UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

SIMULADOR DE LAS REDES DE BASE RADIAL

Autor: Víctor Serrano Buitrago

Tutor: Inés Mª Galván León

Agradecimientos

Quiero expresar mi enorme agradecimiento a mi madre, Felicidad, por su ayuda y comprensión en los momentos más difíciles y por el enorme esfuerzo que ha realizado a largo de toda su vida con el único objetivo de ofrecerme un buen futuro y darme la mejor educación posible.

Gracias a mi hermana Laura, por sus sonrisas, que muchas veces me hacían reír sin importar lo duro que fuese el día. Gracias a Sandra, que me ha aportado una gran confianza en mí mismo, y a su empuje incondicional, consiguiendo que supere con éxito gran parte de esta carrera. Gracias también a mis abuelos, a mis tíos, tías y primos por sus consejos que siempre me han ayudado.

Quiero agradecer también a Sergio, mi mejor amigo, su contagio de felicidad y energía que me ofrece siempre otro punto de vista sobre las cosas. También a Marisol, Felipe, Ángel, Javi, Seve y Damián, por preocuparse tanto por mí y por su entrega. Agradezco a todos mis amigos los grandes momentos que hemos pasado juntos y su disposición para cualquier cosa. Porque sé que siempre estarán ahí a pesar del paso del tiempo.

También a todas aquellas personas que me han dado la oportunidad de desarrollarme tanto intelectualmente como personalmente.

Por último, quiero agradecer a mi tutora Inés su comprensión, paciencia y dedicación conmigo, sin las cuales este trabajo no hubiera sido posible.

Resumen

Las redes de neuronas de base radial son un tipo de redes de neuronas que son utilizadas para la resolución de problemas de aproximación, predicción o clasificación. Su uso proporciona soluciones que en muchos casos suponen nuevas perspectivas sobre un problema en cuestión.

En este trabajo se ha desarrollado un simulador de redes de base radial, que emula el funcionamiento de dichas redes ante patrones que representan un problema en concreto. El simulador se ha desarrollado en JAVA. Además, el principal objetivo es proporcionar un simulador para uso docente, sencillo y eficaz.

El desarrollo de este simulador conlleva varias actividades asociadas, que se expondrán de una manera detallada a lo largo de la memoria, como el análisis y el diseño del sistema, base teórica sobre la que se basan las redes de neuronas de base radial o un conjunto de pruebas que verifican el correcto funcionamiento del simulador.

Contenido

Índice de figuras

Índice de tablas

Capitulo 1: Introducción

1.1. Introducción a las redes de base radial

Las redes de neuronas artificiales son una simulación de los sistemas nerviosos, que están compuestos por neuronas conectadas entre sí, de ahí su principal similitud. En general, las redes de neuronas artificiales se caracterizan por disponer de un aprendizaje adaptativo, de tal forma que sean capaces de aprender a realizar una tarea en base a un entrenamiento específico, y adaptarse a los cambios recibidos como nuevas informaciones. También pueden crear su propia representación de la información que se recibe durante el aprendizaje. Por último, son capaces de realizar generalizaciones si se elige bien la estructura y se realiza un entrenamiento adecuado.

Por lo general, se suelen encontrar tres tipos de neuronas, las de la capa de entrada, que se encargan de transmitir los datos recibidos; las de la capa oculta, que procesan la información, lugar donde se produce el aprendizaje, y las de la capa de salida, que se encargar de mostrar la salida de la red ante la información recibida. En la figura 1.1 se puede observar la estructura comentada, por la cual se rigen las redes de base radial.

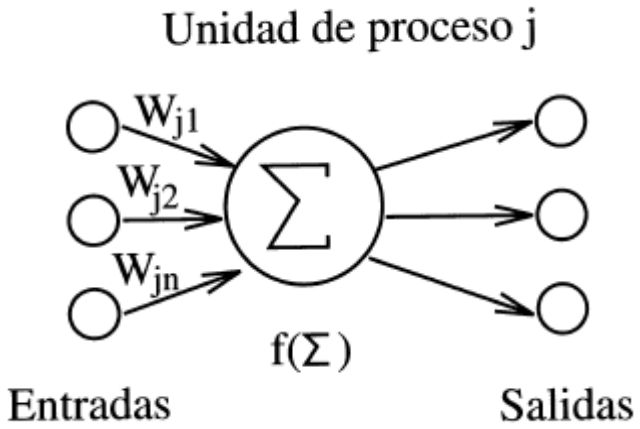

Figura 1.1: Neurona artificial

Las redes de base radial son un tipo de red de neuronas artificiales. Tienen su origen entre 1989 y 1990 gracias a Moody y Drakern [Moody and Darken, 1989], Renals [Renals, 1989], y también a Poggio y Girossi [Poggio and Girosi, 1990]. Su principal objetivo era disponer de redes capaces de resolver problemas de aproximación y debido a su rapidez en la resolución de dichos problemas, se estimó su utilización para aplicaciones en tiempo real. Se caracterizan por ser unas redes multicapa que se propagan hacia delante, obteniendo una salida que varía en función del cálculo de las distancias de un punto denominado centro. Sólo presentan una única

capa oculta, en las que las neuronas presentan un carácter local. De tal modo que se obtienen tiempos más reducidos debido al carácter local de las neuronas ocultas de la red, permitiendo que tan solo unas pocas neuronas de dicha capa tengan que ser procesadas para nuevos patrones de entrada. Dicho carácter local se debe, principalmente, a las funciones de activación que se utilizan en las redes de base radial, que mayoritariamente se trata de la función gaussiana. Las salidas se encargan de realizar una combinación lineal de los resultados de las activaciones de las neuronas de la capa oculta.

Por otra parte, se asemeja con el Perceptrón Multicapa en su uso como aproximador universal, lo cual fue demostrado en 1991 por Park y Sandberg [Park and Sandberg, 1991]. Además, las funciones de base radial que se utilizan en estas redes, llegan a definir hiperesferas o hiperelipses que dividen el espacio de entrada, de modo que cada neurona forma una aproximación local no lineal de una determinada región del espacio de entrada. Entre sus aplicaciones más comunes y eficientes cabe destacar el análisis de series temporales, el procesamiento de imágenes, el reconocimiento automático del habla y los diagnósticos médicos.

1.2. Objetivos

El presente Trabajo Fin de Grado pretende, como principal objetivo, proporcionar un simulador de redes de neuronas de base radial, que incorpore todos los aspectos involucrados en la creación y entrenamiento de las redes de base radial. Las funcionalidades que debe realizar el simulador son:

- Definir la arquitectura
- Entrenamiento híbrido
- Entrenamiento totalmente supervisado
- Realización de validación de la red
- Almacenamiento de la arquitectura de la red
- Almacenamiento de los resultados de entrenamiento
- Almacenamiento de los resultados de test
- Realización de validación cruzada

Además, debe ofrecer una interfaz de usuario sencilla, fácil de usar y eficiente, que permita a cualquier usuario utilizarla de manera ágil, sin la necesidad de disponer conocimientos altos en el uso de la informática. También, debe ser fácilmente exportable, por lo que se ha optado por su realización en el lenguaje Java, ya que

necesita menos requisitos para su correcto funcionamiento que cualquier otro lenguaje. Por último, el simulador estará enfocado al uso docente y público.

1.3. Motivaciones

Las redes de neuronas de base radial proporcionan soluciones eficientes a diversos problemas de clasificación, aproximación o predicción. Por ello, se emplean habitualmente en varios ámbitos, como en ámbitos médicos, financieros, ingenieriles, etc. No obstante, la comprensión de su estructura y de su funcionamiento requiere un apoyo que se consigue mediante aplicaciones o herramientas que ofrezcan un punto de vista más enfocado en la práctica. Pero dada la complejidad de estas redes de neuronas, es necesaria la elaboración de estas herramientas a través de proyectos dedicados a su realización en los diferentes lenguajes de programación.

Por otra parte, en el ámbito de investigación y profesional, muchas veces se opta por utilizar dichas redes de base radial, pero la complejidad de su cálculo y estructura, hacen necesaria la utilización de herramientas específicas que puedan simular los resultados de dichas redes en torno a unos determinados patrones. Sin embargo, no siempre se puede disponer de tales herramientas, o en caso contrario, su uso no es nada intuitivo y genera en muchos casos un rechazo de la utilización de las redes de base radial como solución al problema, buscando nuevas alternativas. Por este principal motivo, se ha optado por desarrollar un simulador que cumpla los objetivos antes descritos, centrándose sobre todo en la facilidad de uso, aportando una alternativa sencilla y cómoda que permita obtener una solución mediante las redes de base radial a un problema en cuestión, si el usuario así lo considera.

Para la puesta en marcha de un nuevo simulador de redes de neuronas de base radial, es necesario, como en el desarrollo de cualquier aplicación, llevar a cabo un estudio de aplicaciones semejantes, que cubran las necesidades que se pretendan satisfacer. Actualmente, existen numerosas herramientas que permiten la simulación de arquitecturas de redes neuronales, muchas de ellas accesibles e incluso intuitivas. No obstante, tan sólo un reducido número de estas herramientas ofrecen la posibilidad de simular las redes neuronales de base radial.

NeuroSolutions es un software destinado principalmente a la resolución de grandes problemas con una complejidad elevada mediante diferentes tipos de arquitecturas. Presenta una interfaz agradable, con varios iconos muy intuitivos, pero su elevado número de opciones dificulta el uso a los usuarios más inexpertos, ya que al mezclar diferentes tipos de arquitecturas, tanto neuronales como no neuronales, el empleo de una arquitectura específica requiere el conocimiento de uso de la totalidad de las alternativas. Después de un estudio realizado con dicho software, se han observado algunos inconvenientes que dificulta su uso para la simulación de las redes de bases radial, entre ellos: permite la modificación de parámetros que no tienen relevancia para una determinada arquitectura, pero su modificación puede afectar a

los resultados; no permite la determinación de ciertos parámetros relevantes en las redes de base radial, lo que imposibilita la total simulación y la obtención de determinadas soluciones. No obstante, el principal inconveniente es su elevado precio, lo que no permite su accesibilidad a todo el público.

SNNS y su versión en Java JavaNNS son simuladores de redes de neuronas que permiten la generación de arquitecturas y aprendizaje para obtener una serie de resultados en función del tipo de arquitectura y los patrones establecidos. La principal ventaja de su versión en Java, es la posibilidad de acceso a un mayor número de usuarios dada su portabilidad. Además, el proceso de instalación es muy reducido y no requiere grandes entornos para ejecutarse. No obstante, presenta una mezcla de parámetros de las diferentes arquitecturas que soporta, de tal forma que es complicado para un usuario que no esté familiarizado con los diferentes tipos de redes de neuronas, especificar los parámetros necesarios para la simulación de las distintas redes.

Neural Network Toolbox es una librería de Matlab que permite la simulación de arquitecturas de redes neuronales. Su principal adversidad es la necesidad de obtener el software Matlab con todo su extenso contenido para poder realizar la simulación de redes de base radial, incluyendo la necesidad de la adquisición de su correspondiente licencia. Se necesita, además, conocimientos sobre los distintos comandos empleados en Matlab para poder realizar la simulación, dificultando en gran medida el uso por parte de usuarios inexpertos. Además, las propias interfaces que presenta esta librería, no son fácilmente manejables dado que presentan una manera muy poco accesible de cargar los patrones, ya que es preciso haberlos cargado mediante comandos antes de iniciar la interfaz. Además, tampoco permite la opción de modificar los datos obtenidos, para seguir entrenando la red, ni realizar una validación cruzada, algo verdaderamente útil en la comprobación de la eficiencia de la red.

1.4. Estructura de la memoria

El presente documento forma la base teórica y la estructura sobre la que se ha realizado la aplicación, con la siguiente organización:

En el segundo capítulo, se detallan los fundamentos teóricos sobre los que se construye la aplicación, de modo que albergará todo el contenido teórico de las redes de neuronas de base radial.

En el tercer capítulo, se realizará el análisis del sistema, especificando los requisitos correspondientes a la aplicación y mostrando las funcionalidades que podrá realizar un usuario.

En el cuarto capítulo, se acometerá la realización del diseño de la aplicación, incluyendo la arquitectura, los subsistemas y las distintas clases que conforman la aplicación.

En el quinto capítulo, se especifican las numerosas pruebas que se requieren para comprobar el correcto funcionamiento del simulador, junto con un análisis de los resultados ofrecidos por el simulador respecto a unos determinados patrones. Finalmente, se encontrará el manual de usuario, que mostrará una breve explicación sobre la utilización del simulador de redes de neuronas de base radial que se ha desarrollado en este trabajo.

En el capítulo sexto, se incluye una planificación del desarrollo de la aplicación, compuesta principalmente por un diagrama de Gantt, con las diferentes fases del proyecto, junto con el presupuesto de la realización de la aplicación.

En el capítulo séptimo, se comentarán las diferentes conclusiones obtenidas con la realización de este Trabajo Fin de Grado y las posibles líneas futuras sobre la aplicación.

Capítulo 2: Redes de neuronas de base radial

En el presente capítulo, se detallarán las características fundamentales de las redes de base radial, de tal forma que sirvan de base teórica sobre el simulador, y para conocer el comportamiento que presentará el simulador cuando sea utilizado en sus distintas funcionalidades. Por otra parte, proporcionará un conocimiento básico sobre las redes de base radial, que probablemente influya positivamente en el buen desarrollo de la aplicación, ya que conocer la base teórica del objetivo a desarrollar, favorece enormemente que el resultado final represente lo más fielmente posible, la teoría sobre la que se basa.

2.1. Arquitectura

Las redes de neuronas de base radial se caracterizan por disponer de una única capa oculta, que se encarga de realizar una transformación local y no lineal de las señales que se transmiten desde las entradas. Además de la capa oculta, estas redes de neuronas están compuestas de dos capas más, formando un total de tres capas de neuronas. La primera capa está formada por las entradas de la red y se encarga de transmitir las señales de entrada hasta la capa oculta, que hemos comentado anteriormente. Cabe destacar que las conexiones entre la capa de entrada y las neuronas de la capa oculta no llevan pesos asociados, por lo que la señal de entrada se propaga directamente hacia la capa oculta. Por último está la capa de salida, encargada de realizar una combinación lineal de las activaciones de la capa oculta, obteniendo así las salidas de la red. Las conexiones entre estas capas (oculta y salida), sí llevan un peso asociado. En lo referente a los umbrales de las neuronas, sólo las neuronas de la capa de salida poseen un umbral, el cual se suele tratar como una conexión más de la neurona cuya entrada es constante e igual a uno.

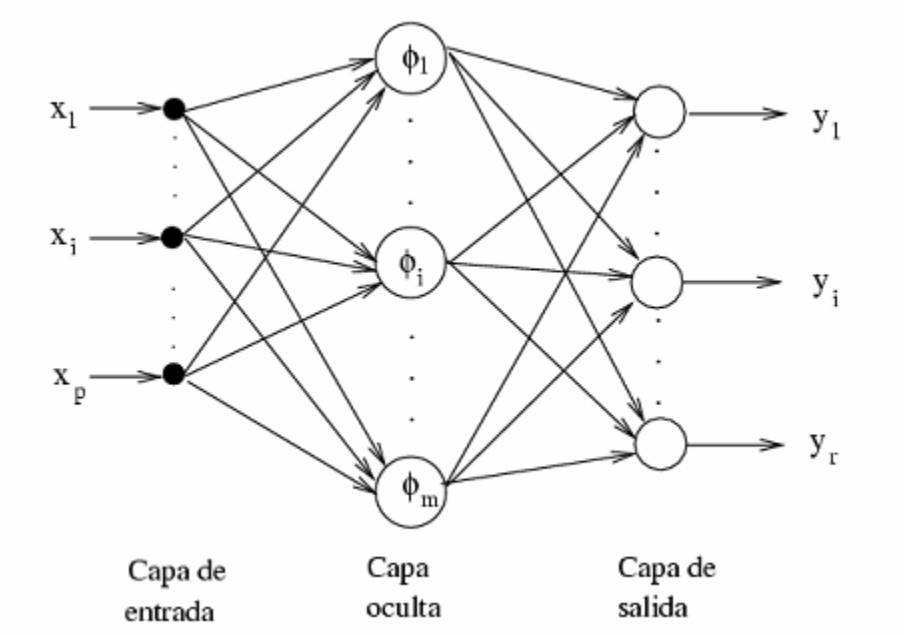

Figura 2.1: Arquitectura de redes de base radial

La propagación en estas redes de neuronas se realiza siempre hacia delante de forma que para un espacio de entrada de **p** dimensiones, un espacio de salida de **r** dimensiones y **m** neuronas ocultas, las activaciones de las neuronas de la capa de salida $\mathbf{y_k(n)}$ para el patrón de entrada **n** serán:

$$y_k(n) = \sum_{i=1}^{m} w_{ik}\phi_i(n) + u_k \text{ para } k = 1, 2, ..., r$$

(Ec. 1)

Siendo $\mathbf{w_{ik}}$ el peso de conexión de la neurona oculta **i** a la neurona de salida **k;** $\boldsymbol{\phi_i(n)}$ es la activación de la neurona oculta **i,** $\mathbf{u_k}$ es el umbral de la neurona de salida **k**.

Respecto a las activaciones de las neuronas de la capa oculta, se caracterizan porque se rigen por las funciones de base radial, las cuales determinan, para cada neurona de dicha capa oculta, si se activa para un determinado patrón de entrada o permanece desactivada. La activación viene dada por la siguiente expresión:

$$\phi_i(n) = \phi\left(\frac{\|X(n) - C_i\|}{d_i}\right) \text{ para } i = 1, 2, ..., m$$

(Ec. 2)

Siendo $\mathbf{X(n)=(x_1(n), x_2(n), ...x_p(n))}$ el patrón de entrada **n**; **ϕ** hace referencia a una función de base radial, que detallaremos posteriormente; $\mathbf{C_i}$ equivale a los vectores que conforman las coordenadas de los centros para la neurona **i**. Dichos vectores tienen el siguiente formato: $\mathbf{(c_{i1}, c_{i2}, \dots c_{ip})}$ siendo **p** el número de entradas de los patrones, es decir, el número de dimensiones para el espacio de entrada; y $\mathbf{d_i}$ representa las desviaciones de las funciones, y se tratan de números reales. Por último, **|| . ||** hace referencia a la distancia euclídea desde el vector de entrada al centro de la función, que viene dada por la siguiente expresión:

$$\|X(n) - C_i\| = \sqrt{\sum_{j=1}^{p} (x_j(n) - c_{ij})^2}$$

(Ec. 3)

De las funciones de base radial (**ϕ**), que hemos mencionado anteriormente, cabe destacar que se pueden utilizar las siguientes:

- Función gaussiana:

$$\phi(r) = exp(\frac{-r^2}{2})$$

(Ec. 4)

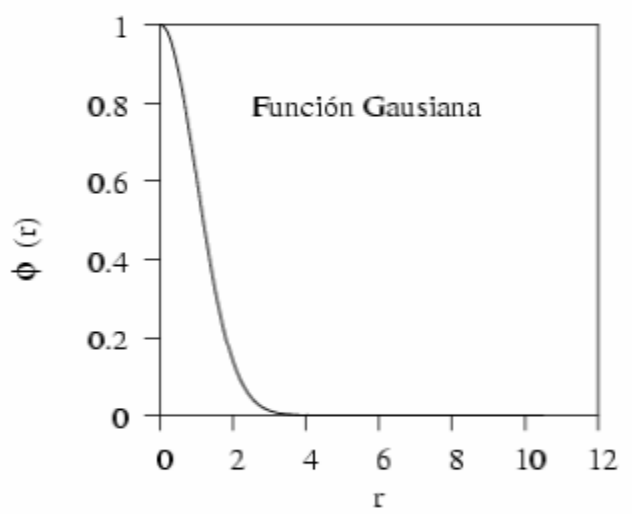

Figura 2.2: Función gausiana

- Función inversa cuadrática:

$$\phi(r) = \frac{1}{1+r^2}$$

(Ec. 5)

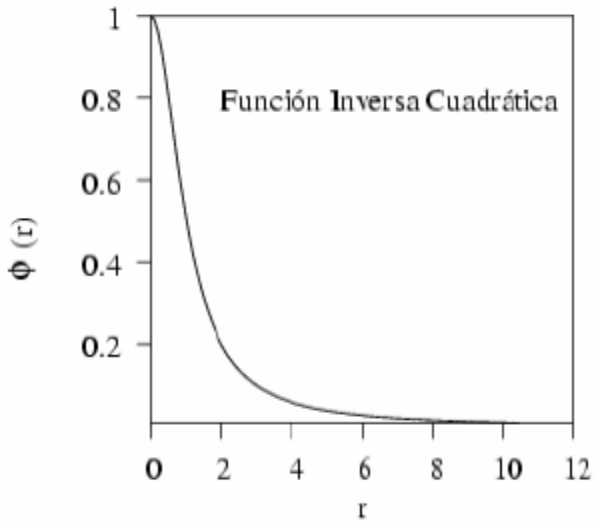

Figura 2.3: Función inversa cuadrática

- Función inversa multicuadrática:

$$\phi(r) = \frac{1}{\sqrt{1+r^2}} \quad \text{(Ec. 6)}$$

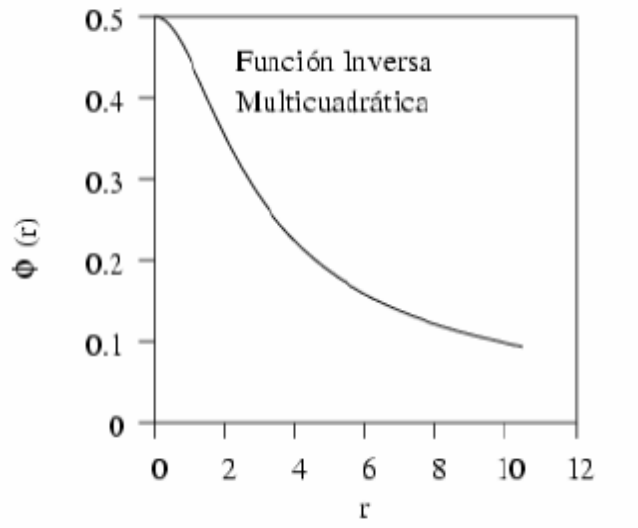

Figura 2.4: Función inversa multicuadrática

No obstante, de entre las mencionadas anteriormente, la más utilizada con diferencia es la función gaussiana, motivo por el cual se ha implementado en el simulador como función de base radial sobre la que realizar el aprendizaje de la red. Por tanto, y teniendo en cuenta las ecuaciones (Eq. 2) y (Eq. 4), las activaciones de las neuronas ocultas vienen dadas por:

$$\phi_i(n) = e^{-\frac{\|X(n)-C_i\|^2}{2d_i^2}} \quad \text{(Ec. 7)}$$

Donde $\mathbf{C_i}$ representa el centro de la función gaussiana para la neurona oculta **i** y $\mathbf{d_i}$ se trata de la anchura o desviación de esa neurona de la capa oculta.

Debido al uso de estas funciones, se puede afirmar que las redes de base radial tienen carácter local. Dado que si el patrón de entrada a la red **X(n)** está cerca del área que cubre el centro $\mathbf{C_i}$, la neurona oculta **i** tendrá un valor alto de activación. A medida que el patrón **X(n)** se va alejando de dicho centro, la activación de dicha neurona disminuye y puede activarse otra neurona oculta de la red. Por lo tanto, para un

patrón en cuestión, sólo las neuronas ocultas que tengan un centro cerca del patrón en cuestión se activarán, permaneciendo el resto inactivas o con un nivel menor de activación. Este carácter local lo podemos observar en las figuras 2.5, 2.6 y 2.7. Dado que se puede observar como cada zona del espacio de entrada está representada por una neurona oculta, de tal forma que para ciertos patrones de entrada, la neurona oculta con mayor activación es diferente que para otro grupo de patrones de entrada.

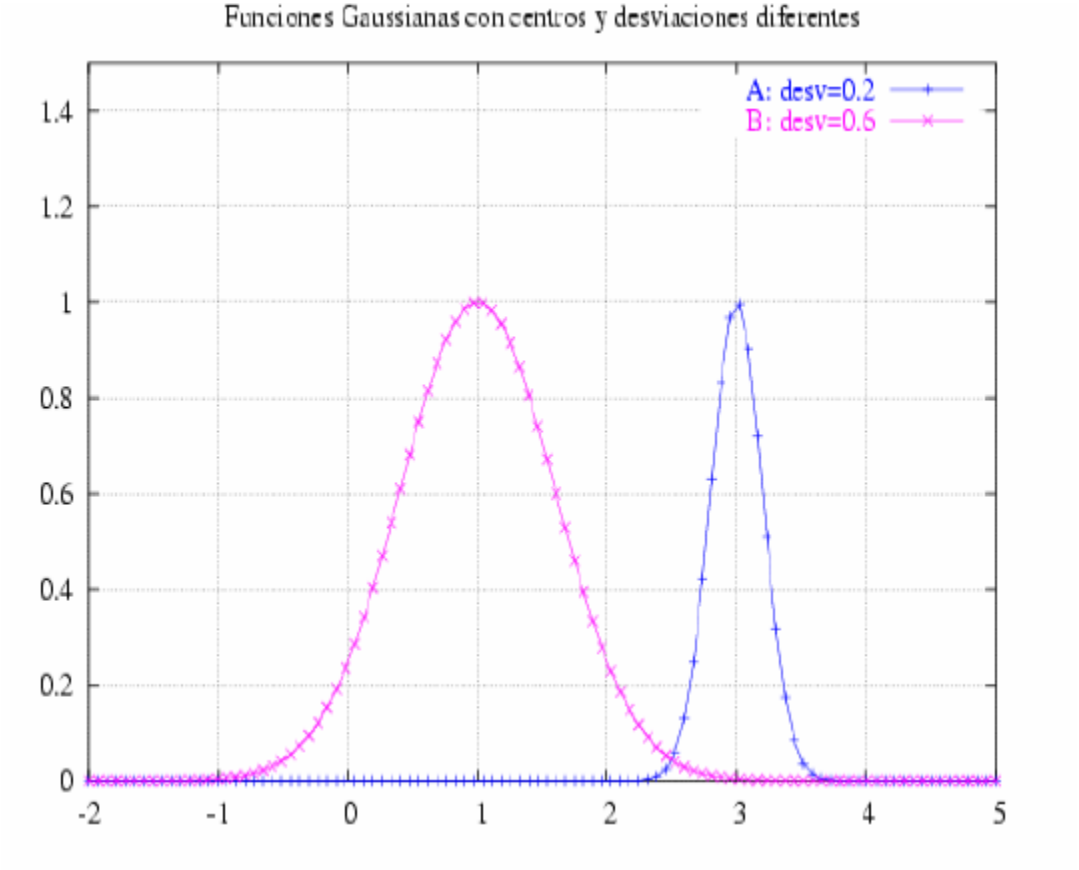

Figura 2.5: Funciones gaussianas con dos centros y desviaciones diferentes

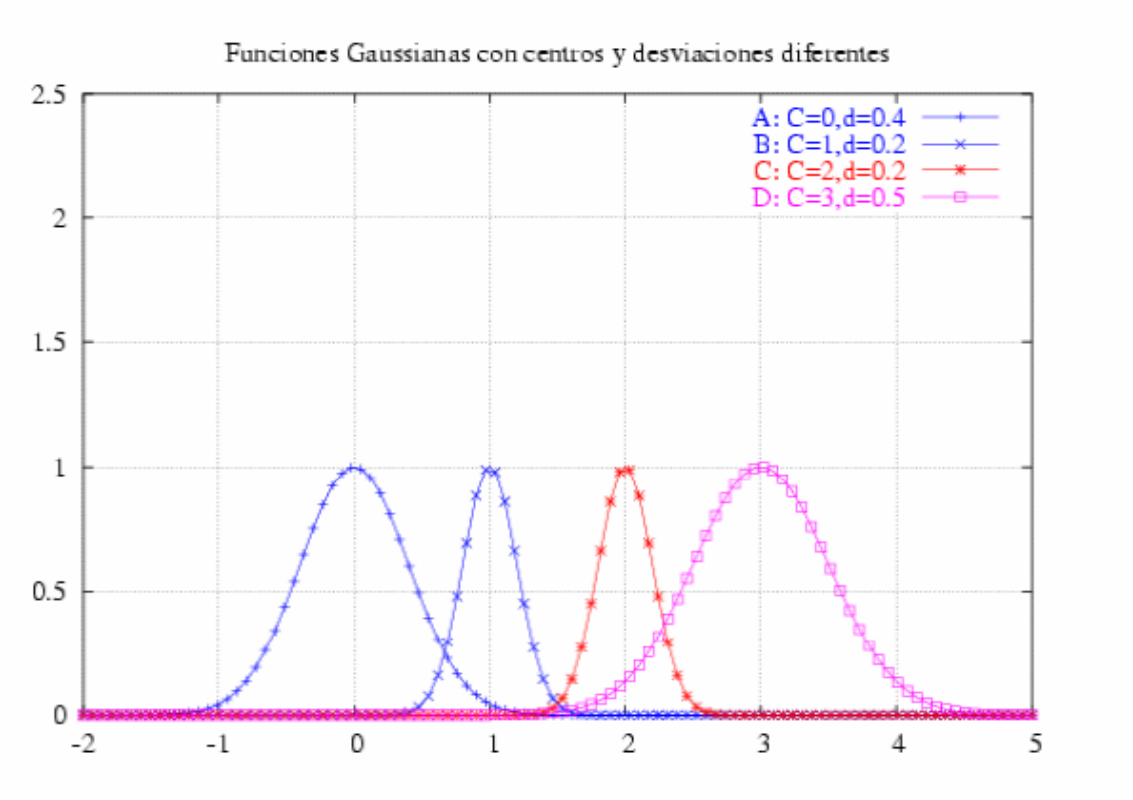

Figura 2.6: Funciones gaussianas con cuatro centros y desviaciones diferentes

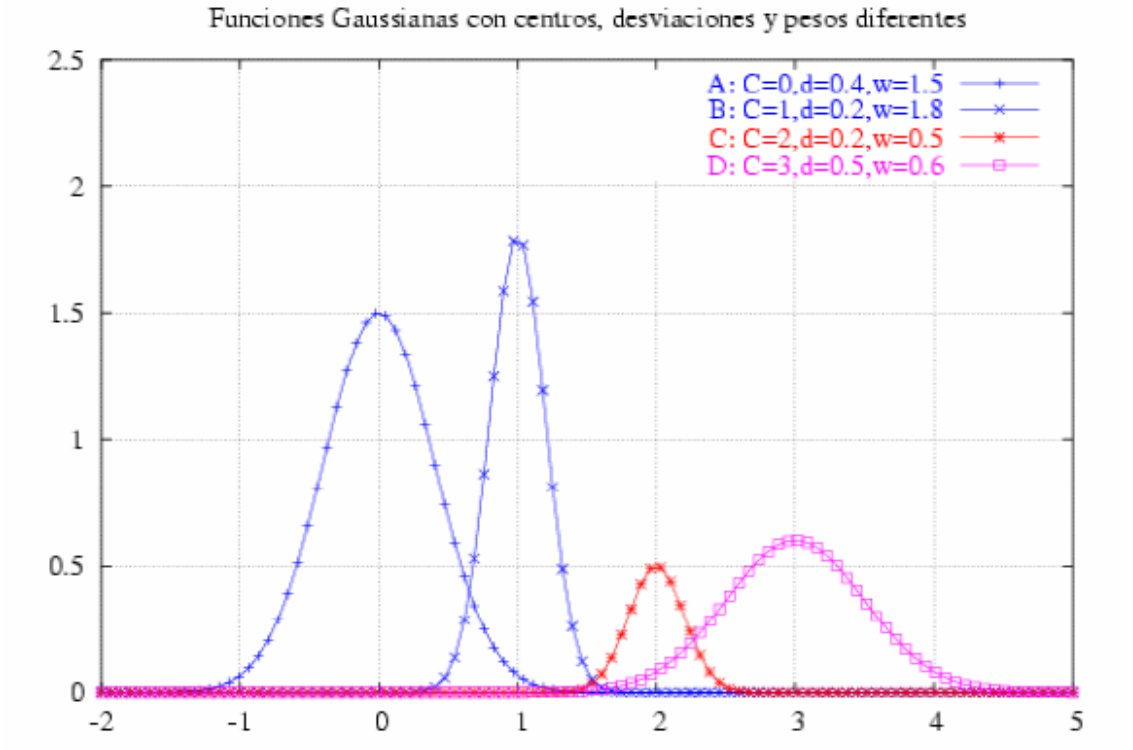

Figura 2.7: Funciones gaussianas con cuatro nuevos centros y desviaciones

2.2 Aprendizaje

Como en todos los sistemas de redes de neuronas, las redes de neuronas de base radial, también cuentan con un proceso de aprendizaje. Dicho proceso de aprendizaje se basa, principalmente en determinar los centros, las desviaciones, los pesos de la capa oculta a la capa de salida y los umbrales. Debido a que las tareas de las capas de red son diferentes, se pueden emplear diferentes procesos para optimizar el entrenamiento de los parámetros de cada una de las capas. De tal forma que para los centros y las desviaciones, se puede realizar un proceso de optimización del espacio de entrada, mientras que para los pesos, se utiliza una optimización en función a las salidas que se desean obtener.

