UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

Implementación de algoritmos espaciales para la estimación del frente de carbonatación en morteros a partir de señales ultrasónicas

Antonio Colomino Granell

Tutor: Jorge Gosalbez Castillo

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2017-2018

Valencia, 11 de Septiembre de 2018

Resumen

Los ultrasonidos son una técnica no destructiva usada para la caracterización y detección en multitud de ámbitos: metalúrgico, construcción, aeronáutico… En el sector de la construcción es de especial interés conocer el estado interno de los diferentes elementos constructivos que forman una edificación y mediante los ultrasonidos podemos averiguarlo de manera eficiente. La carbonatación es un proceso de degradación por gradiente (desde el exterior hasta el interior de la muestra) que deteriora los materiales formados por cemento. Este proyecto se enmarca en un proyecto más grande cuyo objetivo final es la implementación de algoritmos de reconstrucción tomográfica para estimar y generar una imagen a partir de las medidas ultrasónicas. En concreto, este proyecto se ha centrado en el desarrollo de un algoritmo basado en una red neuronal de funciones de base radial (RBFNN, Radial Basis Function Neural Network), entrenada a partir de proyecciones. Se ha realizado un estudio de la misma y se han reformulado las ecuaciones de entrenamiento para adaptarlas a los datos recogidos por el sinograma (proyecciones). Una vez replanteado el sistema de entrenamiento, se ha realizado un estudio evaluando las diferentes variables que conforman la RBFNN: número de funciones, pesos, ruido, número de puntos de entrenamiento… Todo esto se ha realizado bajo entorno MATLAB el cual se ha empleado tanto para generar los datos de proyecciones mediante simulación como la reconstrucción y estudio de los diferentes parámetros.

Resum

Els ultrasons són una tècnica no destructiva usada per a la caracterització i detecció en multitud d'àmbits: metal•lúrgic, construcció, aeronáutic... En el sector de la construcció és d'especial interés conéixer l'estat intern dels diversos elements constructius que formen una edificació i per mitjà dels ultrasons podem esbrinar-ho de manera eficient. La carbonatació és un procés de degradació per gradient (des de l'exterior fins a l'interior de la mostra) que deteriora els materials formats per ciment. Este projecte s'emmarca en un projecte més gran l'objectiu final del qual és la implementació d'algoritmes de reconstrucció tomogràfica per a estimar i generar una imatge a partir de les mesures ultrasónicas. En concret, este projecte s'ha centrat en el desenvolupament d'un algoritme basat en una xarxa neuronal de funcions de base radial (RBFNN, Radial Basis Function Neural Network), entrenada a partir de projeccions. S'ha realitzat un estudi de la mateixa i s'han reformulat les equacions d'entrenament per a adaptar-les a les dades arreplegats pel sinograma (projeccions). Una vegada replantejat el sistema d'entrenament, s'ha realitzat un estudi avaluant les diferents variables que conformen la RBFNN: nombre de funcions, pesos, soroll, nombre de punts d'entrenament... Tot açò s'ha realitzat baix entorn MATLAB el qual s'ha empleat tant per a generar les dades de projeccions per mitjà de simulació com la reconstrucció i estudi dels diferents paràmetres.

Abstract

Ultrasound is a non-destructive technique used for characterization and detection in a multitude of areas: metallurgical, construction, aeronautical ... In the construction sector it is of special interest to know the internal state of the different building elements that make up a building and by means of the ultrasound we can find out efficiently. Carbonation is a degradation process by gradient (from the outside to the inside of the sample) that deteriorates the materials formed by cement. This project is part of a larger project whose final objective is the implementation of tomographic reconstruction algorithms to estimate and generate an image from the ultrasonic measurements. Specifically, this project has focused on the development of an algorithm based on a neural network of radial-based functions (RBFNN, Radial Basis Function Neural Network), trained from projections. A study of the same has been made and the training equations have been reformulated to adapt them to the data collected by the sinogram (projections). Once the training system was redesigned, a study was carried out evaluating the

different variables that make up the RBFNN: number of functions, weights, noise, number of training points ... All this has been done under the MATLAB environment, which has been used as much to generate the data of projections through simulation such as the reconstruction and study of the different parameters.

INDICE

Contenido

Capítulo 1. Introducción 1

1.1 Motivación 1

1.2 Objetivos 1

1.3 Planteamiento del problema 2

Capítulo 2. TEORÍA DE FUNCIONES DE BASE RADIAL Y RECONSTRUCCIÓN TOMOGRÁFICA 3

2.1 Entrenamiento de las RBF 5

2.2 Adaptación del entrenamiento de las RBF para la reconstrucción tomográfica 9

Capítulo 3. DESARROLLO EXPERIMENTAL 11

3.1 Introducción 11

3.2 Parámetro 1: Número de neuronas 12

3.3 Parámetro 2 : Ángulo máximo de la proyección 15

3.4 Parámetro 3: Número máximo de proyecciones 17

3.5 Parámetro 4: Anchura k de la función 19

3.6 Simulación final con parámetros óptimos 21

Capítulo 4. Conclusiones y líneas futuras. 23

Bibliografía 24

Capítulo 1. Introducción

1.1 Motivación

Es una tarea fundamental en muchas industrias llevar a cabo una inspección profunda de las determinadas estructuras que forman una edificación y detectar posibles defectos dentro de éstas.

Por tanto el planteamiento de este trabajo surge de la necesidad de obtener cierta información de los elementos constructivos cuyo interior no es accesible físicamente sin una acción destructiva y por tanto que perjudique a ésta. Cuando hablamos de información relevante, nos referimos a la existencia de defectos distribuidos en dichos elementos, tanto en forma como lugar. Por ello, planteamos el desarrollo de un método de reconstrucción basado en inspección ultrasónica que permite analizar los materiales de manera sencilla y no destructiva.

Cuando nos encontramos ante una estructura, físicamente solo podemos acceder a las paredes exteriores de la misma y a simple vista no podemos ver la parte interna y por tanto no podemos saber si existe algún defecto internamente. Una técnica de evaluación con gran número de aplicaciones en diversos campos es la inspección ultrasónica, que se caracteriza por ser un ensayo no destructivo que no altera de ninguna manera la estructura bajo estudio.

Las limitaciones propias de la inspección ultrasónica son, fundamentalmente, que mediante ésta solo podemos obtener la información de los diferentes caminos de propagación que recorre la onda mecánica a través de la estructura. Para poder reconstruir interiormente la totalidad del área, y así observar si hay algún deterioro, es necesario aplicar un algoritmo de reconstrucción inversa basado en la interpolación de datos para que el número de puntos representados resulte razonable.

Con el fin de obtener tomografías de mayor precisión y resolución, hemos optado por la implementación de un algoritmo de predicción no lineal que consiste en una red neuronal construida a partir de funciones de base radial. El entrenamiento de dicha red permite la optimización de los parámetros de dichas funciones, lo que produce una mejor reconstrucción de los defectos internos que pueda presentar la estructura.

1.2 Objetivos

Este trabajo de investigación tiene por objetivo dar respuesta a la necesidad de detección y localización de los defectos en el interior de diversas estructuras. Para ello, se propone el desarrollo de un método de procesado de señal que se comporta como herramienta de reconstrucción rápida y precisa de los defectos internos de la estructura bajo estudio.

Mediante este trabajo de investigación se buscará cumplir los siguientes objetivos:

- Desarrollar un método de reconstrucción aplicable a todo tipo de materiales que forman las estructuras que podamos inspeccionar mediante ultrasonidos. Los algoritmos desarrollados deben de ser robustos frente a errores y precisos en la detección y localización de los defectos en la estructura bajo estudio.
- Minimizar, en la medida de lo posible, los tiempos de inspección y procesado, así como rebajar el coste computacional requerido en el proceso de reconstrucción tomográfica.
- Optimizar los diferentes parámetros de las funciones de base radial que conforman la red neuronal para cada uno de los análisis para cumplir los objetivos nombrados anteriormente.

1.3 Planteamiento del problema

La labor fundamental de este trabajo de investigación, como hemos comentado anteriormente, es el desarrollo de un método de reconstrucción tomográfica que sea rápido y a su vez eficaz.

Para ello vamos a simular la obtención de proyecciones ultrasónicas de la estructura mediante la herramienta de trabajo MATLAB. Con esta herramienta podemos simular dicha inspección con ultrasonidos que es lo que obtendríamos de la estructura bajo estudio, es decir, las entradas del sistema, generando una imagen reconstruida donde podríamos observar con cierta precisión el interior de la imagen, es decir, la salida del sistema. Esta precisión dependerá de los parámetros de entrada, de los parámetros de configuración de la RBFNN seleccionados así como ubicación y forma que presenten los defectos internos de la estructura.

Con la herramienta de trabajo MATLAB hemos generado una función que reconstruye la imagen final (la de la estructura con sus defectos) mediante las proyecciones de entrada que nosotros le demos. Estas proyecciones entrenarían a la función de base radial y la cual nos ofrecería la imagen final.

Cómo MATLAB, dentro de su biblioteca de funciones, tiene la función *Radon*, la cual genera las proyecciones a partir de una imagen, podemos comparar la imagen original con la imagen final obtenida por nuestra función (mediante las funciones de base radial) y así obtendríamos la eficacia de nuestra función. Cabe indicar, que la función *Radon*, calcula las proyecciones a partir de una integral de línea a lo largo de líneas rectas y paralelas. Esto difiere de la propagación real ultrasónica, la cual está atada a procesos de difracción, pero sirve como primera aproximación.

