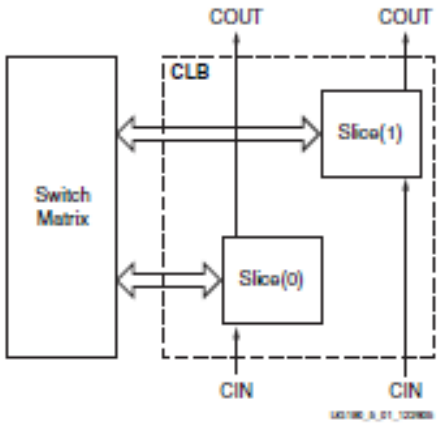


Figura 33: Bloque CLB compuesto por dos Slices.

-DSP (Digital Signal Processor) [17]: los DSPs son bloques programables compuestos por multiplexores, registros, multiplicadores y sumadores. Se pueden realizar diversas funciones: multiplicar, multiplicar y acumular, sumar y multiplicar, sumar tres entradas, multiplexar grandes buses, realizar funciones bit a bit, comparar magnitudes e implementar contadores. Proporcionan una gran flexibilidad y permiten reducir el consumo y aumentar la frecuencia de operación. En la figura 34 se observa un esquema simplificado de la arquitectura interna de un DSP. A continuación, se analizan las señales que se observan en la figura 34:

·A, B son entradas de primer nivel de 30 y 18 bits respectivamente. Aunque A sea de 30 bits el multiplicador es de 25x18bits, el multiplicador selecciona los 25 bits más significativos.
·C es una entrada de segundo nivel de 48 bits. ·A:B es la concatenación de A y B como entrada de segundo nivel.
·PCIN es una señal que permite enlazar dos DSPs.
·P es la señal de salida de otro DSP. Para que esta señal sea la entrada de otro DSP PCIN debe estar activada.
·OPMODE CARRYNSEL y ALUMODE son señales de control. Permiten definir qué operación se desea realizar.

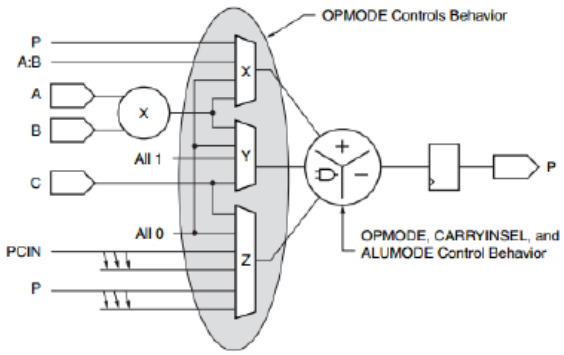
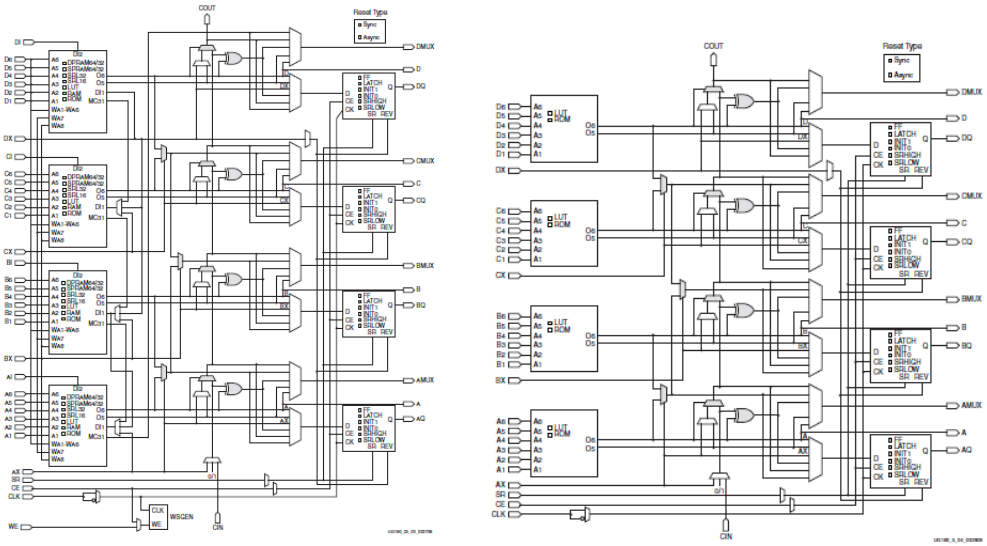


Figura 34: Arquitectura simplificada de un elemento DSP de una FPGA.

Para finalizar, en la figura 35, se puede ver el aspecto de cada uno de los diferentes SLICES. Además en la Tabla 2 se presentan recursos concretos que contiene el dispositivo que se va a utilizar en este trabajo.



(a) (b)

Figura 35: (a) SLICE M. (b) SLICE L.

Table 1: **Virtex-5 FPGA Family Members**

Device	Configurable Logic Blocks (CLBs)			DSP48E Slices[2]	Block RAM Blocks			CMTs[4]	PowerPC Processor Blocks	Endpoint Blocks for PCI Express	Ethernet MACs[5]	Max RocketIO Transceivers[6]		Total I/O Banks[8]	Max User I/O[7]
	Array (Row x Col)	Virtex-5 Slices[1]	Max Distributed RAM (Kb)		18 Kb[3]	36 Kb	Max (Kb)					GTP	GTX		
XC5VLX30	80 x 30	4,800	320	32	64	32	1,152	2	N/A	N/A	N/A	N/A	N/A	13	400
XC5VLX50	120 x 30	7,200	480	48	96	48	1,728	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX85	120 x 54	12,960	840	48	192	96	3,456	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX110	160 x 54	17,280	1,120	64	256	128	4,608	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX155	160 x 76	24,320	1,640	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX220	160 x 108	34,560	2,280	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX330	240 x 108	51,840	3,420	192	576	288	10,368	6	N/A	N/A	N/A	N/A	N/A	33	1,200
XC5VLX20T	60 x 26	3,120	210	24	52	26	936	1	N/A	1	2	4	N/A	7	172
XC5VLX30T	80 x 30	4,800	320	32	72	36	1,296	2	N/A	1	4	8	N/A	12	360
XC5VLX50T	120 x 30	7,200	480	48	120	60	2,160	6	N/A	1	4	12	N/A	15	480
XC5VLX85T	120 x 54	12,960	840	48	216	108	3,888	6	N/A	1	4	12	N/A	15	480
XC5VLX110T	160 x 54	17,280	1,120	64	296	148	5,328	6	N/A	1	4	16	N/A	20	680
XC5VLX155T	160 x 76	24,320	1,640	128	424	212	7,632	6	N/A	1	4	16	N/A	20	680
XC5VLX220T	160 x 108	34,560	2,280	128	424	212	7,632	6	N/A	1	4	16	N/A	20	680
XC5VLX330T	240 x 108	51,840	3,420	192	648	324	11,664	6	N/A	1	4	24	N/A	27	960
XC5VSX35T	80 x 34	5,440	520	192	168	84	3,024	2	N/A	1	4	8	N/A	12	360
XC5VSX50T	120 x 34	8,160	780	288	264	132	4,752	6	N/A	1	4	12	N/A	15	480
XC5VSX95T	160 x 46	14,720	1,520	640	488	244	8,784	6	N/A	1	4	16	N/A	19	640
XC5VSX240T	240 x 78	37,440	4,200	1,056	1,032	516	18,576	6	N/A	1	4	24	N/A	27	960
XC5VTX150T	200 x 58	23,200	1,500	80	456	228	8,208	6	N/A	1	4	N/A	40	20	680
XC5VTX240T	240 x 78	37,440	2,400	96	648	324	11,664	6	N/A	1	4	N/A	48	20	680
XC5VFX30T	80 x 38	5,120	380	64	136	68	2,448	2	1	1	4	N/A	8	12	360
XC5VFX70T	160 x 38	11,200	820	128	296	148	5,328	6	1	3	4	N/A	16	19	640
XC5VFX100T	160 x 56	16,000	1,240	256	456	228	8,208	6	2	3	4	N/A	16	20	680
XC5VFX130T	200 x 56	20,480	1,580	320	596	298	10,728	6	2	3	6	N/A	20	24	840
XC5VFX200T	240 x 68	30,720	2,280	384	912	456	16,416	6	2	4	8	N/A	24	27	960

Tabla 2: Dispositivos familia Virtex-5. Dispositivo en azul el utilizado en este trabajo.

