Caracterización de anomalías mediante aprendizaje automático

Adrián Sanjuán Espejo

Doble Grado en Ingeniería Informática y Matemáticas. Facultad de Matemáticas.
Universidad Computense de Madrid

3 de julio de 2024

Dirigido por:

Ana María Carpio Rodríguez (Universidad Complutense de Madrid)
Macarena Gómez Mármol (Universidad de Sevilla)

Resumen

En este trabajo se analiza la viabilidad de resolver problemas inversos mediante técnicas de aprendizaje automático. Para ello, se estudia una ecuación específica que modela el comportamiento de un medio ante el paso de ondas a través de él.

En una primera parte, se aborda la resolución del problema directo asociado al modelo matemático planteado mediante el método de elementos finitos. La resolución del modelo permitirá generar un conjunto de datos de soluciones de la ecuación a partir de ciertos parámetros determinados, lo cuál facilitará la experimentación posterior con la resolución del problema inverso, en la que se pretende inferir los parámetros a partir de la soluciones obtenidas.

Una vez obtenido el conjunto de datos, se estudian teóricamente arquitecturas de redes neuronales con potencial para aproximar correctamente soluciones al problema inverso. Tras este estudio, se implementan tres redes neuronales: una red neuronal *feedforward*, una red convolucional y una red recurrente, con el fin de analizar su viabilidad para aproximar la solución del problema inverso. El estudio muestra el potencial de las redes neuronales convolucionales y recurrentes para resolver el problema planteado obteniendo errores pequeños en el conjunto de test.

Palabras clave

Ecuación de ondas; Problema inverso; Problema directo; Método de elementos finitos; Aprendizaje automático; Redes neuronales convolucionales; Redes neuronales recurrentes; Python

Abstract

This work analyzes the feasibility of solving inverse problems using machine learning techniques. To do so, a specific equation that models the behavior of a medium as waves pass through it is studied.

In the first part, the solution of the direct problem associated with the proposed mathematical model is addressed using the finite element method. Solvig the model will allow the generation of a dataset of equation solutions given certain parameters of it, which will facilitate the experimentation with the solving of the inverse problem, where the aim is to infer the parameters from the obtained solutions.

Once the dataset is obtained, the theory behind neural network architectures with potential to correctly approximate solutions to the inverse problem is studied. Following this study, three neural networks are implemented: a feedforward neural network, a convolutional neural network, and a recurrent neural network, in order to analyze their viability in approximating the solution of the inverse problem. The study shows the potential of convolutional and recurrent neural networks to solve the proposed problem, achieving low errors in the test set.

Keywords

Wave equation; Inverse problem; Direct Problem; Finite element method; Machine Learning; Convolutional neural network; Recurrent neural network; Python

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Me gustaría expresar mi más sincero agradecimiento a mis tutoras Ana Carpio Rodríguez y Macarena Gómez Mármol por su guía, apoyo y dedicación durante la realización del trabajo de fin de grado.

A mi familia por su apoyo incondicional, no solo durante el periodo de realización de este trabajo, sino durante toda mi etapa universitaria.

No quiero olvidarme de mis compañeros de carrera, con mención especial a Adrián Pérez Peinador y Rubén Gómez Blanco, quienes han sido un pilar fundamental en mi vida durante los últimos 6 años y estoy seguro de que lo van a seguir siendo.

Capítulo 1

Introducción

El estudio del subsuelo mediante técnicas no invasivas es un método comúnmente utilizado en el ámbito de la geofísica. De manera similar, en medicina se emplean técnicas afines, aunque a escalas más pequeñas, para la recreación en imágenes de la estructura interna de órganos y tejidos con fines de diagnóstico de patologías. Estas técnicas se basan en la emisión de ondas que interactúan con el medio y en la recepción del campo de ondas resultante por medio de sensores, permitiendo así inferir la estructura del medio analizado. Dependiendo del tipo de onda utilizada, estas técnicas reciben diferentes nombres, y dicho tipo determina qué clase de medio se puede observar, y por tanto para qué tipo de diagnóstico se utiliza. Algunos de los ejemplos más conocidos por el público general pueden ser radiografías con rayos-x utilizadas para observar principalmente huesos y tejidos densos, ecografías con ondas sonoras utilizadas para obtener imágenes de órganos y tejidos internos, o resonancias magnéticas que utilizan campos magnéticos y ondas de radio para proporcionar imágenes detalladas de tejidos blandos, articulaciones y órganos. Además, existen otras técnicas como ultrasonidos y las elastografías en medicina, o el estudio de ondas sísmicas en geofísica.

Las primeras secciones de este trabajo se centran en explicar brevemente este tipo de técnicas, haciendo especial hincapié en las utilizadas en geofísica, y relacionándolas con el problema que queremos resolver. También se hará referencia al artículo [2], el cual se enfoca en el uso de elastografías en medicina.

1.1. Problema de imagen

El estudio de las variaciones en ondas sísmicas es una técnica que permite a los geofísicos caracterizar el subsuelo. Analizando las variaciones de las ondas sísmicas se puede recrear la estructura del subsuelo identificando posibles capas o vetas. Técnicas análogas se utilizan en medicina con distintos tipos de onda, con el objetivo de caracterizar la estructura del tejido biológico, y estudiar posibles anomalías presentes en él. En este sentido, el artículo [2] presenta un modelo matemático que posibilita el uso de las elastografías. En el trabajo se va a estudiar este modelo y su aplicación en el ámbito geofísico.

Este modelo consiste en una ecuación de ondas escalar que modela el comportamiento de un medio cuando pasa una onda a través de él. Esta ecuación depende de una serie de parámetros que representan distintas propiedades del medio, en este caso modelizarán las propiedades de interés que son la elasticidad y la densidad. El problema puede ser tratado matemáticamente como un problema directo y como un problema inverso. El problema directo consiste en resolver la ecuación mencionada anteriormente para obtener el correspondiente campo de ondas, y así recrear la estructura del medio dados unos parámetros. Por otro lado, el problema inverso se refiere a la tarea de determinar los parámetros de la ecuación a partir de una imagen o un campo específico. Este último enfoque es el más útil desde un punto de vista práctico, ya que el interés reside en poder obtener información sobre las características del medio a partir de una elastografía o de un campo de ondas del subsuelo, y así poder hacer un diagnóstico. En cualquier caso, es importante estudiar con precisión el problema directo para corroborar la exactitud del modelo matemático. En las siguientes secciones se analiza con más profundidad cada uno de los problemas planteados.

1.2. Problema directo

La elastografía mediante ondas de cizalla analiza variaciones en el módulo de cizalladura μ, la cual es una propiedad relacionada con la elasticidad del tejido, que varía en gran medida entre tejido sano y tejido dañado. Esta técnica se puede aplicar a una escala mayor para la

identificación de anomalías o vetas compuestas por inclusiones de materiales en el subsuelo. Las ondas de cizalla a bajas frecuencias prácticamente no experimentan atenuaciones que podría causar el medio, por lo que podemos considerar que se ven gobernadas por ecuaciones de ondas estándar.

Antes de plantear la ecuación vamos a definir los distintos términos que van a aparecer en ella proponiendo a la vez la configuración física del medio, emisores y receptores:

- Consideramos $R \subseteq \mathbb{R}^2$ el medio.
- Las anomalías presentes en el medio las definiremos por $\Omega = \bigcup_{\ell=1}^{L} \Omega_\ell$
- $x_j, \quad j = 1, \ldots, J$, serán los emisores colocados en un parte Σ de la frontera ∂R.
- $r_k, \quad k = 1, \ldots, K$, serán los receptores colocados en la misma región Σ que los emisores.
- Denominaremos la densidad del medio con ρ. Siguiendo la notación, la densidad de las anomalías la denominaremos como ρ_i.
- Denominamos a las constantes elásticas del medio por μ y λ. De la misma manera nos referiremos a las constantes elásticas de las anomalías con μ_i y λ_i.

Es necesario aclarar que, en cuanto a la disposición de los emisores y receptores, podemos considerar distintas configuraciones, como una intercalación de estos o su disposición superpuesta. Para simplificar, vamos a considerar la configuración en la que los emisores y receptores están superpuestos. La disposición general del problema se puede observar en la figura 1.1.

Teniendo en cuenta todo lo anterior, consideremos que las ondas emitidas están gobernadas por la ecuación de ondas:

$$\begin{aligned} &\rho u_{tt} - \operatorname{div}(\mu \nabla u) = f(t)g(x), \quad x \in R, t > 0, \\ &u(x,0) = 0, u_t(x,0) = 0, \qquad\quad x \in R. \end{aligned} \tag{1.1}$$

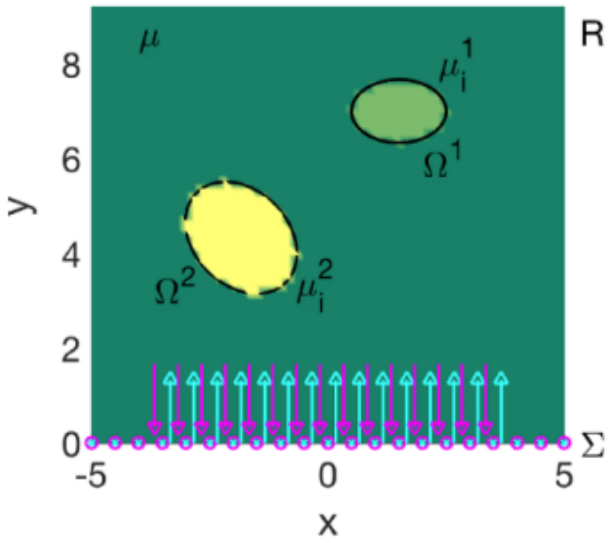

Figura 1.1: *Disposición general del problema. Los emisores (asteriscos azules) y receptores (círculos magenta) se encuentran superpuestos y en una zona Σ del eje X que forma parte de la frontera de R. Nota: La figura está tomada del artículo [2] en el que toman $\rho = \rho_i$, sin embargo en este trabajo se considerará $\rho \neq \rho_i$.*

donde:

$$\rho(x) = \begin{cases} \rho, & \text{if } x \in R \setminus \overline{\Omega}, \\ \rho_i^\ell, & \text{if } x \in \Omega^\ell, \quad \ell = 1, \ldots, L. \end{cases}$$

$$\mu(x) = \begin{cases} \mu, & \text{if } x \in R \setminus \overline{\Omega}, \\ \mu_i^\ell, & \text{if } x \in \Omega^\ell, \quad \ell = 1, \ldots, L. \end{cases}$$

La función $u(t, x)$ representa la evolución de la onda en el espacio y el tiempo, es decir, representa el desplazamiento que sufre un punto x del medio en el instante t. Por tanto, los términos u_t y u_{tt} representan la velocidad y la aceleración de dicho punto respectivamente. Cabe destacar que la ecuación hace énfasis en la aceleración u_{tt} que aparece junto al término de la densidad del medio ρ.

A continuación analizamos el término $-\text{div}(\mu \nabla u)$. El término ∇u representa el gradiente

de la función u o dicho de otra forma, la pendiente o tasa de cambio en cada dirección espacial. En este caso aparece multiplicado por μ que es una función constante a trozos que representa una de las características elásticas del medio. Esto lleva a entender el término $\mu\nabla u$ como un vector que representa la tasa de cambio en cada dirección espacial ponderada por la elasticidad del medio. Por último, la divergencia de un vector nos va a devolver un escalar que representa la diferencia entre el flujo saliente y el flujo de entrada de un campo vectorial, dicho de otra manera, se puede interpretar como la rapidez neta con la que se conduce la materia al exterior de cada punto.

Por último nos queda analizar el término $f(t)g(x)$. Estas funciones describen como actúan las fuentes, en este caso, los emisores. Vamos a asumir que las fuerzas inducidas por los emisores se modelan mediante unos términos de la forma $f(t)g_j(x - x_j)$ [2], donde g_j son funciones suaves de soporte compacto alrededor del 0. Es decir g_j son funciones C^∞ que adquieren valores distintos a cero solamente en un conjunto cuya adherencia es cerrada y acotada alrededor del origen. La suma de los términos g_j da lugar a la función $g(x)$. La función $f(t)$ se va a representar como un pulso de Ricker $f(t) = f_0\,(1 - 2\pi^2 f_M^2 t^2)\,e^{-\pi^2 f_M^2 t^2}$.