Existen diferentes modos de realizar el aprendizaje en las redes de base radial: el entrenamiento híbrido y el totalmente supervisado. El entrenamiento híbrido está formado por dos fases: la fase no supervisada, en la que se determinan los centros y amplitudes de las neuronas de la capa oculta, y la fase supervisada en la que se determinan los pesos y umbrales de la capa de salida. Este tipo de entrenamiento, conserva las características locales de la red y se trata, sin duda, del más utilizado. En cambio, en el entrenamiento totalmente supervisado, como su propio nombre indica, se realiza una adaptación supervisada de todos los parámetros característicos de la red. Con el objetivo de minimizar una función error, perdiéndose las características locales de la red. Ambos tipos de aprendizaje han sido implementados en el simulador desarrollado.

2.2.1. Entrenamiento híbrido

Este tipo de entrenamiento, como hemos mencionado anteriormente, se compone de dos fases claramente distinguidas, la fase no supervisada para la determinación de los centros y amplitudes de las neuronas de la capa oculta, y la fase supervisada para la determinación de los pesos y umbrales de la capa de salida. El proceso de optimización en cada una de las fases se formula con un objetivo diferente y las técnicas para su resolución serán, por tanto, diferentes también.

2.2.1.1 Fase no supervisada

En esta fase, se procede a obtener tanto los centros como las desviaciones, con el fin de poder agrupar el espacio de entrada en distintas clases o grupos. El principal representante de cada clase será el centro de la función de base radial y la amplitud determinará la desviación del mismo.

Para la determinación de los centros, se utiliza el algoritmo que permita realizar una clasificación no supervisada, dividiendo el espacio de entrada en diferentes clases. En el caso de las redes de base radial, el número de clases corresponderá con el número de neuronas de la capa oculta. De todos los algoritmos posibles para su utilización en esta fase, el algoritmo de K-medias, es sin duda, el más utilizado. Dicho algoritmo fue diseñado por J.MacQueen [MacQueen, 1967] y determina una división

del espacio de entrada en **K** clases. El representante de cada una de las clases corresponderá con el centro de la neurona de la capa oculta. No obstante, se podría utilizar cualquier algoritmo de clasificación no supervisado, como por ejemplo los mapas autoorganizados de Kohonen [Kohonen, 1982].

Para la determinación de los centros mediante el algoritmo de K-medias, se pretende minimizar la distancia euclídea entre los distintos patrones de entrada y su centro más cercano correspondiente, es decir se pretende minimizar la siguiente función:

$$J = \sum_{i=1}^{K} \sum_{n=1}^{N} M_{in} \| X(n) - C_i \|$$

(Ec. 8)

Siendo **N** el número de patrones de entrada, **|| . ||** la distancia euclídea definida en (Eq. 3), **X(n)** el patrón de entrada **n**, y $\mathbf{M_{in}}$ la función de pertenencia al centro, valiendo 1 si el patrón pertenece al centro en cuestión, y 0 en caso contrario.

$$M_{in} = \begin{cases} 1 & \text{si } \|X(n) - C_i\| < \|X(n) - C_s\| \ \forall s \neq i, \ s = 1, 2, ..., K \\ 0 & \text{en otro caso} \end{cases}$$

(Ec. 9)

El algoritmo de K-medias es un proceso iterativo y se compone de una serie de pasos a seguir, que son los siguientes:

1. Se inicializan aleatoriamente los valores correspondientes a los centros.
2. Se asignarán cada uno de los patrones de entrada a un centro correspondiente. Para conocer si el patrón **X(n)** pertenece a un centro en concreto debe darse el siguiente caso:

$$\|X(n) - C_i\| < \|X(n) - C_s\|$$
$$\forall \ s \neq i \ \text{con} \ s = 1, 2, ..., K.$$

(Ec. 10)

   De esta forma que cada patrón tendrá asignado el centro más cercano.
3. Se calcula el centro como la media de todos los patrones asignados, es decir:

$$c_{ij} = \frac{1}{N_i} \sum_{n=1}^{N} M_{in} x_j(n) \text{ para } j = 1, 2, ...p \text{ , } i = 1, 2, ..., K$$

(Ec. 11)

4. Se repiten los dos pasos anteriores hasta que la modificación de los centros no sea superior al valor del parámetro épsilon:

$$\| C_i^{nuevo} - C_i^{anterior} \| < \varepsilon \ \forall \ i = 1, 2, ..., K$$

(Ec. 12)

Se trata de un algoritmo sencillo de implementar y utilizar. Además, suele ser bastante eficiente dado que converge rápidamente, aunque en ocasiones, se trate de mínimos locales. Sin embargo, presenta una pequeña desventaja, su dependencia de los valores obtenidos en la inicialización de los centros, de ahí que puedan producirse mínimos locales.

En cuanto a la determinación de las desviaciones, es necesario que se calculen de manera que cada neurona de la capa oculta se active en una región del espacio de entrada, de tal manera que el solapamiento de las zonas de activación de las neuronas sea lo mínima posible. Para ello, una de las formas más efectivas para calcular las desviaciones es la media geométrica de la distancia del centro de la neurona, a sus dos centros más cercanos.

$$d_i = \sqrt{\| C_i - C_t \| \ \| C_i - C_s \|}$$

(Ec. 13)

Siendo $\mathbf{C_t}$ y $\mathbf{C_s}$ los dos centros más cercanos al centro $\mathbf{C_i}$.

2.2.1.2. Fase supervisada

En esta fase se procede a la determinación de los pesos que conectan las neuronas de la capa oculta con la capa de salida, y los umbrales de las neuronas de la capa de salida, con el objetivo de obtener unas salidas, lo más aproximadas posible a las deseadas. Para ello, se trata de una fase supervisada y el objetivo es minimizar la función error medida en la salida de la red, que viene dada por:

$$E = \frac{1}{N} \sum_{n=1}^{N} e(n)$$

(Ec. 14)

Donde **N** es el número de patrones y **e(n)** viene dado por:

$$e(n) = \frac{1}{2}\sum_{k=1}^{r}(s_k(n) - y_k(n))^2$$ (Ec. 15)

Siendo **s(n)=(s_1(n),…,s_r(n))** es el vector de la salida deseada de la red e **y(n)=(y_1(n),…,y_r(n))** es el vector de la salida de la red para el patrón **n**.

Puesto que la salida de la red depende linealmente de los pesos (ver Ec.1), se pueden utilizar para su cálculo métodos directos, como el método de la seudo-inversa o el método de los mínimos cuadrados.

El método de la seudo-inversa es un método que proporciona una solución directa al problema de optimización. Tal solución viene dada por la siguiente expresión matricial:

$$W = G^{+} \cdot S = (G^{t} \cdot G)^{-1} \cdot G^{t} \cdot S$$ (Ec. 16)

Siendo **W** la matriz de pesos y umbrales de la red, de orden **(m+1) x r; S** la matriz que contiene las salidas deseadas de la red, de orden **N x r**; $\mathbf{G^{+}=(G^{t} \cdot G)^{-1} \cdot G^{t}}$ la matriz seudo-inversa de **G** y $\mathbf{G^{t}}$ la matriz traspuesta de **G**. La matriz **G** es una matriz de orden **Nx(m+1)** que contiene las activaciones de las neuronas ocultas de la red para los patrones de entrada. A continuación se indica la expresión de cada una de estas matrices.

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1r} \\ w_{21} & w_{22} & \dots & w_{2r} \\ \dots & \dots & \dots & \dots \\ w_{m1} & w_{m2} & \dots & w_{mr} \\ u_1 & u_2 & \dots & u_r \end{pmatrix}$$ (Ec. 17)

$$\mathbf{G} = \begin{pmatrix} \phi_1(1) & \phi_2(1) & \dots & \phi_m(1) & 1 \\ \phi_1(2) & \phi_2(2) & \dots & \phi_m(2) & 1 \\ \dots & \dots & \dots & \dots & \\ \phi_1(N) & \phi_2(N) & \dots & \phi_m(N) & 1 \end{pmatrix}$$ (Ec. 18)

Donde **ϕ_i(n)** es la activación de la neurona de la capa oculta **i** para el patrón de entrada **X(n)**.

$$\mathbf{S} = \begin{pmatrix} s_1(1) & s_2(1) & \dots & s_r(1) \\ s_1(2) & s_2(2) & \dots & s_r(2) \\ \dots & \dots & \dots & \dots \\ s_1(N) & s_2(N) & \dots & s_r(N) \end{pmatrix}$$

(Ec. 19)

Un método más sencillo para el cálculo de los pesos y umbrales es método de mínimos cuadrados, método que ha sido implementado en el simulador. Se trata de un proceso iterativo mediante el cual los pesos y umbrales se actualizan o modifican de acuerdo a las siguientes leyes:

$$w_{ik}(n) = w_{ik}(n-1) - \alpha_1 \frac{\partial e(n)}{\partial w_{ik}}$$

$$u_k(n) = u_k(n-1) - \alpha_1 \frac{\partial e(n)}{\partial u_k}$$

$$\text{para } k = 1, 2, ..., r \text{ y para } i = 1, ..., m$$

(Ec. 20), (Ec. 21)

Donde **e(n)** es el error para el patrón **n** y $\boldsymbol{\alpha_1}$ es la razón de aprendizaje. Si se tiene en cuenta la expresión del error y que el peso $\mathbf{w_{ik}}$ y el umbral $\mathbf{u_k}$ únicamente afecta a la neurona de salida **k**, obtenemos:

$$\frac{\partial e(n)}{\partial w_{ik}} = -(s_k(n) - y_k(n)) \frac{\partial y_k(n)}{\partial w_{ik}}$$

$$\frac{\partial e(n)}{\partial u_k} = -(s_k(n) - y_k(n)) \frac{\partial y_k(n)}{\partial u_k}$$

(Ec. 22), (Ec. 23)

Derivando la salida $\mathbf{y_k(n)}$ (ver Ec. 1) respecto de los pesos y umbrales, se obtiene:

$$\frac{\partial y_k(n)}{\partial w_{ik}} = \phi_i(n) \qquad \frac{\partial y_k(n)}{\partial u_k} = 1$$

(Ec. 24)

De tal forma que sustituyendo, obtenemos las siguientes ecuaciones para determinar los pesos y los umbrales:

$$w_{ik}(n) = w_{ik}(n-1) + \alpha_1(s_k(n) - y_k(n))\phi_i(n)$$

$$u_k(n) = u_k(n-1) + \alpha_1(s_k(n) - y_k(n))$$

$$\text{para } k = 1, 2, ..., r \text{ y para } i = 1, ..., m$$

(Ec. 25), (Ec. 26)

Cuando se calculan los pesos a partir de las leyes de aprendizaje anteriores, la convergencia es notablemente rápida, por lo tanto se consigue una solución en un número pequeño de iteraciones o ciclos de aprendizaje.

Podemos resumir el procedimiento de aprendizaje híbrido, tomando **X(n)** y **S(n)** como un conjunto de patrones de entrada y sus correspondientes salidas deseadas, en los siguientes pasos:

1. Se realiza el algoritmo K-medias sobre los patrones de entrada **X(n)** para obtener los valores de los centros de las funciones de base radial.
2. Se procede al cálculo de las desviaciones de tales centros mediante la media geométrica de la distancia del centro a sus dos vecinos más cercanos.
3. Se determinan los pesos y umbrales siguiendo los siguientes pasos:
    1) Se inicializan aleatoriamente los pesos y umbrales.
    2) Se toma un patrón del conjunto de entrada y se calcula la salida de la red **Y(n)** para ese patrón.
    3) Se evalúa el error **e(n)** de la red, para el patrón seleccionado.
    4) Se modifican los parámetros de la red utilizando las ecuaciones correspondientes.
    5) Se repiten los pasos 2, 3 y 4 para cada uno de los patrones de entrenamiento.
    6) Se repiten los pasos 2, 3, 4 y 5 hasta que se consigue la convergencia, es decir, que la suma de los errores de los patrones se estabilice.

2.2.2. Entrenamiento totalmente supervisado

Este tipo de entrenamiento se caracteriza por determinar todos los parámetros de la red, es decir, centros, desviaciones, pesos y umbrales, de tal manera que se minimice el error cuadrático medio. No conserva las propiedades locales de las redes de neuronas de base radial, ya que a diferencia del entrenamiento híbrido, no se determinan las desviaciones intentando que el solapamiento entre las amplitudes de las neuronas sea el menor posible, sino que se busca principalmente la minimización del error cuadrático medido en la salida de la red. La dependencia entre los centros y desviaciones de las neuronas de la capa oculta con las salidas de la red pasan a ser no lineales, dado que se utilizan funciones de base radial, que son funciones no lineales.

Debido a ello, se utiliza el método del descenso del gradiente, es decir, una técnica de optimización no lineal. A través de un proceso iterativo, y siguiendo la dirección negativa del gradiente del error, el método proporciona un mínimo, que pudiera ser local, de la función error. De tal forma que los parámetros que se determinan en este tipo de entrenamiento, se adaptan o actualizan teniendo en cuenta las siguientes leyes:

- Centros

$$c_{ij}(n) = c_{ij}(n-1) - \alpha_2 \frac{\partial e(n)}{\partial c_{ij}}$$

(Ec. 27)

- Desviaciones

$$d_i(n) = d_i(n-1) - \alpha_3 \frac{\partial e(n)}{\partial d_i}$$

(Ec. 28)

- Pesos

$$\omega_{ik}(n) = \omega_{ik}(n-1) - \alpha_1 \frac{\partial e(n)}{\partial \omega_{ik}}$$

(Ec. 29)

- Umbrales

$$u_k(n) = u_k(n-1) - \alpha_1 \frac{\partial e(n)}{\partial u_k}$$

(Ec. 30)

Donde $\boldsymbol{\alpha_1}$, $\boldsymbol{\alpha_2}$ y $\boldsymbol{\alpha_3}$ son las razones de aprendizaje y **e(n)** es el error medido en la salida de la red para el patrón **n**.

Dado que el método de descenso del gradiente implica el cálculo de las derivadas del error con respecto a cada uno de los parámetros, se obtienen las siguientes ecuaciones para determinar los parámetros correspondientes:

- Centros

$$c_{ij}(n) = c_{ij}(n-1) + \alpha_2 \left(\sum_{k=1}^{r} (s_k(n) - y_k(n))\omega_{ik}\right)\phi_i(n)\frac{(x_j - c_{ij})}{d_i^2}$$

Para j=$1,2,\ldots, p$ y para $i=1,2, \ldots, m$

(Ec. 31)

- Desviaciones

$$d_i(n) = d_i(n-1) + \alpha_3 \left(\sum_{k=1}^{r} (s_k(n) - y_k(n))\omega_{ik}\right)\phi_i(n)\frac{\|X(n) - C_i\|^2}{d_i^3}$$

Para $i=1,2, \ldots, m$

(Ec. 32)

- Pesos

$$\omega_{ik}(n) = \omega_{ik}(n-1) + \alpha_1 (s_k(n) - y_k(n))\phi_i(n)$$

Para $k=1,2,\ldots, r$ y para $i=1,2, \ldots, m$

(Ec. 33)

- Umbrales

$$u_k(n) = u_k(n-1) + \alpha_1 (s_k(n) - y_k(n))$$

Para $k=1,2,\ldots, r$ y para $i=1,2, \ldots, m$

(Ec. 34)

Finalmente cabe destacar que al tratarse de un método iterativo, existe la necesidad de inicializar todos los parámetros que se van a determinar, con valores aleatorios muy próximos a cero. Asimismo, se recomienda la inicialización de los centros de tal modo que puedan representar las zonas del espacio de entrada, con lo cual se podría realizar una combinación de ambos tipos de entrenamientos, al establecerse los parámetros mediante el método híbrido y adaptándose posteriormente mediante el método totalmente supervisado.

A modo resumen, podemos señalar que dado un conjunto de patrones de entradas **X(n)** y sus salidas deseadas correspondientes **S(n)**, el método de aprendizaje totalmente supervisado se resumiría en los siguientes pasos:

1. Inicializar los parámetros de la red. Para los pesos, umbrales y desviaciones, preferiblemente debe hacerse de manera aleatoria con valores cercanos a cero. En cuanto a los centros, se recomienda el uso de algoritmos de clasificación aplicados en el espacio de entrada.
2. Se toma un patrón del conjunto de patrones de entradas y salida, y se procede a la obtención de la salida de la red **Y(n)** para el patrón **X(n)**.
3. Se evalúa el error **e(n)** cometido por la red, al comparar la salida deseada **S(n)** para el patrón **X(n)**, con la salida de la red **Y(n)**.
4. Se realiza una modificación de los parámetros de la red empleando sus ecuaciones correspondientes.
5. Se repiten los pasos 2, 3 y 4 para cada uno de los patrones de entrenamiento.
6. Se repiten los pasos 2, 3, 4 y 5 hasta que se consiga la convergencia, es decir, que la suma de los errores de cada uno de los patrones se estabilice, momento en el que se alcanza el mínimo de la función.

Capítulo 3: Análisis del sistema

En este capítulo se recogerán las necesidades del usuario, analizándolas para poder representar de la manera más clara posible dichas necesidades de tal forma que se puedan identificar con seguridad y certeza los objetivos del simulador, para que la conformidad del usuario con el mismo, sea la más alta posible.

Además, se establecerán todos los posibles detalles con la intención de facilitar la estructura que va a presentar el simulador. De modo que la realización del diseño del simulador en base a esta estructura en cuestión, proporcionará un aumento de la probabilidad de que el simulador cubra con éxito todas las necesidades del usuario. También, se determinarán las diferentes funcionalidades que podrá desempeñar un usuario, estableciendo una base sólida sobre la que se desarrollará el diseño del simulador.

3.1. Definición de los requisitos del sistema

En este apartado se detallarán los diferentes requisitos de usuario que se han captado para este sistema. Teniendo en cuenta, las capacidades que dispondrá el simulador, que serán recogidas en los denominados requisitos de capacidad, y las restricciones que se deberán tener en cuenta a la hora de desarrollar el simulador, que quedarán recogidas en los requisitos de restricción. Estos requisitos de usuario, reflejarán la idea del usuario sobre el simulador, pero para toda la funcionalidad del sistema, el desarrollador, determinará los requisitos de software, que se verán más adelante, los cuales se basarán en los requisitos de usuario establecidos, verificando así que todas las necesidades del usuario estén cubiertas.

3.1.1. Requisitos de usuario de capacidad

Los requisitos de capacidad reflejan las capacidades que el usuario considera que debe realizar la herramienta a desarrollar. Es decir, todo aquello que dicho usuario considera que debe realizar la aplicación, estará en los requisitos de usuario de capacidad. Para ofrecer una mayor claridad en la comprensión de dichos requisitos, se han establecido y clasificado en diferentes tablas que se mostrarán a continuación.

Cada una de esas tablas, contiene un requisito de capacidad, y para facilitar su identificación, se ha utilizado la siguiente notación: RUC-XX, donde RUC significa Requisito de Usuario de Capacidad y donde XX vendrá especificado por el número de requisito de dos dígitos, es decir para el primer requisito el identificador será RUC-01. Además, para cada uno de los requisitos, se detallará el nombre del requisito, el tipo de requisito, la prioridad con la que se caracteriza el requisito, la necesidad de que esté presente en el simulador final y una breve descripción del mismo.

A continuación se muestran los requisitos de usuario de capacidad determinados para el desarrollo del simulador.

Identificador	RUC-01								
Nombre	Creación y modificación de redes								
Tipo	**Capacidad**		X	**Restricción**					
Prioridad	**Alta**	X	**Media**			**Baja**			
Necesidad	**Obligatorio**		X	**Opcional**					
Descripción	Se podrán crear y modificar redes de base radial.								

Tabla 3.1: Requisito de usuario de capacidad 1

Identificador	RUC-02								
Nombre	Entrenamiento								
Tipo	**Capacidad**		X	**Restricción**					
Prioridad	**Alta**	X	**Media**			**Baja**			
Necesidad	**Obligatorio**		X	**Opcional**					
Descripción	Se podrán realizar un entrenamiento híbrido o totalmente supervisado de la red.								

Tabla 3.2: Requisito de usuario de capacidad 2

Identificador	RUC-03								
Nombre	Test								
Tipo	**Capacidad**		X	**Restricción**					
Prioridad	**Alta**	X	**Media**			**Baja**			
Necesidad	**Obligatorio**		X	**Opcional**					
Descripción	Se podrá realizar test sobre una red de base radial.								

Tabla 3.3: Requisito de usuario de capacidad 3

Identificador	RUC-04								
Nombre	Almacenar los resultados								
Tipo	**Capacidad**		X	**Restricción**					
Prioridad	**Alta**	X	**Media**			**Baja**			
Necesidad	**Obligatorio**		X	**Opcional**					
Descripción	Se permitirá guardar los resultados que proporciona la red para entrenamiento y test.								

Tabla 3.4: Requisito de usuario de capacidad 4

Identificador	RUC-05							
Nombre	Almacenar la red							
Tipo	Capacidad		X	Restricción				
Prioridad	Alta	X	Media			Baja		
Necesidad	Obligatorio		X	Opcional				
Descripción	Se permitirá almacenar los parámetros de una red.							

Tabla 3.5: Requisito de usuario de capacidad 5

Identificador	RUC-06							
Nombre	Validación cruzada							
Tipo	Capacidad		X	Restricción				
Prioridad	Alta		Media		X	Baja		
Necesidad	Obligatorio			Opcional			X	
Descripción	Se permitirá realizar validación cruzada sobre un conjunto de datos.							

Tabla 3.6: Requisito de usuario de capacidad 6

Identificador	RUC-07							
Nombre	Cargar una red							
Tipo	Capacidad		X	Restricción				
Prioridad	Alta		Media		X	Baja		
Necesidad	Obligatorio		X	Opcional				
Descripción	Se permitirá importar los parámetros de una red creada con anterioridad.							

Tabla 3.7: Requisito de usuario de capacidad 7

Identificador	RUC-08							
Nombre	Número de ciclos							
Tipo	Capacidad		X	Restricción				
Prioridad	Alta	X	Media			Baja		
Necesidad	Obligatorio		X	Opcional				
Descripción	Se podrá establecer el número de ciclos que se realizarán durante el entrenamiento y test.							

Tabla 3.8: Requisito de usuario de capacidad 8

Identificador	RUC-09							
Nombre	Número de ciclos de validación							
Tipo	Capacidad		X	Restricción				
Prioridad	Alta		Media		X	Baja		
Necesidad	Obligatorio		X	Opcional				
Descripción	Se podrá establecer el número de ciclos en los que se realizará validación durante el entrenamiento.							

Tabla 3.9: Requisito de usuario de capacidad 9

Identificador	RUC-10							
Nombre	Carga de patrones							
Tipo	Capacidad		X	Restricción				
Prioridad	Alta	X	Media			Baja		
Necesidad	Obligatorio		X	Opcional				
Descripción	Se permitirá cargar los patrones tanto de entrenamiento como de test.							

Tabla 3.10: Requisito de usuario de capacidad 10

3.1.2. Requisitos de usuario de restricción

Los requisitos de usuario de restricción son aquellos que representan las restricciones que se deben respetar durante el desarrollo del simulador. Es decir, si debe estar realizado de alguna manera en concreto, como por ejemplo estar en un idioma en particular. Al igual que los requisitos de capacidad, también están clasificados en tablas con la misma información. En este caso, el identificador utilizado se basa en la siguiente notación: RUR-XX, siendo RUR las siglas de Requisito de Usuario de Restricción, y XX el número de dos cifras que representa el requisito dentro de su clasificación como de restricción.

Identificador	RUR-01							
Nombre	Portabilidad de la aplicación							
Tipo	Capacidad			Restricción				X
Prioridad	Alta		Media		X	Baja		
Necesidad	Obligatorio		X	Opcional				
Descripción	Deberá ser portable en los diferentes equipos.							

Tabla 3.11: Requisito de usuario de restricción 1

Identificador	RUR-02							
Nombre	Idioma de la interfaz							
Tipo	**Capacidad**				**Restricción**	X		
Prioridad	**Alta**		**Media**		**Baja**	X		
Necesidad	**Obligatorio**	X	**Opcional**					
Descripción	Todo el contenido visual deberá estar en inglés.							

Tabla 3.12: Requisito de usuario de restricción 2

Identificador	RUR-03							
Nombre	Visualización de entrenamiento y test							
Tipo	**Capacidad**				**Restricción**	X		
Prioridad	**Alta**		**Media**	X	**Baja**			
Necesidad	**Obligatorio**	X	**Opcional**					
Descripción	Los errores de entrenamiento y test se mostrarán visualmente a lo largo de los ciclos.							

Tabla 3.13: Requisito de usuario de restricción 3

Identificador	RUR-04							
Nombre	Resultados de entrenamiento							
Tipo	**Capacidad**				**Restricción**	X		
Prioridad	**Alta**		**Media**	X	**Baja**			
Necesidad	**Obligatorio**		**Opcional**	X				
Descripción	El almacenamiento de los resultados deberá tener un formato que permita su fácil exportación y visualización.							

Tabla 3.14: Requisito de usuario de restricción 4

Identificador	RUR-05							
Nombre	Mensajes de ayuda y error							
Tipo	**Capacidad**				**Restricción**	X		
Prioridad	**Alta**		**Media**		**Baja**	X		
Necesidad	**Obligatorio**		**Opcional**	X				
Descripción	En caso de introducir algún valor incorrecto, o cargar datos con formato inadecuado, se deberá mostrar un mensaje de ayuda, indicando el error existente.							

Tabla 3.15: Requisito de usuario de restricción 5

3.2. Estudio de las alternativas de la solución

Tras el estudio de las necesidades del sistema, junto con los requisitos de usuario se van a estudiar y proponer diferentes alternativas de solución con el fin de analizar y seleccionar la que más satisfaga al usuario. Estas alternativas analizan el sistema desde un punto de vista tecnológico, teniendo en cuenta el sistema operativo, el lenguaje y el entorno de programación.