Capítulo 3: APLICACIÓN: RECONOCIMIENTO DE SUELOS DE IMÁGENES SATELITE

3.1 Presentación

En los capítulos 1 y 2 se ha han introducido las redes neuronales artificiales y la lógica programable. En este capítulo se van a utilizar ambas cosas. En primer lugar se va a crear una red ELM con el objetivo de clasificar diferentes tipos de suelos de imágenes de satélites. Es decir, ya no se va a utilizar la red neuronal con el objetivo de aproximación de funciones sino como de clasificador. Una red que funciona como clasificador funciona prácticamente igual que cuando tiene que aproximar funciones. La mayor diferencia es que habrá una salida por clase. Es decir, cada grupo de datos tendrá asociada una salida concreta y la salida con el valor mayor será la que va asociada a ese grupo de entrada de datos.
La aplicación concreta del clasificador va a ser la de clasificación de suelos de imágenes satélite como bien se ha dicho anteriormente. Estas imágenes son del satélite LandSat [18]. Uno de los mayores problemas consiste en obtener los datos de entrada a partir de las imágenes que produce el satélite a partir de las capturas que obtiene cuando está orbitando alrededor de la tierra. En este caso se van a agrupar bloques de 9 pixeles de la imagen. Además, se le asignarán a cada pixel 3 diferentes parámetros, que corresponden a longitudes de ondas diferentes. Es decir, esos parámetros darán la cantidad de color que hay en cada pixel. En resumen, al agrupar 9 pixeles y cada pixel tener 3 diferentes parámetros, el número de datos de entrada será 36. En esta aplicación habrá 6 clases de suelos diferentes, por lo tanto, esos 36 datos de entrada pertenecerán a una de las 6 clases diferentes que hay en la aplicación que se está tratando.

En las figuras 36, 37 y 38 se ven diferentes imágenes capturadas por el satélite que se ha comentado anteriormente:

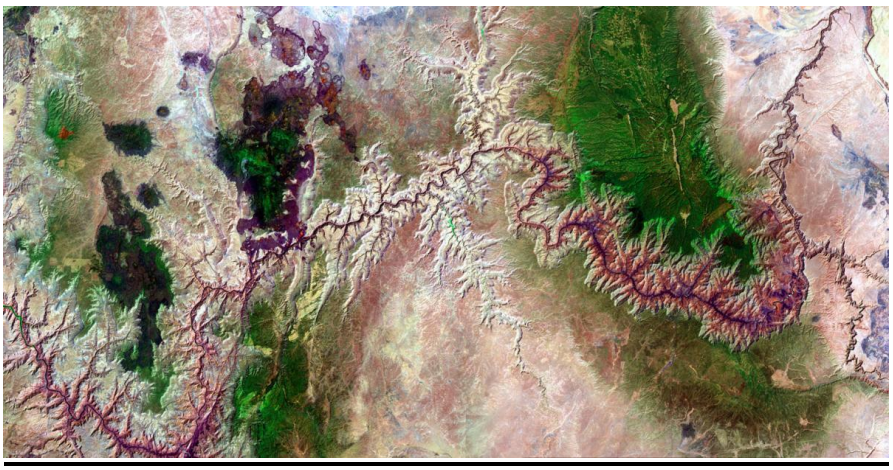


Figura 36: Grand Canyon, Arizona, USA.

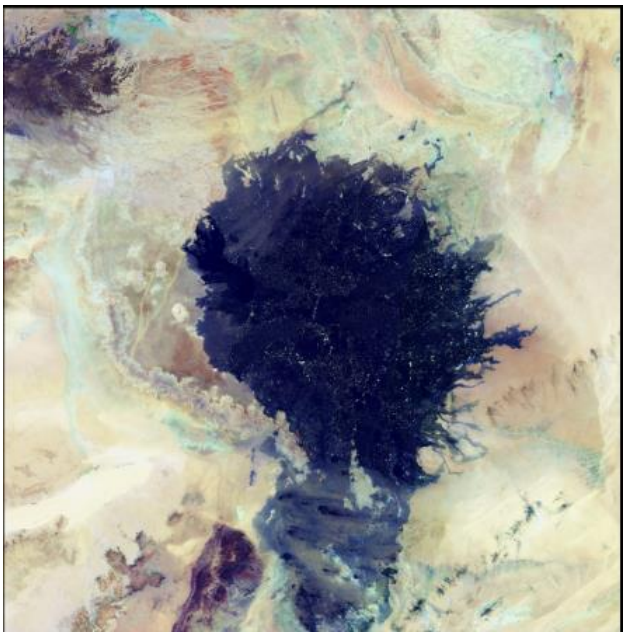


Figura 37: Haruj Volcanic Field, Lybia.

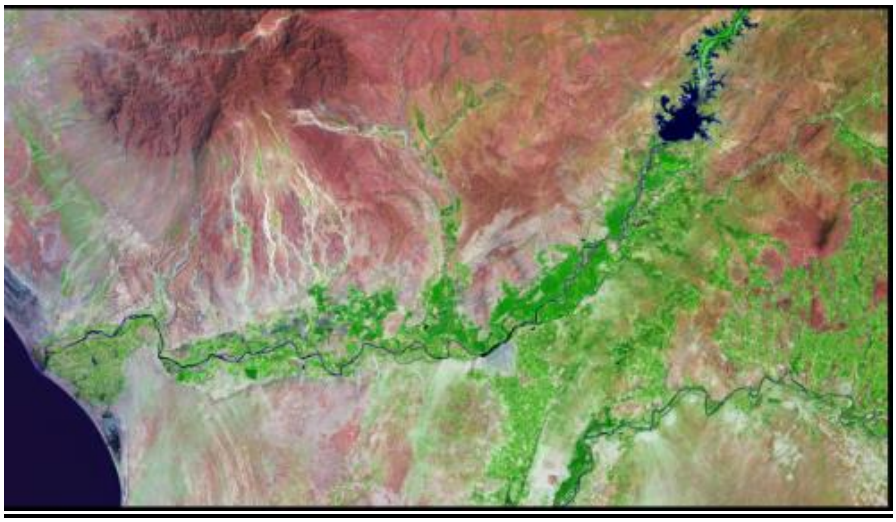


Figura 38: Chira river, Perú.

3.2 Aprendizaje offline

En este apartado se presenta la red neuronal artificial de tipo ELM diseñada con Matlab. Una vez diseñada, se entrena para obtener todos los parámetros necesarios como son los pesos y los bias.