A nuestra función podemos modificarle varios parámetros, por ello, crearíamos una función donde los parámetros variasen entre dos valores y posteriormente al comparar con la imagen original , podríamos ver en qué valores es más eficaz nuestra función y al final sabiendo para todos los parámetros , que valores son los más ideales, podríamos ejecutar una la función introduciendo dichos valores y así encontrar la manera más eficaz de programar nuestra función.

Además, para realizar la simulación lo más real posible, al final introduciremos un ruido en nuestra función, para simular las imperfecciones que pueda encontrar la onda ultrasónica en su camino dentro de la estructura.

Capítulo 2. TEORÍA DE FUNCIONES DE BASE RADIAL Y RECONSTRUCCIÓN TOMOGRÁFICA

1.1 Teoría de las RBF

Las Redes Neuronales Artificiales (RNA) son una herramienta de procesamiento matemático que está basada en el funcionamiento de las redes neuronales del sistema nervioso de los animales que nos rodean. Las características de las RNA son las propias de las redes biológicas y durante el tiempo se han ido empleado en el tratamiento de la información.

Si pasamos a observar detenidamente la arquitectura tipo de una RNA observamos que está formada por un número de neuronas de entrada X, que mediante la "función de propagación o excitación", se conectan entre ellas. En una segunda etapa, obtenemos un valor previo que después de aplicarle "la función de activación", actualizamos la neurona de salida Y. La interconexión de las neuronas que forman la red se hace mediante una suma ponderada. Cuando queremos obtener la red que mejor se adaptaría a nuestro problema se conseguiría mediante la optimización de los parámetros que forman nuestra red, destacando como el más importante los pesos w_i de la función de propagación.

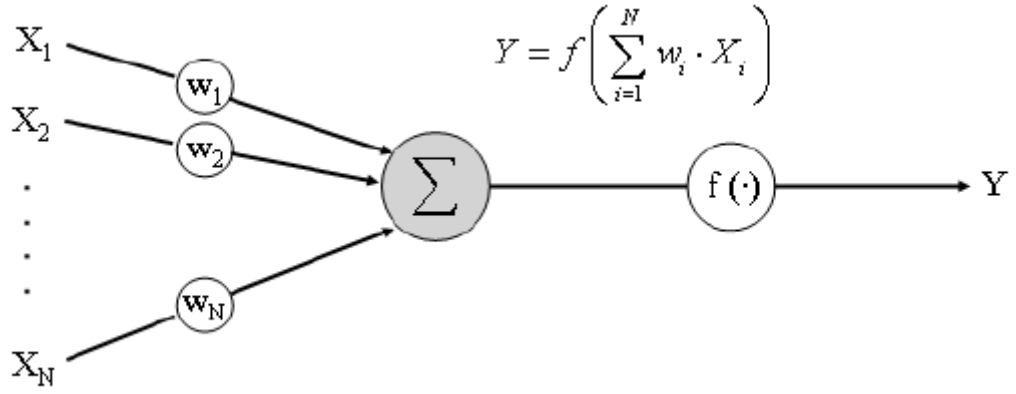


Figura 1. Arquitectura típica de una red neuronal artificial

Hay diferentes tipos de RNA, pero un caso particular es la llamada red de funciones de base radial (RBF), empleada habitualmente para aproximar funciones. Esta red, funciona de manera parecida a lo citado anteriormente, pero con la distinción de que la función de activación de

estas redes son las propiamente dichas funciones de base radial. Podemos encontrar las RBF definidas de muchas maneras, pero nosotros las vamos a definir como:

$$\boldsymbol{\Phi}(\boldsymbol{x}) \equiv \Phi(\boldsymbol{x}, \boldsymbol{c}, \boldsymbol{r}) \equiv \boldsymbol{\Phi}\left(\frac{||\boldsymbol{x} - \boldsymbol{c}||}{\boldsymbol{r}}\right) \quad (1)$$

Siendo:

- $x \rightarrow$ un punto del espacio de entrada.
- $c \rightarrow$ centro de la RBF.
- $r \rightarrow$ radio de la RBF.

Si representamos la estructura de una RNA adaptada a red RBF, tendríamos algo tal que así:

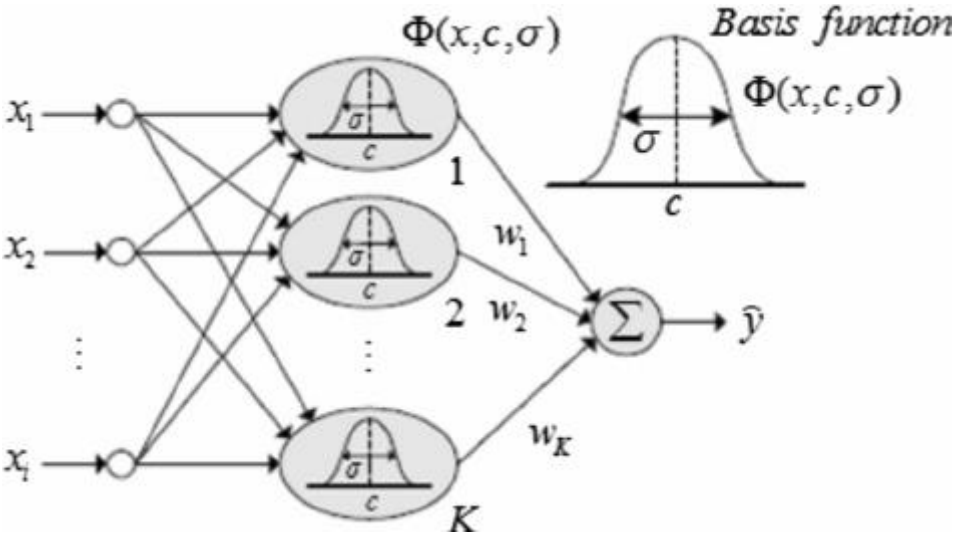


Figura 2. Estructura de una RBF

Donde podemos observar las diferentes capas, donde en la primera capa, llamada capa de entrada nos encontramos los parámetros de entrada X. En la segunda capa, llamada capa oculta, nos encontramos la RBF que llamamos $\Phi(x)$,, y en la última capa, llamada capa de salida, tenemos la combinación lineal de las entradas X_i con la función $\Phi(x)$, para así tener nuestra salida Y. Si observamos detenidamente la función 2, podemos observar en que capa ajustamos nuestra RBF y tras ello ajustar los pesos para poder obtener la función de salida Y deseada.

En la primera capa, llamada ***capa de Entrada***, encontramos los valores de entrada de las RBF X_i, que son los valores que obtenemos del exterior y no les realizamos ningún procesado.

En la siguiente capa, llamada ***capa Oculta***, nos encontramos la función de base radial $\Phi(x, c, r)$, donde aplicaremos la RBF a cada elemento de entrada X_i mediante la función 1. Podemos encontrar como RBF más populares y con su uso más extendido: multicuadrática, spline poli armónico, gaussiana, spline de placa delgada… y en nuestro caso, vamos a utilizar la RBF Gaussiana:

$$\Phi(r) = e^{-(\varepsilon r)^2} \quad (2)$$

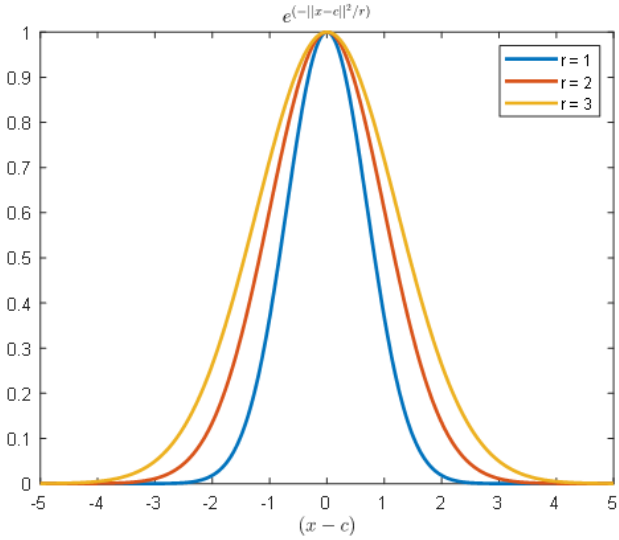


Figura 3 Respuesta función Gaussiana

En la última capa, que llamamos ***capa de Salida***, después de obtener todos los valores que vienen de la capa Oculta de la RBF, se hace una combinación lineal de estos valores mediante un sumatorio, donde aplicaremos los pesos asociados W_i y que al ponderar estos con la salida de la capa Oculta de la RBF finalmente obtendremos nuestra la salida de la función Y

Podemos destacar como característica principal de una RBF su carácter Local, puesto que alcanzan un nivel máximo cuando el valor de entrada X_i esta cercano al centro de la neurona y a medida que el valor de entrada está más lejano del centro, el valor de la función va acercándose a su valor mínimo.

Por tanto, podemos definir las salidas de las redes neuronales de base radial como una combinación lineal de funciones gaussianas que se activan para una determinada posición del espacio bajo estudio definido por los valores de entrada.