- **Alternativa A**

Sistema operativo	Microsoft Windows 7
Lenguaje de programación	Java
Entorno de programación	Netbeans 7.1
Herramienta de diseño	Microsoft Visual Studio 2010

Tabla 3.16: Alternativa A

Esta alternativa, se caracteriza sobre todo por la utilización del lenguaje de programación Java, cuya característica principal es la portabilidad. De este modo, se permitiría la utilización del simulador en prácticamente cualquier entorno que cumpliera unos requisitos mínimos de hardware y tuviera un entorno de ejecución Java. Además destaca la utilización de la herramienta Visual Studio para el diseño, lo cual facilita el desarrollo a la hora de realizar cualquier tipo de diseño, debido fundamentalmente a sus herramientas de modelado. Su coste de utilización podría suponer un problema, pero debido al acuerdo MSDN Academic Alliance, que permite la utilización libre de software propietario de Microsoft para las personas vinculadas con la Universidad Carlos III de Madrid, supondría un coste cero.

- **Alternativa B**

Sistema operativo	Microsoft Windows 7
Lenguaje de programación	C#
Entorno de programación	Microsoft Visual Studio 2010
Herramienta de diseño	Visual Paradigm

Tabla 3.17: Alternativa B

Esta alternativa ofrece garantías de un desarrollo más que aceptable para las necesidades del sistema, añadiendo la facilidad para su realización debido a la amplia variedad de recursos de los que se podrían disponer para realizar cada una de las tareas necesarias para el desarrollo del simulador. No obstante, la utilización de lenguaje C#, no permitiría la exportación a sistemas operativos no pertenecientes a Microsoft. Además la utilización del Visual Paradigm, podría suponer un aumento en los costes totales, dado que no se

trata de una herramienta de uso gratuito, aunque su presencia en los equipos de las aulas informáticas de la Universidad Carlos III, evitaría dicho aumento.

- **Alternativa C**

Sistema operativo	GNU/Ubuntu Linux
Lenguaje de programación	C++
Entorno de programación	Eclipse
Herramienta de diseño	DIA

Tabla 3.18: Alternativa C

Es la alternativa más económica, dado que todas las herramientas son de código abierto, por lo que el coste de su adquisición sería nulo. No obstante, la herramienta de diseño propuesta no es todo lo fácil de utilizar que debería, dado que si nunca se ha trabajado con ella, puede suponer un esfuerzo extra que demoraría los plazos de desarrollo del simulador.

3.2.1. Valoración de las alternativas y selección de la solución

Tras el análisis de las ventajas e inconvenientes de las tres alternativas de solución propuestas se realiza una valoración de las mismas, con el fin de seleccionar la alternativa que más se ajusta a las necesidades del usuario.

La alternativa A, presenta a simple vista un menor tiempo de desarrollo que el resto, debido a la facilidad de uso del conjunto de herramientas, y a la experiencia del desarrollador con ellas. Por ello junto con su ausencia de incremento del coste total del simulador, supone un riesgo bajo.

La alternativa B, supone un riesgo medio, dado que no se cuenta con la aportación de la portabilidad de uso que se espera para el usuario.

La alternativa C, presenta un riesgo moderado, dado que no dispone de una experiencia previa en la utilización de sus herramientas, pudiendo suponer un retraso en la planificación estimada para la realización del simulador.

Tras este análisis se estima conveniente la selección de la alternativa A, debido a que cubre con garantías todas las necesidades que se plantean en este sistema, ofreciendo además un riesgo bajo y un plazo de entrega lo más reducido posible.

3.3. Requisitos de software y casos de uso

En este apartado se detallarán los diferentes requisitos de software que identificarán las distintas funcionalidades identificadas a partir de los requisitos de usuario detallados anteriormente. Además, se mostrarán los casos de uso que servirán para mostrar de manera gráfica la relación del sistema con el usuario, lo cual facilitará la interpretación de las necesidades del sistema, incluyendo sus funcionalidades, de manera que la representación de la idea final del simulador sea lo más clara posible.

3.3.1. Casos de uso

A continuación se muestra el esquema representativo de los casos de uso, en el cual se podrán apreciar las relaciones mediante interacciones del usuario con el sistema, representando claramente las funcionalidades que dispondrá el simulador, con las cuales podrá interactuar el usuario.

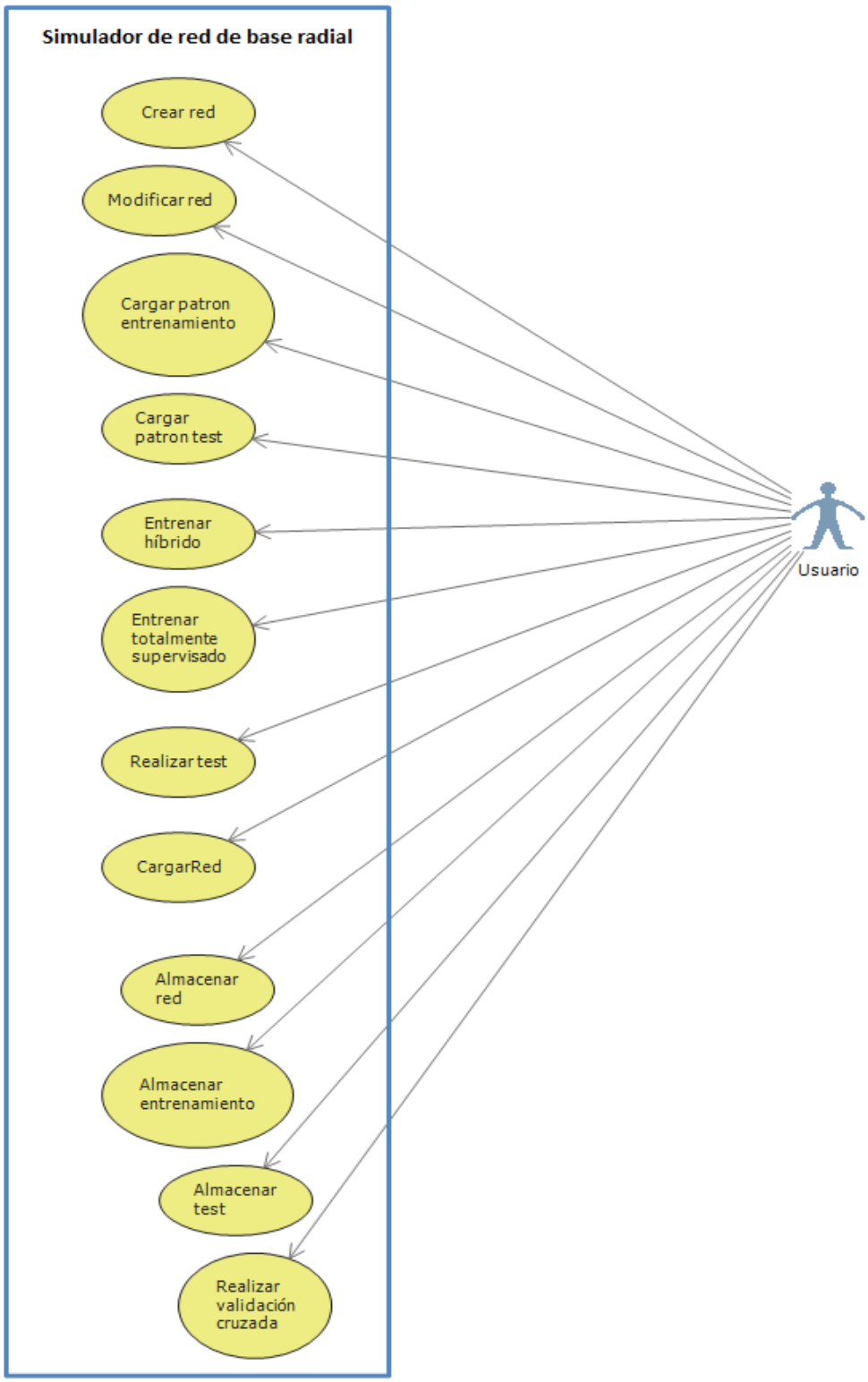

Figura 3.1: Diagrama de casos de uso

A continuación se desglosa las características de cada uno de los casos de uso, indicando para cada uno de ellos, los actores que intervienen, el objetivo del caso de uso y los escenarios previos y finales.

Caso de uso	Crear red
Actores	Usuario
Objetivo	Definir la arquitectura de una nueva red de base radial
Escenario previo	Iniciación del simulador
Escenario final	Se define la red con el número de entradas, ocultas y salidas determinado

Tabla 3.19: Caso de uso para crear una red

Caso de uso	Modificar red
Actores	Usuario
Objetivo	Modificar la red definida con anterioridad
Escenario previo	Haber creado o cargado una red definida con anterioridad
Escenario final	Se modifica la red actual con el número de entradas, ocultas y salidas determinado

Tabla 3.20: Caso de uso para modificar una red

Caso de uso	Entrenar una red de manera híbrida
Actores	Usuario
Objetivo	Realizar entrenamiento híbrido sobre una red, a través de un conjunto de patrones de entrenamiento y test
Escenario previo	Haber cargado un conjunto de entrenamiento y test, establecer el número de ciclos de entrenamiento, cada cuantos ciclos se realizará el test, la razón de aprendizaje y el valor de épsilon
Escenario final	Se obtiene una red entrenada

Tabla 3.21: Caso de uso para entrenamiento híbrido

Caso de uso	Entrenar una red de manera totalmente supervisada
Actores	Usuario
Objetivo	Realizar entrenamiento supervisado sobre una red, a través de un conjunto de patrones de entrenamiento y test
Escenario previo	Haber cargado un conjunto de entrenamiento y test, establecer el número de ciclos de entrenamiento, cada cuantos ciclos se realizará el test y las razones de aprendizaje
Escenario final	Se obtiene una red entrenada

Tabla 3.22: Caso de uso para entrenar totalmente supervisado

Caso de uso	Realizar test
Actores	Usuario
Objetivo	Obtener los resultados de test de una red
Escenario previo	Haber definido una red, cargado el conjunto de patrones de test y haber entrenado una red
Escenario final	Obtención de la salidas de la red para un conjunto de datos de test

Tabla 3.23: Caso de uso para realizar test

Caso de uso	Almacenar red
Actores	Usuario
Objetivo	Recoger en un fichero los parámetros relativos a una red
Escenario previo	Debe haberse creado la red
Escenario final	Se obtendrá un fichero de texto con todos los parámetros de la red

Tabla 3.24: Caso de uso para almacenar una red

Caso de uso	Almacenar entrenamiento
Actores	Usuario
Objetivo	Recoger la información del entrenamiento en un fichero
Escenario previo	Haber realizado entrenamiento sobre una red creada
Escenario final	Se obtiene un fichero con las salidas de la red para los datos de entrenamiento

Tabla 3.25: Caso de uso para almacenar las salidas del entrenamiento

Caso de uso	Almacenar test
Actores	Usuario
Objetivo	Recoger la información del test en un fichero
Escenario previo	Haber realizado test sobre una red creada
Escenario final	Se obtiene un fichero con las salidas de la red para los datos de test

Tabla 3.26: Caso de uso para almacenar las salidas del test

Caso de uso	Cargar patrón entrenamiento
Actores	Usuario
Objetivo	Disponer de los patrones de entrenamiento
Escenario previo	Debe haberse creado una red
Escenario final	Se establecen los patrones de entrenamiento

Tabla 3.27: Caso de uso para cargar el patrón de entrenamiento

Caso de uso	Cargar patrón test
Actores	Usuario
Objetivo	Disponer de los patrones de test
Escenario previo	Debe haberse creado una red
Escenario final	Se establecen los patrones de test

Tabla 3.28: Caso de uso para cargar el patrón de test

Caso de uso	Cargar una red creada
Actores	Usuario
Objetivo	Obtener los datos de una red creada con anterioridad
Escenario previo	Haberse creado la red, y guardado en un fichero
Escenario final	Se obtiene la red entrenada mediante la carga

Tabla 3.29: Caso de uso para cargar una red

Caso de uso	Realizar validación cruzada
Actores	Usuario
Objetivo	Obtener los resultados de aplicación de validación cruzada
Escenario previo	Haberse creado la red y cargado un conjunto de patrones
Escenario final	Se obtiene la red entrenada y validada mediante validación cruzada

Tabla 3.30: Caso de uso para realizar validación cruzada

3.3.2. Requisitos de software funcionales

Los requisitos funcionales establecen la funcionalidad de la que se dispondrá en el simulador, definiendo el comportamiento interno y externo del simulador y estableciendo el comportamiento del sistema.

A continuación se detallan los requisitos funcionales contenidos, cada uno de ellos en tablas, indicando su identificador, el nombre, el tipo de requisito de software, la prioridad, la necesidad y una breve descripción. Para el identificador se ha seguido la siguiente notación: RSF-XX, siendo RSF la abreviatura de Requisitos de Software Funcionales y XX el número de requisito de dos cifras.

Identificador	RSF-01					
Nombre	Creación de redes					
Tipo	**Funcional**		X	**No funcional**		
Prioridad	**Alta**	X	**Media**		**Baja**	
Necesidad	**Obligatorio**		X	**Opcional**		
Descripción	Se podrá crear redes de base radial.					

Tabla 3.31: Requisito de software funcional 1

Identificador	RSF-02					
Nombre	Modificación de redes					
Tipo	**Funcional**		X	**No funcional**		
Prioridad	**Alta**	X	**Media**		**Baja**	
Necesidad	**Obligatorio**		X	**Opcional**		
Descripción	Se podrá modificar redes de base radial.					

Tabla 3.32: Requisito de software funcional 2

Identificador	RSF-03							
Nombre	Entrenamiento híbrido							
Tipo	Funcional		X	No funcional				
Prioridad	Alta	X	Media		Baja			
Necesidad	Obligatorio		X	Opcional				
Descripción	Se podrá realizar un entrenamiento híbrido de la red.							

Tabla 3.33: Requisito de software funcional 3

Identificador	RSF-04							
Nombre	Entrenamiento supervisado							
Tipo	Funcional		X	No funcional				
Prioridad	Alta	X	Media		Baja			
Necesidad	Obligatorio		X	Opcional				
Descripción	Se podrá realizar un entrenamiento totalmente supervisado de la red.							

Tabla 3.34: Requisito de software funcional 4

Identificador	RSF-05							
Nombre	Realizar test							
Tipo	Funcional		X	No funcional				
Prioridad	Alta	X	Media		Baja			
Necesidad	Obligatorio		X	Opcional				
Descripción	Se podrá realizar test sobre una red de base radial.							

Tabla 3.35: Requisito de software funcional 5

Identificador	RSF-06							
Nombre	Almacenar los resultados de entrenamiento							
Tipo	Funcional		X	No funcional				
Prioridad	Alta	X	Media		Baja			
Necesidad	Obligatorio		X	Opcional				
Descripción	Se permitirá guardar los resultados que proporciona la red para el entrenamiento.							

Tabla 3.36: Requisito de software funcional 6

Identificador	RSF-07							
Nombre	Almacenar los resultados de test							
Tipo	Funcional		X	No funcional				
Prioridad	Alta	X	Media			Baja		
Necesidad	Obligatorio		X	Opcional				
Descripción	Se permitirá guardar los resultados que proporciona la red para test.							

Tabla 3.37: Requisito de software funcional 7

Identificador	RSF-08							
Nombre	Almacenar los datos de red							
Tipo	Funcional		X	No funcional				
Prioridad	Alta	X	Media			Baja		
Necesidad	Obligatorio		X	Opcional				
Descripción	Se permitirá almacenar los parámetros de una red.							

Tabla 3.38: Requisito de software funcional 8

Identificador	RSF-09							
Nombre	Realizar validación cruzada							
Tipo	Funcional		X	No funcional				
Prioridad	Alta		Media		X	Baja		
Necesidad	Obligatorio			Opcional			X	
Descripción	Se permitirá realizar validación cruzada sobre un conjunto de datos.							

Tabla 3.39: Requisito de software funcional 9

Identificador	RSF-10							
Nombre	Cargar una red creada							
Tipo	Funcional		X	No funcional				
Prioridad	Alta		Media		X	Baja		
Necesidad	Obligatorio		X	Opcional				
Descripción	Se permitirá importar los parámetros de una red creada con anterioridad.							

Tabla 3.40: Requisito de software funcional 10

Identificador	RSF-11									
Nombre	Número de ciclos de entrenamiento									
Tipo	Funcional		X	No funcional						
Prioridad	Alta	X	Media		Baja					
Necesidad	Obligatorio		X	Opcional						
Descripción	Se podrá establecer el número de ciclos que se realizarán durante el entrenamiento.									

Tabla 3.41: Requisito de software funcional 11

Identificador	RSF-12									
Nombre	Número de ciclos de test									
Tipo	Funcional		X	No funcional						
Prioridad	Alta	X	Media		Baja					
Necesidad	Obligatorio		X	Opcional						
Descripción	Se podrá establecer el número de ciclos que se realizarán durante el test.									

Tabla 3.42: Requisito de software funcional 12

Identificador	RSF-13									
Nombre	Número de ciclos de validación durante entrenamiento									
Tipo	Funcional		X	No funcional						
Prioridad	Alta		Media		X	Baja				
Necesidad	Obligatorio		X	Opcional						
Descripción	Se podrá establecer el número de ciclos en los que se realizará validación durante el entrenamiento.									

Tabla 3.43: Requisito de software funcional 13

Identificador	RSF-14									
Nombre	Carga de patrones de test									
Tipo	Funcional		X	No funcional						
Prioridad	Alta	X	Media		Baja					
Necesidad	Obligatorio		X	Opcional						
Descripción	Se permitirá cargar los patrones de test.									

Tabla 3.44: Requisito de software funcional 14

Identificador	RSF-15							
Nombre	Carga de patrones de entrenamiento							
Tipo	Funcional		X	No funcional				
Prioridad	Alta	X	Media			Baja		
Necesidad	Obligatorio		X	Opcional				
Descripción	Se permitirá cargar los patrones de entrenamiento.							

Tabla 3.45: Requisito de software funcional 15

3.3.3. Requisitos de software no funcionales

Los requisitos no funcionales especifican criterios que deben usarse para juzgar las operaciones dentro de un sistema. De modo que limitan ciertos comportamientos del simulador, no aportando en ningún caso, funcionalidad al sistema.

A continuación se muestran los requisitos no funcionales definidos para este sistema, siguiendo la misma clasificación que para los funcionales, con la diferencia de la notación del identificador que en este caso es la siguiente: RSNF-XX, siendo RSNF las siglas de Requisito de Software No Funcional, y XX el número de dos cifras que determina el requisito no funcional.

Identificador	RSNF-01							
Nombre	Lenguaje de programación							
Tipo	Funcional				No funcional			X
Prioridad	Alta	X	Media			Baja		
Necesidad	Obligatorio		X	Opcional				
Descripción	El simulador se realizará en Java.							

Tabla 3.46: Requisito de software no funcional 1

Identificador	RSNF-02							
Nombre	Lenguaje de la interfaz							
Tipo	Funcional				No funcional			X
Prioridad	Alta		Media			Baja		X
Necesidad	Obligatorio		X	Opcional				
Descripción	El texto de los distintos botones de la interfaz estará en inglés.							

Tabla 3.47: Requisito de software no funcional 2

Identificador	RSNF-03							
Nombre	Visualización de entrenamiento y test							
Tipo	Funcional				No funcional		X	
Prioridad	Alta		Media		X	Baja		
Necesidad	Obligatorio		X		Opcional			
Descripción	Los resultados de entrenamiento y test se mostrarán visualmente.							

Tabla 3.48: Requisito de software no funcional 3

Identificador	RSNF-04							
Nombre	Parámetros de entrenamiento							
Tipo	Funcional				No funcional		X	
Prioridad	Alta		Media		X	Baja		
Necesidad	Obligatorio				Opcional		X	
Descripción	El almacenamiento de los datos deberá tener un formato que permita su fácil exportación y visualización.							

Tabla 3.49: Requisito de software no funcional 4

Identificador	RSNF-05							
Nombre	Mensajes de ayuda y error							
Tipo	Funcional				No funcional		X	
Prioridad	Alta		Media			Baja	X	
Necesidad	Obligatorio				Opcional		X	
Descripción	En caso de introducir algún valor incorrecto, o cargar datos con formato inadecuado, se deberá mostrar un mensaje de ayuda, indicando el error existente.							

Tabla 3.50: Requisito de software no funcional 5

3.4. Análisis de las clases

Una vez establecidos los distintos casos de uso los requisitos, se procede a determinar la estructura de las distintas clases que conformarán el simulador. Para cada clase se mostrará a continuación, sus responsabilidades dentro del simulador, los atributos que contendrán y las distintas operaciones que deberán realizar.

Clase	**ProyectoJFrame**
Responsabilidades	Ofrecer una interfaz al usuario y recoger las peticiones que se van seleccionando
Atributos	Objeto de la clase Train para disponer de las operaciones de entrenamiento. Objeto de la clase Test para disponer de las operaciones de la clase test.

	Objeto de la clase Arquitectura para establecer la arquitectura de la red
Operaciones	CargarPatronEntrenamiento: se encarga de cargar el conjunto de patrones de entrenamiento CargarPatronTest: carga el conjunto de patrones de test. CargarRed: carga una red a partir de un fichero. CrearRed: define una nueva red de base radial. Entrenar: realiza el proceso de entrenamiento de la red. GuardarRed: almacena los datos de la red definida. GuardarSalidaTest: almacena las salidas obtenidas durante el test. GuardarSalidaTrain: almacena las salidas obtenidas durante el entrenamiento. RealizarTest: realiza la validación de test de la red. RealizarValidacionCruzada: realiza la validación cruzada de la red.

Tabla 3.51: Clase ProyectoJFrame

Clase	**Train**
Responsabilidades	Realizar las acciones necesarias para aplicar las fórmulas necesarias efectuando el entrenamiento sobre un determinado patrón
Atributos	Patrones []: contienen los patrones de entrenamiento. Epsilon: representa el valor de épsilon en el entrenamiento.

	Razones de aprendizaje (1,2 y 3): representan las razones de aprendizaje, que según el tipo se usarán todas o solo la primera.
Operaciones	Get: devuelve los valores de los atributos. Set: establece los valores de los atributos. CalcularKMedias: aplica el algoritmo de K-Medias sobre un conjunto de patrones de entrada. CalcularGauss: aplica la función de base radial gaussiana. Calcular Pesos: calcula los pesos de la red durante el entrenamiento CalcularUmbrales: calcula los umbrales de la red durante el entrenamiento. CalcularDesviaciones: calcula las desviaciones de la red durante el entrenamiento híbrido. CalcularDesviacionesSupervisado: En el caso del entrenamiento totalmente supervisado, calcula las desviaciones. CalculaCentrosSupervisado: En el caso del entrenamiento totalmente supervisado, calcula los centros. CalculaValidacionCruzada: Obtiene los diferentes conjuntos de patrones de entrenamiento y test para la validación cruzada.

Tabla 3.52: Clase Train

Clase	**Test**
Responsabilidades	Realizar las acciones necesarias para aplicar las fórmulas necesarias efectuando el test sobre un determinado patrón.
Atributos	Patrones[]: contiene los patrones de test
Operaciones	Get: devuelve los valores de los atributos.

	Set: establece los valores de los atributos. CalcularError: calcula el error para un patrón en cuestión. CalcularGauss: aplica la función de base radial gaussiana. CalcularSalida: obtiene la salida de la red para un patrón.

Tabla 3.53: Clase Test

Clase	Arquitectura
Responsabilidades	Ofrecer la base de la red, almacenando los parámetros que la definen
Atributos	Centros[][], desviaciones[], entradas, ocultas, salidas, pesos[][] y umbrales[]
Operaciones	Inicialización de atributos: se encarga de inicializar los valores de los atributos entre 0 y 1, de forma aleatoria. Get: devuelve los valores de los atributos. Set: establece los valores de los atributos

Tabla 3.54: Clase Arquitectura

Clase	GestionarFichero
Responsabilidades	Leer y escribir en ficheros de texto, para poder cargar los datos y almacenarlos
Atributos	Sin atributos
Operaciones	Escritura: realiza la escritura en un fichero de los parámetros de la red. EscrituraSalidas: realiza la escritura de las salidas de entrenamiento o test Lectura: realiza la lectura de los datos de patrones de un fichero. LecturaRed: realiza la lectura de los parámetros de una red contenidos en un fichero.

Tabla 3.55: Clase GestionarFichero

3.5. Matriz de trazabilidad

Dada la necesidad de satisfacer todos los requisitos establecidos por el usuario, se han establecido las matrices siguientes, en la cuales se representa las relaciones de los requisitos de software y los requisitos de usuario, de tal modo que se garantice la presencia final, en los requisitos establecidos por el desarrollador de todas las necesidades del usuario.

3.5.1. Trazabilidad entre requisitos de usuario de capacidad y funcionales

	RUC-01	RUC-02	RUC-03	RUC-04	RUC-05	RUC-06	RUC-07	RUC-08	RUC-09	RUC-10
RSF-01	X									
RSF-02	X									
RSF-03		X								
RSF-04		X								
RSF-05			X							
RSF-06				X						
RSF-07				X						
RSF-08					X					
RSF-09						X				
RSF-10							X			
RSF-11								X		
RSF-12								X		
RSF-13									X	
RSF-14										X
RSF-15										X

Tabla 3.56: Matriz de trazabilidad de requisitos de capacidad y funcionales

3.5.2. Trazabilidad entre requisitos de usuario de restricción y no funcionales

	RUC-01	RUC-02	RUC-03	RUC-04	RUC-05
RSF-01	X				
RSF-02		X			
RSF-03			X		
RSF-04				X	
RSF-05					X

Tabla 3.57: Matriz de trazabilidad de requisitos de restricción y no funcionales

Capítulo 4: Diseño del sistema

En el presente capítulo se detallará el diseño sobre el que se desarrollará la aplicación, además de establecer las diferentes bases, en cuanto a los que deberá realizar el desarrollo del simulador. Se debe tener en cuenta que el diseño mostrado en este capítulo estará realizado de tal forma que en caso de querer realizar cambios sobre la estructura del simulador, los tiempos de comprensión y entendimiento del funcionamiento de dicha estructura sean los mínimos posible, facilitando así, la aplicación de modificaciones en el menor tiempo posible y utilizando el menor número de recursos necesarios. Además, se incluirán las distintas excepciones que se podrán producir en caso de un mal uso del simulador, junto con la normativa de código que se emplea para desarrollar el simulador. De tal forma que cualquier persona que quiera manejar el código del programa, puede comprenderlo lo más rápidamente posible. Por último, se detallará el entorno tecnológico que será necesario tanto para el desarrollo del simulador, como para la ejecución del mismo.

4.1. Definición de la arquitectura del sistema

La arquitectura correspondiente al simulador de redes de base radial está enfocada en la sencillez y eficacia de dicho simulador, teniendo en cuenta también la vista que se proporciona al usuario y las diferentes interacciones que éste realiza sobre la interfaz. Por este motivo, se ha optado por establecer una capa que sirva de vista para el usuario, además de receptor de peticiones del mismo. Dicha capa, compuesta en este caso por una clase, se encargará de realizar las peticiones necesarias a la segunda capa, para devolver los resultados que se esperan en función de la utilización de la interfaz que proporciona la primera capa.

La segunda capa de la arquitectura se centra en la realización de los diferentes cálculos, como por ejemplo la función de base radial de la red, o la realización del algoritmo K-medias, etc. tomando como parámetros los diferentes datos necesarios, como patrón, o neuronas ocultas de la red. De esta forma la separación por funciones lo más concretas posible, facilitará en gran medida las posibles modificaciones que se pudieran realizar en un futuro.

4.2. Descripción de las iteraciones de las funcionalidades principales

En este apartado, se procederá a detallar las diferentes iteraciones o procesos que siguen para cada una de las funcionalidades principales. De forma que se detallará como es el funcionamiento interno, indicando las llamadas a métodos y clases, mostrando el funcionamiento del simulador a nivel interno, con el fin de facilitar la comprensión de dicho funcionamiento.

4.2.1. Crear red

La figura 4.1, determina las iteraciones que se realizan a nivel interno para realizar la creación de una red. En ella se puede observar cómo a partir de los parámetros de la red recogidos mediante la interfaz, se establecen como atributos de una nueva red, y se procede a inicializar la red con valores aleatorios entre 0 y 1. Finalmente se activan los botones que se pueden utilizar una vez definida la red.

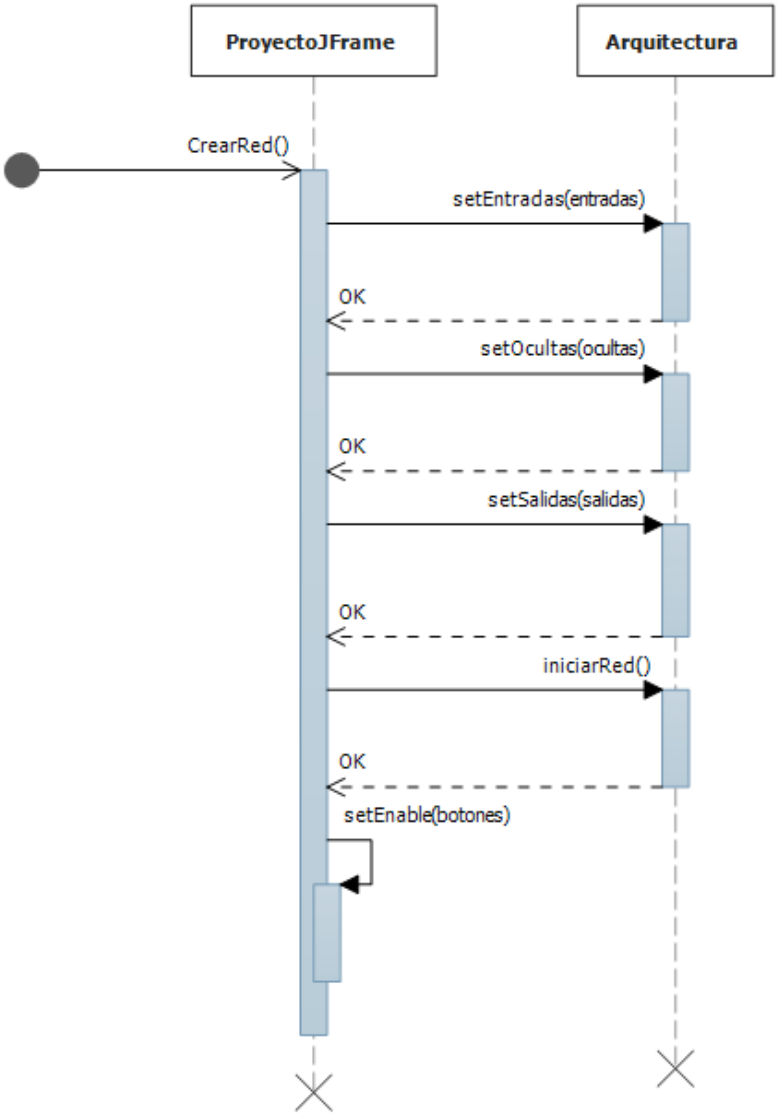

Figura 4.1: Crear red

4.2.2. Modificar red

La figura 4.2, especifica las iteraciones necesarias para llevar a cabo la modificación de la red. Como se puede apreciar, el procedimiento es el mismo que para crear una nueva red, de este modo se edita siempre una sola red, para evitar posibles problemas.