La red consta de 36 neuronas de entrada, una por cada dato de entrada, 20 neuronas en la capa oculta y 6 neuronas en la capa de salida, una por cada clase posible. El número de neuronas de la capa oculta, es decir 20 neuronas, está decidido por dos razones principales. La primera es que hay que tener en cuenta que se quiere implementar en una FPGA el diseño, por ello, no existen

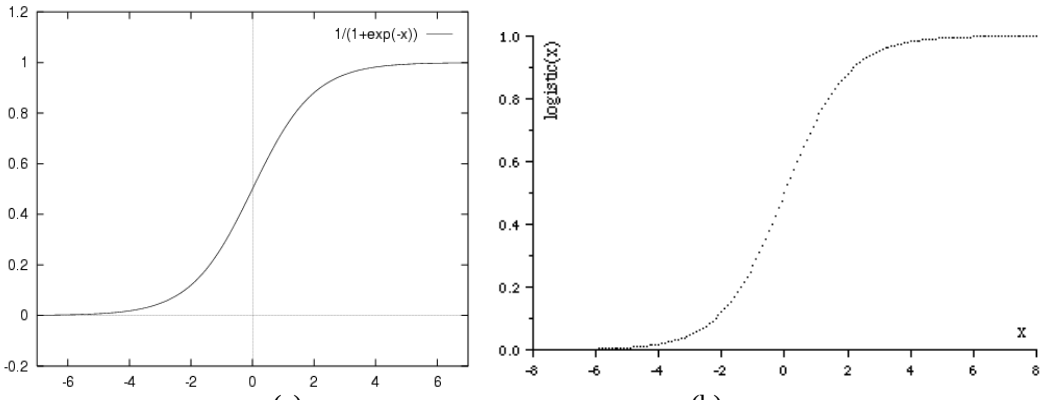
recursos ilimitados. En segundo lugar, que con esa cantidad de neuronas los resultados de test den un alto porcentaje de aciertos. Por último, la función de activación elegida es la sigmoide.
Para realizar el proceso de aprendizaje y fijar los pesos de la capa de salida, es necesario tener una base de datos de los suelos de las imágenes que se quieren reconocer. El profesor G.B. Huang facilita estos datos a través de su página web [19].
Los datos de entrada deben estar normalizados. Es decir, todos los datos tienen que estar entre los rangos [-1, 1]. Los archivos de texto para el entrenamiento y test de la red deben organizarse como se muestra en la figura 39.
La información debe estar dividida en diferentes columnas, en la primera columna deberán aparecer los datos de salida que en este caso será el número de la clase al que están asociados el resto de datos de esa misma línea. Por otro lado, en el resto de columnas aparecerán los datos de entrada. El número de columnas para este caso será 37 por lo tanto.

% Target %	Input Attr 1	Input Attr 2	Input Attr 3	Input Attr 4	Input Attr 5	Input Attr 6	Input Attr 7	Input Attr 8	Input Attr 9
7	-0.38462	-0.34545	-0.70115	-0.63636	-0.26154	-0.34545	-0.6	-0.59375	-0.28125
3	0.630769	0.545455	0.310345	-0.05785	0.753846	0.545455	0.452632	-0.04688	0.75
2	-0.13846	-0.12727	-0.01149	-0.23967	0.107692	0.018182	-0.03158	-0.15625	0.21875
3	0.353846	0.236364	0.103448	-0.17355	0.476923	0.236364	0.073684	-0.15625	0.34375
2	0.261538	0.054545	-0.14943	-0.42149	0.015385	0.054545	-0.07368	-0.32813	0
4	0.138462	0.054545	0.034483	-0.28926	0.015385	-0.01818	-0.24211	-0.32813	-0.125
4	0.076923	0.090909	-0.28736	-0.38843	0.076923	0.090909	-0.11579	-0.35938	0.0625
1	-0.26154	0.236364	0.195402	-0.05785	-0.26154	0.309091	0.263158	-0.04688	-0.28125
1	-0.13846	0.545455	0.517241	0.07438	-0.13846	0.472727	0.452632	0.078125	-0.03125

Figura 39: Ejemplo codificación.

Una vez se tienen los datos en la codificación necesaria para realizar el aprendizaje, se obtienen los pesos y los bias implementado el algoritmo ELM a través de un programa en Matlab. Para realizar el test se ha utilizado la base de datos y el mismo algoritmo ELM proporcionado por Huang, ya que el algoritmo ELM es capaz de realizar el entrenamiento y el test.
Antes de realizar el test, hay que tener en cuenta que los resultados obtenidos a través de Matlab no van a ser completamente iguales que si los obtuviésemos a través de un test mediante la red neuronal que se implementará en una PFGA. Por ello, antes de realizar el testeo se va a intentar que este mismo se lo más real posible. Para ello, se va a modificar la función sigmoide que utiliza el algoritmo ELM. Esta función es la que proporciona la plataforma Matlab y con una precisión muy superior a la que se va a implementar. Por ello, mediante Matlab se va a diseñar una función sigmoide con una precisión igual que la que se va a implementar en la FPGA. En la figura 40 se puede ver la diferencia entre una función sigmoide continua y una función sigmoide similar a la que se ha implementado en la FPGA. Los datos de la función diseñada son los siguientes: datos de entrada de 8 bits en complemento a dos, 4 para la parte entera y 4 para la fraccional. Los valores de la sigmoide son de 6 bits, todos ellos son la parte fraccional.
Por último, y después de realizar estas modificaciones, se conseguirá un resultado del test más real. Aun así se compararán los dos resultados, el primero de ellos con la función predefinida en

Matlab y en segundo lugar con la que se ha discretizado a semejanza de la que se va a implementar en la FPGA. Los resultados son los siguientes:
81% de acierto para la función sigmoide de Matlab y 78% para la función sigmoide discretizada.



(a) (b)

Figura 40: (a) Función Sigmoide continua (b) Función discretizada.

3.3 Implementación Hardware de la Red Neuronal

3.3.1 Arquitectura

Mediante el programa de diseño de Xillinx ISE se ha creado la red neuronal de tipo ELM que se va a usar para clasificar los suelos de las imágenes satélite. Esta red se va a implementar en el dispositivo XC5VLX50T de la familia Virtex-5.
Para una buena organización y comprensión, la red se ha dividido en diferentes estructuras. Se va a empezar a describir la red comenzando por el nivel superior denominado Red_completa.

i) Bloque "Red_completa"

Este bloque superior consta de 4 entradas y 6 salidas. Red_completa se ha dividido en dos subestructuras: El contador (el controlador que maneja los dispositivos de la red en cada momento) y las neuronas. Las entradas y salidas de este bloque se distribuyen internamente para poder llegar a todos los bloques que sea necesarios. Las señales de entrada y salida son las siguientes:

·Entrada_1: Datos de las imágenes de satélite.
·Clock: Reloj de todo el sistema
·Enable: Activa o desactiva los elementos internos de la red.
·Reset: Resetea todos los registros de la red.
·Salidas: 6 diferentes, una por cada neurona de salida. Es decir, una por clase.

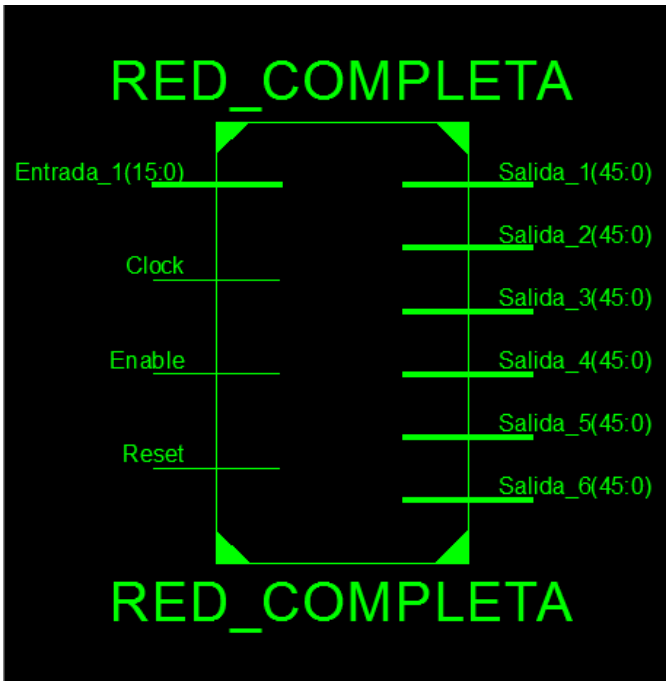


Figura 41: Entradas y salidas de la red neuronal.