Para que el modelo matemático esté bien planteado necesitamos añadir unas condiciones de contorno, en este caso consideramos:

$$\frac{\partial u}{\partial n} = 0, \text{ en } \partial R \tag{1.2}$$

Las ecuaciones (1.1) y (1.2) definen el problema, donde $\rho \in L^\infty(R), \mu \in L^\infty(R), \rho \geq \rho_0 > 0, \mu \geq \mu_0 > 0$. Una vez definido el problema, se enuncia el siguiente resultado:

Teorema 1. *[1][2][9][12] Sean R y Ω dominios $\mathcal{C}^1$ a trozos, donde $\Omega \subset R$. Asumimos $f \in \mathcal{C}^\infty(\mathbb{R}^+) \cup L^\infty(\mathbb{R}^+)$ y $g \in \mathcal{C}^\infty(\mathbb{R}^2) \cup L^\infty(\mathbb{R}^2)$. Entonces, el problema*

$$\begin{aligned} &\rho u_{tt} - div(\mu\nabla u) = f(t)g(x), && x \in R,\ t > 0,\\ &u(x,0) = 0, u_t(x,0) = 0, && x \in R.\\ &\frac{\partial u}{\partial n} = 0, && en\ \partial R \end{aligned}$$

tiene una única solución débil $u \in C([0,\tau]; H^1(R)), u_t \in C([0,\tau]; L^2(R)), u_{tt} \in L^2(0,\tau; (H^1(R))')$, *para cualquier* $\tau > 0$. *Además, si* $u_{tt}(x,0) \in L^2(R)$, *también se tiene* $u_t \in C([0,\tau]; H^1(R))$ *y* $u \in C([0,\tau]; H^2(R\backslash\overline{\Omega}))$.

La demostración de este resultado[1] puede encontrarse en el artículo [1], además en el artículo referencia de esta sección [2] también ses esboza esta demostración.

1.3. Problema inverso

El análisis del problema inverso tiene especial relevancia en ámbitos geofísicos y médicos, ya que en un caso real existen una serie de datos obtenidos a través de una prueba y el objetivo es averiguar qué hay en esos datos. En este caso queremos explorar la existencia de vetas[2] en el subsuelo, representadas por inclusiones de distinto material, y cuáles serían sus características. En este sentido, la mayor parte del trabajo se centrará en explorar una vía potencial para la resolución del problema inverso mediante la aplicación de técnicas de aprendizaje automático.

Desde un punto de vista matemático, el problema inverso consiste en encontrar las inclusiones y sus parámetros de manera que la solución del problema directo con dichos parámetros, coincida con los datos obtenidos en los receptores. Se trata de identificar Ω, ρ_i y μ_i.

La concordancia con los datos recibidos se mide a través de una función de coste. Por tanto, el problema inverso consiste en averiguar para que parámetros Ω, μ_i, ρ_i se minimiza esta función [2]:

$$J(\Omega, \mu_i, \rho_i) = \frac{1}{2} \sum_{k=1} \sum_{m=1} |u_{\Omega,\mu_i,\rho_i}(r_k, 0, t_m) - d_m^k|^2 \tag{1.3}$$

donde $u_{\Omega,\mu_i,\rho_i}(r_k, 0, t_m)$ es la solución del problema directo y d_m^k el valor obtenido en los receptores, $d_k^m = d_{k,\text{true}}^m + ruido$ donde $d_{m,\text{true}}^k = u(r_k, 0, t_m)$. Como se ve en la sección 1.2

[1] El resultado hace referencia a varios espacios que se explican en la sección 2.

[2] En el resto del trabajo nos referiremos a vetas como anomalías, utilizando un temas más genérico que puede ser aplicable tanto en el área de la medicina como de la geofísica.

$k = 1, .., K$ hace referencia al receptor concreto de la red de receptores y $m = 1, .., M$ al momento temporal en el que se mide el valor.

Aunque nos vayamos a centrar en técnicas de aprendizaje automático, es importante mencionar la existencia de otras vías utilizadas para la resolución del problema inverso. Entre los posibles métodos aplicables vamos a destacar la aproximación utilizada en [2] que da lugar a este trabajo. En [2] tratan en profundidad la aplicación de métodos bayesianos para obtener un marco FWI (Full-waveform inversion), técnica que trata de estimar la estructura subsuperficial de un medio. Esta técnica suele ser utilizada para la obtención de imágenes del subsuelo mediante el análisis de ondas sísmicas, y en el artículo plantean su utilización para la obtención de la estructura de un tejido.

En el capítulo tercero se introducen las diferentes técnicas que se van a utilizar para intentar resolver el problema, pero antes conviene hacer un estudio de como obtener la solución al problema directo.

Capítulo 2

Resolución del problema directo: Método de elementos finitos

En este capítulo vamos a tratar la resolución del problema directo, es decir, la obtención de la solución a las ecuaciones (1.1) y (1.2). Para ello vamos a estudiar la aplicación del método de elementos finitos y una implementación de este. Es importante resaltar que la resolución del problema directo no constituye el objetivo principal de este trabajo, y por tanto no vamos a dedicarnos a ello de manera exhaustiva. Sin embargo, conviene analizar y corroborar la exactitud y utilidad del modelo. Esta parte es esencial ya que es la que permite generar un un conjunto de datos sintético con el que entrenar un modelo que resuelva el problema inverso.

Antes de proceder con la resolución, recordamos la ecuación y las condiciones de contorno del problema:

$$\begin{aligned} &\rho u_{tt} - \operatorname{div}(\mu \nabla u) = f(t)g(x), && x \in R, t > 0, \\ &u(x,0) = 0, u_t(x,0) = 0, && x \in R, \\ &\frac{\partial u}{\partial n} = 0, && \text{en } \partial R, \end{aligned} \tag{2.1}$$

donde:

$$\rho(x) = \begin{cases} \rho, & \text{if } x \in R \setminus \overline{\Omega}, \\ \rho_i^\ell, & \text{if } x \in \Omega^\ell, \quad \ell = 1, \dots, L, \end{cases}$$

$$\mu(x) = \begin{cases} \mu, & \text{if } x \in R \setminus \overline{\Omega}, \\ \mu_i^\ell, & \text{if } x \in \Omega^\ell, \quad \ell = 1, \dots, L. \end{cases} \tag{2.2}$$

2.1. Formulación variacional del problema de ondas

Nuestra ecuación tiene un término $\text{div}(\mu\nabla u)$ en el que se pretende calcular la divergencia de una función que involucra un coeficiente discontinuo μ, como vemos en (2.2). La formulación variacional (o formulación débil) de nuestro problema nos va a posibilitar evitar este inconveniente permitiéndonos formular la ecuación en forma integral.

Antes de comenzar, vamos a definir una serie de espacios necesarios para comprender la formulación variacional del problema. Dado un dominio R en el espacio:

- $L^2(R) = \{f : R \longrightarrow \mathbb{R} \mid \int_R |f|^2 dx < \infty\}$
- $L^\infty(R) = \{f : R \longrightarrow \mathbb{R} \mid \sup_{x\in R} |f(x)| < \infty\}$
- $C(\overline{R}) = \{f : \overline{R} \longrightarrow \mathbb{R} \mid f \text{ continua en } \overline{R}\}$
- $C([0,T]) = \{f : [0,T] \longrightarrow \mathbb{R} \mid f \text{ continua en } [0,T]\}$
- $H^1(R) = \{f : R \longrightarrow \mathbb{R} \mid \int_R |f|^2 dx < \infty,\ \int_R |f_{x_i}|^2 dx < \infty\ \forall i\}$
- $C([0,T]; H^1(R)) = \{u : t \in [0,T] \longrightarrow u(x,t) \in H^1(R) \mid \text{u continua en [0,T]}\}$
- $C^1([0,T]; L^2(R)) = \{u : t \in [0,T] \longrightarrow u(x,t) \in L^2(R) \mid$
  . u continua con derivada continua en [0,T]$\}$

Si en lugar de C o C^1 escribimos C^k o C^∞, estaremos trabajando con funciones cuyas derivadas hasta el orden k son continuas o funciones cuyas sucesivas derivadas son todas continuas, respectivamente. Si en lugar de H^1 escribimos H^k, significa que todas las derivadas en el sentido de las distribuciones hasta el orden k son funciones de L^2, véase [12] y [9].

La formulación variacional de la ecuación se lleva a cabo tomando una función test $\varphi(x) \in C^\infty(\overline{R})$. En primer lugar, se multiplica la ecuación por esta función y se integra en R obteniendo

$$\int_R \rho u_{tt}\varphi\, dx - \int_R \text{div}(\mu\nabla u)\varphi\, dx = \int_R f(t)g(x)\varphi(x)\, dx. \tag{2.3}$$

Recordando la ecuación planteada al comienzo, necesitamos una forma de resolver las complicaciones que causa el término $\operatorname{div}(\mu\nabla u)$. Esto lo vamos a hacer a través de una de las extensiones del teorema de Green.

Fórmula de Green. *[12] Sea $R \subset \mathbb{R}^2$ un dominio acotado cuya frontera ∂R es C^1 a trozos y sea* $\mathbf{n}$ *el vector unitario normal exterior a ∂R. Dadas dos funciones $u, v \in H^1(R)$ se tiene*

$$\int_R \frac{\partial u}{\partial x_i} v \, dx = -\int_R u \frac{\partial v}{\partial x_i} \, dx + \int_{\partial R} uv \, n_i dS_x.$$

Donde las funciones u y v están definidas en $L^2(\partial R)$ en un sentido débil, como trazas [12].

Toda función $\varphi \in C^\infty(\overline{R})$ pertenece a $H^1(R)$ y está definida en ∂R en el sentido usual, puntualmente. Suponemos que $\operatorname{div}(\mu\nabla u) \in H^1(R)$. Aplicando la fórmula de Green a la segunda integral de (2.3) obtenemos

$$-\int_R \operatorname{div}(\mu\nabla u)\varphi \, dx = \int_R \mu\nabla u\nabla\varphi \, dx - \int_{\partial R} \mu\frac{\partial u}{\partial n}\varphi dx, \tag{2.4}$$

donde última integral es nula en vista de la condición de contorno $\frac{\partial u}{\partial n} = 0$, en ∂R. Insertando la información de (2.4) en la formulación variacional (2.3) obtenemos que para cualquier función de test $\varphi \in C(\overline{R})$ se tiene:

$$\int_\Omega \rho(x) u_{tt}(x,t)\varphi(x) \, dx + \int_\Omega \mu(x)\nabla u(x,t)\nabla\varphi(x) \, dx = \int_\Omega f(t)g(x)\varphi(x) \, dx, \tag{2.5}$$

suponiendo que $\rho(x)u_{tt}(x,t), f(t)g(x) \in L^2(R)$ para cada t, identidad que se extiende a $\varphi \in H^1(R)$ por densidad.

La formulación variacional va a consistir en encontrar:

$$u \in H = C([0,T]; H^1(\Omega)) \cap C^1([0,T]; L^2(\Omega))$$

tal que:

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}\int_\Omega \rho(x)u(x,t)\varphi(x)\,dx + \int_\Omega \mu(x)\nabla u(x,t)\nabla\varphi(x)\,dx$$
$$= \int_\Omega f(t)g(x)\varphi(x)\,dx, \quad \forall\varphi(x)\in H^1(R),\ t>0, \tag{2.6}$$
$$u(x,0) = u_t(x,0) = 0, \quad x\in R, \tag{2.7}$$

dados $\rho(x), \mu(x) \in L^\infty(R), f(t) \in C([0,T]), g(x) \in C(\overline{R})$.

Como se menciona en el teorema 1, se demuestra en [2] el resultado de existencia y unicidad de soluciones débiles del problema definido por (2.6) y (2.7).

2.2. Método de elementos finitos

El método de elementos finitos va a consistir en realizar una aproximación de la función u mediante bases finitas. Como $H^1(R)$ es un espacio de Hilbert separable, admite bases numerables $\{\varphi_1, \varphi_2, \ldots, \varphi_n, \ldots\}$. Es decir, dado $\varepsilon > 0$ se puede encontrar un n_ε y unos coeficientes $u_1, ..., u_{n_\varepsilon}$ de modo que $||u - \sum_{i=1}^{n_\varepsilon} u_i\varphi_i||_{H^1(R)} < \varepsilon$. Los métodos de elementos finitos proporcionan aproximaciones en dimensión finita usando como bases polinomios a trozos definidos sobre mallados de paso a elegir. Refinando los mallados se tienen aproximaciones en mayor dimensión más precisas.

2.2.1. Mallado de R y espacio de elementos finitos

En nuestro caso vamos a construir un mallado triangular a partir de una malla rectangular del dominio R:

$$x_j = x_0 + jh_x, \quad j = 0, .., J,$$

$$y_k = y_0 + kh_y, \quad k = 0, .., K,$$

donde h_x y h_y son el "paso" con el que se divide el dominio en las dimensiones x e y respectivamente. Dividiendo los rectángulos por la diagonal tenemos un mallado triangular, una triangulación. Llamamos T_ℓ, $\ell = 1, \ldots, L$, a cada uno de los triángulos que forman la triangulación. Para facilitar la comprensión del mallado se proporciona la figura 2.1. Así, cada punto (x_j, y_k) se refiere a un nodo del mallado mostrado en la figura.