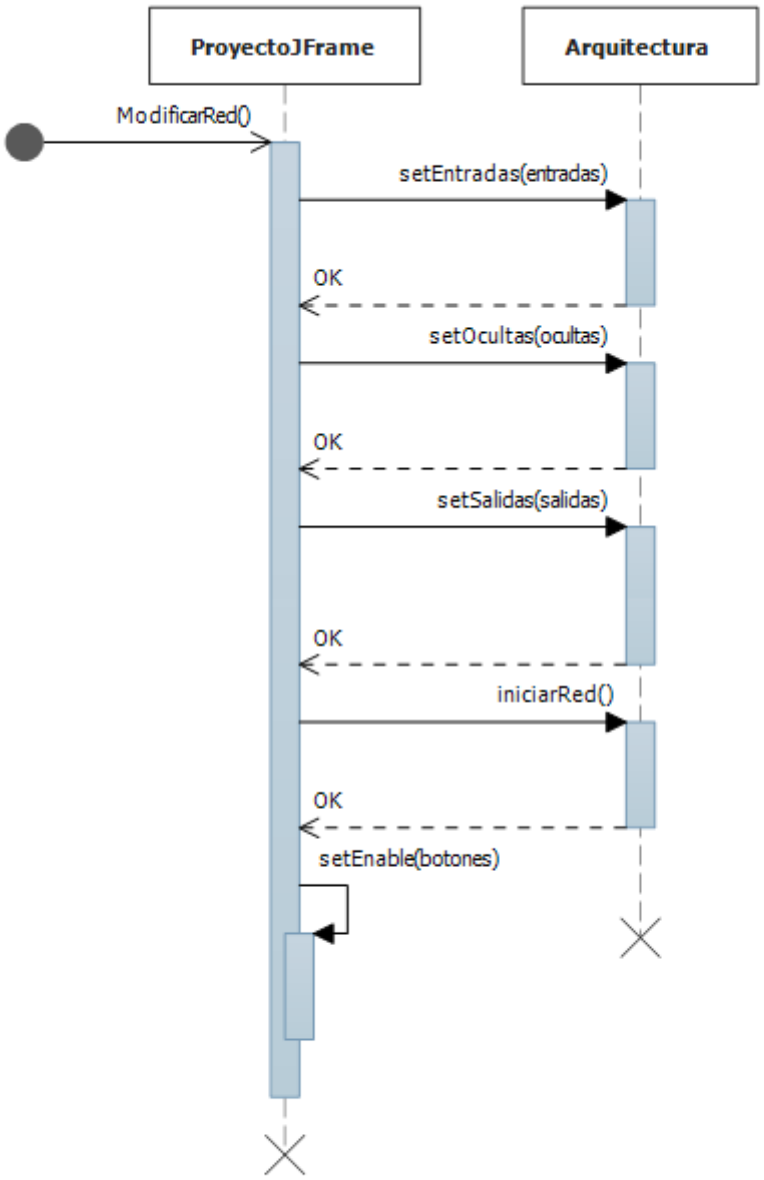

Figura 4.2: Modificar red

4.2.3. Entrenar híbrido

A continuación se especifican las iteraciones que se realizan para llevar a cabo el entrenamiento híbrido. Se puede observar la recogida de los diferentes parámetros relativos al entrenamiento que posteriormente serán establecidos como atributos de la clase "Train". Después, dicha clase se encargará de la realización del algoritmo K-medias para inicializar los centros. Dichos centros serán establecidos en la arquitectura de la red junto con las desviaciones que se calculará a continuación. Para el primer ciclo, se obtendrá la salida y se evaluará el error de la misma, y en función de dicho error, se procederá al cálculo de los pesos y umbrales, que se obtendrán de métodos de la clase "Train". Todo este proceso se realizará tantas veces como ciclos se establezcan y finalizado dicho proceso, se habilitarán los botones correspondientes.

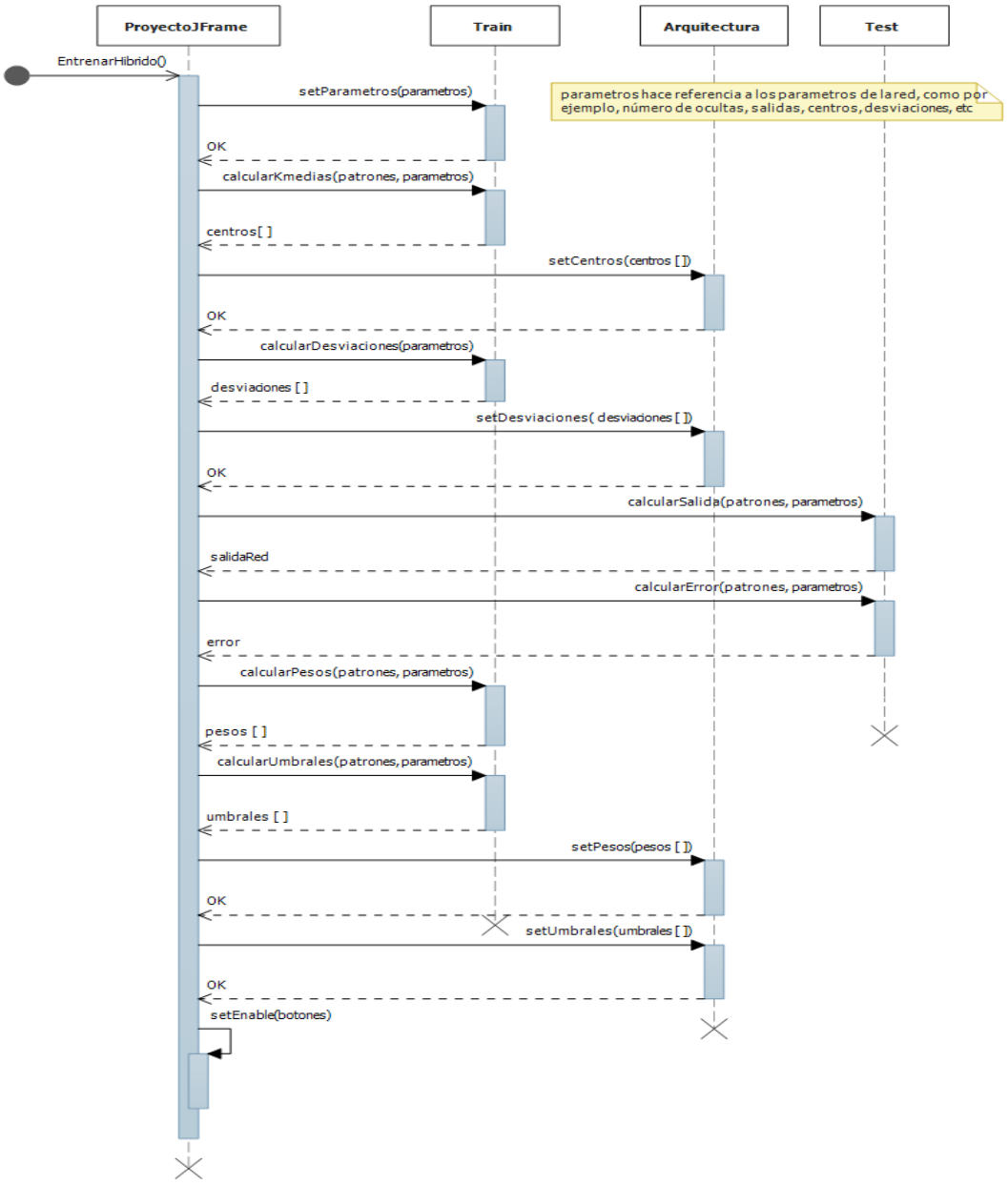

Figura 4.3: Entrenar híbrido

4.2.4. Entrenar totalmente supervisado

A continuación se especifican las iteraciones que se realizan para llevar a cabo el entrenamiento totalmente supervisado. A diferencia del anterior, no se realiza el algoritmo de K-medias, por lo que se ejecutarán los cálculos de pesos, umbrales, centros y desviaciones tantas veces como ciclos, incluyendo el primer ciclo.

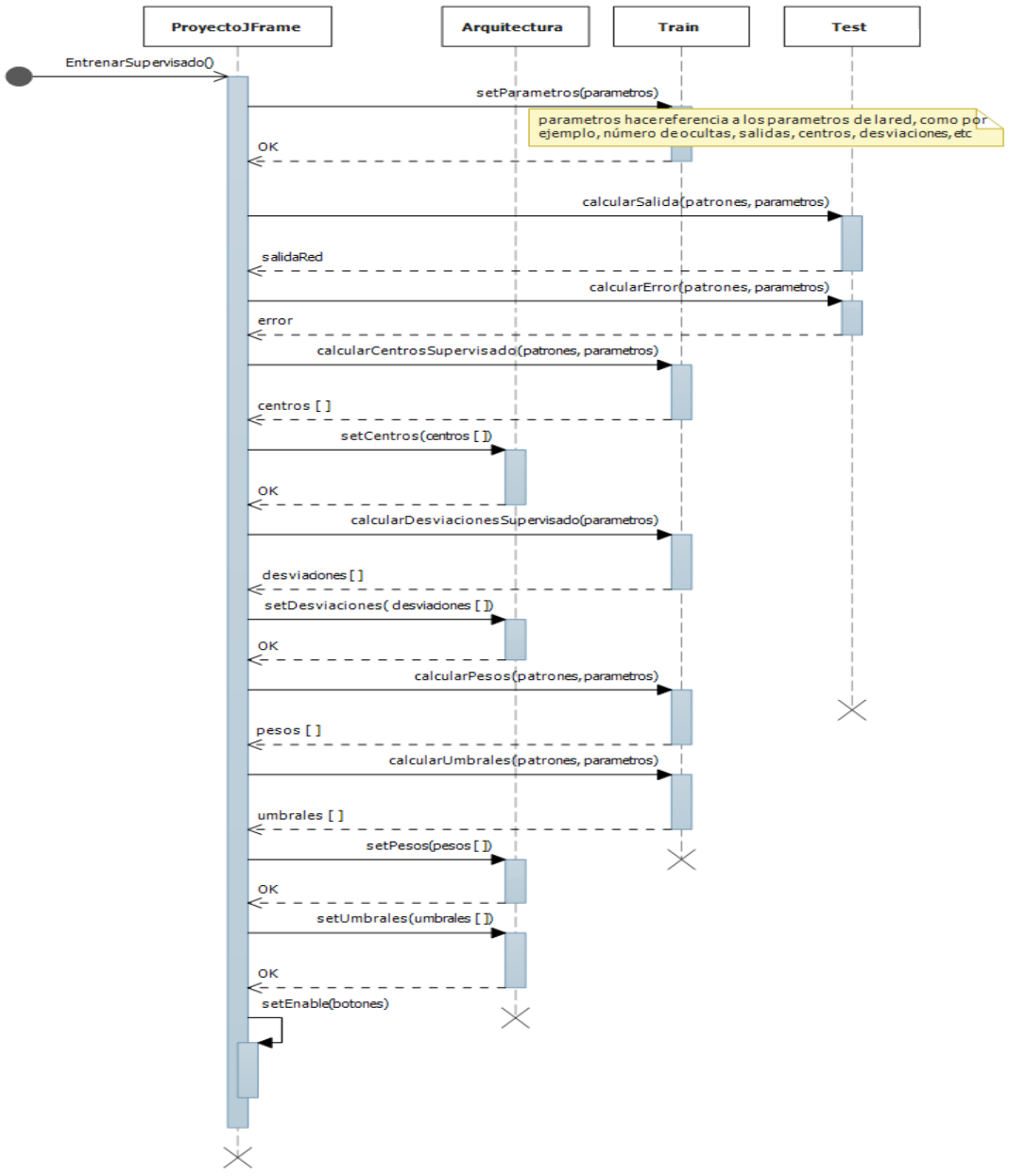

Figura 4.4: Entrenar totalmente supervisado

4.2.5. Realizar test

El siguiente diagrama detalla el proceso de la realización de test. Para este proceso, simplemente se obtiene la salida de la red, para un patrón determinado y posteriormente se evalúa el error correspondiente.

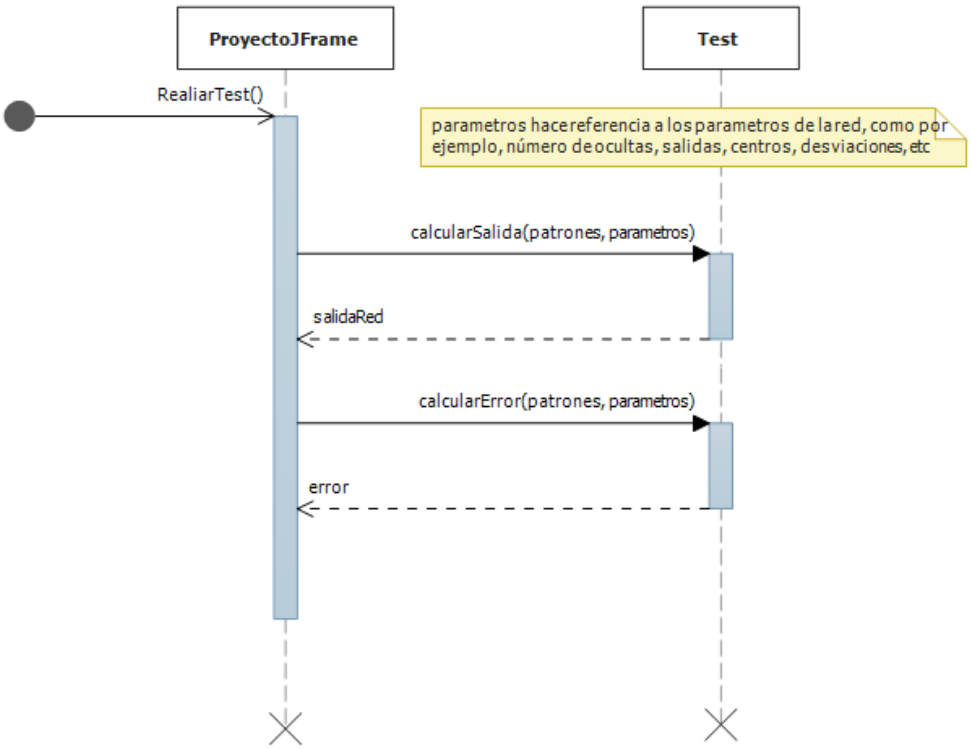

Figura 4.5: Realizar test

4.2.6. Almacenar red

A continuación se especifican la iteración necesaria para almacenar una red. Dado que desde la clase “ProyectoJFrame” se dispone de la información de la red con la que se está trabajando, simplemente se realiza una llamada al método “escribir” de la clase “GestionarFichero” enviando todos los parámetros de la red, para que se proceda a su almacenamiento en un fichero externo.

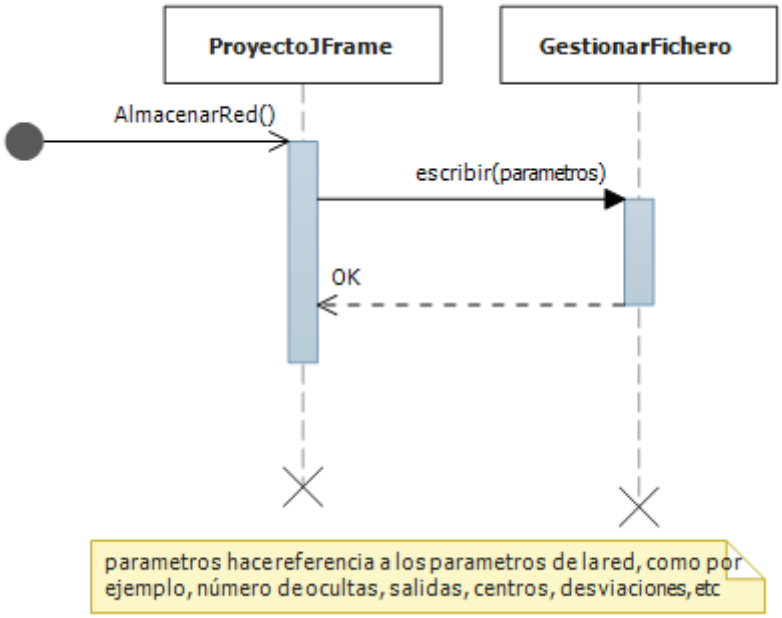

Figura 4.6: Almacenar red

4.2.7. Almacenar salidas entrenamiento

El siguiente diagrama determina el proceso para almacenar las salidas del entrenamiento. Para ello, se realiza un ciclo de test sobre la red con los patrones de entrenamiento, para que no se modifique la red, y posteriormente, del mismo modo que se almacenaba la red, se procede al almacenamiento de las salidas del entrenamiento, llamando al método “escribirSalidas”.

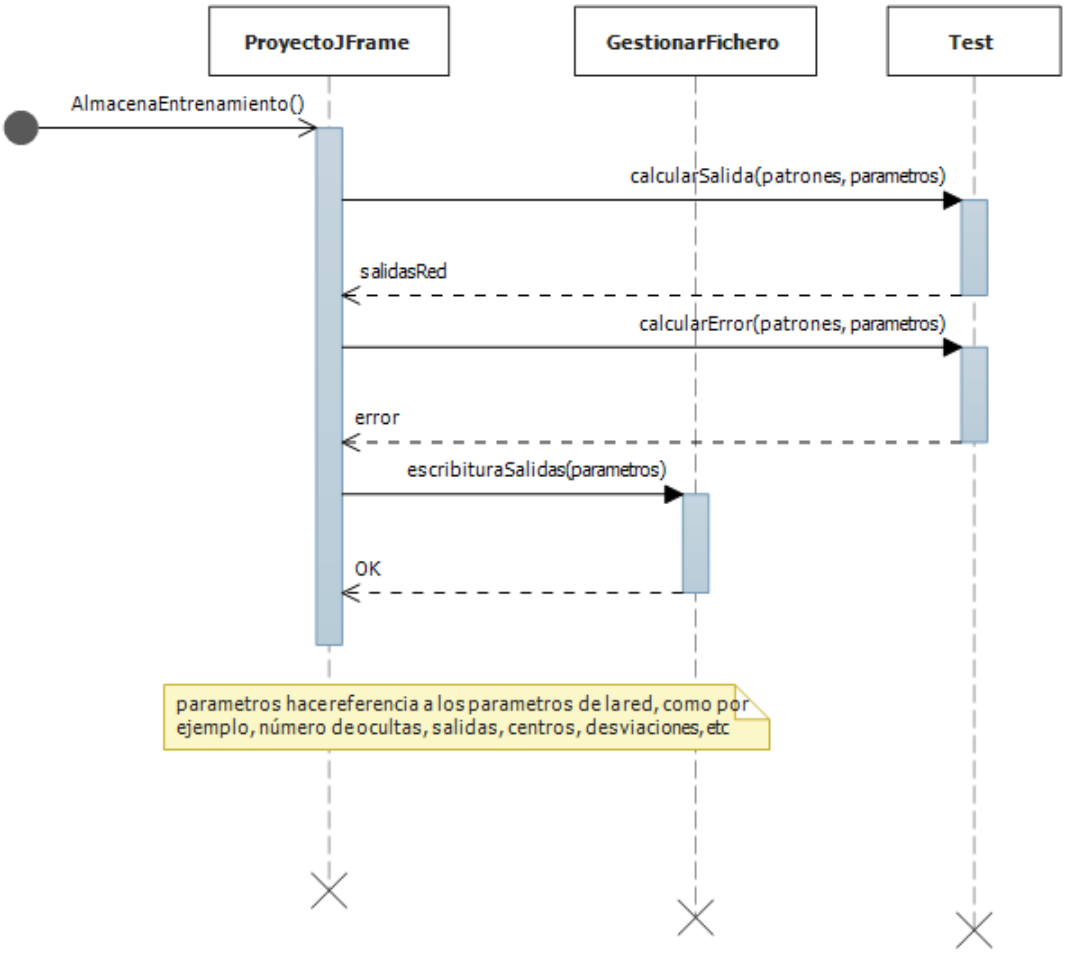

Figura 4.7: Almacenar salidas entrenamiento

4.2.8. Almacenar salidas test

A continuación se especifican las iteraciones para llevar a cabo el almacenamiento de las salidas de test. Como podemos apreciar el procedimiento es similar al almacenamiento de las salidas de entrenamiento, con la única diferencia de utilizar los patrones de test para realizar el test, en lugar de los patrones de entrenamiento.

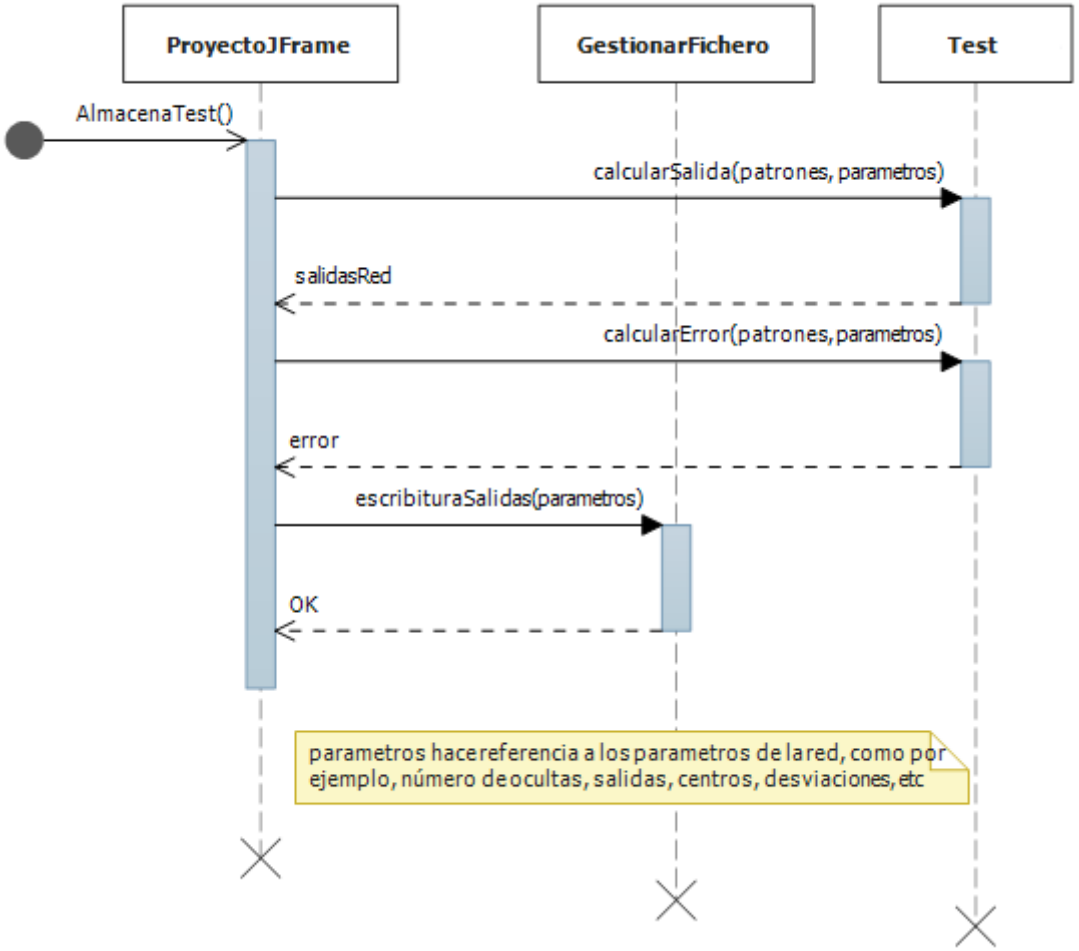

Figura 4.8: Almacenar salidas test

4.2.9. Cargar patrón entrenamiento

La figura 4.9 determina el proceso para realizar la carga de un patrón de entrenamiento. Para ello, se procede a llamar al método "lectura" de la clase "GestionarFichero" que devolverá el conjunto de patrones contenidos en el fichero que se envía como parámetro. Posteriormente, se establecen en la clase "Train" dichos patrones de entrenamiento recibidos.

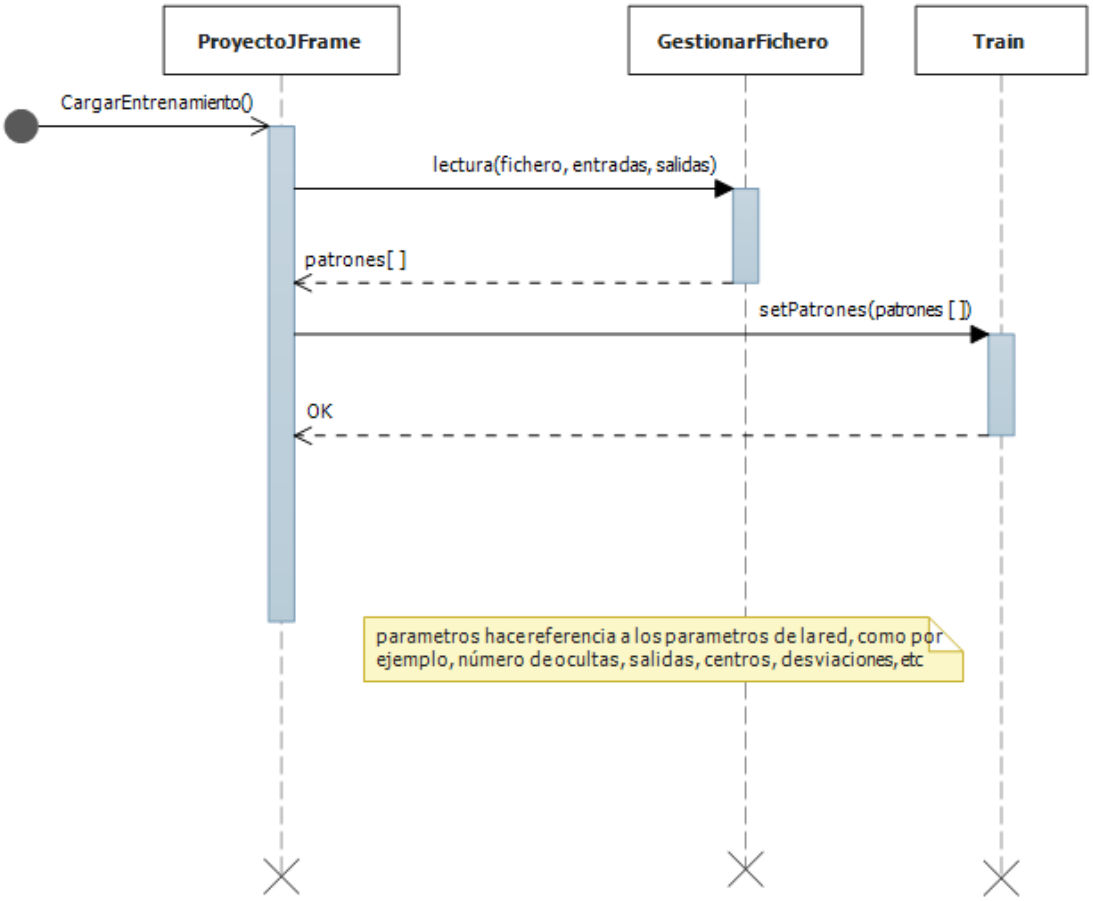

Figura 4.9: Cargar patrón entrenamiento

4.2.10. Cargar patrón test

La siguiente figura determina el proceso para realizar la carga de un patrón de test. Como se puede apreciar, el procedimiento es similar a la carga de patrones de entrenamiento, con la diferencia que los patrones obtenidos se establecen en la clase "Test".