2.1 Entrenamiento de las RBF

Aunque las redes de funciones de base radial comentadas anteriormente se emplean habitualmente para la aproximación de funciones, en este proyecto se han empleado para desarrollar un algoritmo de reconstrucción tomográfica. Para ello es necesario que la red nos devuelva un valor o función estimada *(f)* para unos parámetros de entrada (X). Podemos escribir la función de forma analítica y de forma matricial de la siguiente manera:

$$f(X) = \sum_{j=1}^{Nr} W_j \cdot e^{-\frac{\left(X - X_{cj}\right)^2}{k}} \quad \Rightarrow \quad f(X) = \Phi(X)W \quad (3)$$

Donde:

- N_r = Número de funciones de base radial
- W_j = Pesos de ponderación de las funciones de base radial
- X_{cj} = Centros de las diferentes funciones de base radial
- k = Anchura de la función de base radial

En nuestro caso, el parámetro estimado o valor de salida de la RBF ($f(X)$) podría aproximar velocidad, atenuación, impedancia ultrasónica...que asociaremos a la luminosidad del pixel a la hora de representarlo como una imagen y el parámetro de entrada X corresponderá a la posición dentro de elemento analizado.

Para conseguir esto necesitaremos entrenar la red. Entendemos como entrenamiento de la red al proceso de ajustar los diferentes parámetros de la función de base radial hasta encontrar el valor de ese parámetro que mejor se ajusta a nuestro objetivo. Estos parámetros, ya comentados anteriormente son: centro de las funciones (X_c), número de neuronas (N_r), pesos (W_i) y anchura de la RBF (k). Podemos ajustarlos todos, o podemos tratar de ajustar únicamente uno de ellos. Dependerá del caso concreto.

Podemos pensar por ejemplo, en las proximidades de las zonas donde podríamos tener un defecto en nuestra estructura, debemos intensificar el número de funciones, o afinar el ancho de la RBF para obtener un valor más preciso de dicho efecto.

Si observamos la **figura 2** observamos que las entradas $X1, X2, X3$…son sometidas a una transformación a través de la función que al entrenarla, obtendremos los parámetros más precisos que mejor se adapten a dicho puntos. El proceso se muestra a continuación para un caso unidimensional.

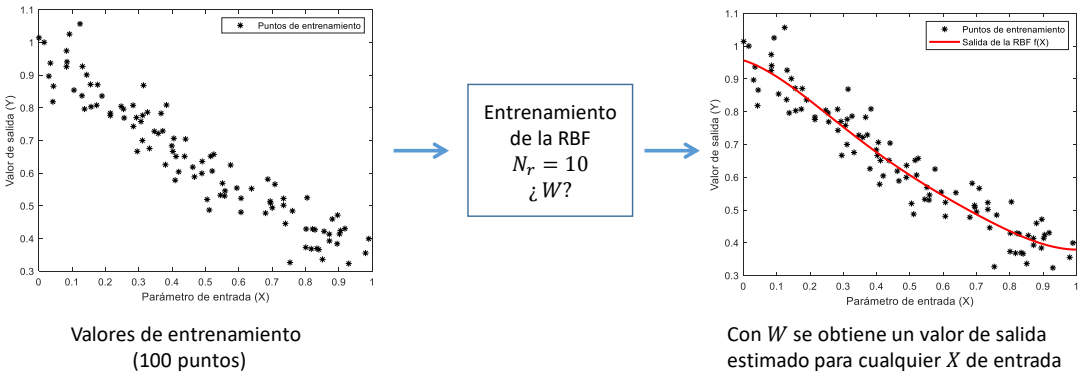


Figura 4. Ejemplo gráfico de entrenamiento para obtención de los pesos

Podemos ver que el entrenamiento de la función requiere un conjunto de valores de entrada (X) y valores de salida (Y) como puntos de entrenamiento o aprendizaje a partir de los cuales se buscará el valor de los parámetros de la RBF que mejor se adapten.

A continuación se realizará la explicación para entrenar los pesos W de la RBF dejando fijos el número de funciones y sus centros. El proceso se va a describir mediante notación matricial, por ello, se define en la siguiente tabla las diferentes variables así como una descripción y su dimensión con el fin de facilitar la lectura del presente documento:

Variable	Descripción	Tamaño
N_r	Número de funciones RBF	Escalar (1×1)
N	Número de puntos proporcionados para el entrenamiento	Escalar (1×1)
N_D	Dimensión del vector de entrada. Si estamos en el caso unidimensional (figura anterior), $N_D = 1$, pero si estamos aproximando una imagen, $N_D = 2$.	Escalar (1×1)
W	Vector de pesos	$N_r \times 1$
X	Valores de entrada de entrenamiento	$N \times N_D$

X_c	Matriz de centros de las RBF	$Nr \times N_D$
$\Phi(X)$	Matriz de activación de la RBF obtenida a partir del vector de entrada de entrenamiento	$N \times N_r$

Cada punto de entrada activa las N_r funciones de basa radial $\Phi(\cdot)$, en nuestro caso nombrado anteriormente y reescrito según 4, donde hemos especificado entre llaves las dimensiones de la matriz)

$$\Phi(X) = e^{-\frac{|X-X_c|^2}{k}} \quad \{(N \times N_r)\} \tag{4}$$

Donde si tomamos (1) y (4), llegamos a la conclusión que de forma matricial la salida estimada $\hat{Y}$ por la RBF la podemos escribir como (5). Dicho valor estimado se puede asociar a una velocidad de propagación ultrasónica v o algún otro parámetro que nos interese:

$$v(X) = \hat{Y} = \Phi(X)W \tag{5}$$
$$\{(N \times 1) = (N \times N_r) \times (N_r \times 1)\}$$

De esta forma y a partir del vector de entrenamiento Y y de la matriz Φ podemos escribir la estimación del vector de pesos óptima como:

$$W = (\Phi^T \Phi)^{-1} \Phi^T Y \tag{7}$$

$$\{(N_r \times 1) = ((N_r \times N) \times (N \times N_r))^{-1} \times (N_r \times N) \times (N \times 1)\}$$
$$\left\{(N_r \times 1) = \left((N_r \times N_r)\right)^{-1} \times (N_r \times 1)\right\}$$
$$\{(N_r \times 1) = (N_r \times N_r) \times (N_r \times 1)\}$$
$$\{(N_r \times 1) = (N_r \times 1)\}$$

Con esto, podemos obtener la estimación de los pesos para aproximar una función cuyo parámetro de entrada presenta una dimensión N_D. Cabe indicar que se ha dejado fijo el número de funciones, N_r, el ancho de las mismas, k, y el centro de los mismos se han distribuido siguiendo una función de densidad de probabilidad uniforme (que no una separación uniforme o constante).

Siguiendo la expresión (7), vamos a mostrar un ejemplo para entrenar una RBF para estimar una imagen. En este caso, emplearemos la imagen proporcionada por MATLAB bajo la instrucción Phantom. En la siguiente **figura 5**, se muestra dicha imagen (a), cuyo tamaño es de 100 x100 puntos (igual a 10000 puntos). Esta imagen se ha empleado como puntos de entrenamiento, X $\{10000 \times 2\}$ e Y $\{10000 \times 1\}$. Indicar que la imagen sigue una estructura 2D, pero dicha estructura se transforman en un vector columna para poder realizar el entrenamiento siguiendo las fórmulas anteriores.

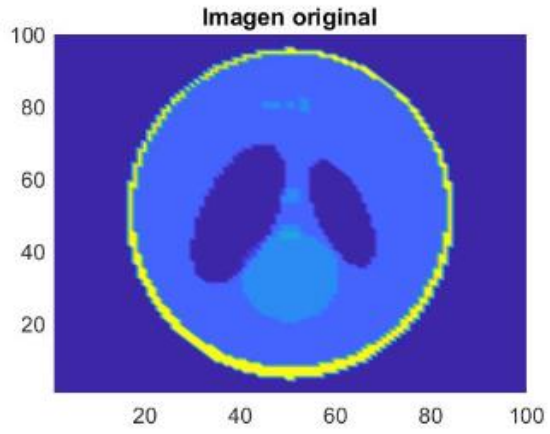


Figura 5 . Imagen creada con función Phantom en Matlab

El número de funciones RBF es igual a 500 ($N_r = 500$) y sus centros siguen una fdp uniforme y (en el apartado experimental se estudiará la influencia del número de funciones). Una vez entrenada la RBF, obtenemos los pesos W (**Figura 6**) y podemos realizar una estimación o una reconstrucción de la imagen usando la RBF entrenada (**figura 7**). En este punto podríamos usar un mallado más fino para reconstruir la imagen por ejemplo de 1000 por 1000 y se esperaría obtener una imagen con mejor resolución (aunque no es así).