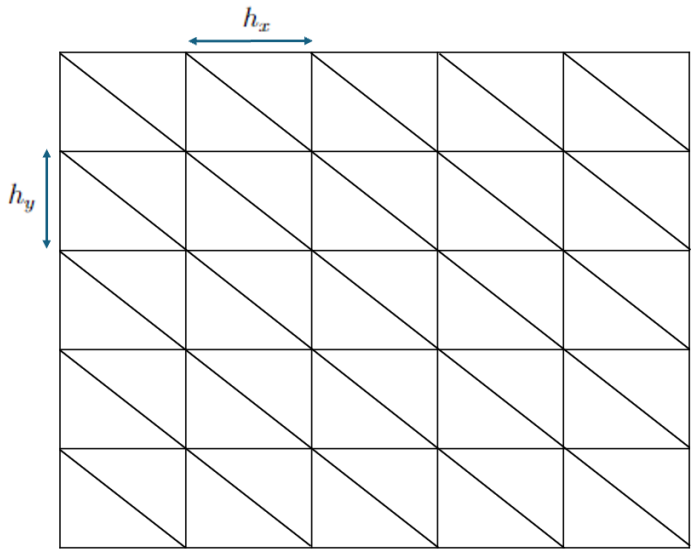

Figura 2.1: *Ejemplo de mallado triangular de un dominio.*

A cada nodo, se le asocia un polinomio a trozos de grado a elegir. En este caso tomaremos grado 1, de modo que $\varphi_{jk}|_{T_\ell}$ es un polinomio de grado 1, es decir, $\varphi_{jk} = a_{jk}^{\ell}x + b_{jk}^{\ell}y + c_{jk}^{\ell}$ con coeficientes a determinar de forma que el polinomio a trozos resultante sea continuo en todo T y cumpla que

$$\varphi_{jk}((x_{j'}, y_{k'})) = \begin{cases} 1 & \text{si } j = j' \text{ y } k = k', \\ 0 & \text{si } j \neq j' \text{ o } k \neq k'. \end{cases}$$

De esta manera el conjunto $\{\varphi_{jk}\}$ para $j = 1, .., J$ y $k = 1, ..., K$ forma una base del espacio de elementos finitos $H^1(R)$.

2.2.2. Discretización de las ecuaciones en espacio

Renombramos los índices tomando, por ejemplo, la función $n(j, k) = K(k-1)+j$ con lo que las funciones de base del espacio de polinomios quedan reordenadas como $\{\varphi_1, \varphi_2, ..., \varphi_N\}$ con $N = JK$. De esta forma

$$V^N = \{\sum_{N=1}^{N} \alpha_n\varphi_n,\ \alpha_n \in \mathbb{R}, \forall n\} = \text{span}\{\varphi_1, \varphi_2, ..., \varphi_N\} \subset H^1(R).$$

La formulación variacional aproximada consistirá por tanto, en buscar $u^N \in V^N$ de la

forma $u^N = \sum_{n=1}^{N} \alpha_n(t)\varphi_n(x)$ tal que se cumpla la ecuación:

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}\int_R \rho(x)u^N(x,t)\varphi_m(x)\,dx + \int_R \mu(x)\nabla u^N(x,t)\nabla\varphi_m(x)\,dx = \int_R f(t)g(x)\varphi_m(x)\,dx \text{ para } m = 1,..,N \tag{2.8}$$

Para resolver este problema tenemos que encontrar los coeficientes $\alpha_m(t)$ para $m = 1,..,N$ de forma que se cumplan las ecuaciones (2.8). Esto da lugar a un sistema formado por N ecuaciones diferenciales con los N coeficientes como incógnitas:

$$M\begin{bmatrix} \frac{\mathrm{d}^2\alpha_1}{\mathrm{d}t^2} \\ \vdots \\ \frac{\mathrm{d}^2\alpha_N}{\mathrm{d}t^2} \end{bmatrix} + A\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{bmatrix} = F(t), \tag{2.9}$$

con datos iniciales $\alpha_n(0) = \alpha'(0) = 0$ para $n = 1,\ldots,N$. En esta ecuación, M y A son matrices cuyas entradas son integrales de la forma $\int_R \rho(x)\varphi_n(x)\varphi_m(x)\,dx$ y $\int_R \mu(x)\nabla\varphi_n(x)\nabla\varphi_m(x)\,dx$, respectivamente[1], y $F(t)$ es un vector con componentes $\int_R f(t)g(x)\varphi_m(x)\,dx$, en todos los casos para $n, m = 1,\ldots,N$.

2.2.3. Existencia y unicidad del sistema de ecuaciones

En esta sección vamos a demostrar la existencia y unicidad del sistema (2.9). Para ello, vamos a llevar a cabo una serie de transformaciones para poder aplicar el Teorema 2 directamente.

Teorema 2. ***(Existencia y unicidad de soluciones para sistemas lineales de ecuaciones diferenciales ordinarias)*** *[4] Sean $A(t)$ y $b(t)$ funciones matriciales $n\times n$ y $n\times 1$, respectivamente. Si $A(t)$ y $b(t)$ son continuas en un intervalo $I \subseteq \mathbb{R}$, que contiene al punto t_0, entonces el problema de valores iniciales*

$$\begin{cases} X' = A(t)X + b(t) \\ X(t_0) = X_0 \end{cases}$$

para cualquier vector $X_0 \in \mathbb{R}^n$ dado, tiene una única solución definida en el intervalo I.

[1]Estos términos se obtienen sustituyendo $u^N = \sum_{n=1}^{N} \alpha_n(t)\varphi_n(x)$ en la ecuación (2.8)

En primer lugar, tomamos $X = \begin{bmatrix} \alpha_1 \\ \vdots \\ \vdots \\ \alpha_L \end{bmatrix}$ para reescribir el sistema (2.9) como

$$MX'' + AX = F(t).$$

M es inversible por las propiedades de la base, por lo que podemos multiplicar la ecuación por M^{-1} obteniendo:

$$X'' + M^{-1}AX = M^{-1}F(t),$$

$$X'' = -M^{-1}AX + M^{-1}F(t).$$

Haciendo el cambio de variable $Y' = X$ obtenemos el sistema:

$$\begin{cases} X' = Y \\ Y' = -M^{-1}AX + M^{-1}F(t) \end{cases} \tag{2.10}$$

Tomando $Z = \begin{bmatrix} X \\ Y \end{bmatrix}$ podemos reescribir el sitema (2.10) como:

$$Z' = \begin{bmatrix} 0 & I \\ M^{-1}A & 0 \end{bmatrix} Z + \begin{bmatrix} 0 \\ M^{-1}F(t) \end{bmatrix} \tag{2.11}$$

Por cómo se ha construido tanto $\begin{bmatrix} 0 & I \\ M^{-1}A & 0 \end{bmatrix}$ como $\begin{bmatrix} 0 \\ M^{-1}F(t) \end{bmatrix}$ sabemos que ambas son funciones continuas. Con estas premisas podemos aplicar el teorema 2 al sistema (2.11) quedando probada la existencia y unicidad de las soluciones de nuestro sistema original (2.9).

2.2.4. Discretización en tiempo

Para construir numéricamente la solución hay que discretizar en tiempo. Tomaremos un paso τ para discretizar $[0, T]$ dando lugar a $[t_0, t_1, ..., t_P]$ donde $t_0 = 0$, $t_p = t_0 + p\tau$ y lógicamente $t_P = T$, $\tau = T/P$.

Aproximando la derivada segunda mediante el método de diferencias finitas podemos tomar:

$$\frac{\mathrm{d}^2\alpha_i}{\mathrm{d}t^2}(t_p) \simeq \frac{\alpha_i(t_p + \tau) - 2\alpha_i(t_p) + \alpha_j(t_p - \tau)}{\tau^2}. \tag{2.12}$$

Sustituyendo en el sistema de ecuaciones (2.9) la aproximación (2.12) obtenemos la recurrencia:

$$M\frac{\alpha_i(t_p+\tau)-2\alpha_i(t_p)+\alpha_j(t_p-\tau)}{\tau^2}=-A\alpha_j(t_p)+F(t_p)$$

que se transforma en

$$M\alpha_i(t_p+\tau)=2M\alpha_i(t_p)-M\alpha_j(t_p-\tau)-\tau^2A\alpha_j(t_p)+\tau^2F(t_p) \tag{2.13}$$

Sabemos que $\alpha_j(0)=0$ y $\alpha_j(0)=0$, lo que nos permite resolver la recurrencia resolviendo un sistema matricial en cada nivel. Hemos de elegir el paso temporal suficientemente pequeño en comparación con el paso espacial para que el problema sea estable (condición CFL, ver [12]). La recurrencia anterior será la que resolvamos computacionalmente.

2.3. Adimensionalización de los parámetros

Para fines computacionales necesitamos llevar a cabo una adimensionalización de los parámetros como se indica en [2]. Para ello se utilizan una longitud L y un tiempo T característicos (a elegir según el problema que se quiera modelizar). Se establecen los cambios de variable $x=x'L$, $t=t'T$, $u=u'L$ y $\Omega=\Omega'L$, $R=R'L$ y $\Sigma=\Sigma'L$. Para adimensionalizar el parámetro ρ se divide la ecuación por una constante ρ_0. Haciendo los cambios de variables correspondientes y omitiendo el símbolo $'$ por simplificar la notación, obtenemos:

$$\begin{aligned}
&\frac{\rho}{\rho_0}u_{tt}-\operatorname{div}(\frac{\mu T^2}{\rho_0L^2}\nabla u)=\frac{T^2}{\rho_0L}f(tT)g(xL)=\tilde{f}(t)\tilde{g}(x), && x\in R, t>0,\\
&u(x,0)=0, u_t(x,0)=0, && x\in R,\\
&\frac{\partial u}{\partial n}=0, && \text{en } \partial R,
\end{aligned} \tag{2.14}$$

Los términos $\frac{\rho}{\rho_0}$ y $\frac{\mu T^2}{\rho_0L^2}$ se pueden seguir denotando como ρ y μ, respectivamente, por simplicidad. Sin embargo, ahora hay que tener en cuenta que estos parámetros representarán valores sin dimensiones, como pretendíamos.

2.4. Generación del dataset

La resolución del sistema anterior se he implementado en MATLAB haciendo uso de subrutinas diseñadas para la resolución de ecuaciones diferenciales. Como se menciona anteriormente, uno de los propósitos de esta fase del trabajo es generar un dataset sintético para explorar la resolución del problema inverso mediante técnicas de aprendizaje automático. El objetivo es generar varias soluciones para un dominio que incluya anomalías con distintas características. En el capítulo 4 se entrenarán diferentes tipos de redes neuronales con las soluciones generadas para aprender a caracterizar las anomalías, es decir, para aprender a resolver el problema inverso.

Se ha decidido generar anomalías elípticas, y los parámetros que caracterizan estas anomalías se han generado siguiendo patrones de distribuciones normales. A continuación se explica cada parámetro y la distribución que siguen sus valores[2].

- cx: Coordenada x del centro de la elipse. Los valores siguen una distribución normal con $\nu = 0{,}5$ y $\sigma = 1$.
- cy: Coordenada y del centro de la elipse. Los valores siguen una distribución normal con $\nu = 3{,}5$ y $\sigma = 1$.
- a: Longitud del semieje horizontal de la elipse. Los valores siguen una distribución normal con $\nu = 1$ y $\sigma = \sqrt{0{,}5}$ excluyendo los valores negativos.
- b: Longitud del semieje vertical de la elipse. Los valores siguen una distribución normal con $\nu = 0{,}4$ y $\sigma = \sqrt{0{,}5}$ excluyendo los valores negativos.
- θ: ángulo de inclinación (en radianes) de la elipse en el medio. Los valores siguen una distribución normal con $\nu = 0{,}2$ y $\sigma = \sqrt{0{,}1}$.

[2] ν va a hacer referencia a la media de la distribución normal. Se ha elegido esta notación para no confundir al lector con el término μ de la ecuación.

- ρ: Parámetro ρ de la anomalía. Los valores siguen una distribución normal con $\mu = 2{,}1$ y $\sigma = \sqrt{2}$ excluyendo valores negativos.

- μ: Parámetro μ de la anomalía. Los valores siguen una distribución normal con $\nu = 4{,}4$ y $\sigma = \sqrt{5}$ excluyendo valores negativos.

*Los valores de los parámetros ρ y μ se han elegido atendiendo a posibles casos reales de inclusiones de diferentes materiales en el subsuelo.