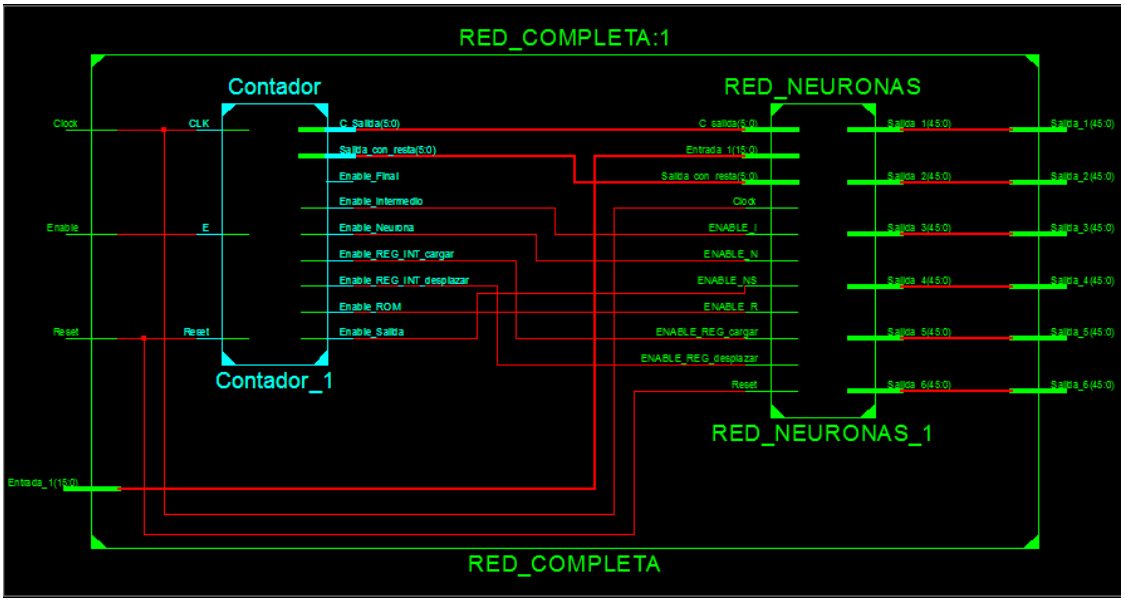


Figura 42: Los dos bloques en los que está dividida la Red completa. Contador y neuronas.

ii) Bloque “Red neuronas”

El bloque Red neuronas está dividido en 3 módulos. El primero de ellos es la capa oculta de neuronas. Esta capa contiene las 20 neuronas que se han asignado para esta capa. El segundo bloque es un registro que se encarga de guardar y suministrar los datos de salida de la capa oculta a la capa de salida o neuronas de salida. Y por último la capa de salida que contiene 6 neuronas (ver figura 43).

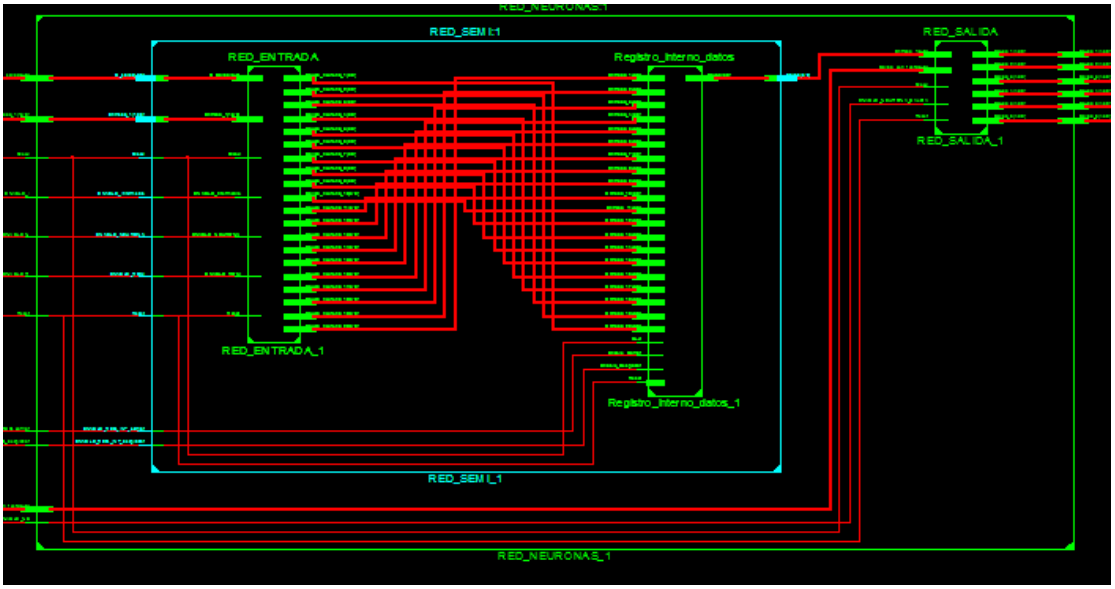


Figura 43: Los tres bloques en los que está dividido el bloque de neuronas.

iii) Bloque Red_entrada

El bloque contiene las 20 neuronas de la capa oculta. Cada neurona debe recibir su dato de entrada y además el peso asociado. Por ello, como ya están calculados los pesos gracias al proceso de entrenamiento realizado en el anterior apartado, se guardarán los pesos asociados a cada neurona para cada dato en un bloque ROM que se conectará a una de las entradas de la neurona. Además, en el caso de la capa oculta se guardará en último lugar de la ROM el bias también y se insertará en las neuronas con un dato de entrada de valor unidad. Por lo tanto cuando los 36 datos de entrada se hayan enviado a la neurona, se enviará un último dato de valor unidad al mismo tiempo que el último dato que hay almacenado en la ROM, que en este caso será el bias. Esta estructura es la misma que se encontrará en la capa de salida de neuronas con excepción del bias que no aparece en las neuronas de salida.