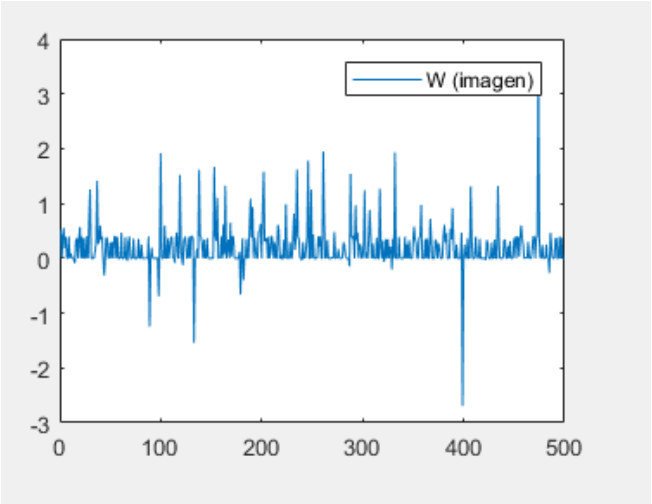


Figura 6. Pesos entrenados para imagen original

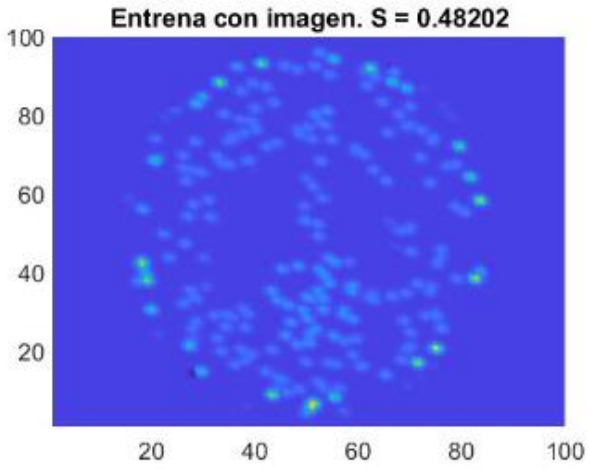


Figura 7. Entrenamiento con imagen original

Esto se junta con el aspecto un poco "incoherente" de entrenar la red con la propia imagen que queremos reconstruir. Ya que el objetivo es estimar el interior de la imagen, pero sin tener los valores de dicho interior, Esta "incoherencia" la solucionaremos en el siguiente apartado.

2.2 Adaptación del entrenamiento de las RBF para la reconstrucción tomográfica.

Como ya hemos nombrado en el apartado anterior, la función *Phantom* de MATLAB nos dibuja la imagen que queremos reconstruir, por tanto entrenar con ella resulta un poco incoherente. Por ello, en nuestro caso, lo que tenemos no son los puntos interiores de la muestra, si no las proyecciones del interior de la muestra a lo largo de la línea:

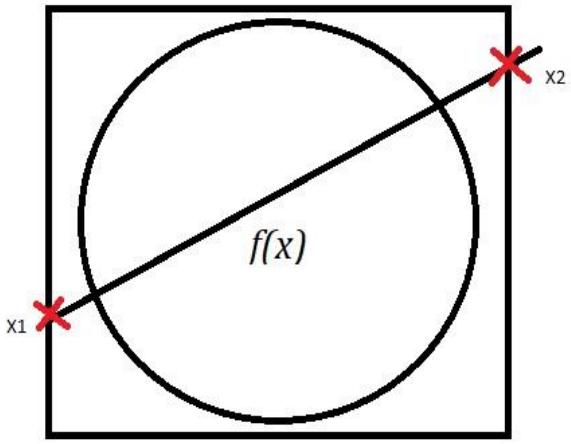


Figura 8. Proyección de la imagen

Como observamos en la **figura 8**, lo que tenemos es proyección de la imagen que va desde $X1$ hasta $X2$, por tanto resulta que:

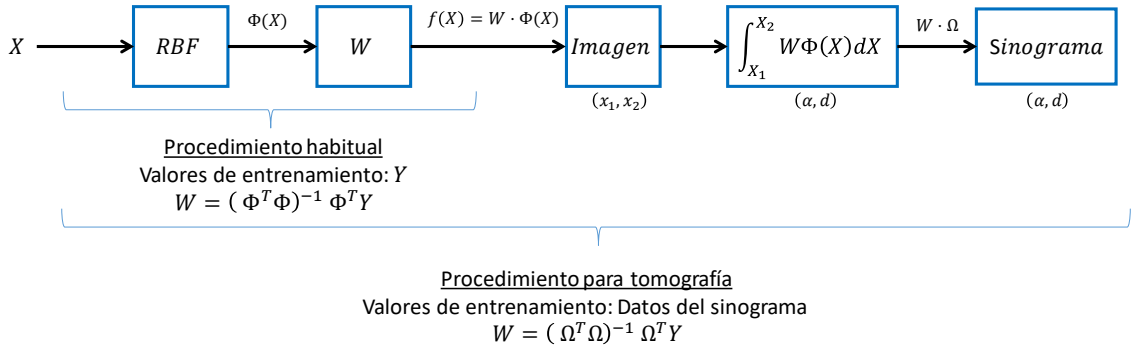
$$v = \frac{1}{L}\int_{x_1}^{X_2} v(x)\cdot dx = \frac{1}{L}\cdot W \int_{X_1}^{X_2} \Phi(x)\cdot dx \qquad (6)$$

Considerando $Y(X) = \int_{X_1}^{X_2} \Phi(x)dx$ y combinando (5) y (6), finalmente obtenemos:

$$W = (Y^T Y)^{-1}\,\Omega^T V L \qquad (7)$$

Y así obtenemos los pesos óptimos para nuestra función donde ahora los valores Y no corresponden al valor de la propia imagen sino al valor integrado entre dos puntos (valor de la proyección). Calculados ya los pesos óptimos en nuestra función, el siguiente paso es recalcular nuestra función 1, donde obtendríamos los valores óptimos de la muestra estimados de los parámetros (pesos) ajustados para las funciones de base radial.

Para finalizar el apartado, cabe destacar que hemos adaptado está teoría y hemos creado unos ficheros de MATLAB que se encargan de entrenar la función probando diferentes valores en la función. Hemos simulado diferentes valores para los centros, los pesos, el número de neuronas, y como mediante la función *Phantom* de MATLAB tenemos lo que sería la imagen original, comparando la imagen original con la imagen entrenada con proyecciones podemos determinar qué valores se adaptan mejor a nuestra función dando como resultado una imagen clara donde podamos observar los defectos de nuestra estructura.



En el dibujo de arriba podemos observar de manera visual el entrenamiento que realizaría MATLAB con las proyecciones de la imagen dándonos como resultado el valor (que representamos en el valor de luminosidad del pixel) del punto bajo estudio de la imagen.

Capítulo 3. DESARROLLO EXPERIMENTAL

3.1 Introducción

Como ya hemos comentado anteriormente, hemos adaptado la teoría de las RBF a la reconstrucción tomográfica y para ello hemos creado unos ficheros en el entorno MATLAB que nos van a ayudar ajustar los diferentes parámetros de entrada de las funciones para así ajustar que valores son los óptimos para obtener con mayor claridad la imagen con los defectos de la estructura. También, vamos a generar las proyecciones con una imagen controlada y con la función de *Radon* de MATLAB que aunque no modela una propagación ultrasónica propiamente dicha, es una buena aproximación para evaluar la capacidad de reconstrucción de este tipo de redes neuronales. Así mismo, podremos calcular un índice de error (que llamaremos similitud con la imagen original) dado que al disponer de la imagen original, podemos comparar la imagen que obtenemos con las proyecciones con la original y observar cuánto se parecen. Para ello, gastaremos la función morfológica *ssim* de MATLAB.

Los parámetros que vamos a variar para así obtener el valor óptimo de estos son: número de neuronas, número máximo de proyecciones con las que incidimos en la estructura, el ángulo máximo de las proyecciones y la anchura de la RBF. Cabe destacar que hemos programado nuestros ficheros en MATLAB de manera que definimos entre que valores queremos definir nuestro parámetro y el programa ejecuta un barrido de las funciones con los diferentes valores del parámetro y vamos almacenando un resultado gráfico (imagen entrenada con proyecciones) y así poder ver de manera visual cuanto de óptimo es el valor que hemos definido para ese parámetro. Finalmente, hacemos una comparativa de los pesos óptimos entrenados con la imagen original y los pesos óptimos entrenados con las proyecciones, calculamos el índice de similitud (cuanto se parece la imagen entrenada con proyecciones con la imagen real entrenada) así como también los costes computacionales.

La imagen empleada para todo esto proceso es una imagen patrón empleada para evaluar algoritmos tomográficos, y la devuelve la función *phantom* de Matlab. Emplearemos una imagen de 100×100 y se muestra en la siguiente figura.

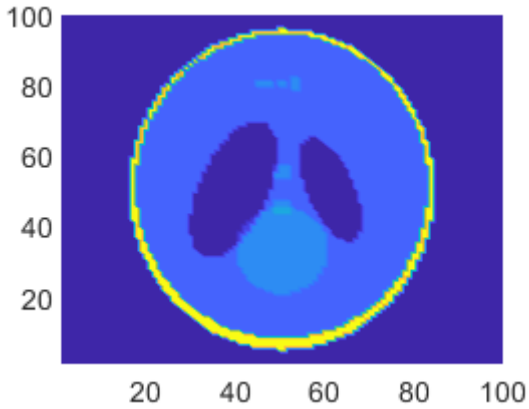


Figura 9 Imagen que devuelve la función *Phantom* de MATLAB

3.2 Parámetro 1: Número de neuronas

A continuación vamos a ejecutar nuestro fichero variando el número de neuronas de la RBF. Para ello definimos que el número de neuronas (número de funciones de la RBF) sea entre 10 y 1000 en saltos de 100, por lo que el programa ejecutará la simulación para 10 , 110, 210, 310, 410, 510, 610, 710, 810 y 910 neuronas.