Se han considerado 4400 pasos temporales para generar un total de 20000 soluciones a partir de diferentes combinaciones de estos parámetros. Sin embargo, debido a que los parámetros se han generado de forma pseudo-aleatoria, alguna combinación de ellos ha dado lugar a una solución inestable para el paso de tiempo considerado. Tras descartar las soluciones inestables, el dataset se reduce a un total de 14094 soluciones.

Para la generación del dataset sintético, las soluciones se han calculado en un dominio bidimensional, aunque en la práctica las mediciones se suelen tomar en un extremo del dominio, como se muestra en la figura 1.1. Por esta razón, para la sintetización del dataset, se va a considerar la solución a lo largo de los 4400 pasos temporales en 17 puntos espaciales, simulando mediciones obtenidas por 17 receptores o sensores colocados en el extremo del medio, como se indica en la figura 1.1. En consecuencia, nuestros datos consistirán en una matriz de dimensiones 17×4400 (donde la primera dimensión es espacial, y la segunda temporal) y una combinación de parámetros asociada, la que se ha utilizado para generar la solución que dicha matriz representa. Es importante destacar que se ha introducido un ruido del 1 % a los datos de las matrices para simular las variaciones y errores que podrían presentarse en un caso real, donde los receptores tienen cierto margen de error en las mediciones. En resumen, nuestro dataset sintético consta de 14094 pares solución-parámetros, que nos permitirán entrenar y evaluar modelos de aprendizaje automático para resolver el problema inverso, es decir, para caracterizar y localizar las anomalías en el medio estudiado.

El capítulo 3 es un paréntesis teórico, en el que se explican en profundidad las técnicas

de aprendizaje automático que se van a utilizar más adelante en el trabajo para intentar resolver el problema inverso. Será por tanto en el capítulo 4, cuando se utilice este dataset para entrenar distintos modelos de aprendizaje automático con el objetivo de aprender a caracterizar las anomalías presentes en el medio.

Capítulo 3

Aprendizaje automático: Tipos de redes neuronales utilizadas

Tal y como se ha mencionado en secciones anteriores, el objetivo principal del trabajo es explorar el potencial de diversas técnicas de aprendizaje automático para resolver problemas inversos, o al menos para obtener aproximaciones suficientemente buenas de la solución. Con este fin, vamos a evaluar la viabilidad de aplicar diferentes tipos de redes neuronales.

3.1. Introducción a las redes neuronales

Antes de entrar en el detalle de los distintos tipos de redes neuronales que existen y el análisis de cuales de ellos pueden ser útiles para abordar este problema, vamos a estudiar brevemente los fundamentos matemáticos de las redes neuronales.

Las redes neuronales están compuestas por un conjunto de unidades básicas, llamadas neuronas, interconectadas entre sí. Estas neuronas se organizan en capas intentando imitar la estructura de una red neuronal biológica

Una neurona artificial, mejor conocida como perceptrón simple, es la unidad fundamental de las redes neuronales. Se puede representar como una función matemática que recibe un conjunto de entradas y produce una salida.

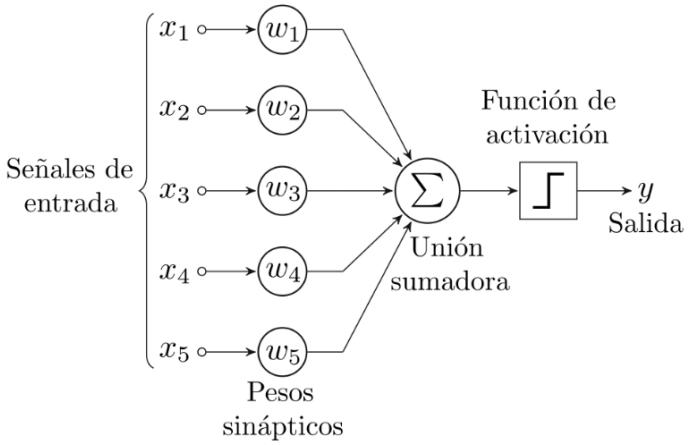

Figura 3.1: *Esquema general del perceptrón simple o de una neurona artificial. Figura de [19].*

El comportamiento general [18] de cada neurona i como perceptrón simple se puede determinar por:

$$y_i(t) = f_i(\sigma_j(w_{ij}, x_j(t))) \tag{3.1}$$

donde:

- x_j son las variables de entrada que recibe la neurona.
- w_{ij}[1] son los pesos sinápticos (o ponderaciones) asociados a cada una de las entradas.
- σ es la regla de propagación, función que toma las entradas ponderadas por los pesos sinápticos y produce una salida denominada potencial post-sináptico. $h_i(t) = \sigma(w_{ij}, x_j(t))$. En la práctica suele consistir en el sumatorio $h_i(t) = \sum_j (w_{ij}, x_j(t))$, donde se ha tomado $\sigma_j = \sum_j$.
- f_i es la función de activación, función que toma el potencial post-sináptico y produce la salida de la neurona $a_i(t) = f_i(h_i(t))$. La función de activación es uno de los elementos principales que distinguen las capas de una red neuronal[2].

[1] Aunque estemos describiendo un perceptrón simple (una sola neurona), para la notación vamos a considerar que nos estamos refiriendo a la neurona i.

[2] Todas las neuronas de una misma capa tienen la misma función de activación.

3.1.1. Tipos de función de activación

Existen multitud de tipos de funciones de activación (véase [16]) por lo que solamente se van a formular aquellas que se utilizan más adelante en el trabajo. En general, las funciones de activación suelen ser funciones simples:

$$\text{ReLU} : \begin{cases} 0 & si\ x < 0 \\ x & si\ x \geq 0 \end{cases} \quad \textit{Leaky}\ \text{ReLU} : \begin{cases} 0{,}1x & si\ x < 0 \\ x & si\ x \geq 0 \end{cases} \quad \text{ELU} : \begin{cases} \alpha(\exp x - 1) & si\ x < 0 \\ x & si\ x \geq 0 \end{cases}$$

donde α es un parámetro a elegir, en este trabajo se utilizará $\alpha = 1$.

Una vez aclarado el funcionamiento de un perceptrón, procedemos a explicar como una red neuronal aprende a modelar una solución para un problema específicos. El objetivo del entrenamiento de una red neuronal es ajustar los pesos sinápticos para que la red aprenda a realizar una tarea específica. Como podemos observar en la figura 3.1, una neurona no puede modelar casos demasiados complejos, pero una combinación de varias neuronas apiladas por capas puede aprender patrones más complicados. En el contexto del aprendizaje automático supervisado, se proporciona a la red neuronal un conjunto de datos con entradas (X) y salidas deseadas (y). La red utiliza estos datos para aprender la relación entre las entradas y las salidas, ajustando los pesos sinápticos mediante un algoritmo llamado retropropagación. Antes de profundizar en este algoritmo, es crucial entender cómo una red neuronal transforma las entradas para producir una salida y cómo se calcula el error que comete en el proceso.

3.1.2. Funciones de coste

Siguiendo el mecanismo, una vez la red neuronal recibe una entrada, propaga estos valores hacia delante (*forward propagation*), calculando las entradas y salidas de cada capa de neuronas según sus pesos y funciones de activación, para generar unos valores de salida. Una vez se obtiene la salida, se calcula el coste o error con respecto a la salida esperada utilizando una función de coste. Esta función de coste va a ser la que se intente minimizar en el entrenamiento de la red neuronal.

Tipos de funciones de coste

En función de si las predicciones de la red neuronal son valores continuos o discretos (en problemas de clasificación se obtienen valores categóricos o booleanos, mientras que en problemas de regresión obtendremos valores continuos) y como queramos entrenar el modelo, se pueden utilizar diferentes funciones de coste. A continuación explicamos algunas de las funciones de coste más utilizadas para problemas de regresión como el que nos incumbe [6], y los efectos que tiene cada una durante el entrenamiento. Para las fórmulas consideraremos:

- N: Número total de datos de entrenamiento
- y_i: salida esperada
- $\hat{y}_i$: predicción

***Mean Absolute Error* (MAE)**

Calcula la distancia Manhattan (o norma L^1) entre las predicciones y las salidas esperadas:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

***Mean Squared Error* (MSE)**

Calcula la distancia Euclidiana (o norma L^2) entre entre las predicciones y las salidas esperadas:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

El cuadrado de los errores va a provocar que se penalice en mayor medida errores grandes en el cálculo de la función de coste. Puede ser interesante utilizar esta función de coste durante el entrenamiento si queremos evitar errores grandes en nuestras predicciones, a costa de sacrificar un ajuste detallado la red en casos en los que cometa errores más pequeños.

***Root Mean Squared Error* (RMSE)**

Calcula la raíz cuadrada del MSE:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

No existe gran diferencia entre entrenar un modelo con función de coste MSE o RMSE. Sin embargo, hacer la raíz cuadrada de la media de los errores al cuadrado devuelve un valor en las unidades originales de los datos, lo que puede ser útil en términos de interpretabilidad del error del modelo.

Aunque la función de coste elegida sea la que se va a tener en cuenta durante el entrenamiento para el cálculo del error y el posterior ajuste de los pesos mediante retropropagación, es una práctica común monitorizar varias métricas.

3.1.3. Retropropagación

Una vez se ha calculado el error del modelo, se resuelve un problema de optimización mediante un algoritmo llamado retropropagación, a través del cual se ajustan los pesos de la red neuronal. No vamos a entrar en el detalle del algoritmo de retropropagación que se explica en detalle en [13], pero sí que vamos a exponer los fundamentos básicos. Se trata de una técnica que se basa en el descenso de gradiente y sigue los siguiente pasos:

- Cálculo del gradiente del error: Se calcula el gradiente del error con respecto a cada peso de la red utilizando la regla de la cadena.

- Actualización de pesos: Los pesos se ajustan en la dirección opuesta al gradiente del error para minimizar la función de coste. Esto se realiza mediante la siguiente regla de actualización de pesos

  $$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}},$$

  donde w_{ij} es el peso j de la neurona i, η es la tasas de aprendizaje, y $\frac{\partial E}{\partial w_{ij}}$ representa el gradiente. La tasa de aprendizaje es un hiperparámetro muy importante en el entrenamiento de redes neuronales ya que va a determinar la "velocidad" con la que los

pesos se ajustan y convergen a la solución. Una tasa de aprendizaje alta puede acelerar la velocidad a la que converge el algoritmo, pero también puede provocar que el algoritmo oscile alrededor de una solución óptima. Por otro lado, una tasa de aprendizaje reducida provoca una convergencia lenta del algoritmo, pero también puede provocar que este se atasque en soluciones sub-óptimas.

El entrenamiento de una red neuronal consiste en un proceso iterativo de predicción y ajuste mediante retropropagación, en el que cada iteración sobre un conjunto de datos de entrenamiento se denomina época. El objetivo del entrenamiento es iterar el número de épocas necesario para que los pesos converjan a una solución aceptable para la predicción.

3.2. Tipos de redes neuronales

Una red neuronal se puede configurar de varias maneras en función del número de capas, las funciones de activación elegidas, la manera en la que están conectadas las neuronas en cada capa y muchos otros hiperparámetros. Estas configuraciones dan lugar a distintos tipos de arquitecturas de red neuronal. A continuación, se introducen algunas de las arquitecturas con las que se explora la resolución del problema inverso.

3.2.1. *Feedforward*

Las redes unidireccionales son el tipo más simple de red neuronal. En ellas, la información fluye en una sola dirección desde la capa de entrada hasta la capa de salida. Se caracterizan por estar formadas principalmente por capas densas.

- Capa densa: Cada neurona en una capa densa está conectada con todas las neuronas de la siguiente capa.

Las redes unidireccionales se pueden utilizar para casuísticas muy variadas de regresión y clasificación. Sin embargo, si los datos de entrada son matriciales o secuenciales, es posible que la arquitectura no sea lo suficientemente compleja como para capturar una relación espacial o temporal entre los datos.

3.2.2. Redes Neuronales Convolucionales (CNN)

Las redes convolucionales (CNN) son un tipo de red neuronal que se utilizan para procesar datos espaciales, como matrices. Se utilizan principalmente para el análisis de imágenes, entendiendo estas como matrices de píxeles. Este tipo de redes suelen estar formadas por bloques convolucionales que utilizan capas convolucionales y de agrupación como se explica en [11]. El propósito de estos bloques es extraer características espaciales significativas, tales como bordes, texturas y formas, directamente de los datos de entrada:

- Capa convolucional: Aplican filtros a los datos de entrada para extraer características.
- Capa de agrupamiento o *pooling*: Reducen la dimensionalidad de los datos de entrada.

Aunque una red convolucional esta caracterizada por tener capas convolucionales, suelen tener capas densas que aprenden a relacionar las características extraídas por las capas convolucionales con la salida.