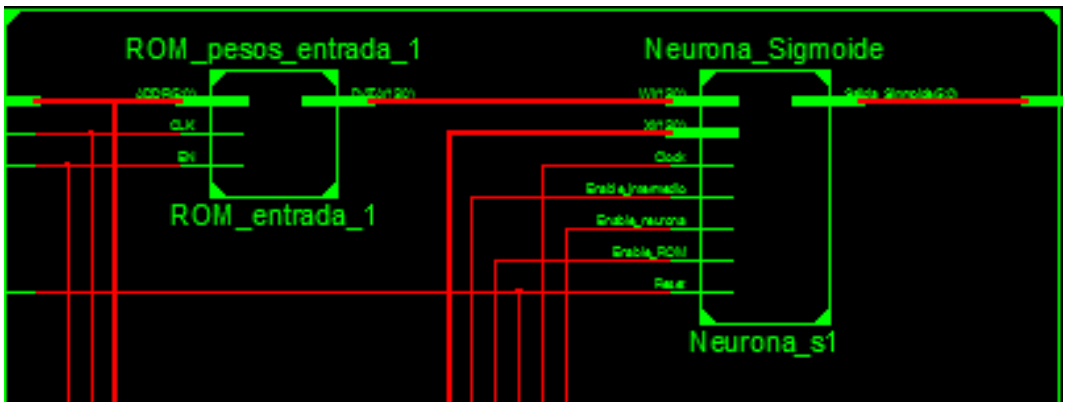


Figura 44: Una de las neuronas de la capa de entrada con su respectiva ROM que contiene los pesos y bias.

iv) Bloques de neuronas: “Neuronas_sigmoide” y “Neurona pura”

Se tiene dos tipos de neuronas. Las de la capa oculta contienen la parte de la función de activación al contrario que las de salidas que no tienen función de activación.
En primer lugar se tiene la neurona “pura”, la cual se encarga de multiplicar y sumar los datos de entrada. En segundo lugar hay un bloque que hace que la salida de la neurona sea compatible con la precisión de nuestra función sigmoide. Ésta viene representada con el tercer bloque que es el ROM.
Por lo tanto, las neuronas de la capa oculta tendrán estos tres bloques comentados anteriormente, en cambio, las neuronas de salida solamente tendrán el bloque el cual se ha definido como neurona “pura”.

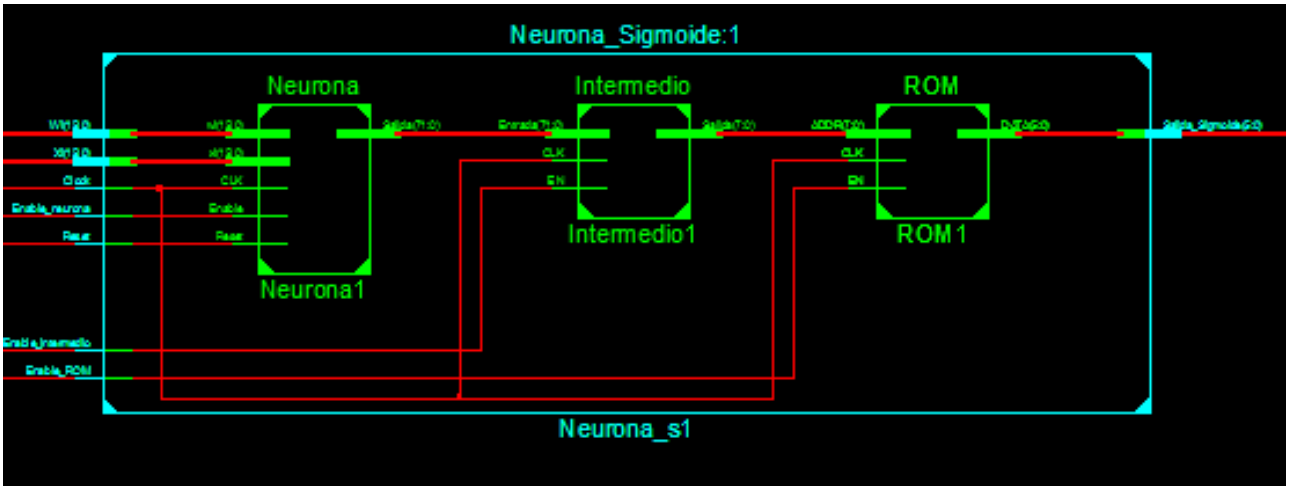


Figura 45: Bloque de una de las neuronas de la capa oculta. La neurona de salida es idéntica pero sin los bloques intermedio ni ROM.

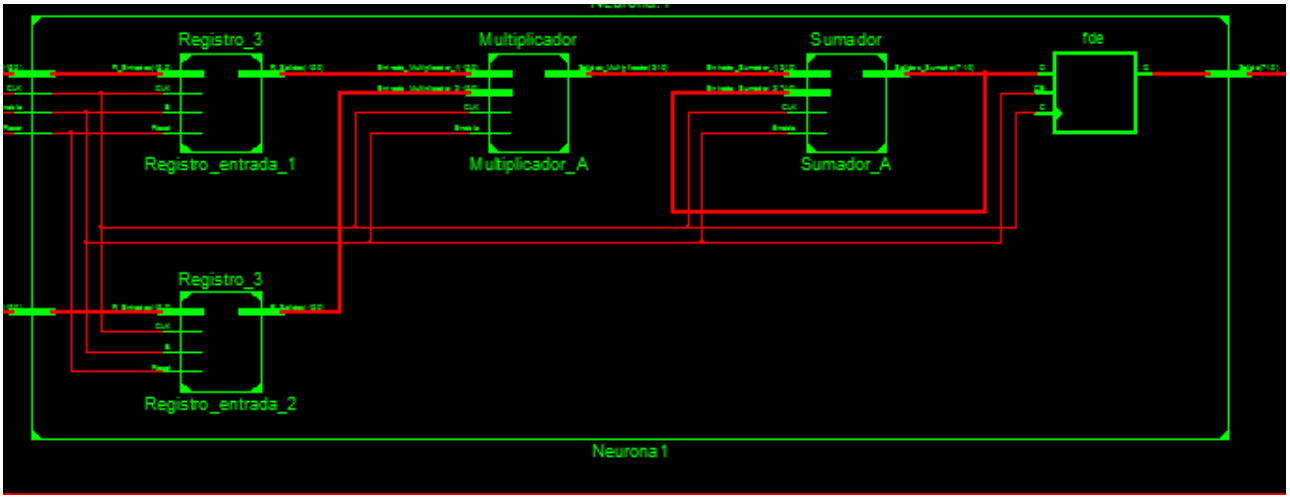


Figura 46: Neurona “pura”. Compuesta por un multiplicador, un sumador y dos registros.

v) Bloque Registro_interno

El registro interno se encarga de recibir los datos de las neuronas de la capa oculta y de transferirlos a los de salida. Es un registro de desplazamiento. Se compone de 20 registros individuales. Estos registros individuales contienen dos entrada de datos, una única salida y dos enables. Gracias a ellos la señal de salida será una de las dos entradas. Una de las entradas está recibe un dato que llega de fuera del Registro_interno, en cambio, la otra entrada está conectada a la salida del anterior registro conectado en serie. Por lo tanto nuestro registro interno recibirá los 20 de datos de las neuronas en paralelo activando el enable correspondiente de cada registro individual. Y después activando el otro enable los datos se irán desplazando por el registro interno pasando por cada registro individual. Y así se transmitirán en modo serie los datos a la capa de salida.

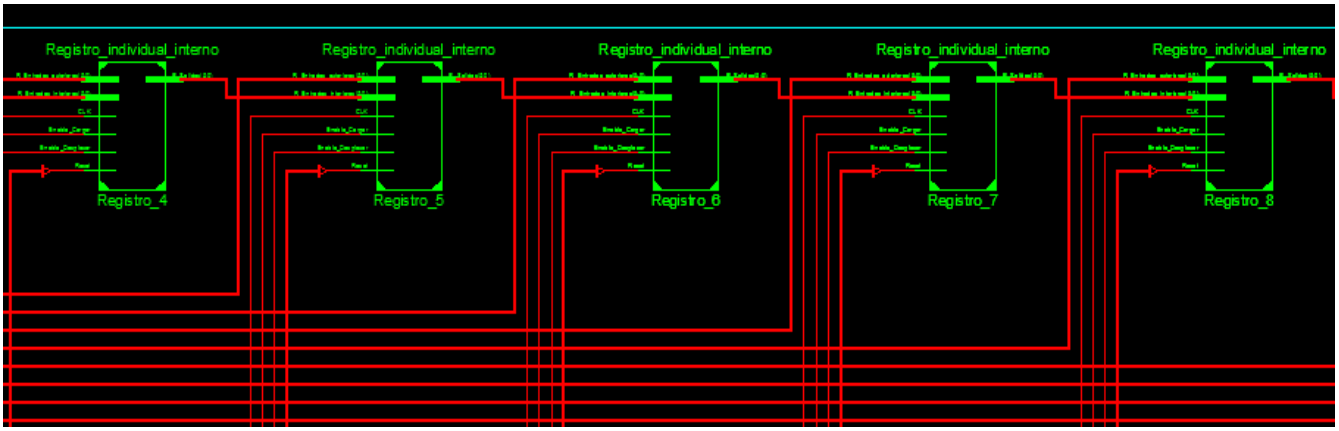


Figura 47: 5 de los 20 registros individuales conectados en serie en el registro interno.