Después de ejecutar nuestro programa, lo primero es observar detenidamente la gráfica de similitud:

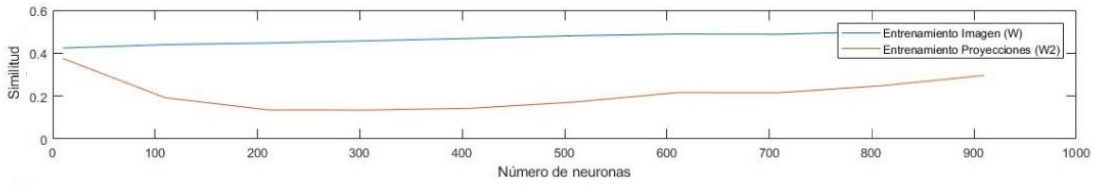


Figura 10. Similitud entre imagen real e imagen con proyecciones

Como podemos observar con pocas neuronas ya obtenemos casi una similitud de casi el 40% con pocas neuronas, pero dado que las neuronas se reparten de forma aleatoria, podemos afirmar que las pocas neuronas han caído en los lugares donde estaban las deformaciones y por ello tenemos ese índice tan alto. Observamos que no es hasta casi 600 neuronas cuando empezamos a aumentar la similitud, de manera que nos lleva a pensar que cuantas más neuronas añadamos, más precisa será nuestra simulación.

A continuación vamos a observar de manera visual la comparativa entre las 2 imágenes (la original y la entrenada con las proyecciones):

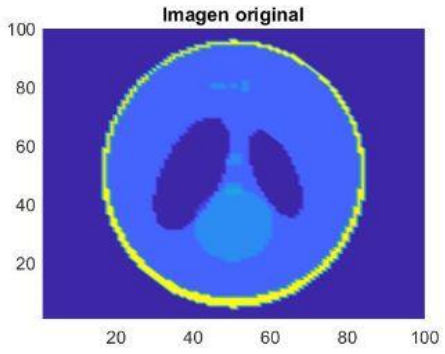


Figura 5. Imagen original

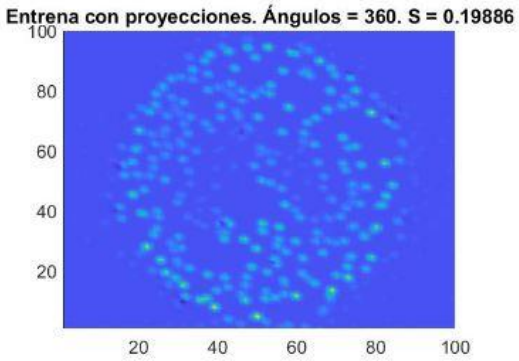


Figura 10. Entrenamiento con proyecciones para 610 neuronas

Como podemos observar, a nivel visual no queda nada claro donde están las irregularidades de lo que sería nuestra estructura, por ello podemos afirmar que el índice de similitud del aproximadamente 20 % no sería suficiente.

Por ello aumentamos las neuronas hasta 910, donde observamos la siguiente imagen:

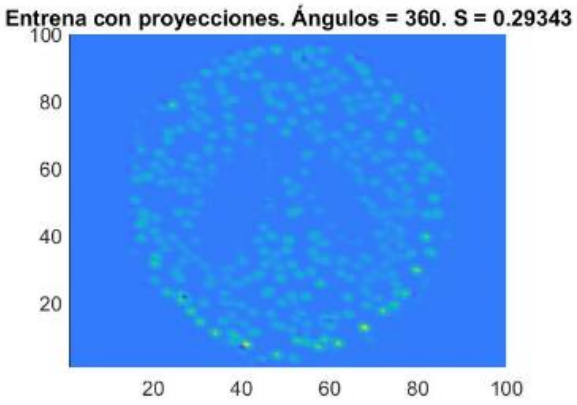


Figura 11. Entrenamiento con proyecciones para 910 neuronas

Como podemos observar en la **figura 11** tenemos una similitud de casi el 30%, y con esto podemos observar que las irregularidades quedan un poco más claras a nivel visual, pero aún no sería suficiente para decir que lo vemos con claridad.

Por ello, vamos a doblar el número de neuronas a partir del cual la similitud empieza a subir (**figura 9**). Probamos con 1600 neuronas y el resultado es el siguiente:

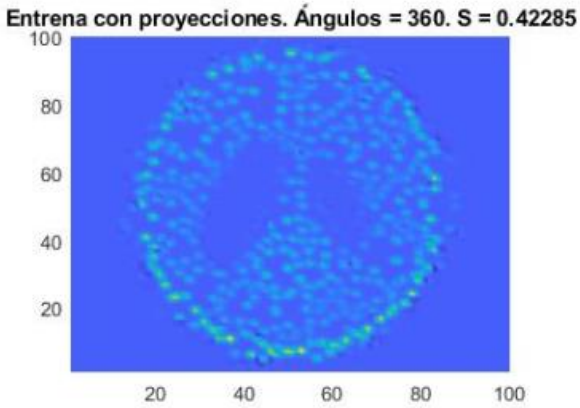


Figura 12. Entrenamiento con proyecciones para 1600 neuronas

Como podemos observar, para un número de neuronas de 1600 con similitud del 42%, ya se vuelve bastante perceptible al ojo humando y por ello podríamos ver con mayor facilidad las irregularidades de nuestra estructura.

El último parámetro, y no por ello menos importante, es el coste temporal que lleva realizar la simulación. A continuación observamos la evolución del coste en función del número de neuronas:

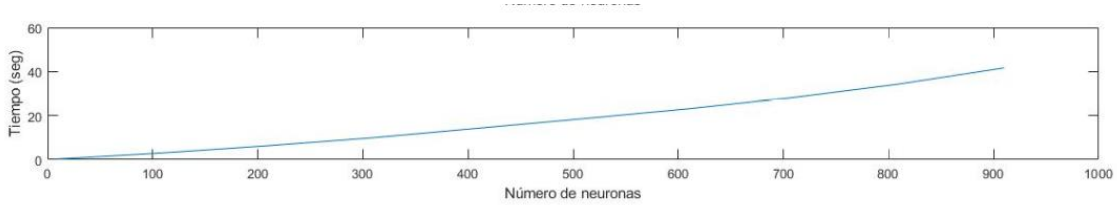


Figura 13. Coste temporal en función del número de neuronas

Si atendemos a la **figura 13**, podemos observar como el coste temporal aumenta de manera cuasi lineal en función del número de neuronas, cuantas más neuronas tengamos, más tiempo le cuesta hacer todo el procedimiento. Podemos afirmar que esto es algo lógico, cuantos más puntos de estudio le pongamos a nuestro programa, más le va a costar procesar toda la información.

A continuación vamos a observar varios ejemplos cuando aumentamos el número de neuronas por encima de 1000:

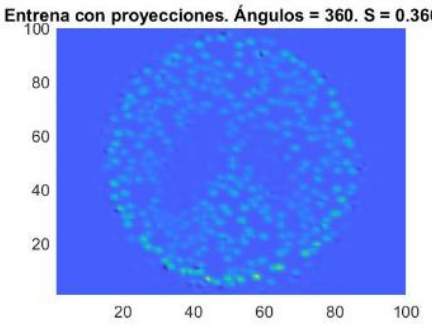


Figura 14 Imagen entrenada con 1100 neuronas

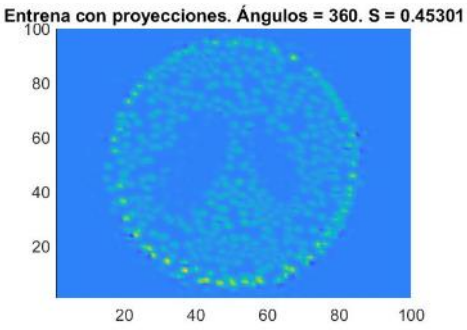


Figura 15 Imagen entrenada con 1500 neuronas

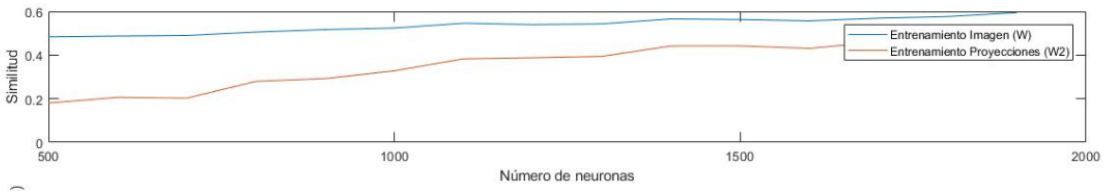


Figura 16 Similitud a lo largo del aumento del número de neuronas

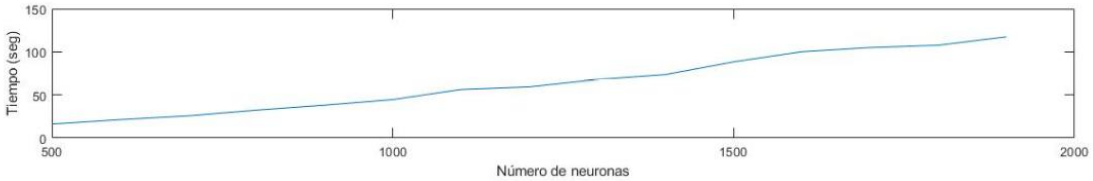


Figura 17 Coste temporal a lo largo del aumento del número de neuronas

Observando ambas gráficas podemos reafirmar que si aumentamos el número de neuronas, conseguiremos mayor claridad en la imagen entrenada (**figuras 14 y 15**) así como aumentaremos el coste de realizar dichos entrenamientos (**figura 17**).