En nuestro caso, los datos de entrada son matriciales y además existe una relación espacial en una dimensión y temporal en otra dimensión. Ello hace que las redes neuronales convolucionales puedan ser mejores candidatas que las redes neuronales densas *feedforward* para poder aproximar la solución del problema inverso de una manera razonable.

Operación de convolución

En esencia, la operación de convolución que caracteriza estas redes, consiste en el desplazamiento de un filtro (o kernel) a través de una matriz, calculando la suma del producto elemento a elemento del filtro y la sección de la matriz original sobre la que se encuentra el filtro. El kernel suele ser una matriz pequeña, normalmente de tamaños 3×3 o 5×5, que se ajusta para detectar patrones en la matriz. La operación de convolución se podría describir matemáticamente como [7]

$$S(i,j) = (I * K)(i,j) = \sum_{m,n} I(i+m, j+n)K(m,n), \tag{3.2}$$

donde:

- $S(i, j)$ es el mapa de características extraídas.
- I es la matriz de entrada.
- K es el kernel
- i, j son las coordenadas del mapa de características.
- m, n son las coordenadas del kernel.
- $*$ denota la operación de convolución.

Esta ecuación nos muestra como cada elemento $S(i, j)$ del mapa de características extraídas es la suma del producto elemento a elemento del kernel K y la sección de la matriz de entrada I sobre la cual está posicionado actualmente el filtro.

La matriz resultante representa un mapa que muestra la intensidad presente en cada posición de la característica que el kernel ha sido entrenado para detectar.

En la figura 3.2 se muestra un ejemplo de la operación de convolución aplicada a una matriz 4×4 con un kernel 3×3.

Una consideración importante a tener en cuenta, son las dimensiones del mapa de características de salida. Estas dimensiones van a depender del tamaño del kernel pero también de otros parámetros importantes en la convolución como el *stride* y *padding*, que se definen a continuación.

Stride

El *stride* indica el número de posiciones que se desplaza el kernel en cada paso (se aplica tanto horizontalmente como verticalmente). Se trata de un parámetro que influye en gran medida en la dimensión del mapa de características de salida. Un *stride* de 1 da lugar a un mapa de características con una dimensión muy similar (dependiendo del *padding*) a la matriz de entrada, ya que se aplica el kernel a cada posición de la matriz de entrada (excepto a los bordes en función del *padding*). Un *stride* más grande da lugar a un mapa de

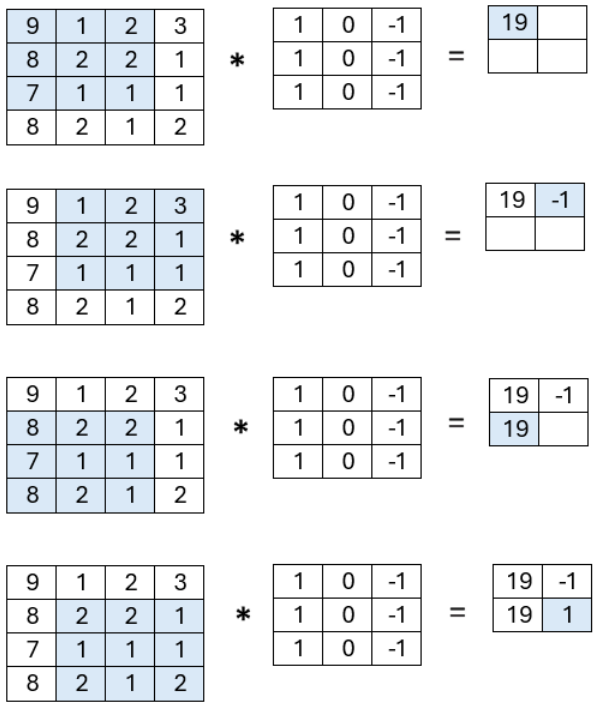

Figura 3.2: *Ejemplo de convolución con un kernel 3 × 3 aplicado a una matriz 4×4.Este ejemplo es representativo de un kernel que detecta bordes verticales. Si la matriz representase una imagen de grises, el kernel serviría para identificar cambios bruscos de intensidad de los píxeles en ejes verticales. Se puede ver como la primera columna de la matriz original tiene valores altos, mientras que el resto tiene valores bajos. Por tanto, la matriz de características extraídas tiene valores altos en la primera columna y valores cercanos a 0 en la segunda.*

características más pequeño ya que el kernel recorre la matriz de entrada en menos pasos, habiendo menos solapamiento entre las posiciones a las que se aplica el filtro.

Por tanto, un *stride* pequeño conserva en mayor medida la dimensionalidad de la matriz original en la salida, y permite una granularidad más fina a la hora de extraer características. Por el contrario, un *stride* mayor permite una cobertura más amplia y rápida de la entrada a costa del detalle del mapa extraído.

El ejemplo de la figura 3.2 utiliza un *stride* de 1. Es decir, el kernel se desplaza una posición en cada paso.

Padding

Como podemos observar, en la figura 3.2, la dimensión del mapa extraído es $(M-m+1)$ x $(N-n+1)$ donde $M \times N$ son la dimensiones de la matriz de entrada y $m \times n$ son las dimensiones del kernel.

Esto es debido a que el kernel no se puede aplicar a los bordes de las matrices porque algunas posiciones del kernel no se encontrarían sobre la matriz. Esto no tiene porque ser un problema, pero en caso de que interese aplicar el kernel a los bordes de la matriz, se introduce el concepto de *padding*. El *padding* consiste en añadir ceros (u otro valor por defecto) alrededor del borde de la matriz, para poder aplicar el kernel a los bordes originales.

La siguiente fórmula da la dimensión del mapa de características de salida en función de la dimensión del kernel, del *stride* y del *padding* [7] es

$$W_{out} = \frac{W_{in} + 2P - F}{S} + 1$$
$$H_{out} = \frac{H_{in} + 2P - F}{S} + 1,$$

donde:

- W_{out} y H_{out} son la anchura y altura del mapa de características extraídas, respectivamente.
- W_{in} y H_{in} son la anchura y altura de la matriz de entrada, respectivamente.
- F es el tamaño del kernel (Entendiendo que tiene una dimensión $F \times F$)
- S es el stride.
- P es la cantidad de padding añadido en cada borde de la matriz de entrada.

Profundidad

Aunque en el ejemplo se este aplicando un filtro o kernel sobre la matriz, en la práctica, las capas convolucionales suelen aprender y aplicar varios filtros, lo que da lugar a un mapa de características en 3 dimensiones. La tercera dimensión o profundidad representa el número de filtros aplicados. De esta manera, cada capa puede aprender a identificar un mayor tipo de características en la entrada.

Un mayor número de filtros implica una mayor capacidad de la red para aprender características complejas. Sin embargo, esto aumenta considerablemente el coste computacional de la red y la cantidad necesaria de datos de entrenamiento para su correcto aprendizaje.

Una estrategia común suele ser colocar un menor número de filtros en la entrada con el objetivo de que aprendan a identificar características más generales, mientras que en las capas más lejanas se suelen disponer más filtros que aprendan a identificar características de más alto nivel en los datos.

Pooling

En la sección anterior se ha señalado como aplicar una capa convolucional a una matriz puede resultar en un mapa de características de una alta dimensionalidad. Las capas de *pooling* son cruciales para reducir la dimensionalidad de la salida de las capas convolucionales, minimizando el sobreentrenamiento y reduciendo el coste computacional de la red neuronal. En esencia, se trata de una simplificación del mapa de características extraído en la capa convolucional que permite a la red mantener robustez ante pequeños cambios en la matriz origen y centrarse en las características más importantes extraídas.

La operación *pooling* consiste en coger un subconjunto de los datos de entrada y transformarlos en una sola salida. Esta operación aplicada a varias secciones de la matriz de entrada, nos da como resultado otra matriz o vector de menor dimensionalidad.

Para explicar los distintos tipos de *pooling* que podemos aplicar, vamos a tener en cuenta las siguientes consideraciones:

- Consideramos F como el mapa de características extraídas por la capa convolucional.
- Consideramos una ventana de tamaño $n \times n$.
- (i, j) representan las coordenadas a las que estamos aplicando la operación.
- s es el *stride* de la ventana, es decir, el desplazamiento de la ventana después de aplicar la operación.
- a, b son variables para iterar sobre las dimensiones de la ventana.

Teniendo en cuenta la notación anterior, definimos los tipos de *pooling* más comunes [7]:

- *Max Pooling*: El valor máximo de un conjunto de valores es seleccionado y propagado a la siguiente capa.

$$P_{max}(i, j) = \max_{a=0}^{n-1} \max_{b=0}^{n-1} F(i \cdot s + a, j \cdot s + b)$$

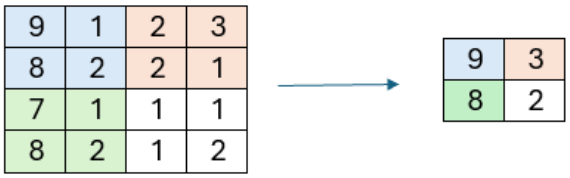

Figura 3.3: *Ejemplo de max pooling en una matriz 4 × 4 con una ventana 2 × 2 y un stride de 2.*

- *Average Pooling*: Propaga a la siguiente capa la media del conjunto de valores considerado.

$$P_{avg}(i, j) = \frac{1}{n^2} \sum_{a=0}^{n-1} \sum_{b=0}^{n-1} F(i \cdot s + a, j \cdot s + b)$$

- *Global Pooling*: Todo el mapa de características extraídas es reducido a un solo valor estadístico, como puede ser el máximo o la media.

$$\begin{aligned} P_{gmax} &= \max_{a=0}^{M-1} \max_{b=0}^{N-1} F(i \cdot s + a, j \cdot s + b) \\ P_{gavg} &= \frac{1}{MN} \sum_{a=0}^{M-1} \sum_{b=0}^{N-1} F(i \cdot s + a, j \cdot s + b) \end{aligned} \tag{3.3}$$

  Los últimos tipos de agrupamiento (3.3) reducen cada campo de características a un solo valor, lo que suele ser particularmente útil para conectar los bloques convolucionales con una sección de capas densas para clasificación o regresión.

3.2.3. Redes Neuronales Recurrentes (RNN)

Las redes neuronales recurrentes (RNN) son un tipo de modelos utilizados para procesar datos secuenciales, como texto o series temporales, es decir, datos con una relación temporal entre sí [14]. Estas redes utilizan capas recurrentes para aprender dependencias a corto y a largo plazo de las entradas.

- Capa recurrente: Procesan los datos de entrada de forma secuencial. Existen distintos tipos de neuronas recurrentes como las celdas *SimpleRNN*, LSTM o GRU. En nuestro caso vamos a utilizar las segundas para implementar la red neuronal recurrente.

  - *Long-Short Term Memory* (LSTM): Neuronas o celdas capaces de aprender dependencias a corto y largo plazo entre los datos de entrada.

Al igual que las redes convolucionales, las redes recurrentes también suelen tener capas densas que aprenden a relacionar la salida de las capas recurrentes con la salida del modelo.

Nuestros datos matriciales son en realidad secuencias de vectores de dimensión 17, es decir, son datos secuenciales. Ello hace que las redes recurrentes puedan ser también mejores candidatas a la hora de intentar dar una solución razonablemente aproximada del problema inverso, que una red con solamente capas densas.

SimpleRNN

Antes de explicar la celda LSTM en profundidad vamos a definir las celdas *SimpleRNN* ya que facilitarán la comprensión de la primera.

Un red recurrente toma como entrada una secuencia de datos, es decir, un conjunto de datos con un orden temporal específico. En este sentido, una celda *SimpleRNN* es la versión más simple de una neurona con capacidad de aprender dependencias temporales en una secuencia de datos.

Este tipo de neuronas toman como entrada el valor del instante temporal actual y el estado generado en en el instante temporal anterior.

Las ecuaciones matemáticas correspondientes al cálculo del estado y la salida en cada instante temporal t son [5]:

$$\begin{aligned} h_t &= \sigma(W_x x_t + W_h h_{t-1} + b_h) \\ y_t &= \sigma(W_y h_t + b_y) \end{aligned} \tag{3.4}$$

donde:

- x es la secuencia de valores de entrada.
- W_x son los pesos o ponderaciones de las entradas.
- h_t y h_{t-1} son el estado actual y previo respectivamente.
- y es la secuencia de salida.
- W_y son los pesos o ponderaciones aplicados al estado para producir la salida.
- b_h y b_y son los *biases* o sesgos.

*h_{-1} se inicializa a un vector nulo.