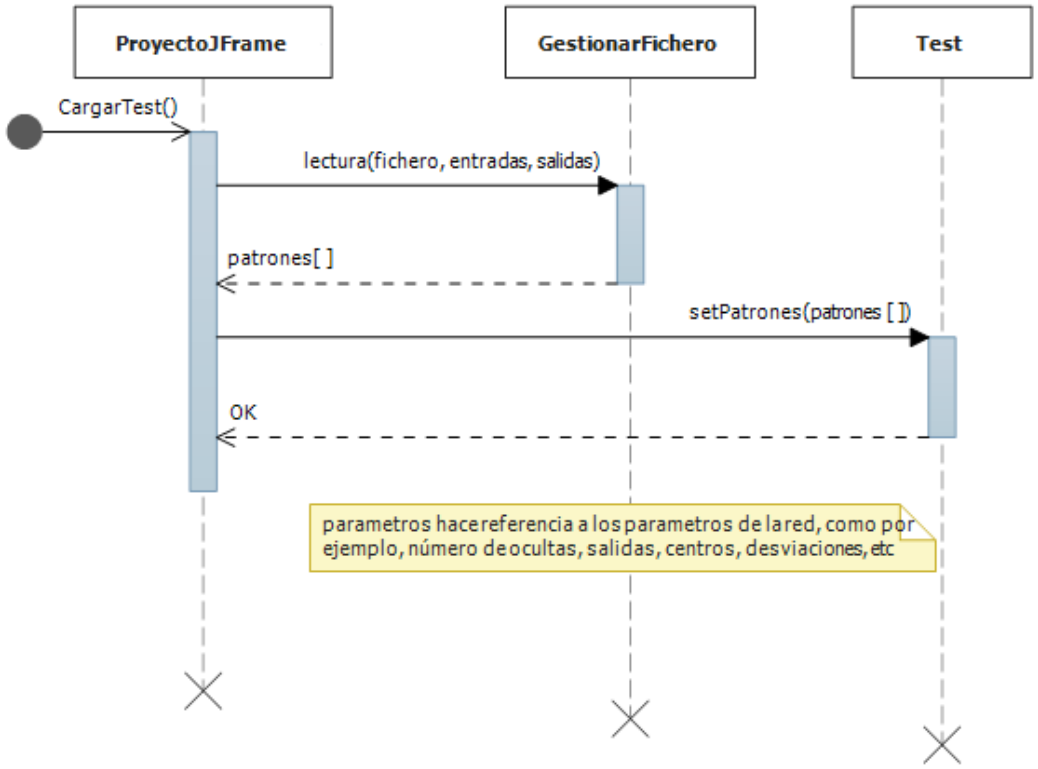

Figura 4.10: Cargar patrón test

4.2.11. Cargar Red

La siguiente figura determina el proceso para realizar la carga de una red. Para ello, se procede a llamar al método “lecturaRed” de la clase “GestionarFichero” que a partir de un fichero pasado como parámetro extrae toda la información y la devuelve en un array de datos. A partir de dicho array, se van estableciendo todos los parámetros de la red cargada realizando las llamadas correspondientes a la clase “Arquitectura”.

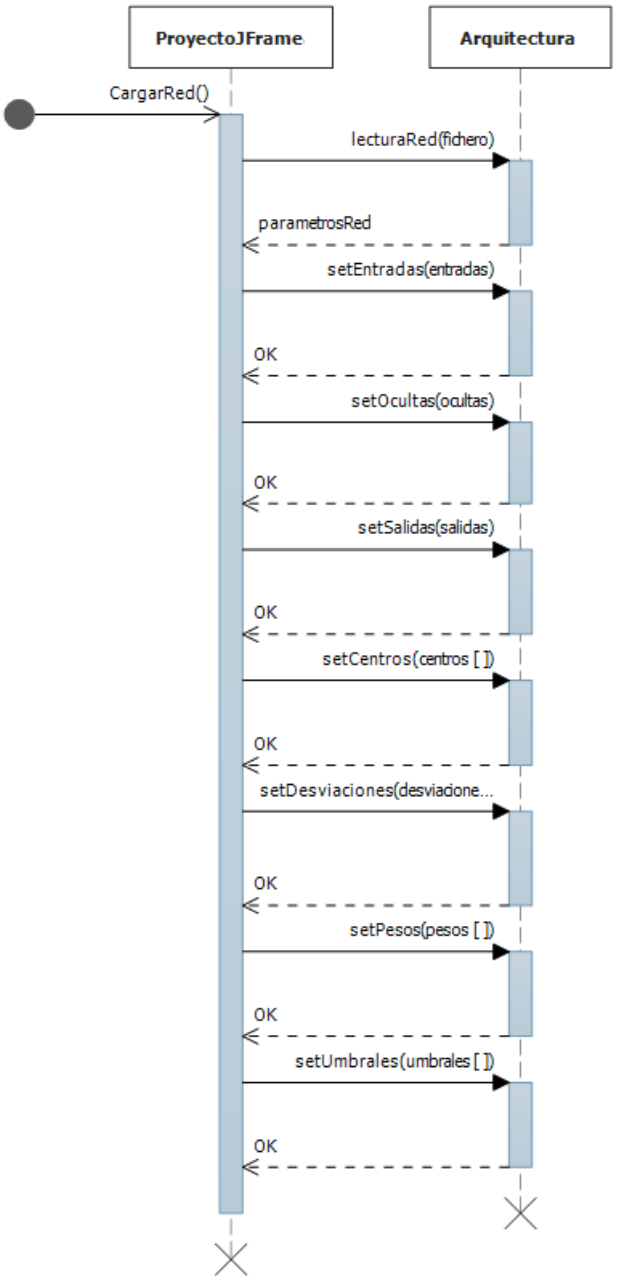

Figura 4.11: Cargar red

4.2.12. Realizar validación cruzada

A continuación se detalla el procedimiento para realizar la validación cruzada. Se puede apreciar como primero, en función del conjunto de grupos con los que se desee dividir el conjunto de patrones para realizar la validación cruzada, se obtienen a través del método “obtenerGrupos” de la clase “Train”. Una vez obtenido el conjunto de entrenamiento y de test, se procede a realizar el entrenamiento y su correspondiente test. A continuación se volverá a reorganizar el conjunto total de patrones para realizar el entrenamiento y test con otros conjuntos diferentes obtenidos a partir de la división del conjunto original. Esto se realizará tantas veces como grupos se determinen.

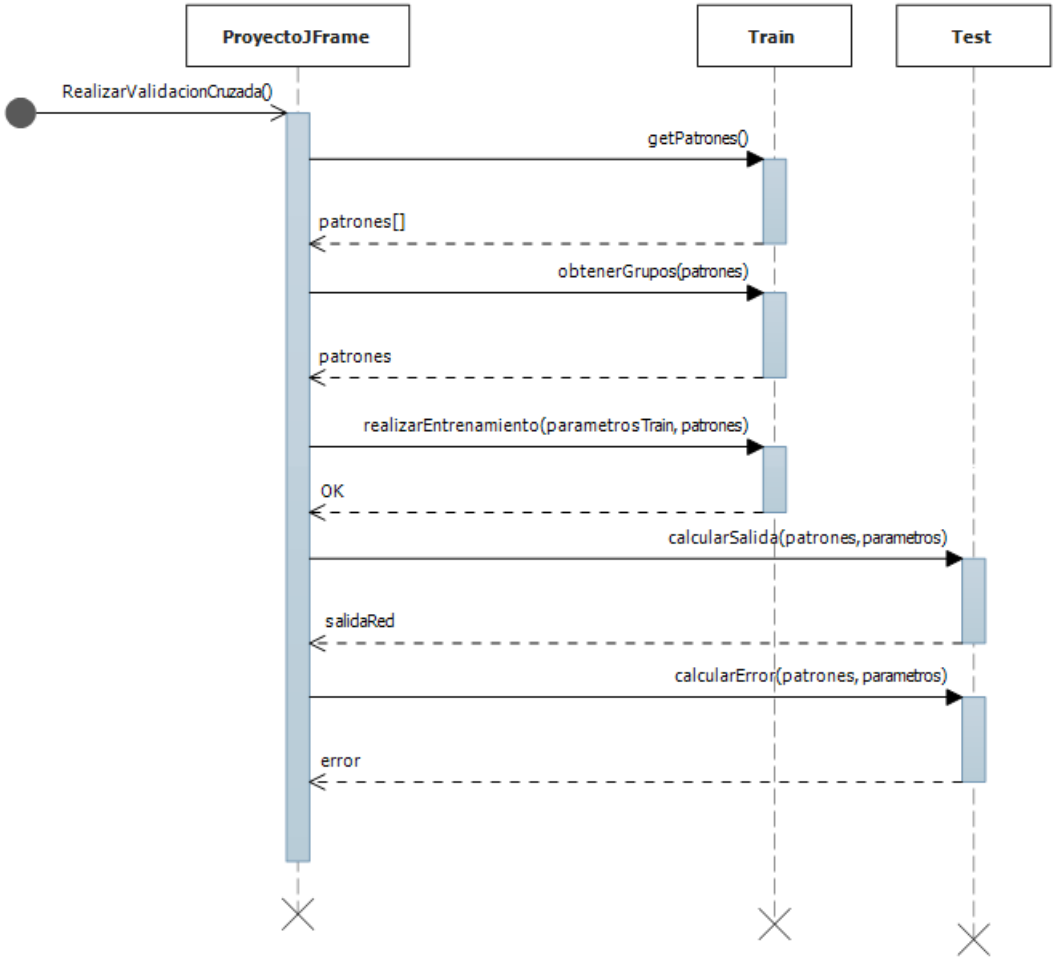

Figura 4.12: Realizar validación cruzada

4.3. Diseño de clases

Una vez detalladas las diferentes funcionalidades por separado del simulador, se va a proceder a representar y explicar el diseño en su conjunto de las clases implicadas en el simulador. En la figura 4.13 podemos ver como se relacionan dichas clases entre sí. Como se puede comprobar, se han utilizado varias clases, con el fin de representar las funcionalidades más generales, de tal forma que para todas las funcionalidades relativas al entrenamiento, se han desarrollado dentro de una clase denominada "Train", y de igual modo para el resto. Para todas las funciones que suponen un manejo de un gestor de ficheros, se ha empleado una clase denominada "GestionarFichero". Para las funcionalidades relativas al test, se emplean funciones de la clase "Test", y para conservar los datos de la arquitectura de la red, se utiliza la clase "Arquitectura".

Finalmente, para visualizar la interfaz y recoger todas las peticiones recibidas a través de los botones que figuran en dicha interfaz, se utiliza la clase "ProyectoJFrame". Dicha clase también gestiona en cierta manera el proceso de cada funcionalidad, realizando llamadas a las diferentes clases enviando los parámetros correspondientes, pero nunca en gran volumen para disponer de una cohesión elevada.

Por último, cabe destacar que el máxima división de las principales funcionalidades para que realizan pequeñas funcionalidades con el objetivo de facilitar el entendimiento del proceso de la funcionalidad, y la modificación que se pueda realizar en un futuro, con el objetivo de incluir nuevas funcionalidades. Por ejemplo, en el caso de realizar entrenamiento o test, se utiliza la función de base radial de gauss, lo cual constituye un método por si mismo, de modo que si queremos añadir otra función de base radial, tan solo sería necesario añadir dicho método y sustituir la llamada correspondiente.

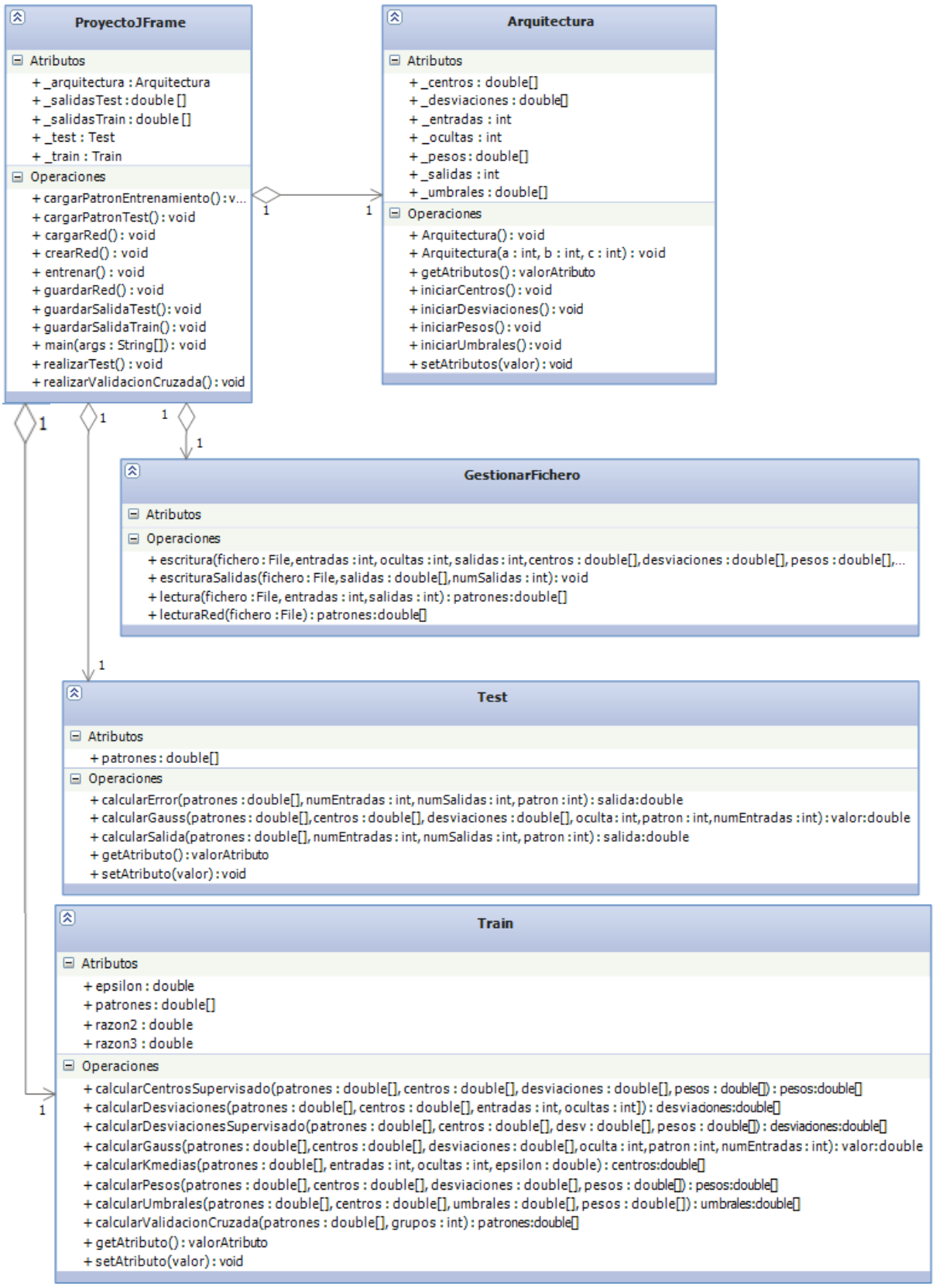

Figura 4.13: Diagrama de clases

4.4. Especificaciones de excepciones

En esta tarea se definen situaciones anómalas que se puedan producir en el funcionamiento y ejecución del simulador. Se crea un catálogo con estas situaciones como ayuda para el diseño del simulador, y como guía en la especificación técnica de las pruebas.

EX_01	**Números negativos**
Descripción	Se ha introducido un valor negativo en alguno de los parámetros del simulador
Estado previo	Usuario utilizando la aplicación
Respuesta	Mensaje de error con el texto “Debe introducir un valor numérico positivo”

Tabla 4.1: Excepción 1

EX_02	**No es un número**
Descripción	Se ha introducido un valor que no representa un valor numérico en alguno de los parámetros del simulador
Estado previo	Usuario utilizando la aplicación
Respuesta	Mensaje de error con el texto “Debe introducir un valor numérico”

Tabla 4.2: Excepción 2

EX_03	**Archivo inexistente**
Descripción	Se ha intentado cargar un archivo que no existe, o no se encuentra en el directorio en cuestión
Estado previo	Usuario utilizando la aplicación
Respuesta	Mensaje de error con el texto “No se ha encontrado el fichero”

Tabla 4.3: Excepción 3

EX_04	**Patrón inadecuado**
Descripción	Se han cargado patrones cuyas características (atributos entrada y salida) no coincide con los parámetros de la red definida
Estado previo	Usuario utilizando la aplicación
Respuesta	Mensaje de error con el texto “EL patrón no se ajusta a la red”

Tabla 4.4: Excepción 4

EX_05	**Parámetros de red incompletos**
Descripción	No se han definido todos los parámetros necesarios para la creación de una red
Estado previo	Usuario utilizando la aplicación
Respuesta	Mensaje de error con el texto “No se han definido todos los parámetros necesarios”

Tabla 4.5: Excepción 5

4.5. Normativa de código

Para proporcionar una mayor sencillez a la hora de comprender el código por el cual se desarrolla el simulador, por parte de personas que no han formado parte del equipo de desarrollo, y para establecer una determinada coherencia entre los diferentes métodos de una clase, y a su vez, entre diferentes clases, se han establecido una serie de normas y estándares de código.

Para dejar constancia del autor de la clase del código en cuestión, se especifica, mediante comentarios el nombre y apellidos del autor, del siguiente modo:

//Autor: Nombre y Apellidos

Cada clase en el código fuente se representa en un fichero independiente, de tal modo que en un mismo fichero no coincidan las especificaciones de dos o más clases. Además, para cada fichero correspondiente al código fuente, se indicará, justo después de la información de los autores, una breve descripción de dicho archivo, igualmente comentado.

Respecto al estándar para el nombrado de clases, métodos y variables, se destacan los siguientes estilos:

- Estilo “PascalCase”: Palabras compuestas con las iniciales en mayúsculas. Por ejemplo: EstoEsUnEjemplo. Se utilizarán para el nombrado de las clases.

- Estilo “CamelCase”: Palabras compuestas con las iniciales en mayúsculas, excepto la primera. Por ejemplo: estoEsUnEjemplo. Se utilizará para el nombrado de métodos y variables. A excepción de las variables de carácter global, que irán precedidas de un guion bajo”_”.

- Estilo “Mayúsculas”: Todas las letras en mayúsculas. Se debe usar sólo en constantes. Por ejemplo: **const** PI = 3.1416.

En cuanto al estilo del código, se han seguido los siguientes consejos:

- Llaves: "{" Una por línea. Siempre usar llaves cuando sea opcional.

- Indentación: Usar tabulaciones de tamaño de 4 espacios en blanco.

- Comentarios: Usar el // para los comentarios de línea y /* para abrir y */ para cerrar un comentario de párrafo.

- Variables: Una variable por declaración.

Para la estructura de los métodos, no se establece ningún límite de líneas, en cuanto a su extensión. Además, tampoco se considera necesario la separación del código perteneciente al mismo por bloques lógicos, ni la descripción de algunas fases

del mismo mediante comentarios. No obstante, sí se ha separado el código por bloques, definidos a juicio del propio autor, así como sus correspondientes comentarios si estima que existe una complejidad elevada.

Para facilitar la legibilidad del código fuente de cada archivo, se considera necesaria la aplicación de una indentación. Dicha indentación consistirá en la presencia de una tabulación en cada línea de código con respecto a la sentencia a la que pertenece. De tal forma, que todo el código relativo a un método, deberá presentar al menos una tabulación con respecto a la declaración del mismo.

Para todas las llaves correspondientes al código fuente, se establecerá la primera llave, o llave de apertura justo al final de la primera línea correspondiente al bucle, condición o sentencia que la requiera; y la última llave, o llave de cierre, se situará justo una línea después de la última sentencia correspondiente, quedando dicha llave aislada del resto de código en dicha línea.

4.6. Entorno tecnológico

En ese apartado se definirá la infraestructura técnica que da soporte al simulador, estimando las necesidades de rendimiento junto con los recursos necesarios para satisfacerlas, seleccionando las implementaciones de hardware y software necesarias para el correcto funcionamiento del simulador.

4.6.1. Hardware

Para el correcto desarrollo de la codificación y ejecución por parte del usuario, se requerirá disponer de un equipo con al menos las siguientes características:

- Procesador Pentium IV. 2.5 GHz

- 1 GB de RAM

- 4 GB de espacio libre en disco duro

- Unidad CD-ROM

- Monitor SGVA

- Ratón y teclado

- Equipo conectado a una red de comunicaciones

4.6.2. Software

Al utilizarse el lenguaje Java para la codificación del simulador se requerirá de la presencia en el equipo, en el cual se vaya a desarrollar el simulador, del Java SE versión 6.

Para el correcto desarrollo del simulador, se utilizará el entorno de programación Netbeans 7.1, el cual facilitará el diseño de la interfaz gráfica y la codificación de las diferentes clases necesarias.

Para la realización de los diferentes diagramas del diseño, tanto de los casos de uso, como de las clases, se utilizará la herramienta de modelado que proporciona Microsoft Visual Studio 2010 Ultimate.

La realización de la documentación de todo el trabajo fin de grado, se realizará utilizando el paquete de gestión ofimática Microsoft Office 2010, disponible en las aulas informáticas de la Universidad Carlos III de Madrid.

Todo esto tendrá como base el sistema operativo Microsoft Windows 7**.**

Finalmente, cabe destacar que para la utilización del simulador por parte de un usuario, solamente será necesaria la presencia de la máquina virtual de Java, dada las características de portabilidad de este lenguaje de programación, sin importar el sistema operativo.

Capítulo 5: Pruebas del sistema y manual de usuario

En este capítulo se procederá a la especificación de las pruebas del simulador desarrollado, así como la verificación del resultado de las mismas. De este modo se podrá garantizar un buen funcionamiento por parte del simulador, junto con la presencia de las funcionalidades que se requerían para dicho simulador. Por último, se detallará el manual de usuario, para facilitar la utilización del simulador.

5.1. Definición de pruebas

La determinación de las pruebas irá en función de la comprobación del cumplimiento de la funcionalidad requerida para el simulador, junto con la confirmación de varias funcionalidades que permitirán afirmar en gran medida, el correcto funcionamiento del simulador, independientemente de la forma de uso que se realice sobre dicho simulador. También, se analizarán los resultados obtenidos para distintos conjuntos de patrones con características diferentes (número de atributos de entrada y salida y número de patrones), para comprobar que el funcionamiento del simulador, respecto al entrenamiento y a la validación mediante test, es adecuado.

A continuación se especificarán las pruebas realizadas para la comprobación del correcto funcionamiento del simulador y el cumplimiento de todos los objetivos asignados. En la especificación de cada prueba se incluyen también las dependencias de los requisitos tratados en la sección 3.4.2 del capítulo 3 que indican la comprobación del funcionamiento correcto de las funcionalidades requeridas durante el desarrollo del simulador. De este modo se asegura que todas las funcionalidades recogidas mediante los requisitos de funcionalidad, funcionan correctamente.

Identificador	**PR-01**
Descripción de la prueba	Se iniciará una red y se almacenará para comprobar que se ha guardado correctamente.
Elementos de entrada	Entradas: 1 Ocultas: 1 Salidas: 1
Especificaciones de salida	Se creará un fichero denominado “red.txt” que contendrá la arquitectura de la red.
Dependencias de requisitos	RSF-01, RSF-08

Tabla 5.1: Prueba 1

Identificador	**PR-02**
Descripción de la prueba	A partir de una red creada, se modificarán las neuronas ocultas de la misma, y se almacenará en un fichero para comprobar dicho cambio.
Elementos de entrada	Debe haber una red con los siguientes parámetros: Entradas: 1 Ocultas: 1

	Salidas: 1 Posteriormente se introduce: Ocultas: 10
Especificaciones de salida	Se modifica, y posteriormente se crea un fichero llamado “redModificada.txt” que contiene la nueva arquitectura de la red.
Dependencias de requisitos	RSF-02, RSF-08

Tabla 5.2: Prueba 2

Identificador	**PR-03**
Descripción de la prueba	Se realizará entrenamiento híbrido sobre una red definida.
Elementos de entrada	Entradas: 1 Ocultas: 5 Salidas: 1 Patrón de entrenamiento: TrainHermite.txt Patrón de test: TestHermite.txt (Ambos patrones contienen la función de Hermite). Ciclos: 100 Mostrar test cada 5 ciclos. Epsilon: 0.0001 Razón de aprendizaje: 0.02
Especificaciones de salida	Una vez seleccionado la opción de entrenar, se mostrará por pantalla el error de la red y el error de test cada 5 ciclos, que estarán enumerados.
Dependencias de requisitos	RSF-01, RSF-03, RSF-11, RSF-13, RSF-14, RSF-15

Tabla 5.3: Prueba 3

Identificador	**PR-04**
Descripción de la prueba	Se realizará el entrenamiento totalmente supervisado sobre una red definida.
Elementos de entrada	Entradas: 1 Ocultas: 10 Salidas: 1 Patrón de entrenamiento: TrainHermite.txt Patrón de test: TestHermite.txt (Ambos patrones contienen la función de Hermite). Ciclos: 200 Mostrar test cada 15 ciclos. Razón de aprendizaje: 0.02 Razón de aprendizaje 2: 0.01 Razón de aprendizaje 3: 0.5
Especificaciones de salida	Una vez seleccionado la opción de entrenar, se mostrará por pantalla el error de la red y el error de

	test cada 15 ciclos, que estarán enumerados.
Dependencias de requisitos	RSF-01, RSF-04, RSF-11, RSF-13, RSF-14, RSF-15

Tabla 5.4: Prueba 4

Identificador	**PR-05**
Descripción de la prueba	Se realizará el test sobre una red creada.
Elementos de entrada	Entradas: 1 Ocultas: 10 Salidas: 1 Patrón de test: TestHermite.txt
Especificaciones de salida	Después de crearse la red, se seleccionará la opción de realizar test y mostrará por pantalla el error de test.
Dependencias de requisitos	RSF-01, RSF-05, RSF-12, RSF-14

Tabla 5.5: Prueba 5

Identificador	**PR-06**
Descripción de la prueba	Se almacenarán las salidas deseadas y las de la red para los conjuntos de entrenamiento y test.
Elementos de entrada	Entradas: 1 Ocultas: 5 Salidas: 1 Patrón de entrenamiento: TrainHermite.txt Patrón de test: TestHermite.txt (Ambos patrones contienen la función de Hermite). Ciclos: 150 Mostrar test cada 10 ciclos. Epsilon: 0.0001 Razón de aprendizaje: 0.02
Especificaciones de salida	Se creará un fichero “Salidas-test.txt” que contendrá las salidas deseadas y las de la red para cada patrón de test, y otro fichero “Salidas-train.txt” que contendrá lo mismo pero para cada patrón de entrenamiento.
Dependencias de requisitos	RSF-01, RSF-03, RSF-06, RSF-07, RSF-11, RSF-13, RSF-14, RSF-15

Tabla 5.6: Prueba 6

Identificador	PR-07
Descripción de la prueba	Se guardará una red.
Elementos de entrada	Entradas: 1 Ocultas: 5 Salidas: 1 Una vez almacenada la red, se modificará la red: Ocultas: 1
Especificaciones de salida	Se almacenará un fichero “red1.txt” que contendrá la información sobre la red. Posteriormente, al modificar la red, se perderán los pesos entrenados, por lo tanto al cargarlos, mediante el archivo “red1.txt” se recuperarán.
Dependencias de requisitos	RSF-01, RSF-10

Tabla 5.7: Prueba 7

Identificador	PR-08
Descripción de la prueba	Se realizará la validación cruzada sobre un conjunto de patrones.
Elementos de entrada	Entradas: 4 Ocultas: 5 Salidas: 3 Patrón de entrenamiento: balance-scale.txt Ciclos: 100 Mostrar test cada 50 ciclos. Epsilon: 0.00001 Razón de aprendizaje: 0.02 Conjuntos validación cruzada: 10
Especificaciones de salida	Una vez finalizado el entrenamiento, deberá mostrarse por pantalla, los 10 errores, para cada conjunto establecido por la validación cruzada.
Dependencias de requisitos	RSF-01, RSF-03, RSF-09, RSF-11, RSF-13, RSF-14, RSF-15

Tabla 5.8: Prueba 8

Identificador	PR-09
Descripción de la prueba	Se comprobará que no se permiten números negativos en los parámetros de entrenamiento y datos de la red.
Elementos de entrada	Entradas: 1 Ocultas: 10 Salidas: 1

	Patrón de entrenamiento: TrainHermite.txt Patrón de test: TestHermite.txt Ciclos: -100 Mostrar test cada 50 ciclos. Razón de aprendizaje: 0.02 Razón de aprendizaje 2: 0.03 Razón de aprendizaje 3: 0.05 Tipo de entrenamiento: totalmente supervisado
Especificaciones de salida	Deberá mostrar un mensaje de error, indicando que no son validos los números negativos.
Dependencias de requisitos	RSF-01, RSF-04

Tabla 5.9: Prueba 9

Identificador	**PR-10**
Descripción de la prueba	Se comprobará que solamente se admiten valores numéricos para los parámetros de entrenamiento y datos de la red.
Elementos de entrada	Entradas: 1 Ocultas: 10 Salidas: 1 Patrón de entrenamiento: TrainHermite.txt Patrón de test: TestHermite.txt Ciclos: 10 Mostrar test cada: **cincuenta** ciclos. Razón de aprendizaje: **cinco** Tipo de entrenamiento: Híbrido
Especificaciones de salida	Debe mostrar un mensaje de error indicando que solo se aceptan números.
Dependencias de requisitos	RSF-01, RSF-03

Tabla 5.10: Prueba 10

Identificador	**PR-11**
Descripción de la prueba	Se comprobará que se han incluido valores para todos los parámetros que interviene en el entrenamiento.
Elementos de entrada	Entradas: 1 Ocultas: 10 Salidas: 1 Patrón de entrenamiento: TrainHermite.txt Patrón de test: TestHermite.txt Ciclos: 100 Mostrar test cada 50 ciclos. Razón de aprendizaje: 0.02

	Razón de aprendizaje 3: 0.05 Tipo de entrenamiento: totalmente supervisado
Especificaciones de salida	Se mostrará un mensaje indicando que faltan por completar parámetros del entrenamiento.
Dependencias de requisitos	RSF-01, RSF-04

Tabla 5.11: Prueba 11

5.2. Trazabilidad de pruebas con requisitos

A continuación se mostrará la relación entre las pruebas detalladas en el apartado anterior, con los requisitos de software, que establecen las diversas necesidades del usuario, destacando el cumplimiento satisfactorio de todas las necesidades señaladas.