3.3 Parámetro 2: Ángulo máximo de la proyección

A continuación vamos a variar el ángulo máximo de incidencia sobre la estructura. En el caso de las neuronas, el ángulo máximo era de 180º. Pero ahora vamos a observar que pasaría si variamos el ángulo máximo con el que incide la proyección. Para ello vamos a variar en ángulo de incidencia máximo de 90º hasta 360ª, dejando el número de neuronas fijo en 1000 y dejando fijo el número de proyecciones a 180.

Con el siguiente comando en MATLAB, introducimos 100 valores de ángulo máximo de la proyección entre 90ª y 360ª:

parametros = linspace(90,360,100);

Para poder saber qué nivel de calidad tendríamos para cada uno de los ángulos, lo mejor es observar la gráfica de similitud:

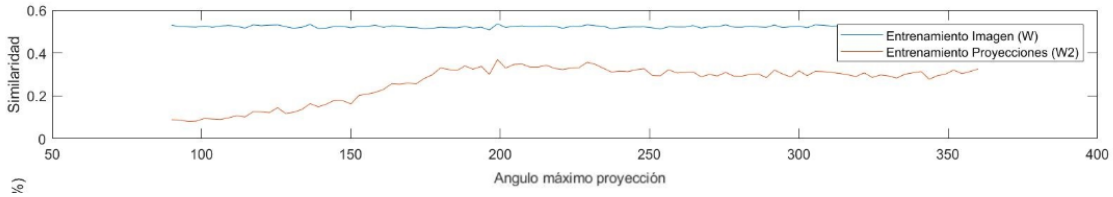


Figura 18. Similitud a lo largo de los diferentes ángulos máximos de proyección

Al observar la gráfica (**figura 18**) podemos darnos cuenta que si limitamos las proyecciones hasta 150º, obtenemos un índice de similitud del 20%, lo cual como ya hemos visto anteriormente no sería suficiente para observar de manera clara las irregularidades que presentaría la estructura. Pero para ello observamos el resultado:

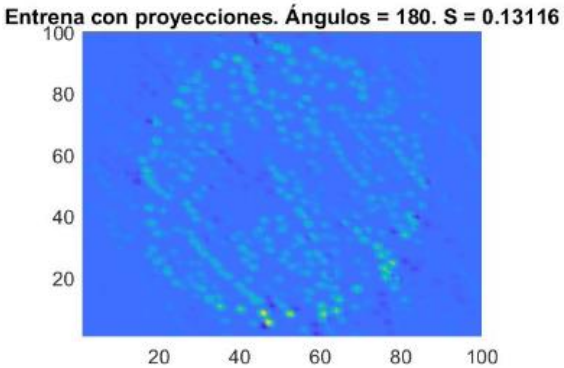


Figura 19. Imagen entrenada con proyecciones con ángulo máximo de la proyección 130.9 º

Si volvemos a la **figura 18**, ésta gráfica nos informa que tendremos un índice de similitud de entre 30-35% aproximadamente sobre 200º, 225º y 300º de ángulos máximos de la proyección. A continuación las imágenes:

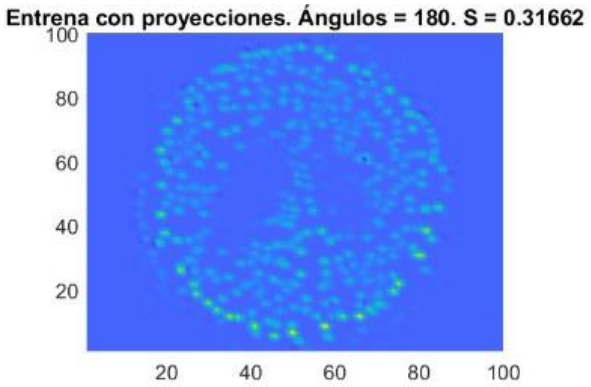


Figura 20. Imagen entrenada con proyecciones con ángulo máximo de la proyección 201.81º

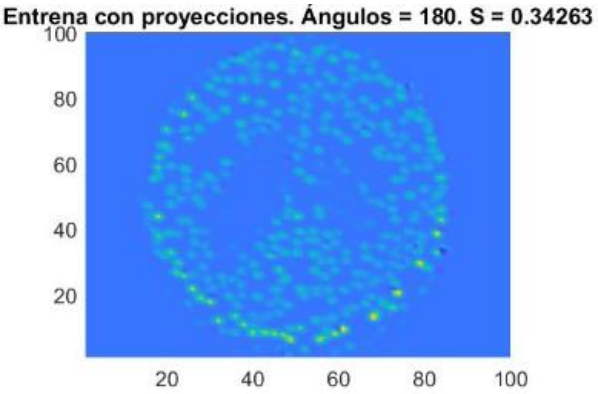


Figura 21. Imagen entrenada con proyecciones ángulo máximo de la proyección 226.36º

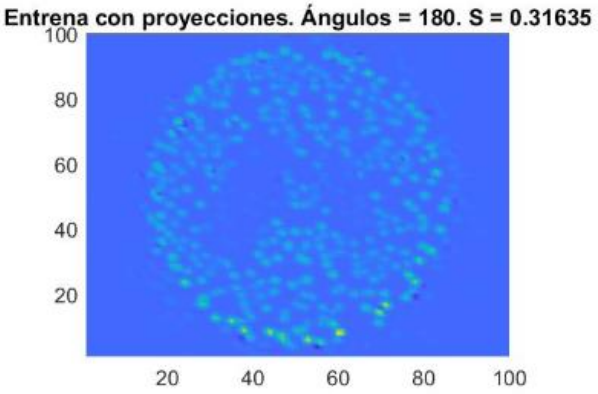


Figura 22. Imagen entrenada con ángulo máximo de la proyección 302.72º

Después de observar detenidamente las imágenes (**figuras 20,21 y 22**), podríamos deducir de manera visual dónde se encuentran las irregularidades, pero bajo mi punto de vista, podemos ver de manera muy clara una de ellas (parte izquierda), pero para la irregularidad de la parte derecha, tenemos dificultades para determinar la otra irregularidad e incluso podríamos pensar que o hay más de 1 irregularidad o que directamente no hay ninguna en la parte derecha de la estructura.

Otro dato importante a destacar es el coste temporal que lleva realizar dicho entrenamiento:

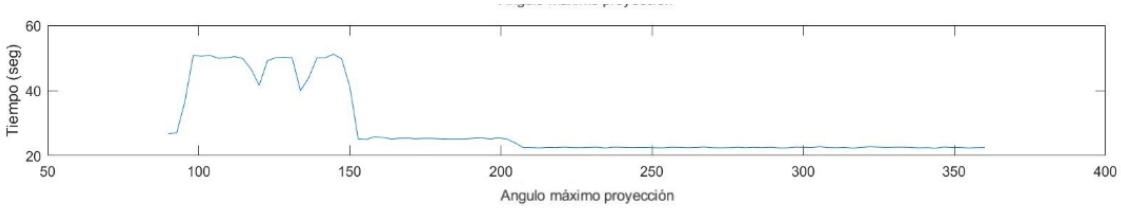


Figura 23. Coste temporal a lo largo de los diferentes ángulos máximos de proyección

Resalta a primera vista si observamos la **figura 23** como el coste varía entre 90º y 180º y a partir de 180º el coste es 0. Si nos paramos a pensar un momento el porqué de esto, enseguida llegamos a la conclusión: a partir de 180º volvemos a donde estábamos al principio, por ello es redundante tener ángulos máximos de proyección mayores a 180º

3.4 Parámetro 3: Número máximo de proyecciones

Anteriormente hemos limitado el ángulo máximo de la proyección pero siempre teníamos el mismo número de proyecciones. Por ello ahora, fijando ahora el ángulo máximo de las proyecciones, vamos a ir variando el número de proyecciones que tendríamos entre los ángulos de éstas. Para ello escribimos en MATLAB:

```
parametros  = unique(round(linspace(10,400,100)));
```

De ésta manera definimos el número máximo de proyecciones entre 100 y 400 y nos aseguramos tener un número entero (no podemos tener un número de proyecciones que no sea entero).

Como primer paso, debemos detenernos a visualizar la gráfica de la similitud a lo largo de la variación del número máximo de proyecciones:

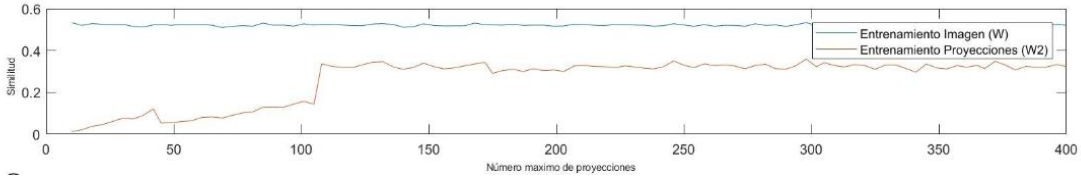


Figura 24. Similitud a lo largo del número máximo de proyecciones

Después de observar detenidamente la **figura 24,** podemos afirmar que con menos de 120 proyecciones tenemos índices de similitud de menos de 20%, lo cual sabemos por los casos anteriores que es insuficiente para poder visualizar los defectos que podría presentar nuestra estructura. Por ello, no es hasta 120-140 proyecciones cuando empezamos a tener unos índices de similitud alrededor del 35%, con el cual podemos distinguir a nivel visual los defectos que podría presentar nuestra estructura.