Es importante observar que el único "mecanismo de memoria" que tiene una celda *SimpleRNN* es el estado h. Este estado es modificado en cada paso temporal por lo que la influencia que tiene este en estados posteriores se va diluyendo con el tiempo. Esto hace

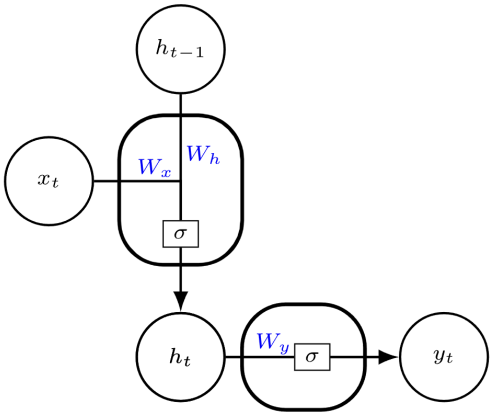

Figura 3.4: *Diagrama de un paso temporal para una celda SimpleRNN. Figura de [5].*

que las celdas *SimpleRNN* no sean aptas para aprender dependencias temporales a largo plazo en los datos de entrada. Para solucionar este problema, como se indica en el artículo intriductorio sobre redes recurrentes [14], se introducen las celdas LSTM.

LSTM

Las celdas LSTM son más complejas, y añaden una variable c, denominada variable de celda, que representa información almacenada para dotar de "memoria a largo plazo" a la celda.

Las ecuaciones que rigen el comportamiento de una celda LSTM son las siguientes [5]:

$$
\begin{aligned}
i_t &= \sigma(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \\
f_t &= \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \\
\tilde{c}_t &= \tanh(W_{cx}x_t + W_{ch}h_{t-1} + b_c) \\
o_t &= \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o)
\end{aligned}
\tag{3.5}
$$

$$
\begin{aligned}
c_t &= f_t c_{t-1} + i_t \tilde{c}_t \\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$

* c_{-1} y h_{-1} se inicializan a vectores nulos.

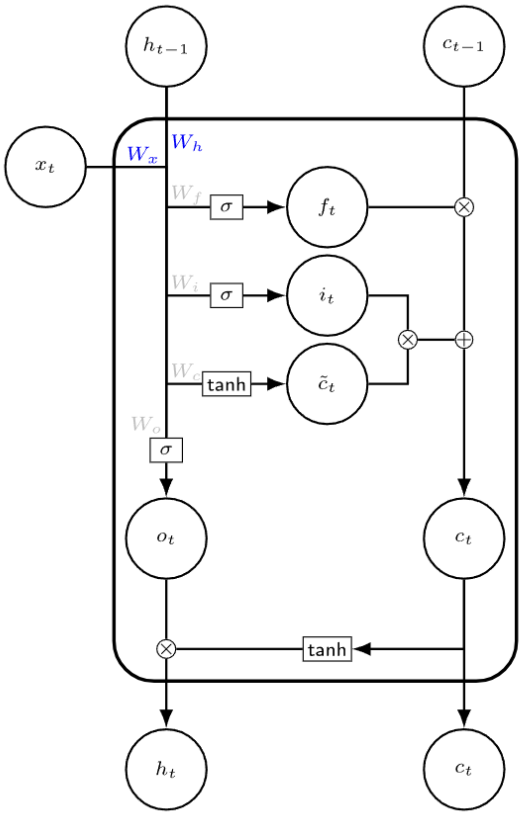

Figura 3.5: *Diagrama de un paso temporal para una celda LSTM. Figura de [5].*

donde:

- f_t controla cuanta información de c_{t-1} se conserva.
- $\tilde{c}_t$ representa la nueva información a guardar.
- i_t son pesos que controlan cuanta información de $\tilde{c}_t$ se conserva.
- La combinación de las variables anteriores da lugar a c_t, la variable de la celda.
- o_t controla cuanta información de $\tanh(c_t)$ se guarda en h_t.
- Las variables que comienzan por W son matrices con pesos que la celda ajusta para aprender dependencias entre los datos.

Como se puede observar, la introducción de la variable de celda c permite crear un mecanismo a través del cual la celda puede guardar información a más largo plazo que una celda *SimpleRNN*. Junto con el estado permite a las celdas LSTM aprender dependencias a corto y a largo plazo de datos de entrada secuenciales, de ahí el nombre *Long-Short Term Memory*.

3.2.4. Capas generales

Existen tipos de capas que se utilizan en varios tipos de arquitecturas de redes neuronales y que por tanto, no están ligadas a un tipo particular de red neuronal. La más común son las ya explicadas capas densas, que suelen ser aplicadas a todos los tipos de red neuronal. Existe una amplia variedad de tipos, por lo que a continuación se explicarán únicamente las que se vayan a utilizar más adelante.

Dropout

Las capas *dropout* suelen situarse después de las capas densas, y su función es la de desactivar aleatoriamente una proporción de las neuronas de una capa durante el entrenamiento. Es una técnica de regularización simple, utilizada para que el modelo aprenda patrones sin utilizar todas las neuronas. En cada iteración del aprendizaje se utiliza la misma proporción de neuronas pero siendo la elección de estas diferente. De esta manera, al no haberse entrenado en todos los casos con las mismas neuronas, se evita el sobreajuste del modelo y se logra una mayor generalización a la hora de resolver el problema. Los valores óptimos en cuanto a proporción de neuronas desactivadas utilizando capas *dropout* se encuentran entre el 20 % y el 50 % según el artículo [17] que introduce esta técnica.

Flatten

Esta capa permite utilizar una técnica que consiste en aplanar cualquier estructura de datos multidimensional, en una estructura unidimensional o vector, cuyo tamaño es el producto del tamaño de cada dimensión del objeto original. De esta manera, esta capa transformaría la matriz $\begin{bmatrix} 9 & 3 \\ 8 & 2 \end{bmatrix}$ en el vector $\begin{bmatrix} 9 & 3 & 8 & 2 \end{bmatrix}$. Este tipo de capas tienen una función similar a las capas de agrupación globales (3.3), se suelen utilizar para aplanar estructuras de datos antes de transmitir los datos a una capa densa que solo puede tomar vectores como entrada.

Una vez explicada la teoría matemática que fundamenta los tipos de redes neuronales que se van a utilizar, en el capítulo 4 se procede a su implementación.

Capítulo 4

Resolución del problema inverso: Implementación de redes neuronales

En este capítulo se estudia la viabilidad de resolver el problema inverso mediante aprendizaje automático. Se van a implementar los tres tipos de redes neuronales mencionados en el capítulo 3 y se van a analizar los resultados de cada una de ellas en cuanto a aproximación de la solución y coste computacional. Para ello, se van a utilizar las implementaciones de *Keras*[1] tanto de los tipos de capas mencionadas como de los métodos necesarios para la creación y entrenamiento de redes neuronales. Se programará tanto la creación de los modelos como su entrenamiento y análisis utilizando *Python* en *notebooks* de *Google Collab*. Todo el código desarrollado para la implementación y entrenamiento de los modelos se puede encontrar en el siguiente repositorio de *Github*: *Anomaly detection*.

4.1. Dataset

En esta sección vamos a visualizar el conjunto de datos con los que vamos a entrenar y probar los diferentes modelos. En la figura 4.1 se muestra la distribución de los valores de cada parámetro a lo largo de todo el dataset. No vamos a entrar en el detalle de las distribuciones de los parámetros ya que en la sección 2 se mencionan las distribuciones normales que estos siguen.

[1] *Keras* [3] es un entorno del lenguaje de programación de *Python* creado para implementar modelos de aprendizaje automático.

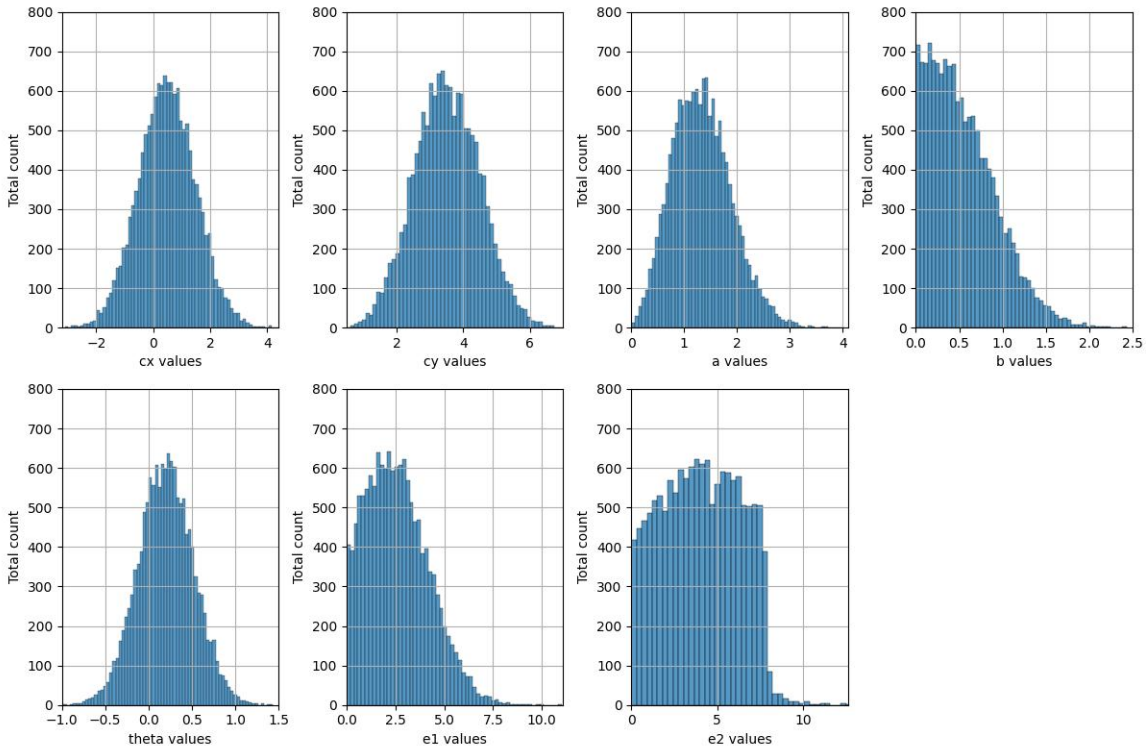

Figura 4.1: *Distribución de los valores de cada parámetro en el dataset.*

En la figura 4.2 se pueden apreciar algunos ejemplos de las matrices de datos visualizadas por colores. Colores más claros representan valores más altos en la matriz que representa la solución de la ecuación, mientras que colores más oscuros representan valores más bajos.

4.2. Preparación de los datos

Como se ha mencionado en varias ocasiones, nuestros datos están compuestos por pares de matrices y parámetros. Para nuestro modelo, la entrada son las matrices, y la salida que tiene que aprender a predecir son los parámetros que caracterizan la anomalía elíptica. Nuestro conjunto de datos está formado por 14094 pares de entrada-salida y una práctica esencial a la hora de estudiar si un modelo de aprendizaje automático es útil o no, es dividir estos datos en conjuntos de entrenamiento, validación y test:

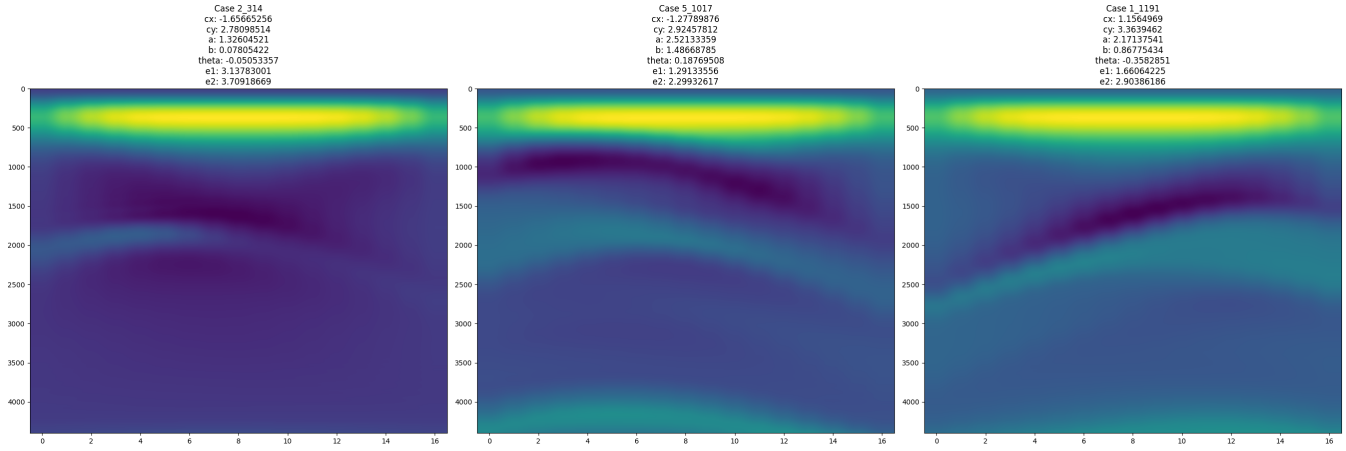

Figura 4.2: *Ejemplos de matrices de datos con sus parámetros correspondientes.*

- Entrenamiento: Se trata del conjunto de datos que el modelo utiliza para ajustar sus pesos. Se suele dedicar entre un 60 % y un 90 % del dataset total al conjunto de entrenamiento.