	RSF-01	RSF-02	RSF-03	RSF-04	RSF-05	RSF-06	RSF-07	RSF-08	RSF-09	RSF-10	RSF-11	RSF-12	RSF-13	RSF-14	RSF-15
PR-01	X							X							
PR-02		X						X							
PR-03	X		X								X			X	X
PR-04	X			X							X			X	X
PR-05	X				X							X		X	
PR-06	X		X			X	X				X		X	X	X
PR-07	X		X							X	X		X	X	X
PR-08	X		X						X		X		X	X	X
PR-09	X			X											
PR-10	X		X												
PR-11	X			X											

Tabla 5.12: Matriz de trazabilidad entre pruebas y requisitos funcionales

5.3. Resultados de las pruebas realizadas

Es necesario probar que el simulador funciona correctamente, y para ello se han realizado un conjunto de pruebas experimentales. Estas pruebas nos permitirán comprobar que el proceso de aprendizaje de las redes converge a los resultados esperados. Para ello, se requiere la observación de los errores para los patrones y comparar si a medida que el aprendizaje va avanzando, la red va consiguiendo los resultados deseados. Para la realización de estas pruebas se han utilizado varios dominios, que se describen brevemente a continuación:

- Función de Hermite. Esta función presenta una arquitectura de una entrada y una salida. Además está compuesto por 40 patrones de entrenamiento y 200 patrones de test.

- Predicción de las casas en los suburbios de Boston. En este dominio, se muestra una arquitectura de trece entradas y una salida. Está formado por un total de 506 instancias, de las cuales se han utilizado 355 como patrones de entrenamiento y 151 como patrones de test. El objetivo de este dominio es obtener el valor medio de una vivienda a partir de trece características del suburbio y la propia casa.

- Balance-scale. Dicho dominio hace referencia a la actuación de una balanza que contiene dos pesos a cada lado, de tal forma que a partir de dichos pesos, normalizados entre cero y uno, se debe determinar si la balanza está en equilibrio, o se inclina hacia alguno de sus dos lados. Está formado por cuatro entradas y tres salidas. Presenta un total de 625 instancias de las cuales 500 se han utilizado como patrones de entrenamiento y las 125 restantes como patrones de test.

Para cada uno de los conjuntos de entrenamiento y test de los que se dispone, se han realizado numerosas pruebas, pero se han escogido cinco de entre todas esas pruebas para compararlas entre sí y con el resto de conjuntos que representan problemas diferentes. Para cada conjunto se han realizado pruebas con cinco neuronas de la capa oculta, diez y quince, y se realizará tanto para el entrenamiento híbrido como para el totalmente supervisado. Para todas las pruebas se realizarán 500 ciclos, con una razón de aprendizaje de 0.02 (modificación de pesos). En el caso del híbrido se utilizará 0.00001 como valor de épsilon y para el totalmente supervisado se utilizarán 0.01 para la razón de aprendizaje 2 (modificación de centros), y 0.03 para la razón de aprendizaje 3 (modificación de desviaciones). Cada una de las cinco pruebas realizadas para cada caso, se diferencian en la inicialización de la red, por lo tanto la principal diferencia entre dichas pruebas reside en la inicialización aleatoria de los centros de la red.

5.3.1. Función de Hermite

Este conjunto de datos requiere de una arquitectura de una neurona en la capa de entrada y otra de salida. Se ha elegido como primer conjunto de prueba, debido fundamentalmente a su simplicidad.

Cada una de las cinco pruebas realizadas para cada caso, se diferencian en la inicialización de la red, por lo tanto la principal diferencia entre dichas pruebas reside en la inicialización aleatoria de los centros de la red. En las tablas 5.13, 5.14 y 5.15 se muestran los resultados en términos de error de entrenamiento y test para las pruebas

con 5, 10 y 15 neuronas cuando se realiza un entrenamiento híbrido de las redes de base radial.

Prueba	Error de entrenamiento	Error de test
1	0.008160244216099682	0.007764653112694836
2	0.0099305673058412 6	0.009343335642719873
3	0.008369984773198964	0.007778772780883902
4	0.009070375701979194	0.008413639272847762
5	0.009321887847740917	0.008649631963843506

Tabla 5.13: Resultados para 5 neuronas ocultas en híbrido, para la función Hermite

Prueba	Error de entrenamiento	Error de test
1	3.612058001671265E-5	3.473397403227064E-5
2	4.6420072292399146E-4	4.5132813026116907E-4
3	8.128964244561038E-5	7.936135947595095E-5
4	2.269407268486999E-4	1.962978304584373E-4
5	1.8847422843066872E-4	1.757460562644713E-4

Tabla 5.14: Resultados para 10 neuronas ocultas en híbrido, para la función Hermite

Prueba	Error de entrenamiento	Error de test
1	3.0291742195613273E-4	3.312648142149345E-4
2	1.4441800030944783E-4	1.2358827654697734E-4
3	5.576140347010075E-5	5.0936421212799294E-5
4	8.991587695351005E-5	1.0234957030058247E-4
5	6.069167711831389E-5	9.293916625665933E-5

Tabla 5.15: Resultados para 15 neuronas ocultas en híbrido, para la función Hermite

En las tablas 5.16, 5.17 y 5.18, se muestran los resultados obtenidos cuando se lleva a cabo el aprendizaje totalmente supervisado.

Prueba	Error de entrenamiento	Error de test
1	0.01155429471873897	0.011233926150475812
2	0.006526348862489112	0.0063019467484624865
3	0.012241048154804634	0.012485038053811945
4	0.015295978154972873	0.015737194825709365
5	0.011569379174621868	0.011021646921357217

Tabla 5.16: Resultados para 5 neuronas ocultas en supervisado, para la función Hermite

Prueba	Error de entrenamiento	Error de test
1	0.007780317279774828	0.008382211338919133
2	0.009510587767090718	0.009564590691185298
3	0.01201160804303957	0.015132590287956825
4	0.007256934890293026	0.0071469492344631924
5	0.013025588714553709	0.013996147806004618

Tabla 5.17: Resultados para 10 neuronas ocultas en supervisado, para la función Hermite

Prueba	Error de entrenamiento	Error de test
1	0.007164576610413022	0.007996481818339098
2	0.003763356675284655	0.004068500016375204
3	0.00649427135066709	0.007117249128639233
4	0.004572626711483251	0.005216121202889098
5	0.005332942382218498	0.006132561113382816

Tabla 5.18: Resultados para 15 neuronas ocultas en supervisado, para la función Hermite

A la vista de los resultados obtenidos podemos señalar que las redes de base radial aproximan correctamente a la función de Hermite ya que el error obtenido es adecuado, en algunos casos mejor que en otros debido al número de neuronas ocultas seleccionado. Además, si se observan las salidas deseadas y las que se obtienen con la red, tanto en el entrenamiento como en el test, se puede comprobar que en la mayoría, la diferencia entre ambas es mínima.

A continuación se muestran unas gráficas que representa las salidas deseadas y las obtenidas por la red, para el entrenamiento híbrido de 10 neuronas definido anteriormente, tanto para el entrenamiento como para el test correspondiente.

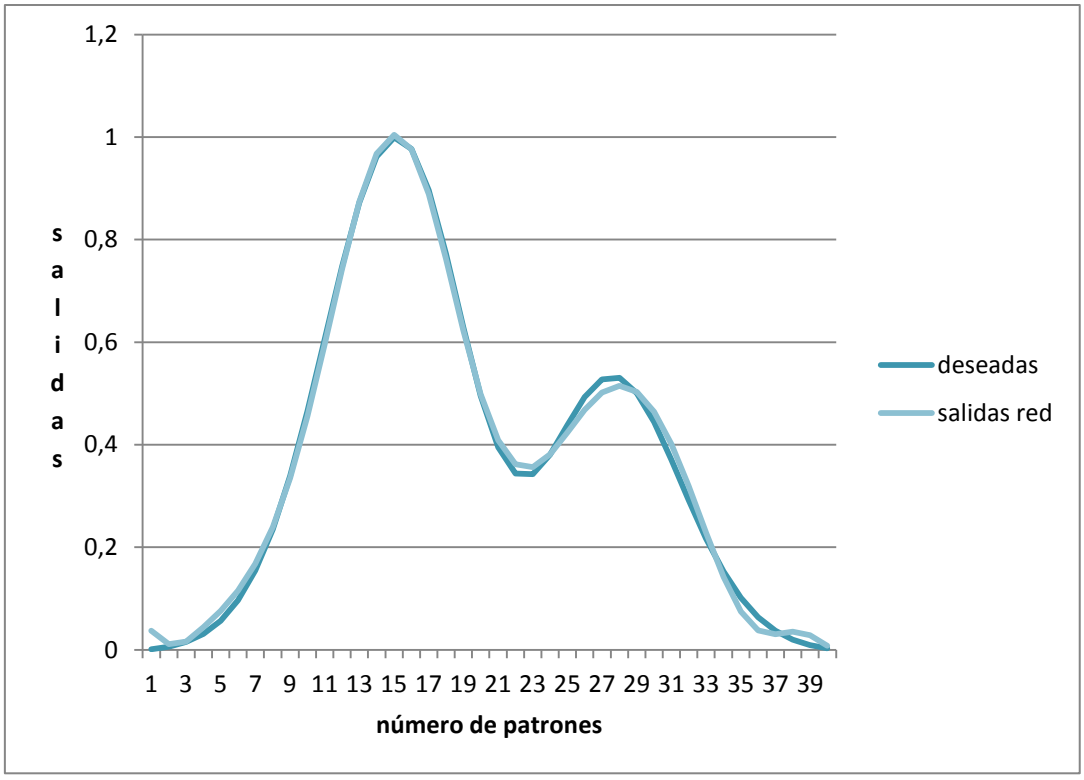

Figura 5.1: Gráfico de entrenamiento para la función Hermite

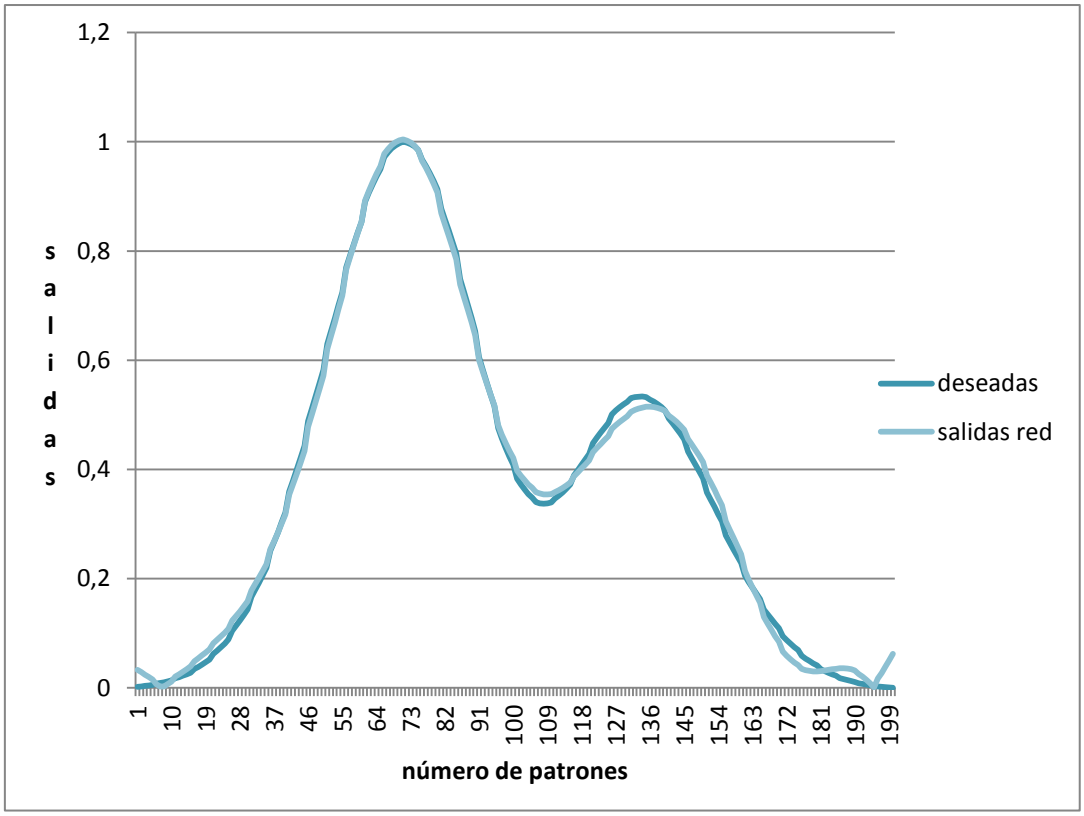

Figura 5.2: Gráfico para salidas de test de la función Hermite

Por último, también se puede afirmar que para este problema en cuestión, la utilización del entrenamiento híbrido es mejor que el entrenamiento totalmente supervisado, pero si se utiliza este último, se obtienen unos resultados buenos igualmente.

5.3.2. Predicción del precio de las casas en los suburbios de Boston

Para el siguiente conjunto de pruebas, se ha utilizado este dominio dado que presenta un mayor número de entradas, además de ser más complejo de resolver.

Se han realizado cinco pruebas para cada caso ya que cada una de tales pruebas representa una inicialización aleatoria diferente de los centros de la red. En las tablas 5.19, 5.20 y 5.21 se muestran los resultados en términos de error de entrenamiento y test para las pruebas con 5, 10 y 15 neuronas cuando se realiza un entrenamiento híbrido de las redes de base radial.

Prueba	Error de entrenamiento	Error de test
1	0.01324998510208144	0.012825972384980918
2	0.013395646052868767	0.012871989975658225
3	0.01464437771724022	0.013530939709650908
4	0.014653277388597258	0.0133300422300821
5	0.014568966593133911	0.013849722186262788

Tabla 5.19: Resultados para 5 neuronas ocultas en híbrido

Prueba	Error de entrenamiento	Error de test
1	0.006356005279459481	0.007061894459629951
2	0.009602489189976105	0.008285668633004728
3	0.008899940816829748	0.008368876575806839
4	0.007773526156982228	0.007076611026280842
5	0.009386911312533231	0.010315660603910629

Tabla 5.20: Resultados para 10 neuronas ocultas en híbrido

Prueba	Error de entrenamiento	Error de test
1	0.006204196104986834	0.0058069381041166235
2	0.006674955257558799	0.006371695811543367
3	0.007008005758911149	0.006182658140601244
4	0.005601624536940624	0.005260838569589892
5	0.0070686787702027585	0.006005621153300741

Tabla 5.21: Resultados para 15 neuronas ocultas en híbrido

En las tablas 5.22, 5.23 y 5.24, se muestran los resultados obtenidos, cuando se lleva a cabo el aprendizaje totalmente supervisado.

Prueba	Error de entrenamiento	Error de test
1	0.017330053421044368	0.015427080964267719
2	0.019662605973207074	0.01917193618104115
3	0.01379669000344062	0.01211315024746232
4	0.017886652492718546	0.017333293364309388
5	0.019512139857806293	0.01848121146141456

Tabla 5.22: Resultados para 5 neuronas ocultas en supervisado

Prueba	Error de entrenamiento	Error de test
1	0.015180334641431967	0.013108584112951911
2	0.013330231597992633	0.0126204771324899
3	0.01245271713763403	0.011522789824579816
4	0.015358551689117494	0.014175862479637596
5	0.010563022038070668	0.010081113200760446

Tabla 5.23: Resultados para 10 neuronas ocultas en supervisado

Prueba	Error de entrenamiento	Error de test
1	0.010543984243145559	0.009607826241076188
2	0.006865712685004886	0.00672791802830579
3	0.011598915870804966	0.011952831171748893
4	0.008609962755152854	0.008452079310550954
5	0.011456002885893483	0.012481816910124234

Tabla 5.24: Resultados para 15 neuronas ocultas en supervisado

A continuación se muestran unas gráficas que representan las salidas deseadas y las obtenidas por la red para este dominio, para entrenamiento y test, teniendo como base el entrenamiento híbrido de 10 neuronas definido anteriormente.

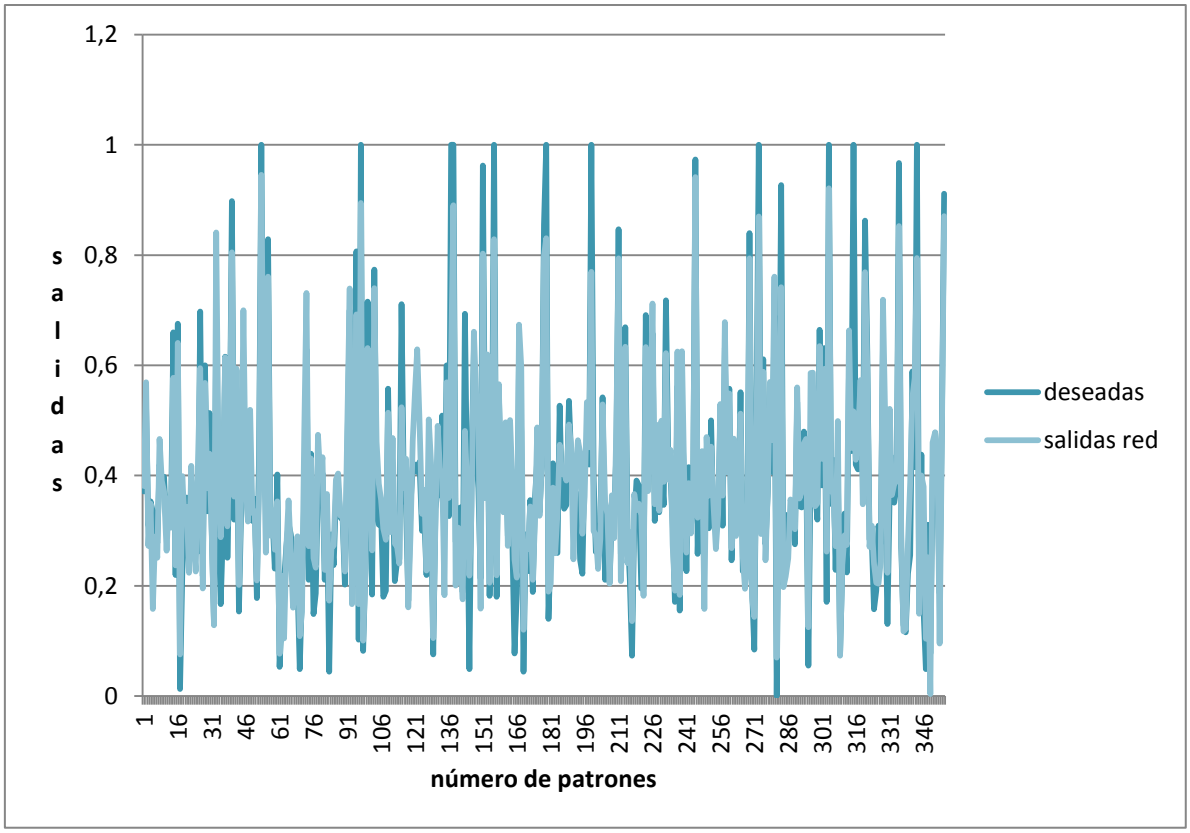

Figura 5.3: Gráfico de las salidas de entrenamiento

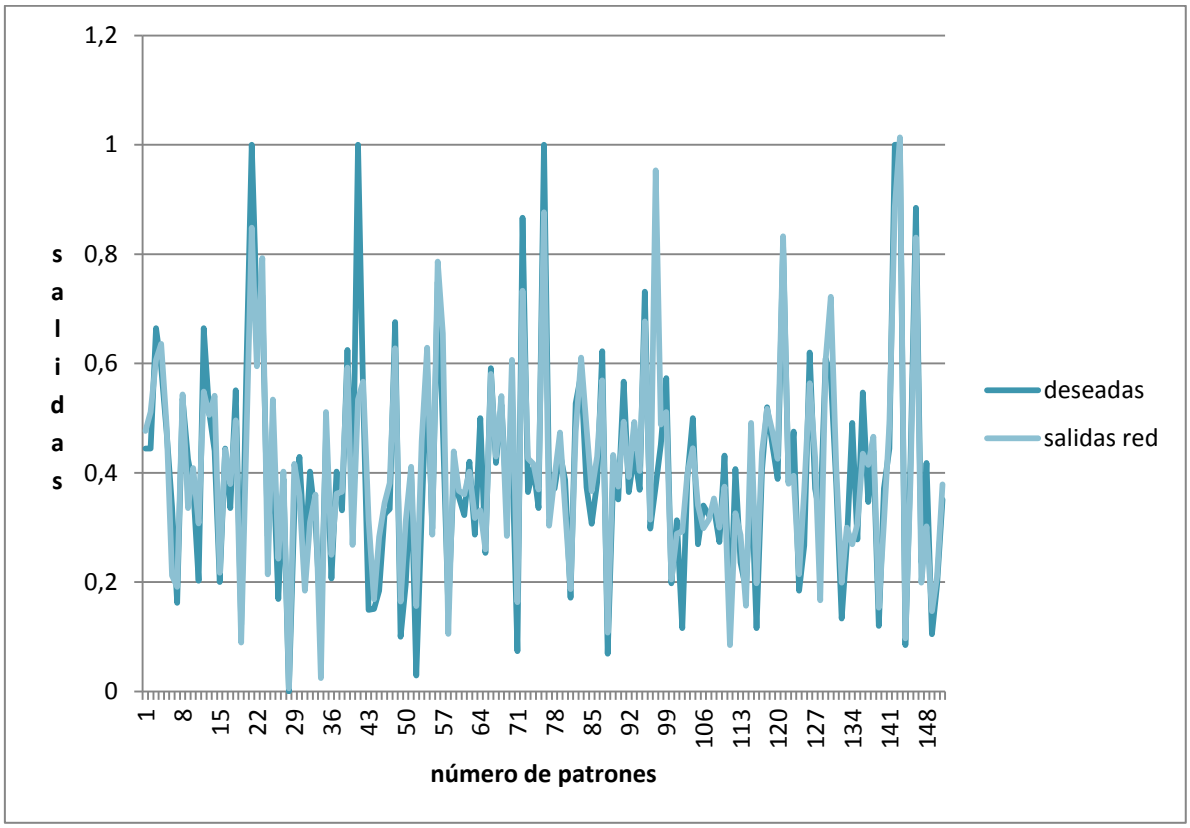

Figura 5.4: Gráfico de las salidas de test

Observando los resultados obtenidos podemos destacar que no existe gran diferencia entre ambos tipos de entrenamiento para este dominio, aunque de nuevo se obtienen mejores resultados con el entrenamiento híbrido. No obstante, existe una cierta relación entre el número de neuronas ocultas utilizado, pero que podría llegar a ignorarse porque en algunos casos es inapreciable.

5.3.3. Balance-scale

Por último se ha escogido un tercer dominio que está formado por cuatro entradas y tres salidas, representando un problema de clasificación en tres clases, dado que representa un problema con un mayor número de salidas que el resto, es decir, mayor que una.

Cada una de las cinco pruebas realizadas para cada caso, se diferencian en la inicialización de la red, lo que implica una distinta inicialización aleatoria de los centros de la red. En las tablas 5.25, 5.26 y 5.27 se muestran los resultados en términos de error de entrenamiento y test para las pruebas con 5, 10 y 15 neuronas cuando se realiza un entrenamiento híbrido de las redes de base radial.

Prueba	Error de entrenamiento	Error de test
1	0.12834327370229742	0.15341568715654624
2	0.1266982495005789	0.14801138782623977
3	0.12434396935976519	0.15448547804755985
4	0.12356405766468648	0.14509397383703648
5	0.12603916000511148	0.1520779341656406

Tabla 5.25: Resultados para 5 neuronas ocultas en híbrido, para Balance-scale

Prueba	Error de entrenamiento	Error de test
1	0.11532589574265696	0.14393736176996408
2	0.11326950404114139	0.13247653867061862
3	0.11401647156329363	0.14203280561651835
4	0.11542156918431938	0.13745607429595713
5	0.11459184882884815	0.1379817418489281

Tabla 5.26: Resultados para 10 neuronas ocultas en híbrido, para Balance-scale

Prueba	Error de entrenamiento	Error de test
1	0.10833522193807421	0.13430540225440513
2	0.10052464662552286	0.1307247937028568
3	0.09983063870284417	0.12994885582380583
4	0.10049374917313879	0.12854928709309374
5	0.09755522636121858	0.13126658669230784

Tabla 5.27: Resultados para 15 neuronas ocultas en híbrido, para Balance-scale

En las tablas 5.28, 5.29 y 5.30, se muestran los resultados obtenidos, cuando se lleva a cabo el aprendizaje totalmente supervisado.

Prueba	Error de entrenamiento	Error de test
1	0.14813182076078907	0.17601720025209697
2	0.16319249259670807	0.19296804377749183
3	0.21070792134343855	0.22369117378671052
4	0.1702235501293887	0.1808466370054482
5	0.18386131054178084	0.19937345790668834

Tabla 5.28: Resultados para 5 neuronas ocultas en supervisado, para Balance-scale

Prueba	Error de entrenamiento	Error de test
1	0.12398708737162879	0.163816329133766
2	0.14514303296302863	0.1964717152979835
3	0.1455349743206971	0.18443483131058158
4	0.1401685424802217	0.14781621836948403
5	0.13806857350832136	0.15837049772010592

Tabla 5.29: Resultados para 10 neuronas ocultas en supervisado, para Balance-scale

Prueba	Error de entrenamiento	Error de test
1	0.11243186769886085	0.1353412961179578
2	0.1236053436403176	0.14902163841881824
3	0.11878843466580974	0.1415238695594315
4	0.12043261787350874	0.15327780814034936
5	0.12368911016406298	0.15069780301398228

Tabla 5.30: Resultados para 15 neuronas ocultas en supervisado, para Balance-scale

En este caso, podemos observar como el número de neuronas óptimo para la obtención de buenos resultados, depende del tipo de entrenamiento empleado, que si lo comparamos a igual número de neuronas, podría indicarse que el entrenamiento híbrido es ligeramente mejor. Sin embargo, la diferencia de errores entre uno y otro es mínima. En cualquier caso, de nuevo se obtiene un error aceptable y se muestra el correcto funcionamiento del simulador.

5.3.4. Conclusiones

Analizando los resultados obtenidos para los diferentes dominios podemos concluir que el simulador realiza su funcionalidad de manera apropiada, ya que en todos los casos, entrena y valida la red definida, obteniendo los resultados esperados.

Éstos varían en función del tipo de entrenamiento, de los parámetros de dicho entrenamiento, de las neuronas de la capa oculta de la red, además del propio dominio del problema. Podemos afirmar, por tanto, que el simulador representa fielmente la estructura y funcionamiento de las redes de neuronas, debido a que se ajusta a las características de cada problema y ofrece una solución adecuada.

Por último, es necesario destacar que la elección de la estructura de la red y de los diferentes parámetros relativos al entrenamiento, así como el tipo de entrenamiento puede afectar considerablemente a los resultados obtenidos, ya que en muchos casos, si no se seleccionan los parámetros adecuados, se pueden obtener unos resultados no del todo satisfactorios, lo cual depende de la propia red de base radial, no de su simulación.

5.4. Manual de usuario

En esta sección se procederá a explicar cada una de las posibles acciones que puede realizar un usuario, de forma que pueda utilizar todas las funcionalidades del simulador, sin necesidad de tener algún tipo de conocimiento sobre el uso de simuladores de este tipo.

5.4.1. Crear una red

Una de los primeros pasos que se debe hacer para empezar a utilizar el simulador es definir la arquitectura de la red que se quiere simular. Para ello, se debe elegir el número de neuronas de la capa de entrada, el número de neuronas ocultas de la red y el número de neuronas de salidas. Todos estos parámetros deben ser valores numéricos positivos.

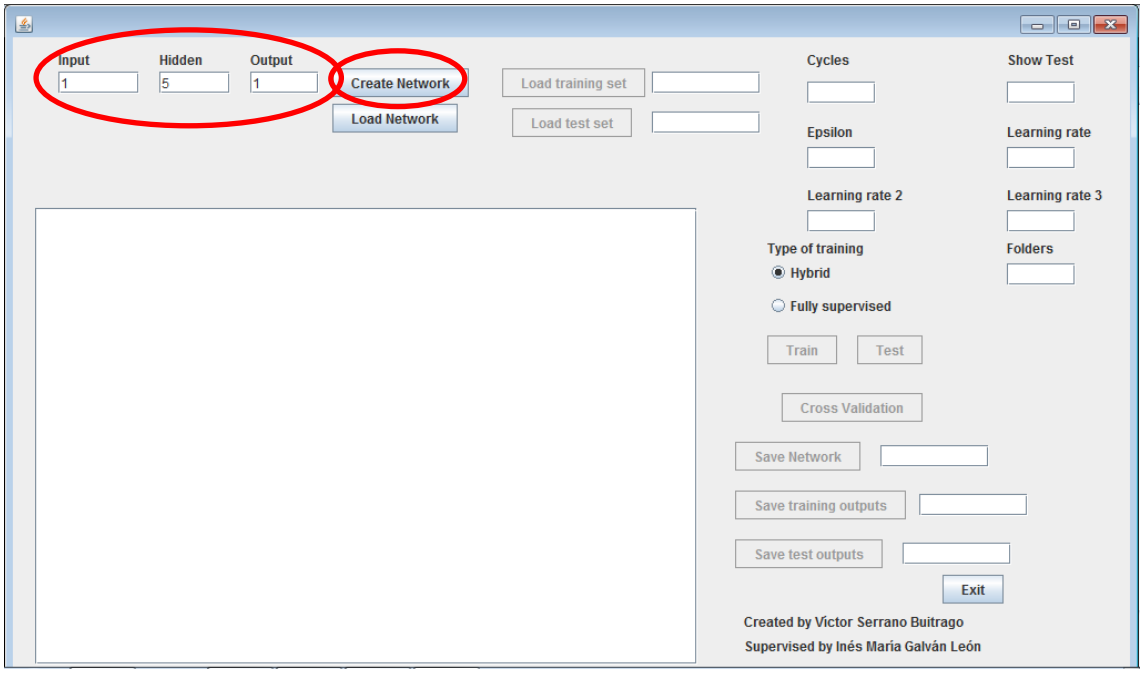

Figura 5.5: Creación de una nueva red

Una vez que se han definido estos valores, se debe pulsar sobre el botón “Create Network”. Con esto ya se tendrá la red definida. Y el botón en cuestión será remplazado por el botón “Modify Network”.