A continuación vamos a observar distintas imágenes con diferentes números máximos de proyecciones para reforzar aquella información que nos da la **figura 24**.

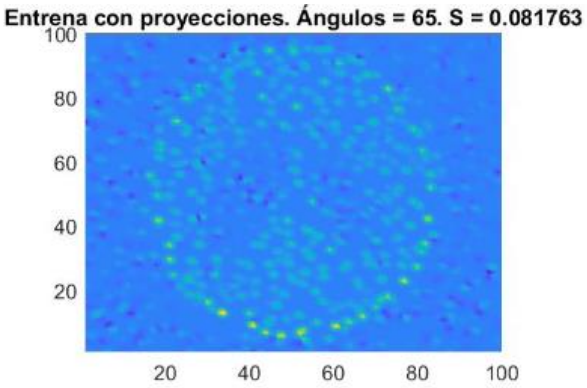


Figura 25. Imagen entrenada con 65 proyecciones

Como cabría esperar, observamos que para 65 proyecciones, nos es insuficiente para poder visualizar con claridad las deformaciones que pueda presentar nuestra estructura. Observamos que el nivel de similitud es del 8%, lo cual es demasiado bajo.

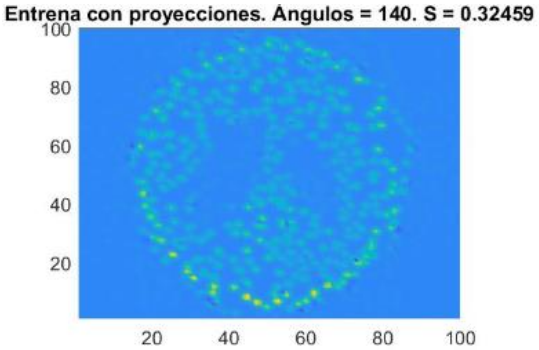


Figura 26. Imagen entrenada con 140 proyecciones

En la **figura 26**, observamos cómo quedaría la imagen entrenada con 140 proyecciones, que como bien nos indicaba la **figura 24**, tenemos un índice de similitud del 32% y a nivel visual podemos observar sin mucha dificultad las irregularidades que presenta la estructura.

Si seguimos observando la gráfica de similitud (**figura 24**) podemos ver que a partir de 130-140 proyecciones, aunque aumentemos el número de proyecciones, no vamos a conseguir aumentar el nivel de similitud y por ello no conseguiremos aumentar la claridad con la que podríamos distinguir los defectos en nuestra estructura.

A continuación voy a presentar varios ejemplos de cómo a pesar de haber aumentado el número de proyecciones, no conseguimos aumentar la claridad para observar las imágenes entrenadas.

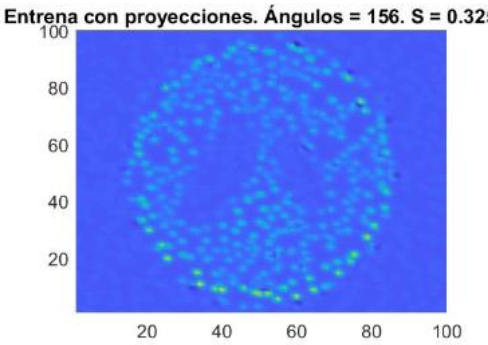


Figura 27. Imagen entrenada con 156 proyecciones

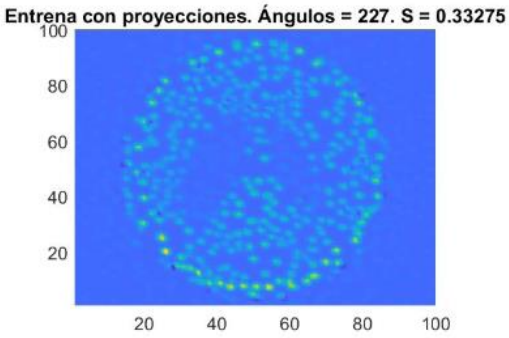


Figura 28. Imagen entrenada con 227 proyecciones

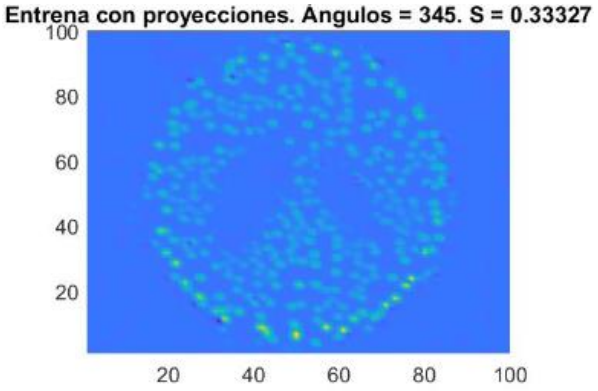


Figura 29. Imagen entrenada con 345 proyecciones

Después de observar la **figura 26** y las **figuras 27,28** y **29** podemos afirmar que a pesar de que éstas últimas 3 tiene muchísimas más proyecciones que la **figura 26**, no hemos conseguido que la imagen entrenada salga más nítida y con más claridad que la **figura 26**. Por tanto, teniendo en cuenta esto, y observando la gráfica del coste temporal (**figura 30**):

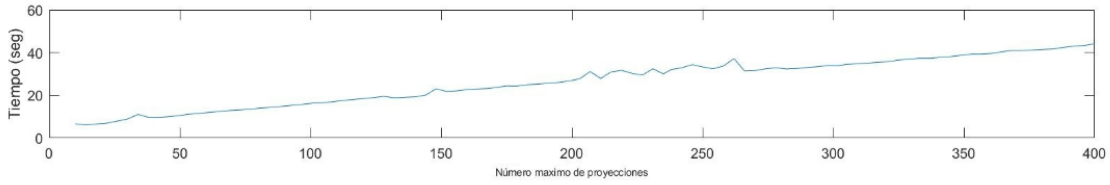


Figura 30. Coste temporal a lo largo del número máximo de proyecciones

Podemos afirmar que no vale la pena aumentar el número de proyecciones más allá de 140-150, porque lo único que hacemos es aumentar el coste temporal que lleva entrenar las proyecciones y no por ello conseguimos aumentar la nitidez y claridad de la imagen resultado.

3.5 Parámetro 4: Anchura k de la función

A continuación vamos a variar el parámetro anchura k de la función. Este hará que la imagen resultante se vea con mayor o menor nitidez. Para ello, fijamos las neuronas en 1000, el ángulo

máximo de la proyección 180º y un valor de 360 al número de proyecciones. A todo esto, añadimos la siguiente línea a nuestro programa:

```
parametros = 1:5:50;
```

Y con esto nuestra anchura k variará entre 1 y 50 en intervalos de 5.

Como hemos hecho anteriormente, lo primero que debemos hacer es echarle un ojo a la gráfica de similitud en función de la anchura. A continuación la gráfica:

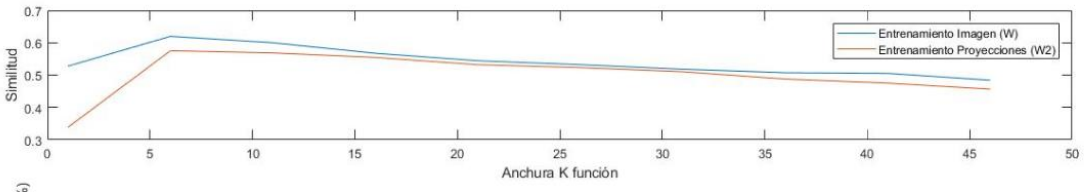


Figura 31. Similitud a lo largo de la anchura k de las RBF

Como podemos observar , en un primer momento, cuando la anchura es menor de 5, el índice de similitud es menor del 40%, y por lo que sabemos de los apartados anteriores, este % es demasiado bajo para obtener una imagen clara de las irregularidades en nuestra estructura. Pero luego observamos que a partir de 5 hasta 10, el índice de similitud ronda entre 40%-50% lo que es un valor de similitud suficientemente alto para observar alguna irregularidad. Luego a partir de una anchura de 12-13, el índice de similitud disminuye de forma lenta y progresiva, lo que nos lleva a pensar que si seguimos aumentando la anchura, iremos perdiendo nitidez en nuestra imagen.