3.3.2 Tipos de datos, características temporales y ocupación

El número de bits de cada señal y su codificación es importante para poder realizar su función. Como señales de entrada se tienen el reset, enable y el clock con 1 bit. La señal de entrada llamada entrada_1 es de 16 bits en complemento a dos con solo el primer bit entero. En la salida se tienen 6 señales iguales de 46 bits en complemento a dos y de esos 46 bits, los 22 bits más significativos como parte entera.

Además de las señales de entrada y salida, la red neuronal tiene señales internas las cuales conectan los diferentes componentes, por ello, es necesario analizar estas también. En primer lugar encontramos los pesos de la primera capa que son de la misma precisión y codificación que las señales de entrada de datos. Las señales de salida de la primera capa de neuronas, es decir, los las señales que salen de las funciones de activación, son de 6 bits sin complemento a 2 y todo ello fraccional. Los pesos de la segunda capa son de 19 bits en complemento a 2 con los 4 bits más significativos enteros.

Por último la señal del contador que sirve para controlar cada uno de los diferentes elementos es de 6 bits sin complemento a 2 y todos los bits enteros.

Después de implementar el diseño en el dispositivo XC5VLX50T de la familia Virtex-5 de Xillinx, se han analizado sus características temporales. Las características temporales del diseño son también importantes, ya que puede limitar su funcionamiento. Estudiando la mayoría de caminos más largos que pueden recorrer las señales, el tiempo medio de propagación es de 5ns aproximadamente, es decir, ese es el retardo que puede limitar el funcionamiento de nuestro diseño. Se ha calculado el periodo mínimo y la frecuencia máxima para el diseño. En la figura 48, se puede observar una captura de imagen del interfaz gráfico del Timing analyzer que facilita la plataforma ISE de Xillinx. Como puede verse, la frecuencia de operación máxima es 168.095 MHz.

```
Timing summary:
---------------

Timing errors: 0  Score: 0  (Setup/Max: 0, Hold: 0)

Constraints cover 17079 paths, 0 nets, and 1577 connections

Design statistics:
   Minimum period:  5.949ns{1}   (Maximum frequency: 168.095MHz)
```

Figura 48: Características temporales.

En tercer lugar y para acabar este apartado se hablará sobre la ocupación de la red neuronal artificial que se ha creado y que se ha implementado en la FPGA.
La herramienta ISE de Xillinx proporciona una tabla de 5 columnas y diferentes filas. En las filas se encuentran cada uno de los diferentes recursos
La ocupación de los recursos más importantes es la siguiente:

-SLICE Registers: el nº de SLICE Registers usados es de 2858. La FPGA contiene 28800. Por lo tanto se ha utilizado un 9% de la capacidad.
-SLICE LUTs: el nº de SLICE Registers usados es de 3455. La FPGA contiene 28800, por lo tanto se ha utilizado un 11% de la capacidad.
-DSP48Es: el nº de SLICE Registers usados es de 26. La FPGA contiene 48. Por lo tanto se ha utilizado un 54% de la capacidad.

3.3.3 Simulación

En este apartado se presentan las simulaciones de la red neuronal. Es decir, se va a verificar que para un grupo de datos de entrada, la red neuronal consigue identificar el suelo de la imagen satélite correctamente. Lo primero que hay que explicar es que la velocidad de reloj utilizada en las simulaciones ha sido de 100 MHz. En el caso concreto que se ha analizado, el grupo de datos de entrada está asociado a la clase 3.
Además de exponer y explicar la simulación de la red completa, se añadirán las simulaciones de 3 bloques más, los que son la base de la estructura de la red neuronal y los más importantes. Estos tres bloques son: El contador, la neurona “pura” y la neurona con la función de activación

sigmoide. Para poder apreciar a simple vista que estos tres bloques principales funcionan bien, se utilizarán datos de entrada que lo hagan posible.

i) Contador o controlador (Figura 49)

Se pueden apreciar diferentes fases en la simulación del contador. La primera fase es en la que los datos están entrando a la neurona y se están multiplicando y sumando. En esta fase tenemos habilitada la señal enable neurona, con 39 flancos ascendentes de duración (36 datos + bias+2 ciclos en llegar a la salida de la neurona el primer dato). Luego una fase en las que los datos pasan por el convertidor y por la sigmoide donde sus enables son convertidor y rom. Más adelante pasan al regisitro interno con el enable cargar y se van desplazando por él con el enable desplazar, finalmente los datos entran a las neuronas de salida con su enable neurona_salida el cual se activa en el mismo momento que se desplazan los datos por el registro interno. Este sería un proceso de 22 flancos (20 datos +2 ciclos en llegar a la salida de la neurona el primer dato) salida.

ii) Bloque neurona (Figura 50 y 51)

El bloque neurona, el cual se refiere al que en arquitectura se ha llamado como neurona “pura”. Se utilizan como datos de entrada (el peso y los datos) -1 y -1 para poder apreciar a simple vista los resultados. De este modo se puede ver como se convierten en 1 al multiplicar (El resultado de la multiplicación pasa a tener dos bits para los enteros) y luego se van sumando de uno en uno y así se puede apreciar cómo va aumentado como si fuese un contador la salida.

Por último en la neurona sigmoide lo que se hace es utilizar los mismos datos de entrada. El objetivo de esta simulación es ver cómo pasan los datos por el convertidor, es decir, como la precisión disminuye para poder acceder a la ROM donde está implementada la función sigmoide. Y leer para ese dato, la salida de la sigmoide.

iii) Simulación de la red completa (Figura 52)

Como bien se ha dicho antes mediante 36 datos de entrada la red tendrá que predecir qué clase de superficie es, a priori en el caso que estamos analizando se sabe que es la tercera ya que se tienen los resultados de antemano. Por lo tanto, respecto a la simulación, se puede ver una primera fase donde entran los 37(36datos+la unidad para el bias) datos, es decir 37 flancos, más los dos flancos que tarda el primer dato en llegar a la salida, luego una fase en la que los datos pasan por el convertidor, la rom y el registro interno. Y la última fase cuando los datos pasan por la neurona de salida. Se puede apreciar que en la última parte debería haber 20 cambios de señal en la señales de salida. Esto no ocurre porque los datos de salida de las neuronas de la capa oculta pueden ser 0, en el caso de que el dato que entre a la función de activación haga que el valor de salida sea 0. Para acabar se ve como la salida 3 es la salida en la que el número es el mayor de los 6, ya que en los que comienzan por 1, los datos son negativos, debido a la codificación que hemos utilizado en las señales de salida (complemento a 2). En este caso la salida 3 será 0.75 y la 2 aproximadamente 0.5 (Los 22 bits más significativos son la parte entera contando el signo). Como puede verse, la simulación es totalmente compatible con los procesos del controlador y eso hace ver que la sincronización es correcta.