5.4.2. Modificar la red

En cualquier momento, si se desea modificar la arquitectura de la red, o simplemente probar con otra distinta, se podrá realizar dicho cambio, introduciendo la nueva red, es decir, el número de entradas, ocultas y salidas. Una vez determinada la nueva arquitectura, se deberá pulsar sobre el botón “Modify Network”, tal y como se aprecia en la siguiente imagen:

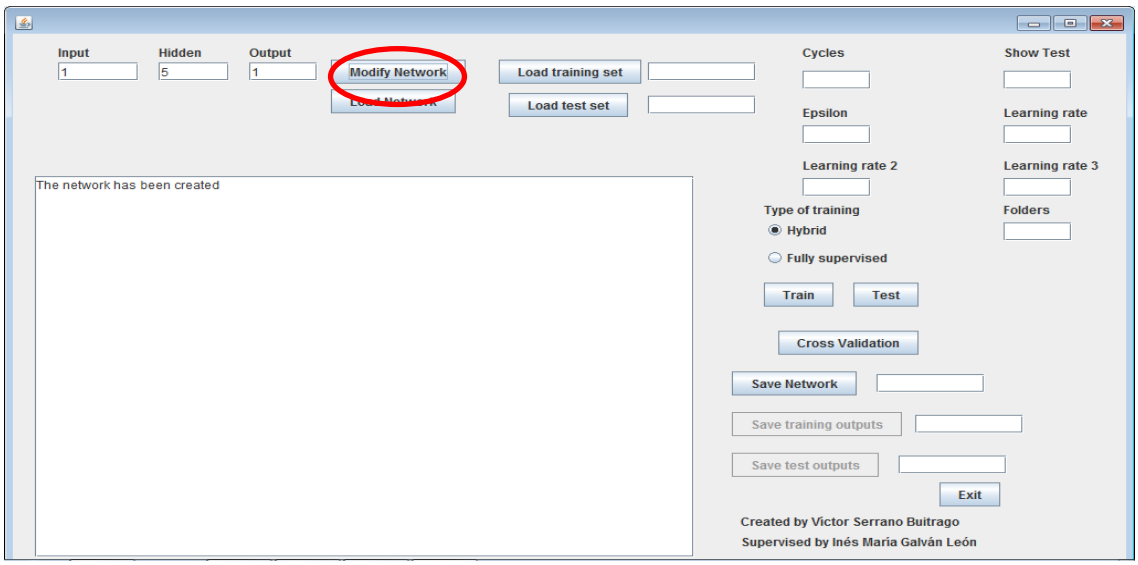

Figura 5.6: Modificación de una red

5.4.3. Cargar una red

En el caso de haber utilizado el simulador y haber definido una red, que bien por sus características obtenidas durante el entrenamiento, o simplemente por su buena inicialización de los parámetros correspondientes, es interesante, se podrá cargar, para seguir trabajando con ella, o sencillamente para observar sus resultados. Para ello, se deberá seleccionar simplemente el botón “Load Network”, sin necesidad de recordad su arquitectura, dado que una vez cargada, automáticamente se establecerá su arquitectura y se reflejará en la zona de los parámetros correspondientes. El fichero deberá estar en formato “**txt**” y debe haber sido creado previamente mediante la funcionalidad que permite almacenar una red.

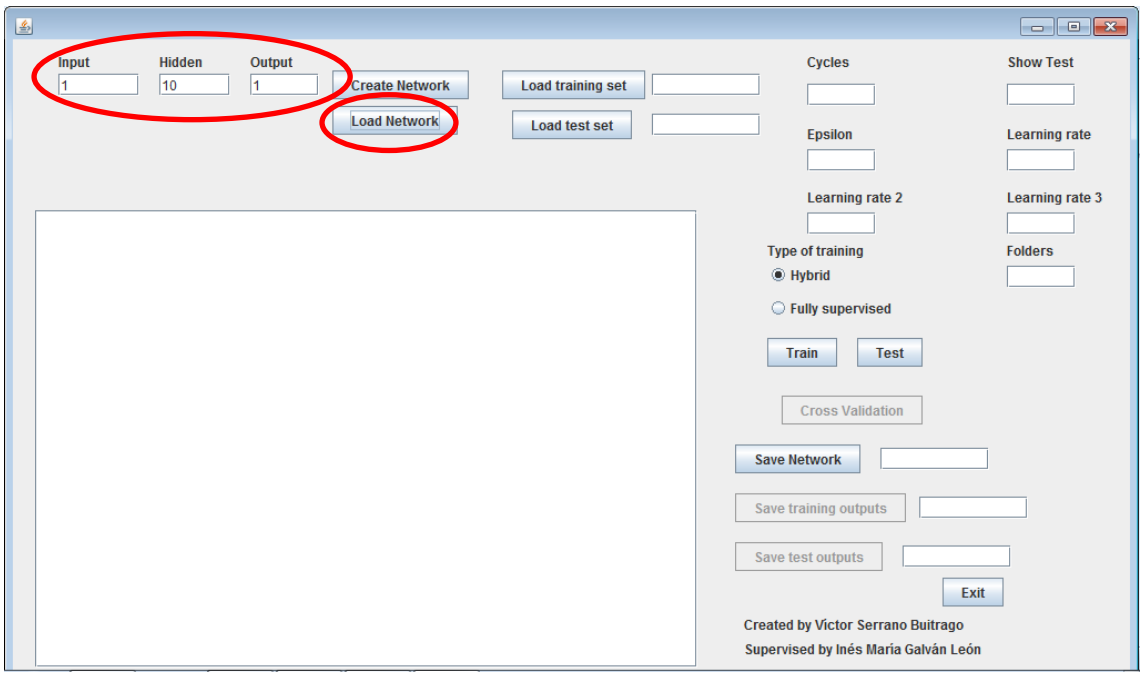

Figura 5.7: Carga de una red

5.4.4. Cargar patrón entrenamiento y test

Finalizada la definición de la arquitectura de la red de base radial, el siguiente paso es realizar la carga del conjunto de patrones que se desea utilizar. Para ello basta pulsar el botón "Load training set" para cargar el conjunto de patrones de entrenamiento y "Load test set" para el conjunto de patrones de test. En las cajas de textos correspondientes aparecerán los nombres de los ficheros cargados. En el caso de querer realizar validación cruzada, tan solo es necesario realizar la carga del patrón de entrenamiento, pulsando el botón "Load training set" puesto que será el fichero tomado como referencia para realizar la validación cruzada.

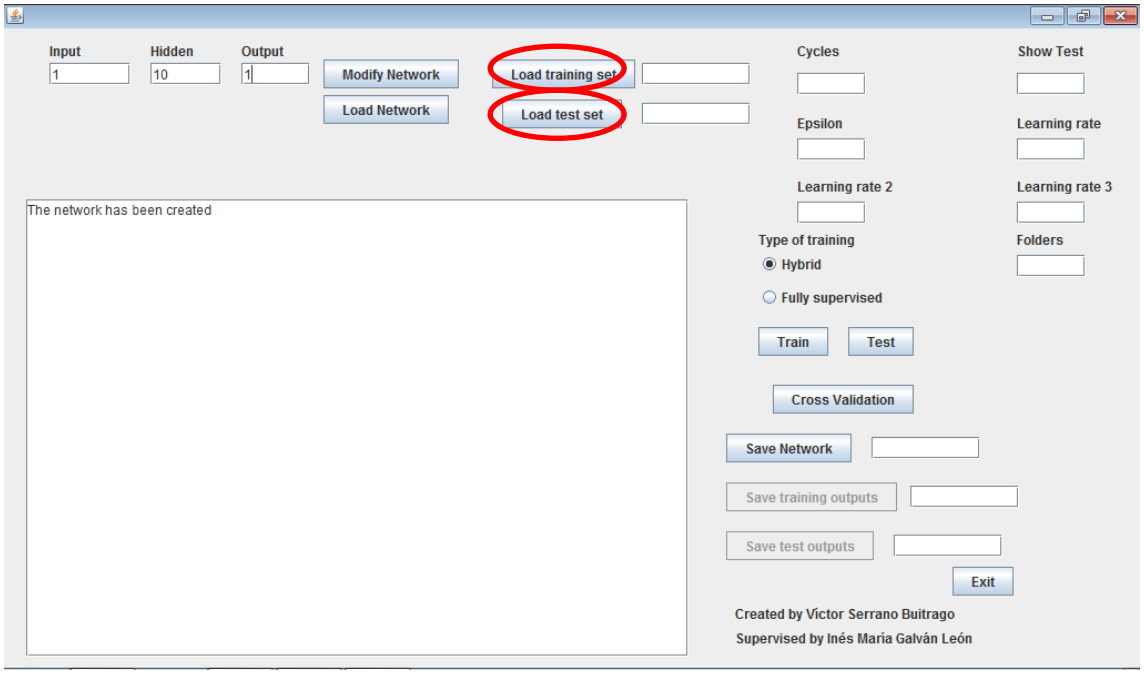

Figura 5.8: Carga de patrones de entrenamiento y test

Para que el simulador cargue sin problemas los patrones del archivo seleccionado, este deberá estar en formato **txt** y los patrones tendrán que seguir una estructura determinada, que se explica a continuación.

Los números decimales deberán estar precedidos de un punto "." Los atributos de cada patrón deberán estar separado por un único espacio, de lo contrario no se detectará tal separación, suponiendo un error en la lectura del patrón. Además, cada patrón deberá estar en una línea distinta y el primero de ellos, deberá estar situado en la primera línea del fichero. Por otra parte, es muy importante, que el número total de atributos de entrada y salida de los patrones coincida con la suma del número de entradas y de salidas de la red de base radial, porque de lo contrario, el simulador detectará que se trata de un patrón no válido para la red definida. Por último, no es necesario ningún tipo de carácter especial ni al principio de la línea, ni al final de la misma, así como tampoco es necesario que todos los números tengan decimales.

A continuación se muestra un ejemplo de un fichero con cuatro entradas y tres salidas:

0.200000 0.200000 0.200000 0.200000 0.000000 1.000000 0.000000

0.400000 0.600000 0.400000 0.600000 0.000000 1.000000 0.000000

0.800000 1.000000 0.800000 1.000000 0.000000 1.000000 0.000000

0.800000 0.600000 1.000000 0.400000 1.000000 0.000000 0.000000

0.800000 0.600000 0.200000 0.600000 1.000000 0.000000 0.000000

0.200000 0.200000 1.000000 0.800000 0.000000 0.000000 1.000000

0.600000 0.600000 1.000000 1.000000 0.000000 0.000000 1.000000

0.800000 0.600000 0.600000 0.800000 0.000000 1.000000 0.000000

5.4.5. Entrenamiento

Una de las principales funcionalidades de este simulador es el entrenamiento que a su vez tiene dos modalidades, el entrenamiento híbrido y el totalmente supervisado. Para poder realizar un entrenamiento se deben completar con valores numéricos positivos los siguientes parámetros: el número de ciclos que realizará el entrenamiento, cada cuántos ciclos se mostrará test y la razón de aprendizaje. En el caso de querer realizar un entrenamiento híbrido además se deberá completar el valor de épsilon. Por otro lado, si lo que se desea es un entrenamiento totalmente supervisado, se deben completar las razones de aprendizaje 2 y 3.

Posteriormente, se deberá seleccionar el tipo de entrenamiento que se realizará y pulsar sobre el botón "Train". Lo cual dará comienzo al entrenamiento, siempre y cuando estén cargados previamente los patrones y esté definida la red a entrenar.

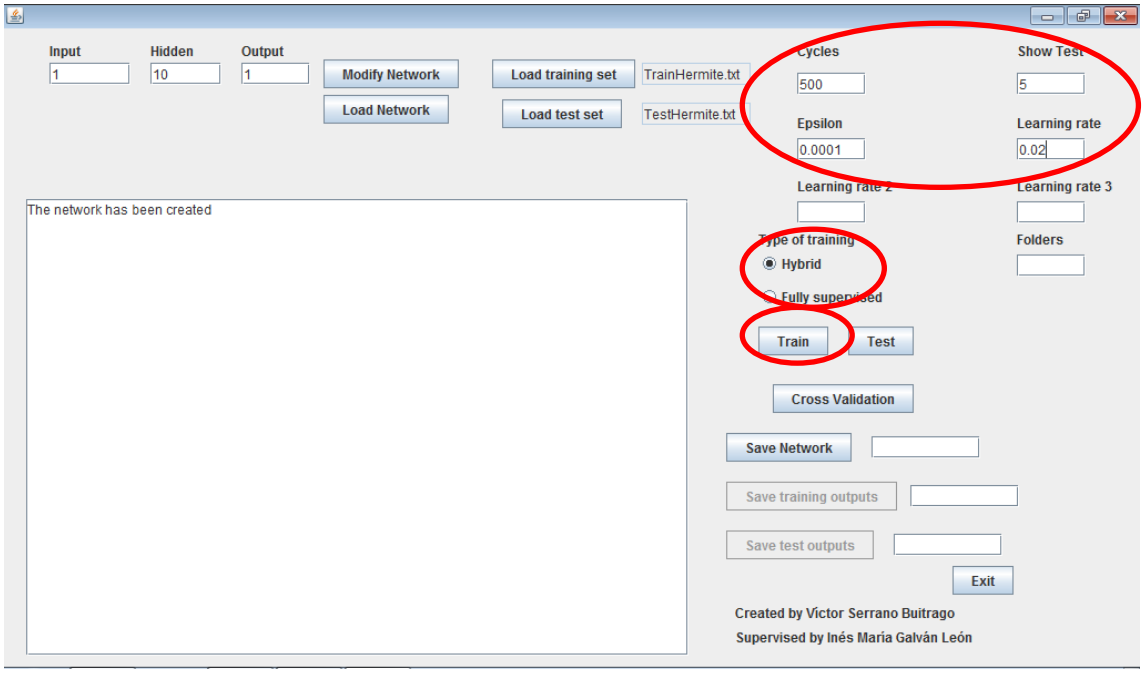

Figura 5.9: Parámetros del entrenamiento híbrido

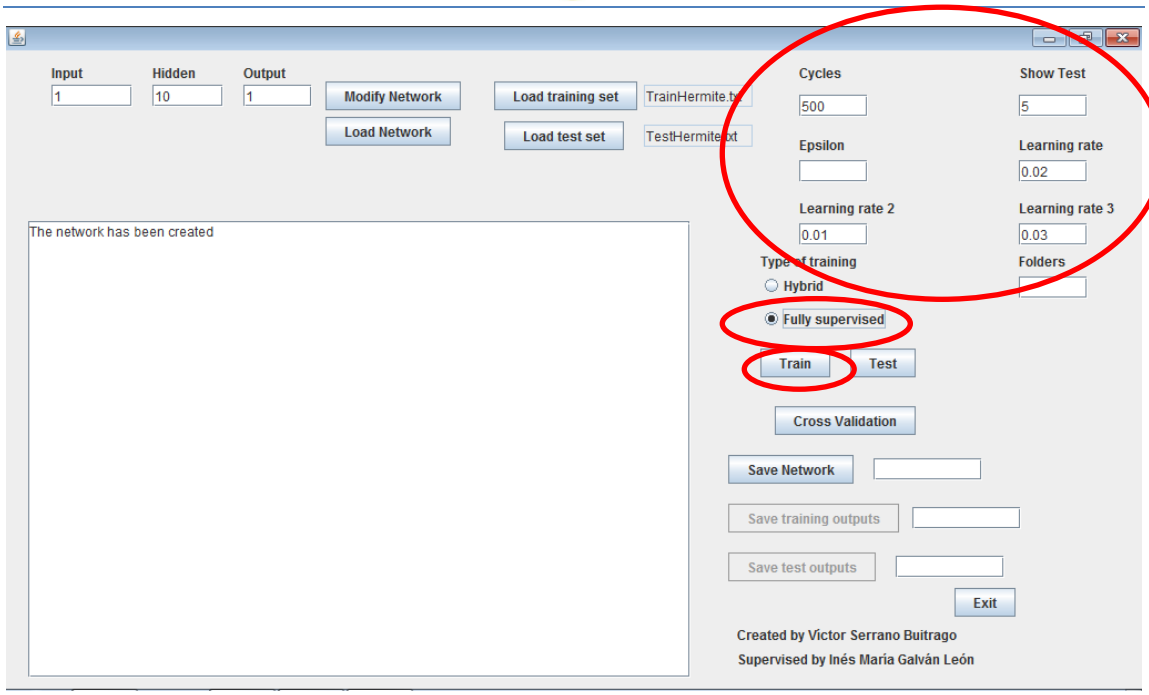

Figura 5.10: Parámetros del entrenamiento totalmente supervisado

Una vez finalizado el entrenamiento híbrido, se mostrarán por pantalla los resultados obtenidos.

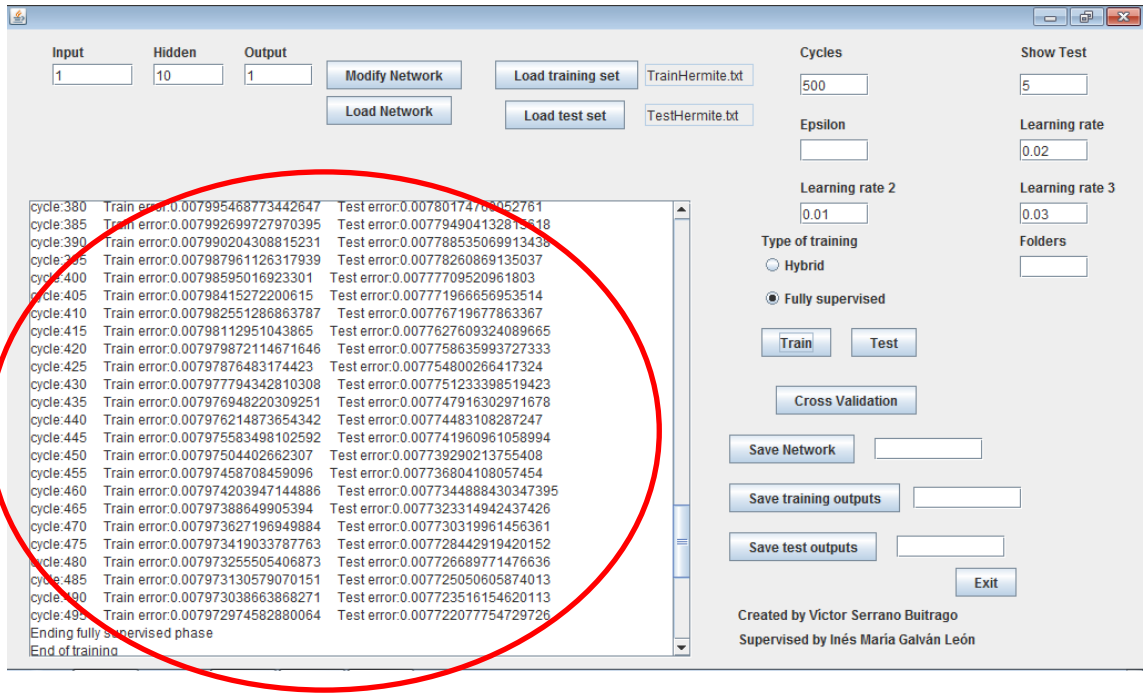

Figura 5.11: Visualización de los resultados

5.4.6. Realizar test

Si se desea analizar cómo se comporta una red para un determinado conjunto de test, basta cargar o definir la red, cargar el conjunto de patrones de test, pulsar sobre el botón “Test”. Aparecerá en pantalla el error cuadrático medio para dicho conjunto.

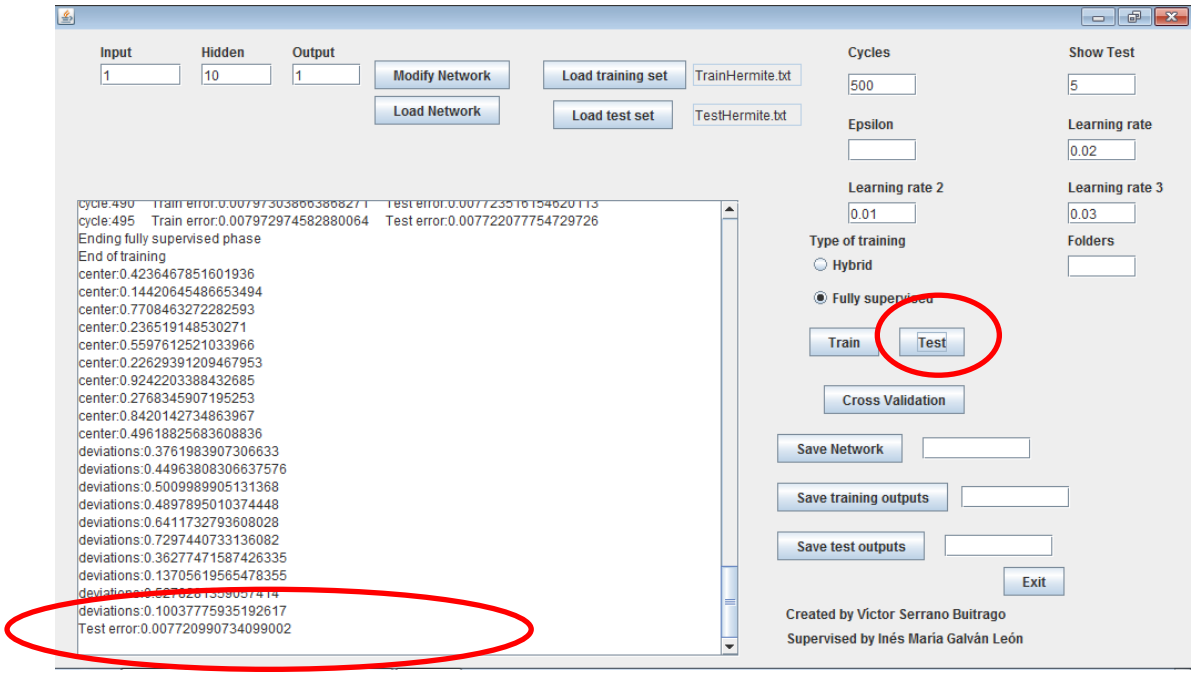

Figura 5.12: Realización de test

Cabe destacar que se podrá realizar la validación en cualquier momento siempre y cuando esté definida la red y se disponga de los patrones que se van a emplear para dicha validación.

5.4.7. Guardar red

Si se desea conservar la red con la que se está trabajando, para futuras utilizaciones, se deberá pulsar el botón “Save Network”. A continuación se abrirá una ventana en la que se deberá situar en el directorio en el que se desea almacenar el fichero que contiene la información de la red, y escribir el nombre de dicho fichero junto con su extensión. Se deberá tener en cuenta que a la hora de realizar la carga de la red, la extensión deberá ser “**txt**”.

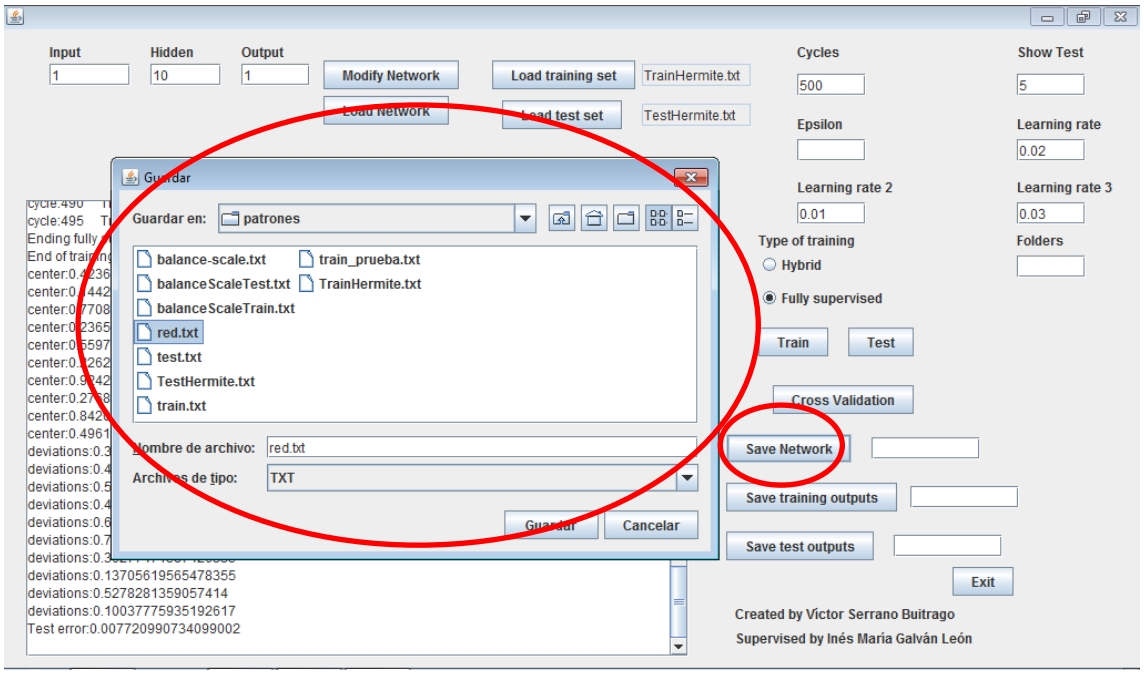

Figura 5.13: Almacenamiento de una red

El fichero en cuestión tendrá un formato que se podrá modificar si se desea cargar una red definida en un entorno distinto al del simulador. El formato utilizado es: número de entradas, número de ocultas, número salidas, centros, pesos, desviaciones y umbrales, en ese orden. Los centros, pesos, desviaciones y umbrales irán ordenados primero por neurona oculta. De tal modo que cada coordenada de los parámetros señalados, irá en una fila diferente, respetando el orden de neurona oculta a la que pertenezca dicha coordenada. Es decir, si disponemos de dos centros y dos entradas, las dos primeras filas corresponderán a las dos coordenadas del centro que corresponde a la primera neurona oculta, y las dos siguientes, a las coordenadas de la segunda neurona de la capa oculta. Del mismo modo se procede con los pesos, desviaciones y umbrales.

A continuación se muestra un ejemplo para una red formada por cuatro entradas, tres salidas y cinco neuronas de la capa oculta:

entradas:4

ocultas:5

salidas:3

centros:0.40614140043178837

centros:0.8670665372120901

centros:0.6988374800865992

centros:0.20874431433531726

centros:0.017377968744466732

centros:0.7259622255178536

centros:0.17067272926817523

centros:0.8393102646197147

centros:0.04903493806694725

centros:0.2787225360150345

centros:0.6805949512230327

centros:0.7448531228266585

centros:0.22569857451881703

centros:0.8922096543529339

centros:0.7992964185326875

centros:0.0033856518579087336

centros:0.9154501687057428

centros:0.19709486702615397

centros:0.8577269369790264

centros:0.22294996677013557

pesos:0.5672833448168505

pesos:0.4924592013688106

pesos:0.3953265120511775

pesos:0.7685757761537351

pesos:0.7424858738212384

pesos:0.06599108562630207

pesos:0.6741613265442539

pesos:0.6225024101820573

pesos:0.7934834952524826

pesos:1.2388223747616944E-4

pesos:0.22310536316484786

pesos:0.31568291695363526

pesos:0.5102443907745703

pesos:0.5276205713859569

pesos:0.6584359913273348

desviaciones:0.5023787347774912

desviaciones:0.6021950399793045

desviaciones:0.024526628757233326

desviaciones:0.09068837626549198

desviaciones:0.9146913406503708

umbrales:0.7126990491043544

umbrales:0.2840762198433596

umbrales:0.2723402588643068

5.4.8. Guardar salidas de entrenamiento y test

Una vez que se haya realizado un entrenamiento de la red, se podrán almacenar las salidas que proporciona la red en un fichero externo, tanto para el conjunto de datos de entrenamiento como el conjunto de datos de test. Para ello se deberá pulsar sobre los botones “Save training outputs” o “Save test outputs”, respectivamente.

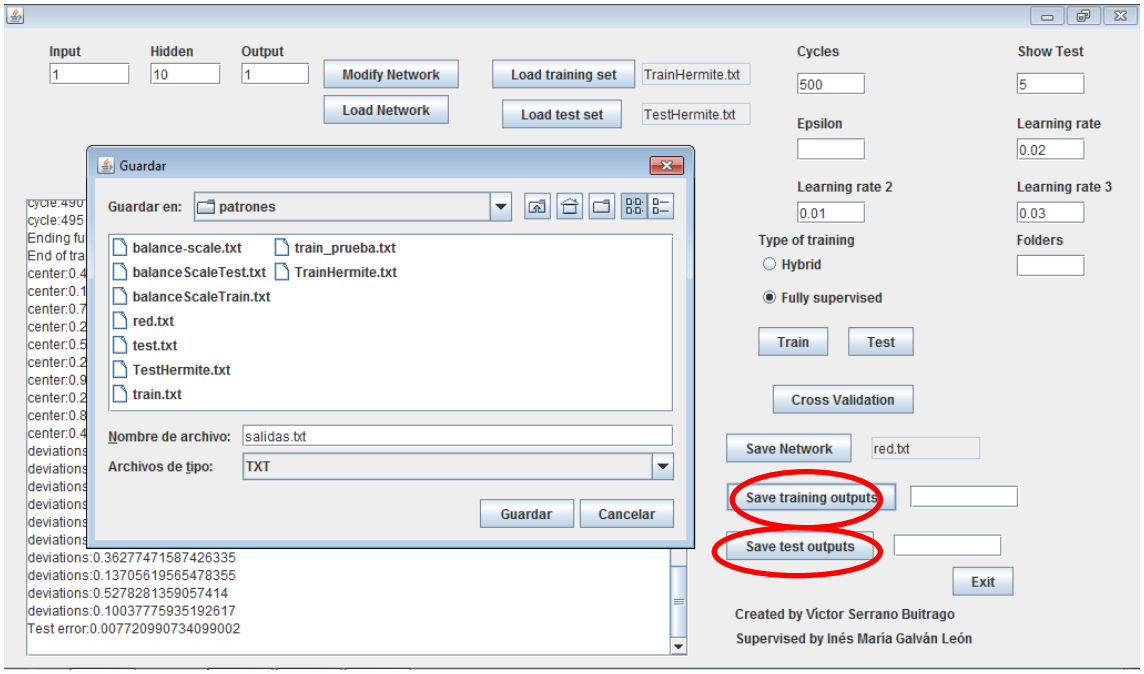

Figura 5.14: Almacenamiento de las salidas de entrenamiento y test

Al igual que al guardar la red, se abrirá una ventana en la cual se deberá seleccionar un fichero, o bien especificar el nombre y la extensión del nuevo archivo.