Por ello, vamos a observar las diferentes imágenes resultantes para diferentes valores de k dónde apreciaremos en peor o mejor forma las irregularidades:

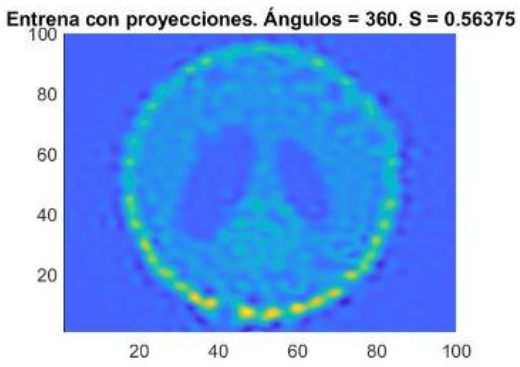


Figura 32. Imagen entrenada para k=6

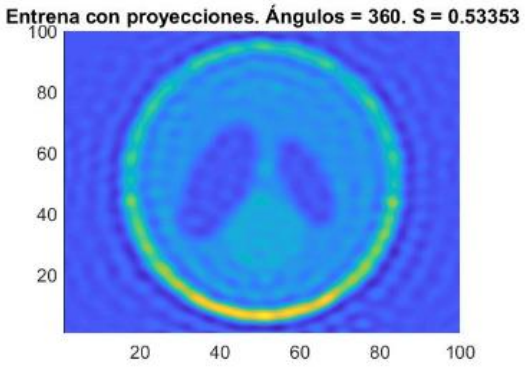


Figura 33. Imagen entrenada para k=22

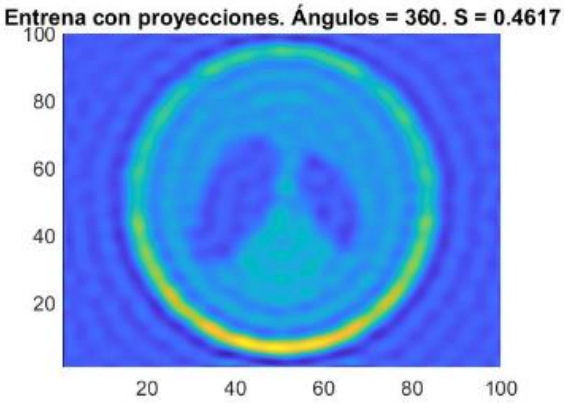


Figura 34. Imagen entrenada para k=46

Como podemos observar, a medida que subimos la anchura de la función , las RBF entre ellas se ayudan , pero hay un momento que se empiezan a superponer demasiado unas a otras lo que lleva que la imagen resultado final empiece a verse borrosa.

Por último vamos a observar la gráfica del coste temporal en función de la anchura k de las funciones:

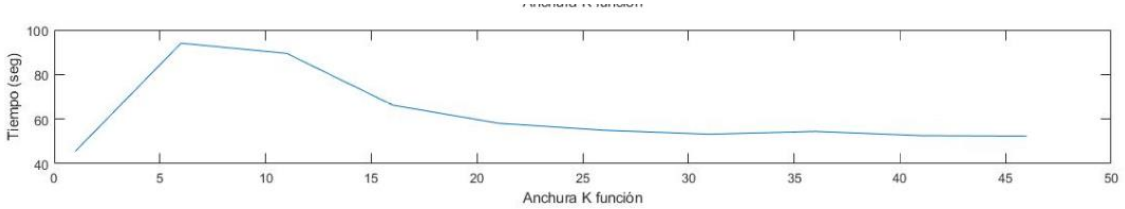


Figura 35. Coste temporal en función de la anchura de la RBF

Como cabía esperar, dado que si las funciones son muy anchas, se solapan entre ellas, por ello el coste de calcular cada punto pasa a ser menor, pero como también si las funciones son muy anchas la imagen final se emborrona, habría que decidir en qué punto es óptimo en función coste/nitidez.

3.6 Simulación final con parámetros óptimos

Una vez analizados todos los parámetros de las funciones que podemos variar, podemos encontrar los valores óptimos de cada uno de ellos para obtener una imagen entrenada con proyecciones limpia y así poder ver con claridad donde se encontrarían los defectos de nuestra estructura. Haciendo un repaso a todo lo anterior, podemos decidir que los valores óptimos para cada parámetro serían:

- Nr= 1600
- Ángulo máximo de la proyección = 200º
- Número máximo de proyecciones = 150
- Anchura k de la RBF= 11

Con los siguientes parámetros, la imagen final optimizada entrenada con proyecciones sería tal que así:

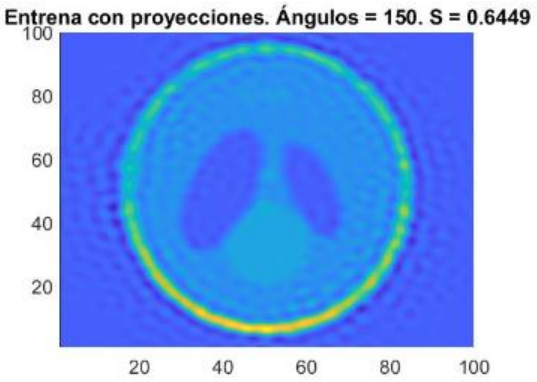


Figura 36. Imagen entrenada con proyecciones con parámetros óptimosAsí mismo, como nuestro programa es reiterativo, al no cambiar ningún parámetro, en cada repetición lo único que varía es la distribución de los centros (dado que es aleatoria, en cada repetición tenemos una distribución de centros diferentes) y por ello tenemos una imagen diferente para cada distribución. Pero por ello, vamos a observar la gráfica de similitud para los parámetros óptimos encontrados en los diferentes casos anteriores:

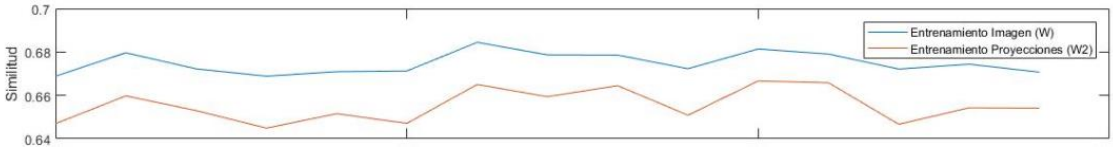


Figura 37.Gráfica con índice de similitud para los valores óptimos

Si observamos la **figura 37** primero podemos observar que no hay valores en el eje de abscisas (porque no hay evolución de ningún parámetro, simplemente la variación entre una repetición y otra es la posición de los centros (que se pone de manera aleatoria, como hemos comentado anteriormente). Fijándonos en el índice de similitud, podemos ver que entre una simulación y otra varía un 4%, lo que a nivel visual tampoco se apreciaría pero, si observamos el valor total, vemos que oscila sobre el 64-66% lo cual es el valor más alto que podríamos conseguir, dado que optimizando los valores de uno en uno, nunca hemos conseguido alcanzar niveles tan altos.

Fijándonos en la **figura 36**, podemos observar que no cabría ningún tipo de duda en que las deformaciones que tendría nuestra estructura son 2, por lo tanto podemos afirmar que la optimización de los parámetros es correcta y el nivel de detalle (similitud 64%) sería suficiente para poder trabajar y encontrar las deformaciones de nuestra estructura.

Capítulo 4. Conclusiones y líneas futuras.

Después de analizar todas las pruebas variando los diferentes y gracias a la posibilidad de comparar el resultado de las proyecciones con una imagen original podemos afirmar que el algoritmo creado por una red neuronal de funciones de base radial nos permite obtener con bastante precisión la obtención de imágenes reconstruidas por las proyecciones en el interior de estructuras dañadas por el proceso de carbonatación. Por ello, no queda ninguna duda de la capacidad de las redes neuronales basada en funciones de base radial para resolver el problema de ingeniería planteado en este trabajo.

Pero también no debemos olvidar que estamos simulando con la herramienta de trabajo MATLAB y con ello son proyecciones ideales de una imagen controlada. Para saber el nivel real de claridad tendríamos que construir las RBF con señales ultrasónicas reales (dado que las señales ultrasónicas no trabajan de manera ideal, no recorrerían la estructura en línea recta porque encontrarían deformaciones que harían que las señales se desviaran, atenuaran…)

Por ello, si quisiéramos simular con señales ultrasónicas, podríamos trabajar con señales simuladas del paquete *wave2000* y contrastar la capacidad entre la imagen original y la reconstruida con las señales ultrasónicas. Una vez probada dicha capacidad, se podrían hacer medidas reales de probetas controladas.

También podríamos ajustar la optimización con un algoritmo recursivo que a partir de las señales de error, se autoajustase para eliminar dicho error y tener una calidad de la imagen mucho más alta.

Cabe destacar que la distribución de los centros es aleatoria, por tanto puede dar la casualidad de que la deformación de la estructura sea demasiado pequeña y que los centros de las RBF no cayesen dentro de la deformación, por tanto no podríamos ver éstas deformaciones y con ello tendríamos un error grande. La manera de solucionar esto sería distribuir de manera uniforme los centros de las funciones, o si tuviésemos una idea de la posición donde encontraríamos la deformación, aumentaríamos el número de funciones dentro de esa zona para así obtener una imagen mucho más clara de la deformación (perderíamos calidad fuera de la zona de trabajo, pero tendríamos una imagen con un nivel de fidelidad más alto dado que nuestras funciones trabajarían muy cerca de las deformaciones y con ello aumentaríamos el rendimiento de los algoritmos)

Para finalizar, si observáramos que tuviésemos materiales de distinto tipo o que las deformaciones fuesen muy grandes, el parámetro que nos convendría ajustar para optimizar el procedimiento sería el ancho k de las funciones.

Bibliografía

- https://es.mathworks.com/matlabcentral/fileexchange/35660-tutorial-funciones-de-base-radial-rbf-redes-neuronales-como-aproximadoras-de-funciones
- http://www.brnt.eu/phd/node11.html#SECTION00634200000000000000
- http://www.webdelprofesor.ula.ve/economia/gcolmen/programa/redes_neuronales/capitulo4_funciones_bases_radiales.pdf
- http://mccormickml.com/2013/08/15/radial-basis-function-network-rbfn-tutorial/
- http://www.varpa.org/~mgpenedo/cursos/scx/archivospdf/Tema5-6.pdf