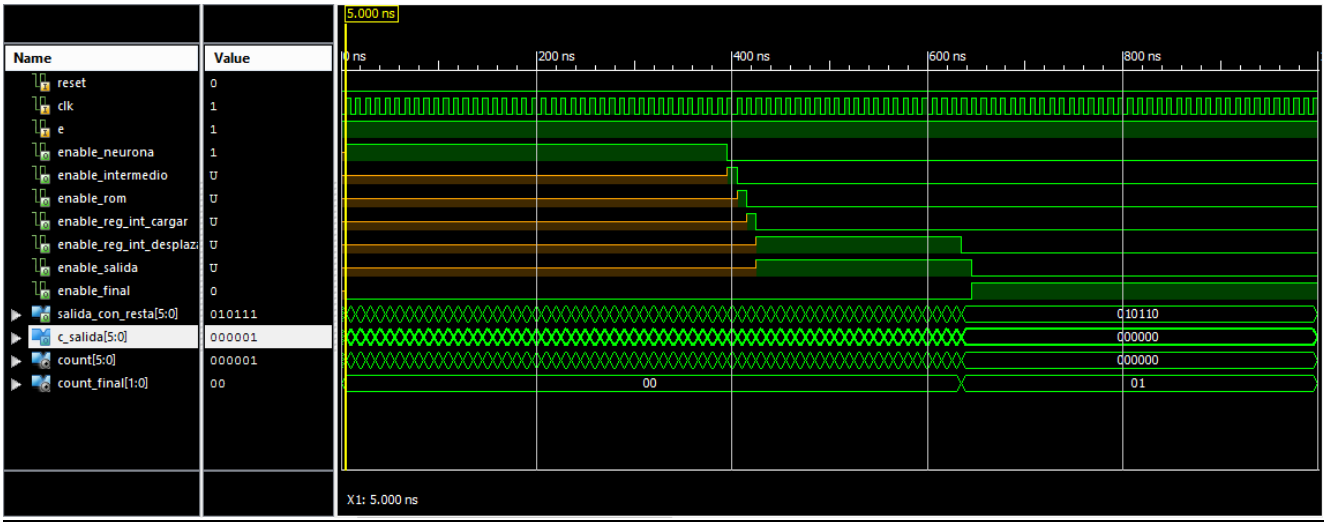


Figura 49: Simulación del contador.

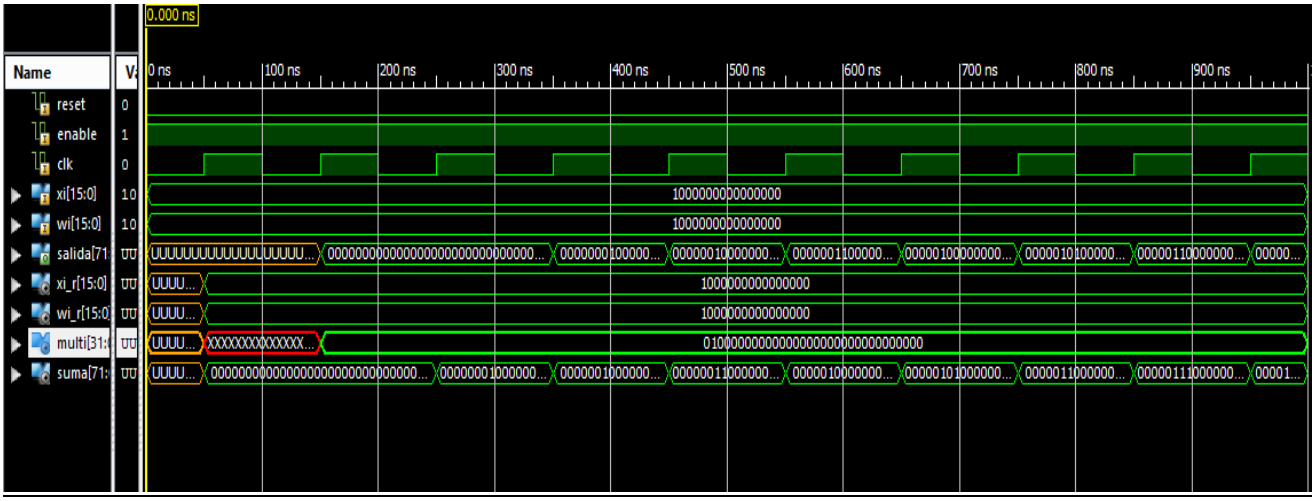


Figura 50: Simulación de la neurona “pura”.

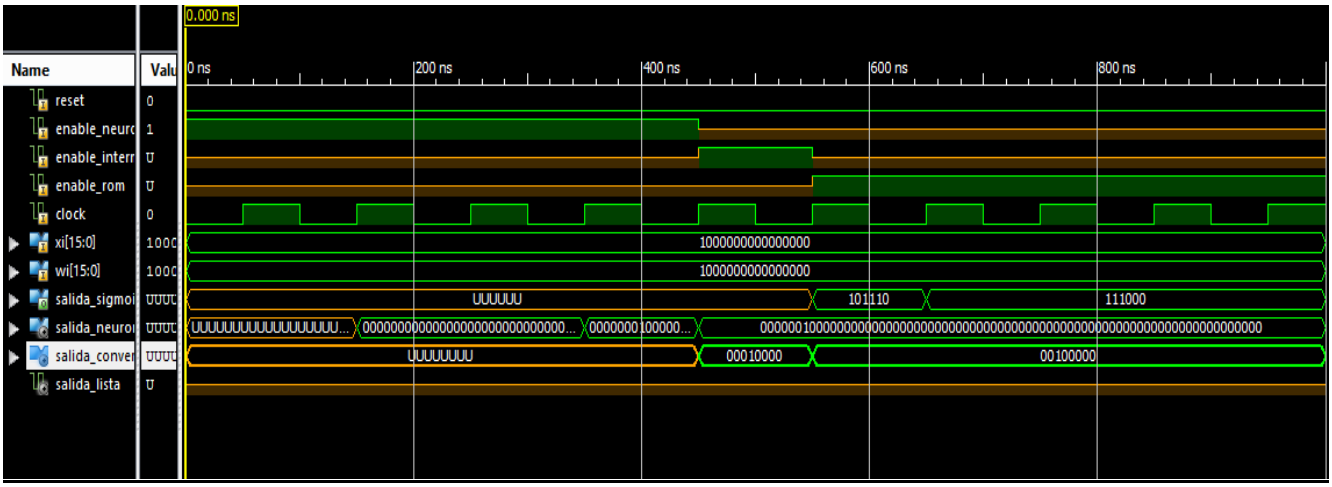


Figura 51: Simulación de la neurona sigmoide.