- Validación: Se utiliza para medir la función de coste en un conjunto que no sea sobre el que se está entrenando. Cuando la función de coste en el conjunto de validación deja de mejorar, pero sigue haciéndolo en el conjunto de entrenamiento, es indicativo de que el modelo se está ajustando demasiado al conjunto de entrenamiento. En nuestro caso el conjunto de validación se va a utilizar para dos propósitos:

  - Durante la fase de exploración de los espacios de hiperparámetros, se va a utilizar la función de coste en el conjunto de validación como métrica para elegir la mejor configuración.
  - Por otro lado, en el entrenamiento final, se va a utilizar para tomar la versión del modelo que mejor valor de la función de coste haya obtenido en validación. Evitamos así tomar una versión que se haya podido sobreajustar demasiado al conjunto de entrenamiento.

  Suele estar formado por la mitad de los datos que no pertenecen al conjunto de entrenamiento, es decir, entre un 5 % y un 20 % del conjunto total.

- Test: Se trata de un conjunto de datos que se aisla de la fase de entrenamiento. Se utiliza para evaluar el modelo final con unos datos que el modelo no ha visto durante el entrenamiento. Suele estar formado por la mitad restante de los datos que no pertenecen al conjunto de entrenamiento, es decir, entre un 5 % y un 20 % del conjunto total.

En nuestro caso, debido a la elevada cantidad de datos de nuestro dataset, no es necesario destinar una gran proporción de estos a los conjuntos de validación y test, ya que con un porcentaje relativamente pequeño del dataset cuentan con suficientes datos. Por ello utilizaremos un 10 % para el conjunto de validación y otro 10 % para el conjunto de test, dejando el 80 % restante al conjunto de entrenamiento.

Al realizar la división en los tres conjuntos, es importante mantener en la medida de lo posible las distribuciones mostradas en la figura 4.1. Para ello, la división se ha llevado a cabo utilizando *MultilabelStratifiedKFold* [15] de la librería *iterative-stratification*. Esta clase está diseñada para separar conjuntos con más de una variable de salida en k particiones, manteniendo la distribución de cada variable de salida, o en nuestro caso, de cada parámetro. Para ello se han generado 10 particiones del conjunto de pares matrices-parámetros y se han tomado 8 para el conjunto de entrenamiento, 1 para el conjunto de validación y la restante para el conjunto de test.

En las tablas 4.1, 4.2 y 4.3 se puede apreciar la media, desviación, mínimos y máximos de cada uno de los parámetros para cada conjunto generado. En ellas se puede apreciar como se mantienen las distribuciones de los parámetros con respecto a las originales mostradas en la sección 2.

4.3. Modelos y entrenamiento

Esta sección detalla la construcción de cada una de las arquitecturas con las que se ha experimentado, así como el proceso de entrenamiento que se ha seguido para cada una. Se empleará la métrica MSE como función de coste y además se monitorizará el RMSE

Parámetro	Media	Desviación típica	Mínimo	Máximo
$\boldsymbol{cx}$	0.485	1.000	-3.102	4.139
$\boldsymbol{cy}$	3.518	0.978	0.276	6.765
$\boldsymbol{a}$	1.333	0.559	0.015	3.852
$\boldsymbol{b}$	0.544	0.385	0.000	2.429
θ	0.199	0.319	-1.009	1.436
ρ	2.603	1.597	0.000	11.031
μ	4.056	2.245	0.001	13.713

Tabla 4.1: *Distribución de los parámetros en el conjunto de entrenamiento formado por 11275 instancias.*

Parámetro	Media	Desviación típica	Mínimo	Máximo
$\boldsymbol{cx}$	0.505	0.991	-2.680	4.378
$\boldsymbol{cy}$	3.523	0.992	0.829	7.287
$\boldsymbol{a}$	1.360	0.564	0.061	4.033
$\boldsymbol{b}$	0.559	0.384	0.000	2.126
θ	0.199	0.317	-0.807	1.229
ρ	2.685	1.565	0.005	8.356
μ	4.164	2.241	0.004	11.215

Tabla 4.2: *Distribución de los parámetros en el conjunto de validación formado por 1409 instancias.*

y el MAE en todos los casos. Es importante remarcar que al tratarse de un problema de regresión múltiple, ya que implica la predicción de 7 parámetros distintos, la métrica general del modelo es el resultado de calcular la media aritmética de los errores en las predicciones de cada parámetro.

4.3.1. Ajuste de hiperparámetros mediante el algoritmo *hyperband*

Antes de entrenar los diferentes modelos con todos los datos durante un número de épocas extenso, vamos a hacer una elección de hiperparámetros de cada tipo de modelo utilizando una versión reducida de los datos.

El objetivo del trabajo no es ajustar una red neuronal de la mejor manera posible para resolver el problema inverso, sino estudiar la viabilidad de diferentes aproximaciones de

Parámetro	Media	Desviación típica	Mínimo	Máximo
$\boldsymbol{cx}$	0.529	1.012	-2.568	3.480
$\boldsymbol{cy}$	3.497	0.982	0.753	6.305
$\boldsymbol{a}$	1.345	0.543	0.041	3.122
$\boldsymbol{b}$	0.548	0.391	0.001	2.397
θ	0.207	0.311	-0.886	1.199
ρ	2.586	1.539	0.002	9.010
μ	4.081	2.240	0.001	10.236

Tabla 4.3: *Distribución de los parámetros en el conjunto de test formado por 1409 instancias.*

redes neuronales para abordar este problema. Aún así, haremos una breve exploración del espacio de hiperparámetros y de las distintas configuraciones de arquitecturas que pueden tener cada tipo de red con la que se va a trabajar. Esto permitirá ejecutar el experimento final con una configuración decente en cada caso, y así poder llevar a cabo una comparación de resultados más relevante.

Dado que el conjunto de entrenamiento y de validación tiene en total 12648 matrices 4400 x 17, el entrenamiento de los modelos puede tener un elevado coste computacional. Para evitar este problema, se van a tener en cuenta las siguientes consideraciones para la exploración del espacio de hiperparámetros:

- Reducción del tamaño de las matrices: Se va a tomar una fila de cada 10 reduciendo el tamaño de las matrices de datos a 440 x 17. Esto permitirá reducir el tiempo de entrenamiento durante la exploración considerablemente.

- Exploración del espacio de hiperparámetros mediante el algoritmo *hyperband*: Este algoritmo permite explorar el espacio de hiperparámetros de manera eficiente y encontrar configuraciones de hiperparámetros con buen rendimiento.

Algoritmo hyperband

El algoritmo *hyperband* [8] se presenta como una alternativa a la exploración del espacio de hiperparámetros de métodos que utilizan optimización Bayesiana para la elección de

Algorithm 1: Algoritmo Hyperband para la optimización de hiperparámetros.

input : R, η (default $\eta = 3$)
initialization: $s_{\text{máx}} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{\text{máx}} + 1)R$
for $s \in \{s_{\text{máx}}, s_{\text{máx}} - 1, \ldots, 0\}$ **do**
  $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil$, $r = R\eta^{-s}$
  `// begin with` (n, r) `inner loop`
  $T =$`get_hyperparameter_configuration`(n)
  for $i \in \{0, \ldots, s\}$ **do**
    $n_i = \lfloor n\eta^{-i} \rfloor$
    $r_i = r\eta^i$
    $L = \{$`run_then_return_val_loss`$(t, r_i) : t \in T\}$
    $T =$`top_k`$(T, L, \lfloor n_i/\eta \rfloor)$
  end
end
return *Configuration with the smallest intermediate loss seen so far.*

configuraciones de forma adaptativa. El algoritmo pretende acelerar la búsqueda aleatoria mediante la asignación adaptativa de recursos y *early-stopping*[2].

Este algoritmo se basa en *halving* sucesivo[3] o divisiones sucesivas. Esto, consiste en entrenar un número n_0 de configuraciones[4] para un número de épocas reducidas, tras las cuales, solo $\frac{1}{\eta}$ configuraciones seguirán siendo entrenadas. Este proceso se sigue sucesivamente hasta entrenar durante el número de épocas máximo R solamente a los modelos que han pasado todos los cortes. Este proceso permite explorar varias configuraciones sin la necesidad de entrenar todas las configuraciones durante el número máximo de épocas. A continuación se analiza como se lleva a cabo este proceso en el algoritmo 1 tomado directamente del artículo [8].

[2] *early-stopping* es una técnica que se utiliza en aprendizaje automático para detener el entrenamiento de un modelo atendiendo a una serie de condiciones. Se suele utilizar para dejar de entrenar un modelo cuando la función de coste en validación no mejora evitando así el sobreentrenamiento.

[3] La palabra *halving* se traduce como "dividir entre 2". A pesar de ello lo utilizaremos de forma genérica para referirnos a "dividir entre η", por lo que nos referiremos a ello como división.

[4] Con el objetivo de facilitar la comprensión del texto en esta sección, vamos a referirnos a entrenar un modelo con una configuración o combinación de parámetros concreta como: "entrenar una configuración".

En primer lugar explicamos los parámetros de entrada del algoritmo:

- R: Parámetro de entrada que va a determinar la cantidad de recursos (épocas) con los que se va a entrenar una configuración como máximo. Como se ha mencionado, se trata de un método de exploración basado en la asignación adaptativa de recursos, por lo que el valor de R va determinar el coste computacional del algoritmo.

- η: Parámetro de entrada del algoritmo que determina el factor por el que se reduce el número de configuraciones cuando se utilizan divisiones sucesivas.

Aparte de los parámetros de entrada del algoritmo, existen otras variables intermedias y métodos auxiliares cuya explicación puede facilitar en gran medida la comprensión de este algoritmo:

- $s_{\text{máx}}$: Cada iteración del bucle externo vamos a denominarlo "banda" y su número de iteraciones viene dado por $s_{\text{máx}} + 1$. $s_{\text{máx}} + 1$ también va a determinar el número máximo de divisiones sucesivas que se van a llevar a cabo en una banda.

- B: Podríamos denominarlo como *budget* o presupuesto asignado para cada banda.

- n: Determina el número inicial de configuraciones aleatorias que se van a tomar en cada banda. n_i es, por tanto, el número de configuraciones que "sobreviven" la división en la iteración i de una banda concreta.

- r: Determina el número de recursos (épocas) que se van a destinar inicialmente al entrenamiento de cada configuración. r_i es, por tanto, el número de recursos que se van a asignar a las configuraciones que sobreviven a la iteración i de una banda en concreto.

- T: Es el conjunto de configuraciones que se van a entrenar en la siguiente iteración del bucle interno correspondiente a una banda concreta.

	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	27	3	9	9	6	27	5	81
1	27	3	9	9	3	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

Tabla 4.4: *Valores de n_i y r_i para las bandas de Hyperband correspondientes a varios valores de s, cuando $R = 81$ y $\eta = 3$.*

- $get\_hyperparameter\_configuration(n)$: Método que toma aleatoriamente n configuraciones del espacio de hiperparámetros.
- L: El valor de la función de coste para cada configuración después de haber sido entrenada por r_i épocas.
- $run\_then\_return\_val\_loss(t, r_i)$: Método que calcula el valor de la función de coste de la configuración t tras haberse entrenado durante r_i épocas.

Una vez comprendidos los parámetros y variables del algoritmo y los conceptos que representan, recorremos el ejemplo mostrado en la tabla 4.4 extraído del artículo [8] para concluir la explicación.

La tabla muestra las bandas, por columnas, y las iteraciones dentro de cada banda, por filas, para $R = 81$ y $\eta = 3$. Con estos valores, se obtiene $s_{\max} = 4$, lo que resulta en la generación de 5 bandas. En la iteración de la primera banda (correspondiente a $s = 4$) se toman aleatoriamente $n_0 = 81$ configuraciones y se entrenan durante r_i épocas. Después, en la iteración $i = 1$ se seleccionan las $\frac{81}{\eta} = 27$ mejores configuraciones y se entrenan durante 2 épocas más, completando un entrenamiento de 3 épocas. Este proceso sigue como se indica en la columna hasta que llega la iteración $i = 4$, en la que solamente se entrena durante las R épocas completas la configuración más prometedora, es decir, la que ha pasado todos los cortes o divisiones. El resto de bandas siguen el mismo proceso, donde la banda correspondiente a $s = 0$ sería equivalente a la búsqueda aleatoria o *RandomSearch* con 5

	$s=2$		$s=1$		$s=0$	
i	n_i	r_i	n_i	r_i	n_i	r_i
0	12	3	6	9	4	25
1	4	9	2	25		
2	2	25				

Tabla 4.5: *Valores de n_i y r_i para las bandas de Hyperband correspondientes a varios valores de s, cuando $R = 25$ y $\eta = 3$.*

configuraciones.