Tanto en el almacenamiento de las salidas de entrenamiento, como en las salidas de test, el fichero resultante tiene un formato establecido. En dicho fichero, se especificará en la primera línea qué columna (o columnas) corresponden a las salidas deseadas, que serán los que estén en primer lugar, y cuales corresponden a las salidas de la red. En las siguientes líneas, se representarán los datos de las salidas deseadas y las de la red, que irán situadas justo en la columna del tipo de salida que sea. De modo que para cada patrón se tendrá una fila con las salidas correspondientes.

5.4.9. Realizar validación cruzada

Otra funcionalidad del simulador es la posibilidad de realizad validación cruzada sobre un conjunto de patrones. Para ello, una vez definida la red, será necesaria la carga de un conjunto de datos que posteriormente se utilizará una parte de él como patrones de entrenamiento y otra parte como test. Al igual que en un entrenamiento, se deberá seleccionar el tipo de entrenamiento a realizar y definir los parámetros correspondientes al tipo de entrenamiento seleccionado. Una diferencia particular de la validación cruzada es el parámetro que determina el total de conjuntos en los que se debe separar el conjunto de datos disponible, también conocido como número de grupos o "folders". Por ello, se deberá especificar dicho parámetro en su casilla correspondiente. Una vez que se haya determinado dicho parámetro con un valor numérico positivo, se deberá pulsar en el botón "Cross Validation".

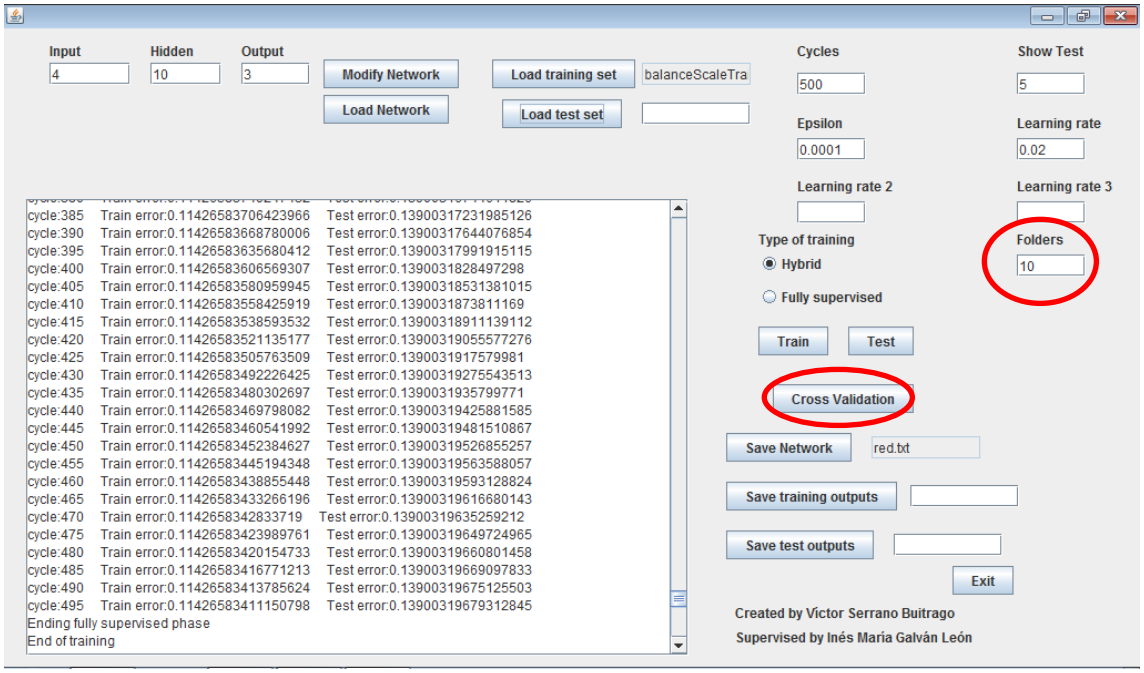

Figura 5.15: Realización de la validación cruzada

5.4.10. Salir del simulador

Si se ha terminado de utilizar el simulador, podemos cerrarlo en cualquier instante, bien mediante el botón "Exit" específico para dicha función, o bien mediante el aspa correspondiente a la ventana, que variará en función del sistema operativo en el que se utilice.

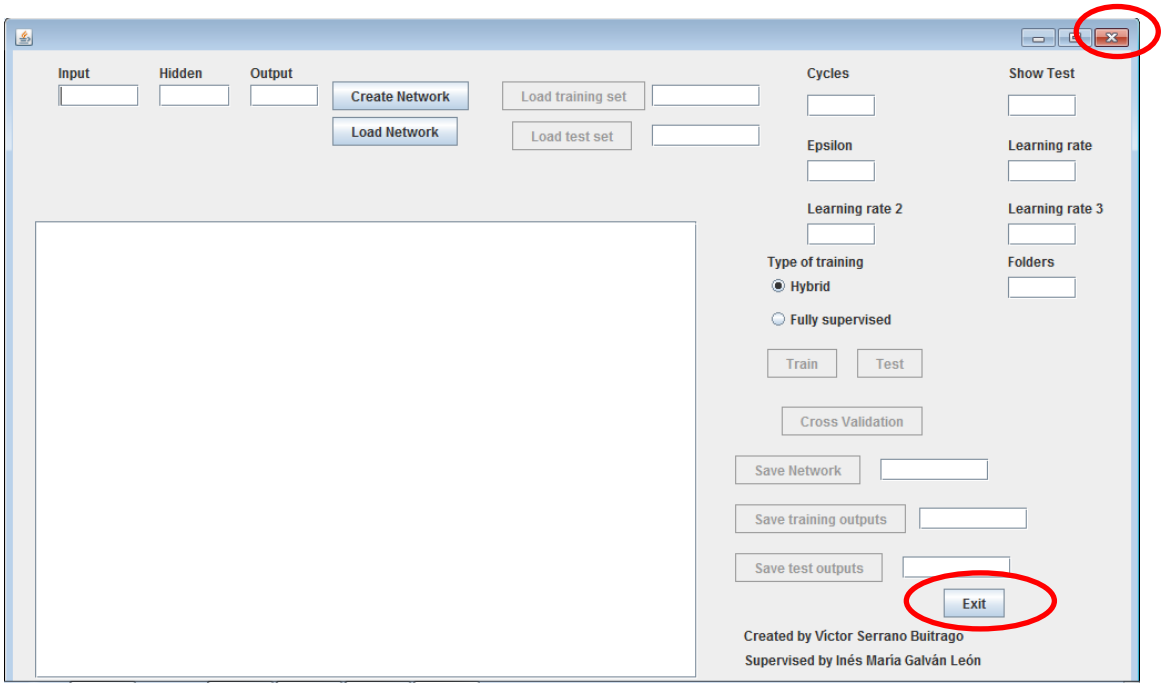

Figura 5.16: Salida del programa

Capítulo 6: Planificación y presupuesto

En este capítulo, se procederá a detallar cual ha sido la planificación que se ha llevado a cabo durante la realización de este Trabajo Fin de Grado, con el objetivo de poder visualizar la evolución del mismo y poder analizar dicha evolución. Además se detallará el presupuesto que supondría llevar a cabo el simulador, incluyendo cada uno de los elementos que participan en el proceso de desarrollo del mismo, que suponen un determinado coste.

6.1. Planificación

Para el desarrollo del simulador, al igual que en cualquier proyecto, es necesaria la elaboración de una planificación que estime los plazos de realización y poder así determinar el esfuerzo de la elaboración del proyecto. Por ello, en primer lugar se realizó una planificación que se detalla en los siguientes apartados.

6.1.1. Definición de las tareas

Cuando se llevó a cabo la planificación, lo primero a tener en cuenta fue la obtención de las diferentes tareas que se debían realizar para poder estimar el esfuerzo de cada una de ellas, obteniendo una estimación del esfuerzo que supondría dicha elaboración del simulador. Una vez analizado el objetivo a realizar en este proyecto, y determinadas las necesidades que requiere el simulador, se han estimado las siguientes tareas:

1. Estudio del sistema. Se estudian los objetivos del sistema a realizar.
2. Análisis. Se lleva a cabo un análisis del sistema para reconocer sus necesidades y la complejidad del mismo.
3. Elaboración del diseño. Se procede a realizar un diseño que cubra las necesidades identificadas durante el análisis.
4. Codificación. Se codifica el diseño realizado.
5. Realización de pruebas. Se prueba el código obtenido, comprobando que cumple todos requisitos estipulados.
6. Elaboración de la documentación. Se realiza la documentación que refleje todo el proceso de desarrollo del sistema.

Cabe destacar, que a raíz del estudio del sistema, se ha determinado que la codificación estaría formada por:

- Interfaz. Se codifica la interfaz que se mostrará al usuario cuando ejecute el simulador, además de implementar el tratamiento de todas las interacciones que realice dicho usuario.
- Arquitectura. Se codifica el soporte para dar estructura a la red que el usuario defina.
- Entrenamiento. Se elaboran todos los métodos necesarios para llevar a cabo el proceso de aprendizaje de una red.

- Test. Se codifican las operaciones necesarias para realizar test sobre una red.
- Gestión de ficheros. Se codifican los métodos para llevar a cabo la escritura y lectura de los ficheros externos al sistema.

6.1.2. Planificación inicial

Una vez definidas las tareas que se deben realizar para el completo desarrollo del simulador, se ha elaborado una planificación, basada en la experiencia del autor del Trabajo Fin de Grado, y en la disponibilidad para la realización del mismo, teniendo siempre presente, las fechas establecidas para la entrega del trabajo.

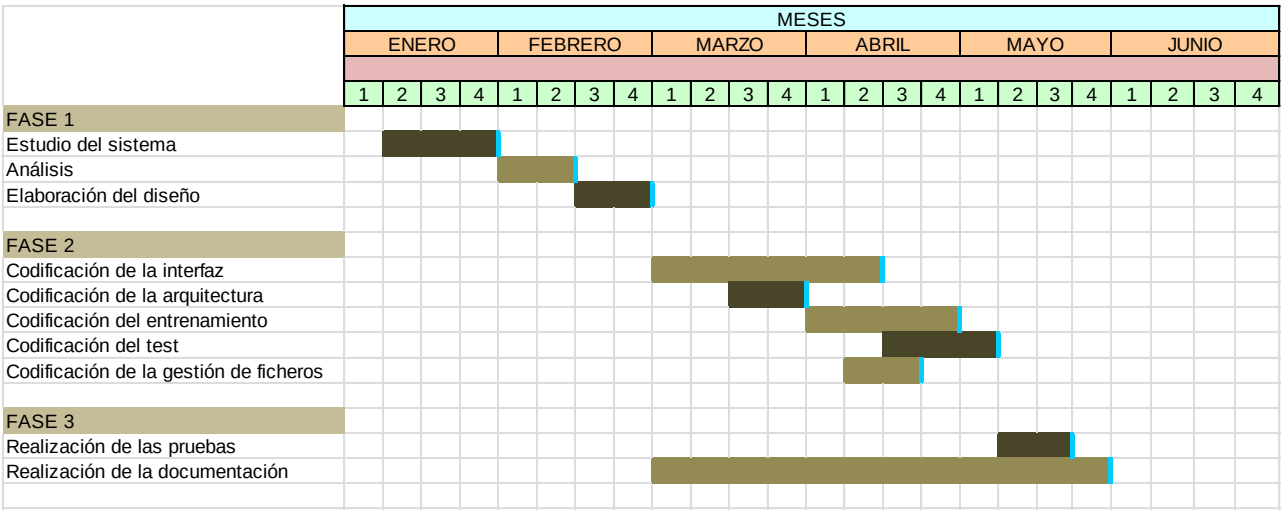

Figura 6.1: Diagrama de Gantt de la planificación inicial

Como se puede observar en el diagrama, se ha establecido una planificación que presenta un claro margen de error de cara al plazo límite para la entrega del simulador, para que en caso de que surjan imprevistos, se pueda disponer de un plazo de subsanación. Además, se ha optado por solapar algunas tareas debido principalmente a su dependencia, dado que para realizar alguna tarea, muchas veces es necesario desarrollar conjuntamente la otra. Finalmente, aunque se represente un largo período para la realización de la documentación, no se empleará un número de horas tan elevado debido a su compaginación con otras tareas. Esto se debe, principalmente, al objetivo de incrementar la sencillez de su elaboración, y poder reflejar lo más fielmente posible en la documentación, la realidad del desarrollo del simulador.

6.1.3. Planificación final

Una vez finalizado el simulador en su totalidad, debido a diversas circunstancias, como la aparición de dificultades imprevistas en la elaboración de ciertas tareas de codificación, y a la eficacia mostrada en determinadas tareas a lo largo de la elaboración del simulador, la planificación que se diseñó en un principio, ha sufrido modificaciones quedando de la siguiente manera:

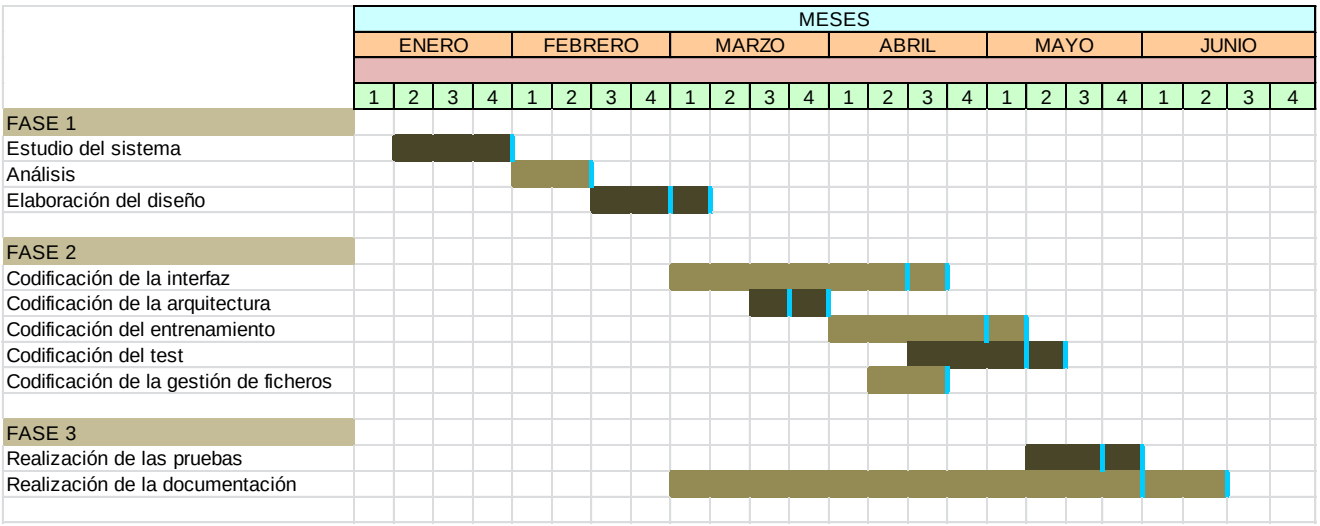

Figura 6.2: Diagrama de Gantt de la planificación final

A la vista de los resultados mostrados en el diagrama, podemos afirmar, que en la mayoría de las tareas se ha requerido de un mayor tiempo de dedicación, con respecto a lo estimado inicialmente. Todo ello ha repercutido en un atraso de dos semanas en la finalización del desarrollo de toda la documentación y por tanto del proyecto. No obstante, en la planificación inicial se contemplaba dicho margen de error, por lo que no ha impedido que se lleve a cabo la entrega dentro del plazo límite establecido.

6.2. Cálculo de costes del sistema

En este apartado se procederá a determinar los diferentes costes que han influido en la elaboración del simulador de red de base radial.

6.2.1. Determinación de los costes

Para realizar una estimación global del coste del desarrollo del simulador, es necesario determinar varios costes particulares que afectarán a la suma total del esfuerzo necesario.

En primer lugar, se determinará el coste del esfuerzo del desarrollador del simulador, teniendo en cuenta que solamente se han trabajado **14** horas por semana:

Persona encargada del desarrollo	Víctor Serrano Buitrago
Horas dedicadas	21 semanas * 14 horas = 294 horas
Coste por hora	15 €
Coste total	**4410€**

Tabla 6.1: Atribución de coste por horas

A continuación se determinarán esfuerzos acumulados, es decir, el número de horas invertidos en cada actividad.

Tarea a realizar	Esfuerzo dedicado
Estudio del sistema	42 horas durante 3 semanas
Análisis	28 horas durante 2 semanas
Elaboración del diseño	32 horas durante 3 semanas
Interfaz	40 horas durante 7 semanas
Arquitectura	8 horas durante 2 semanas
Entrenamiento	29 horas durante 5 semanas
Test	21 horas durante 4 semanas
Gestión de ficheros	13 horas durante 2 semanas
Realización de las pruebas	20 horas durante 3 semanas
Elaboración de la documentación	61 horas durante 14 semanas

Tabla 6.2: Esfuerzo dedicado por tareas

Una vez estimado los esfuerzos de las tareas, se indicarán los costes de las amortizaciones y otros gastos.

Para las amortizaciones tenemos la siguiente tabla:

Amortizaciones	Coste
Ordenador durante el período de 21 semanas	50€
Lugar de trabajo durante 21 semanas	425€
Total	**475€**

Tabla 6.3: Coste de amortizaciones

Los costes varios contemplados son los siguientes:

Coste	Gastos
Material de oficina variado (papel, bolígrafos)	20€
Trasporte durante 21 semanas	140€
Total	**160€**

Tabla 6.4: Otros gastos

6.2.2. Costes totales

A partir del esfuerzo realizado durante el desarrollo del simulador, podemos obtener la siguiente gráfica en la que se representa para cada tarea, el esfuerzo que se ha ido requiriendo a lo largo de la realización del proyecto. Además, refleja el esfuerzo total que se ha ido acumulando conforme se ha ido avanzando, así como los distintos

solapamientos, en muchos casos necesarios para obtener un esfuerzo dentro de lo planificado.

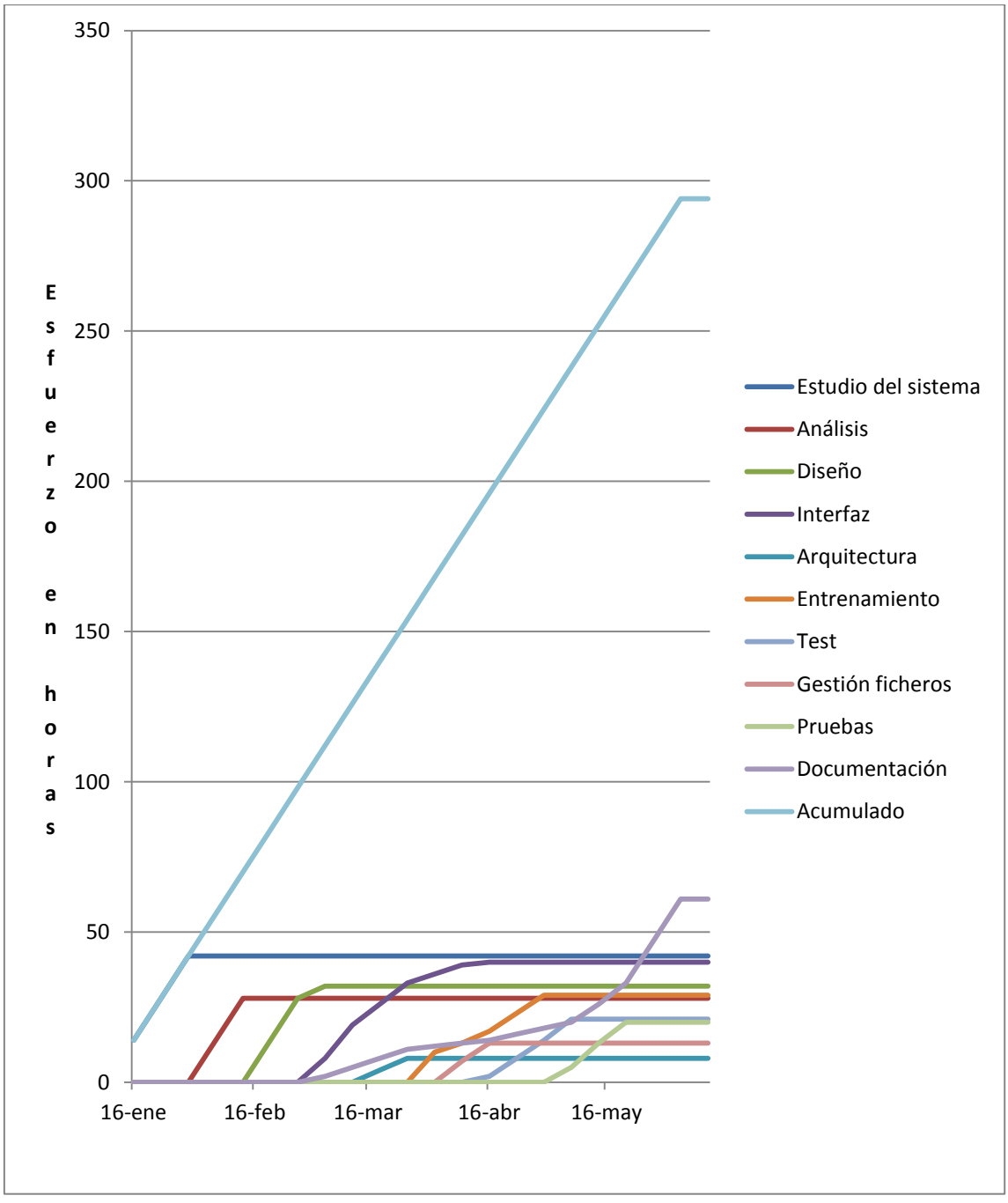

Figura 6.3: Gráfico de esfuerzos totales

Como podemos observar, la mayoría de las tareas ha supuesto un esfuerzo aproximado de unas 30 horas aproximadamente, salvo la documentación, que debido a la amplia duración y a la dependencia de otras tareas ha supuesto un esfuerzo mayor. Por otra parte, cabe destacar de una buena planificación dado que, los

resultados obtenidos, se deben a una buena división de tareas y asignación de esfuerzos a cada una, obteniendo un resultado final acorde con lo exigido.

Finalmente el coste total de la realización del presente Trabajo Fin de Grado se desglosa de la siguiente manera:

Coste de las actividades	4410€
Coste total de amortizaciones	475€
Costes indirectos	160€
15% de beneficio	706,5€
Coste total (sin I.V.A.)	5116,5€
Coste total (con I.V.A.)	**6037,47€**

Tabla 6.5: Presupuesto total

Por lo tanto podemos afirmar, que la realización del simulador de las redes de base radial ha supuesto un coste total de **seis mil treinta y siete con cuarenta y siete céntimos**.

Capítulo 7: Conclusiones y futuras mejoras

En este capítulo se procederá a presentar las distintas dificultades que se han ido encontrando durante la realización del proyecto. Después se extraerán las conclusiones derivadas de la realización del presente Trabajo Fin de Grado. Por último, se explicarán las diferentes líneas futuras a seguir para futuros trabajos.

7.1. Dificultades encontradas

Durante el desarrollo del simulador han surgido varios problemas que han ralentizado la evolución planificada. Entre dichos problemas destacan los siguientes:

- Falta de experiencia en la elaboración de interfaces. Dado que no se había trabajado nunca con interfaces, el aprendizaje para utilizar dichas interfaces ha sido mayor de lo esperado, retrasando en cierta medida el tiempo estimado para terminar la interfaz.
- Resultados incorrectos obtenidos durante el entrenamiento. Una vez finalizado toda la funcionalidad correspondiente al entrenamiento, fue necesario revisar todas las partes que participaban en dicho proceso debido a que los resultados que aportaban, no eran los esperados pues no se llegaba a realizar un aprendizaje correcto.
- Dificultad a la hora de reflejar los conceptos de aprendizaje y test sobre los que se basan las redes de neuronas de base radial en la codificación. Las diferentes ecuaciones que son necesarias, además del proceso de unión de los resultados obtenidos de las distintas ecuaciones, ha representado una cierta dificultad, dado que se requiere que todo el proceso sea codificado de manera perfecta, para poder realizar una simulación adecuada.

No obstante, a pesar de los inconvenientes reflejados, se ha llevado a cabo con éxito el desarrollo del simulador, dado que todas las dificultades se han resuelto en un breve período de tiempo.

7.2. Conclusiones

En este Trabajo Fin de Grado se ha desarrollado un simulador que permite simular las redes de base radial, de una manera sencilla e intuitiva. Dicho simulador permite la creación, carga y almacenamiento de redes de base radial. Además permite la carga de ficheros de entrenamiento y test para su posterior utilización en los procesos de entrenamiento y test que dispone. En cuanto al entrenamiento, se han incorporado dos tipos, el entrenamiento híbrido y el entrenamiento totalmente supervisado. También permite la realización de validación cruzada a partir de un conjunto de patrones. En cualquier caso, se permite el almacenamiento de las salidas tanto de entrenamiento como de test, para su posterior visualización.

El simulador se ha realizado en Java con el principal propósito de permitir una facilidad en la portabilidad, además de estar destinado principalmente al uso docente

y público. De modo que se ha realizado en inglés y de manera que resulte lo más sencilla y eficiente posible para el usuario.

Cabe destacar que se han conseguido todos los objetivos marcados para este trabajo, aprendiendo durante el desarrollo del mismo y mejorando varias habilidades de las que ya se disponía. De forma que se puede concluir el trabajo con una valoración del mismo más que satisfactoria en todos los sentidos.

7.3. Futuras mejoras

Como la mayoría de las herramientas, este simulador también está abierto a posibles mejoras, entre las cuales cabría destacar las siguientes:

- Representación de las salidas de entrenamiento y test de manera gráfica. A pesar de que se ha definido el formato de dichas salidas, de tal forma que sean fácilmente exportables para trabajar con ellos, siempre es conveniente facilitar dicha información de una manera más gráfica.
- Introducción de nuevos algoritmos de inicialización de los centros en el entrenamiento hibrido. Aunque la utilización del K-medias está demostrado que funciona de manera excelente, existen otros algoritmos que pueden realizar la misma función, en algunos casos suponiendo una relativa mejora. Por ello se podría incluir dichos algoritmos con la opción de elegir el más apropiado.
- Inclusión de nuevas funciones de base radial. En este simulador se utiliza la función gaussiana, por ser la más usada en el contexto de redes de base radial. Sin embargo, la incorporación de nuevas funciones de base radial en el simulador, de manera que el usuario pueda elegir la función deseada, podría ser una ampliación interesante del simulador.
- Inclusión de herramientas que permitan la automatización del formato de los datos de entrada. De este modo, no sería necesaria la normalización, ni la dotación del formato a los datos que se necesitan para simular las redes de neuronas.
- Integración a nuevas herramientas. Como hemos señalado en diversos puntos a lo largo de esta memoria, las redes de base radial no siempre son la mejor solución, por ello se podría incorporar el presente simulador en otro más general que ofrezca la posibilidad de simular con diversas arquitecturas.

Bibliografía

- [Eckel, 2007] Eckel, B. (2007)

  Piensa en Java.

  Prentice Hall.

- [Isasi and Galván, 2004] P. Isasi, I. M. Galván. (2004).

  Redes de Neuronas Artificiales. Un Enfoque Práctico.

  Pearson Educación S.A.

- [Kohonen, 1982] Kohonen, T. (1982).

  Self-organized formation of topologically correct feature maps.

  Biological Cybernetics. 43:59-69.

- [MacQueen, 1967] MacQueen, J. (1967).

  Some methods for classification and analysis of multivariate observations.

  In Proceedings of the fifth Berkeley Symposium on Mathematics, Statistics and Probability. Berkeley: U. California Press.

- [Myatt, 2008], Myatt, A. (2008).

  Pro Netbeans IDE 6: rich client platform edition.

  APress.

- [Moody and Darken, 1989] Moody, J.E. and Darkenn C.J. (1989).

  Fast learning in networks of locally-turned processing units.

  Neural Computation 1: 281-294.

- [Park and Sandberg, 1991] Park, J. and Sandberg, I. (1991).

  Universal approximation using radial-basis-function networks.

  Neural Computation, 3:246-257.

- [Poggio and Girosi, 1990] Poggio, T. and Girosi, F. (1990).

  Networks for approximation and learning.

  In Proceeding of the IEEE, volume 78, 1481-1497.

- [Renals, 1989] Renals, S. (1989).

  Radial basis function network for speech pattern classification.

  Electronics Letters, 25:437-439.

- [Weiss, 2007], Weiss, M.A. (2007)

  Data structures and algorithm analysis in Java.

  Addison-Wesley.