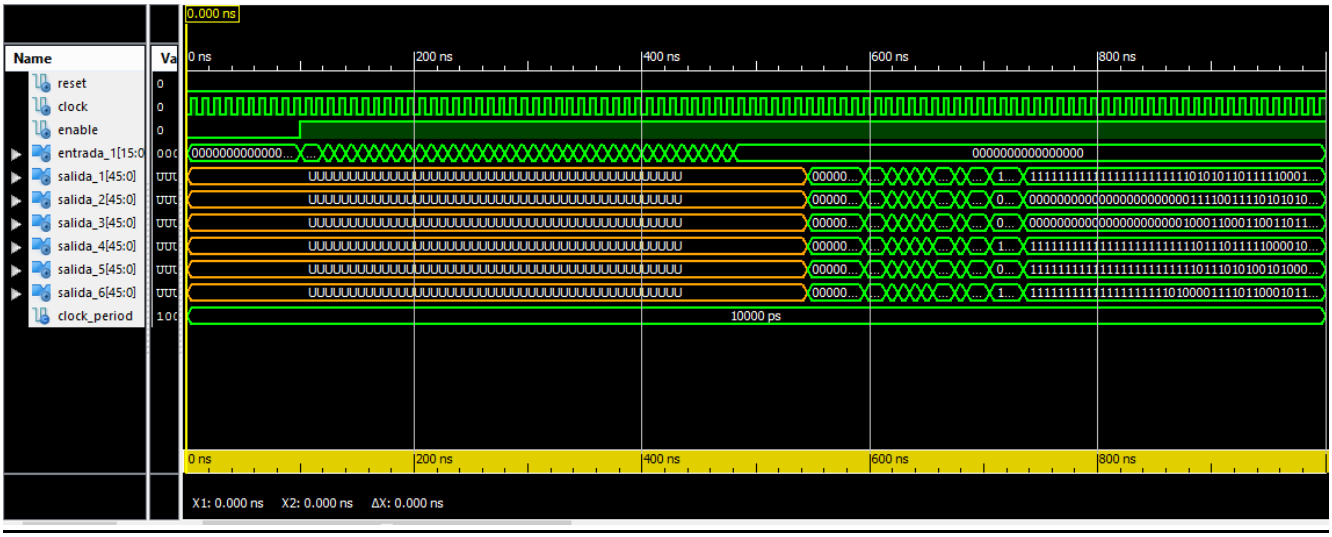


Figura 52: Simulación de la red completa.

CONCLUSIONES

Se han cumplido los tres objetivos principales de este trabajo, estos son: el estudio de las redes neuronales artificiales, el estudio de dispositivos reconfigurables, entre ellos, los de alta velocidad (como son las FPGAs) y su aplicación a un ejemplo práctico como es el reconocimiento de suelos de imágenes satélite.

En el capítulo 1 se han analizado tres diferentes arquitecturas de redes neuronales, MLP, RBF y ELM. Con cada una de ellas se han realizado aproximaciones de funciones de una y dos variables. Se ha visto que las redes ELM son las más indicadas para nuestra aplicación. La estructura es muy parecida entre las redes MLP y ELM, ya que la única diferencia es que las neuronas de salida no tienen bias ni función de activación. Por otro lado, el error cometido para un número de neuronas concreto es mayor que para una red RBF en general. Por lo tanto, la razón principal de utilizar la red ELM es el proceso de aprendizaje. Las ventajas respecto al entrenamiento BP son las siguientes: se pueden fijar los pesos y los bias antes de tener los datos de entrenamiento ya que se calculan aleatoriamente y lo único que hay que obtener durante el proceso de aprendizaje son los pesos de salida. Además de estas dos ventajas también hay que decir que el proceso de aprendizaje ELM no funciona por iteraciones como en el caso del BP. Estas características hacen que el proceso de entrenamiento sea mucho más rápido y sencillo. Por lo tanto, la red ELM es la más indicada para las aplicaciones en las que el proceso de aprendizaje y test se realice en tiempo real.

En Capítulo 2 se han estudiado los diferentes tipos de circuitos integrados que disponibles actualmente. Además se han analizado en profundidad los dispositivos programables y las tecnologías de programación de los mismos. Se concluyó que las FPGAs son los dispositivos que mejor se adaptan a las necesidades de velocidad, área y consumo para la aplicación, permitiendo realizar modificaciones una vez se ha implementado el diseño.

En el Capítulo 3 se ha diseñado la red ELM e implementado en un dispositivo FPGA. Una vez definida la red, mediante el programa ISE de diseño de Xillinx, se han creado mediante el lenguaje VHDL todos los componentes necesarios para poder implementar la red diseñada en Matlab en una FPGA. Se ha realizado un diseño modular, es decir, lo primero que se han creado han sido los subcircuitos con funciones más simples y después se han ido utilizando esos subcircuitos para ir creando módulos más complejos en base a un diseño jerárquico. Otra de las tareas que se ha realizado ha sido importar los parámetros obtenidos en el proceso de aprendizaje mediante la herramienta Matlab. Asismismo, ha sido necesario transformar el formato de los datos convirtiéndolos a códigos binarios fraccionales teniendo en cuenta de que tipo de datos se trataban. Otra de las conclusiones de este apartado es que a pesar de no utilizar un dispositivo de alta capacidad dentro de la familia Virtex-5, como hemos visto en el capítulo 3, todavía se podrían implementar más bloques para poder mejorar el diseño y el rendimiento de la aplicación. Por otro lado, se han realizado diferentes simulaciones para comprobar el correcto funcionamiento de la red neuronal implementada tanto de las entradas y salidas como de las señales internas más relevantes. Se podrían hacer análisis más exhaustivos analizando todas las señales internas y viendo cómo van cambiando a lo largo de la simulación. Sin embargo, la complejidad de esta tarea es inviable. Para finalizar sabiendo la velocidad del clock con la que

hemos hecho las simulaciones y observando el tiempo que tarda en realizar todo el ciclo de reconocimiento se puede decir que cada 750ns el sistema sería capaz de realizar el reconocimiento de 9 píxeles.

BIBLIOGRAFÍA

[1]Web: https://en.wikipedia.org/wiki/Internet_of_Things

[2] Web: https://es.wikipedia.org/wiki/Red_neuronal_artificial

[3] Guang Bin Huang, Extreme Learning machines: Learning Without Iterative Tuning, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, a Tutorial in IJCNN2012/WCCI2012, Brisbane, Australia, June 10 2012

[4] J.S.R.Jang, C.T.Sun and E.Mizutani, Neuro-Fuzzy and Soft Computing, Prentice Hall, UPPER SADDLE RIVER, NJ, USA, 1997.

[5] J.WESLEY HINES, Fuzzy and Neural Approaches in Engineering, John Wiley and Sons, New York, NY, 1997.

[6] Web: http://proton.ucting.udg.mx/posgrado/cursos/idc/neuronales2/AntecedentesP.htm (historia)

[7] Heikki N.Koivo, Neural networks: Basic using MATLAB; Neural Network Toolbox, febuary 1, 2008.

[8] Inés del Campo Hagelstrom, apuntes de la asignatura Diseño de Sistemas Digitales, grado en Ingeniería electrónica, Universidad del País Vasco UPV/EHU.

[9] Peter J. Ashenden, "Digital Design:An Embedded Systems Approach Using VHDL", California: Morgan Kaufmann, 2008.

[10] Jose L. Martín Gonzalez, "Electronica digital", Madrid, Spain: Delta publicaciones, 2007.

[11] Adel S. Sedra and Kenneth C. Smith, "Microelectronic Circuits", New York: Oxford University Press, 2010.

[12] Kevin Skahill, "VHDL for programmable logic" Massachusetts :Addison-Esley, 1996.

[13] Cristian Sisterna. *Field programmable gate arrays* (FPGAS). [online]. Available: dea.unsj.edu.ar/sisdig2/

[14] Web: http://www.xilinx.com

[15] Xilinx. (2012, March, 16). *Virtex-5 FPGA User Guide*. [online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug190.pdf

[16] Xilinx. (2009, February, 6). *Virtex-5 Family Overview Virtex.* [online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf

[17] Xilinx. (2012, January, 26). *Virtex-5 FPGA XtremeDSP Design Considerations.User Guide*. [online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug193.pdf

[18] Web: http://landsat.usgs.gov/

[19] Web: http://www.ntu.edu.sg/home/egbhuang/