En el caso mostrado en la tabla, se exploran un mínimo de $n_{0_{s=4}} = 81$ configuraciones y como máximo $\sum_{s=0}^{4} n_0 = 128$ configuraciones. Por tanto, en caso de que el número total de combinaciones del espacio de hiperparámetros sea mayor que 81, se corre el riesgo de no explorar todas las soluciones. Este efecto se puede mitigar ejecutando el algoritmo un número arbitrario de veces atendiendo a los recursos disponibles.

En el trabajo, para cada tipo de red neuronal, se planteara un espacio de hiperparámetros basado en las diferentes arquitecturas que puede seguir cada tipo de red neuronal. Mediante la implementación de *Hyperband* de *Keras* [10] se explorará el espacio y se elegirá una configuración prometedora. Es importante destacar que en la implementación de *Keras* del algoritmo, el cálculo de B se lleva a cabo computando $B = (\lceil \log_\eta(R) + 1 \rceil)R$, donde varia el término $\lceil \log_\eta(R) + 1 \rceil$, que en el algoritmo original se calcula con $s_{max} = \lfloor \log_\eta(R) + 1 \rfloor$.

Considerando los recursos computacionales disponibles, los parámetros escogidos para la exploración del espacio de hiperparámetros de cada una de las redes propuestas mediante el algoritmo *hyperband* (planteado en las siguientes secciones) han sido $R = 25$ y $\eta = 3$. Estos parámetros dan lugar a la ejecución del algoritmo mostrada en la tabla 4.5 cuya interpretación debe hacerse de la misma manera que se ha hecho para 4.4.

En este caso nos encontramos con que cada iteración del algoritmo para los parámetros $R = 25$ y $\eta = 3$ explora un mínimo de 12 configuraciones y un máximo de 22 configuraciones, consideración que habrá que tener en cuenta en la siguiente sección para iterar el algoritmo en función del tamaño del espacio de hiperparámetros en cada caso.

En las siguientes secciones planteamos un espacio de hiperparámetros para las arquitecturas con las que se va a experimentar y mostramos los resultados de la exploración mediante el algoritmo *hyperband* en cada caso. Una vez elegido la mejor configuración de hiperparámetros para cada arquitectura, se entrenarán estos modelos utilizando todos los datos, es decir, utilizando las matrices 4400×17

4.3.2. Red neuronal *Feedforward*

Espacio de hiperparámetros

Hiperparámetro	**Nº de opciones**	**Valores**
Número de capas	4	2, 3, 4 o 5
Funciones de activación	3	ReLU, *Leaky* ReLU o ELU
Inclusión de capas *Dropout*	2	Sí o No

Tabla 4.6: *Espacio de hiperparámetros considerado para hacer la exploración de configuraciones de la red feedforward.*

Para la red neuronal densa se van a explorar las combinaciones de hiperparámetros y variaciones de su arquitectura[5] mostradas en la tabla 4.6. La estructura de las capas se ha establecido según la cantidad elegida en la primera fila de la tabla de la siguiente manera:

- 2 capas: Formadas por 1024 y 256 neuronas respectivamente.
- 3 capas: Formadas por 1024, 256 y 64 neuronas.
- 4 capas: Formadas por 1024, 512, 256 y 64 neuronas.
- 5 capas: Formadas por 1024, 512, 256, 128 y 64 neuronas.

Como observamos en la tabla 4.6 existen $4 \times 3 \times 2 = 24$ posibles combinaciones de hiperparámetros. Como hemos visto, la configuración elegida del algoritmo *hyperband* nos va a permitir explorar un mínimo de 12 configuraciones y un máximo de 22 configuraciones

[5]Recordamos que el objetivo no es construir el mejor modelo posible, por lo que la elección del espacio de hiperparámetros ha sido determinada arbitrariamente tras una breve experimentación.

(dependiendo de las elecciones aleatorias al comienzo de cada banda) por lo que una iteración del algoritmo nos va a permitir explorar gran parte del espacio de hiperparámetros.

Arquitectura final

En la tabla 4.7 se muestran las combinaciones de hiperparámetros que mejor resultado han obtenido en el algoritmo *hyperband*.

	Nº de capas densas	**F. de activación**	***Dropout***	**MSE**
Config. 1	5	ReLU	No	0.510
Config. 2	4	ReLU	No	0.521
Config. 3	5	*Leaky* ReLU	No	0.532

Tabla 4.7: *Valores de los hiperparámetros de las tres mejores configuraciones obtenidas por el algoritmo Hyperband para la arquitectura feedforward ordenadas ascendentemente por el valor de la función de coste (MSE) evaluada en el conjunto de validación.*

Siguiendo estos resultados, se va a proceder a entrenar la *configuración 1* con la versión de las matrices de tamaño 4400×17 (es decir, con todos los datos) durante 100 épocas. La arquitectura exacta del modelo *feedforward* utilizado finalmente se puede observar en el apéndice A.1. En la figura 4.3 podemos apreciar los valores de la función de coste (MSE) a lo largo del entrenamiento en el conjunto de entrenamiento y en el de validación. Podemos observar como la curva en validación se estabiliza a partir de la época 20 mientras que la de entrenamiento sigue mejorando. Esto podría indicar cierto sobreentrenamiento, por lo que se han tomado como versión final del modelo los pesos que mejor optimizaron la función de coste en validación, lo que ocurrió durante la época 71 en la que se obtuvo un MSE de 0.487. Por otro lado, el mejor valor de la función de coste en el conjunto de entrenamiento se obtuvo en la época 96 con un valor de 0.286.

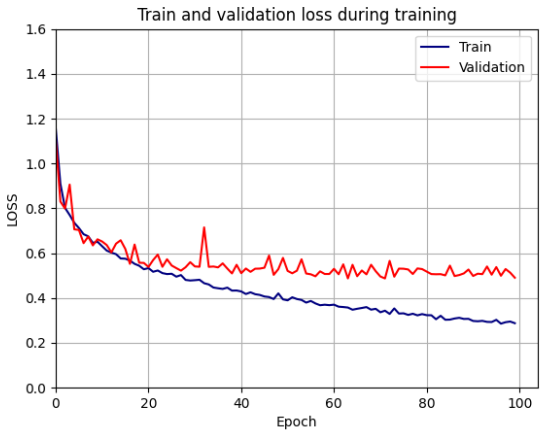

Figura 4.3: *Valores de la función de coste en el conjunto de entrenamiento y en el de validación durante el entrenamiento de la configuración 1 de la red neuronal feedforward.*

4.3.3. Red Neuronal Convolucional (CNN)

Espacio de hiperparámetros

Hiperparámetro	**Nº de opciones**	**Valores**
Número de capas densas	1	5
Funciones de activación	1	ReLU
Inclusión de capas Dropout	2	Sí y No
Número de bloques convolucionales	2	2 y 3
Tamaño del kernel de la convolución	2	3×3 o 5×5
Capa de agrupamiento de la convolución	2	*Max Pooling* o *Avg. Pooling*
Capa de transición	3	*Max Pooling*, *Avg. Pooling* o *Flatten*

Tabla 4.8: *Espacio de hiperparámetros considerado para hacer la exploración de configuraciones de la red convolucional.*

Para la red neuronal convolucional se van a explorar las combinaciones de los hiperparámetros y variaciones de su arquitectura mostradas en la tabla 4.8. Con el objetivo de no ampliar en exceso el espacio de hiperparámetros, vamos a fijar el número de capas densas y la función de activación a los valores para los que mejor resultados se han obtenido en la red neuronal *feedforward*. Esto nos va permitir centrarnos en explorar los hiperparámetros que caracterizan a la red convolucional.

La estructura que se ha considerado para los bloques convolucionales es la siguiente:

- Capa convolucional: Utilizando el tamaño de kernel elegido por el algoritmo *hyperband* de entre las opciones de la tabla[6].
- Capa de agrupamiento: Capa de agrupamiento del tipo elegido por el algoritmo *hyperband* de entre las opciones de la tabla.
- Capa de normalización: Es una práctica común, normalizar las características extraídas por cada bloque convolucional.

Como observamos en la tabla 4.8 existen $1\times1\times2\times2\times2\times2\times3 = 48$ posibles combinaciones de hiperparámetros. Debido a que en este caso tenemos un espacio de hiperparámetros más amplio, vamos a realizar 2 iteraciones del algoritmo *hyperband* con la configuración elegida. Esto nos va a permitir explorar un mínimo de 12 combinaciones y un máximo de 44.

Arquitectura final

En la tabla 4.9[7] se muestran las combinaciones de hiperparámetros que mejor resultado han obtenido en el algoritmo *hyperband*.

	Nº convoluciones	**Tamaño del kernel**	**Agrupamiento**	**Transición**	**MSE**
Config. 1	3	5×5	*Max Pooling*	*Global Max Pooling*	0.420
Config. 2	2	3×3	*Max Pooling*	*Flatten*	0.453
Config. 3	2	3×3	*Avg. Pooling*	*Flatten*	0.466

Tabla 4.9: *Valores de los hiperparámetros (solamente incluidos para los que se consideró más de una opción) de las tres mejores configuraciones obtenidas por el algoritmo Hyperband para la arquitectura convolucional, ordenadas ascendentemente por el valor de la función de coste (MSE) evaluada en el conjunto de validación.*

De acuerdo con estos resultados, se va a llevar a cabo el mismo entrenamiento que en el caso anterior (con todos los datos durante 100 épocas) con la *configuración 1* de la red

[6]El tamaño kernel para el primer bloque convolucional se ha fijado en 3×3, por lo que solamente varían los tamaños del kernel del segundo y tercer bloque si aplica.

[7]La tabla es análoga a las tablas 4.7 y 4.11 pero se ha dividio en dos subtablas por razones de formato.

neuronal convolucional, cuya arquitectura completa se puede observar en el apéndice A.2. Podemos apreciar en la figura 4.4 como los valores de la función de coste en el conjunto de validación tienen picos repentinos durante el entrenamiento. Esto puede ser debido a una mala elección de hiperparámetros que se discutirá en la sección 5. Aún así, al igual que en el caso anterior, se ha tomado la versión del modelo que mejor ha optimizado la función de coste en el conjunto de validación para evitar tomar un modelo sobreajustado al conjunto de entrenamiento. El mejor valor de la función de coste en validación ocurrió durante la época 23 con un valor de 0.420, mientras que el mejor valor en entrenamiento fue de 0.197 durante la época 99.

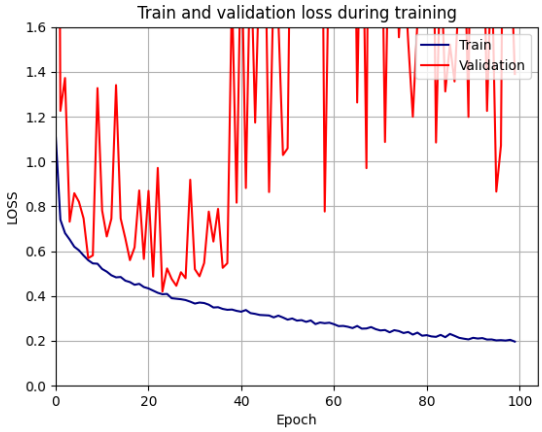

Figura 4.4: *Valores de la función de coste en el conjunto de entrenamiento y en el de validación durante el entrenamiento de la configuración 1 de la red neuronal convolucional.*

4.3.4. Red Neuronal Recurrente LSTM (RNN)

Espacio de hiperparámetros

Hiperparámetro	**Nº de opciones**	**Valores**
Número de capas densas	1	5
Funciones de activación	2	ReLU o *Leaky* ReLU
Inclusión de capas *Dropout*	2	Sí o No
Número de capas LSTM	2	1 o 2
Devolución de secuencias	2	Sí o No

Tabla 4.10: *Espacio de hiperparámetros considerado para hacer la exploración de configuraciones de la red recurrente utilizando celdas LSTM.*

Para la red neuronal recurrente utilizando celdas LSTM se van a explorar las combinaciones de hiperparámetros y variaciones de su arquitectura mostradas en la tabla 4.10. Al igual que en el caso anterior, con la intención de centrar la exploración en los hiperparámetros que caracterizan la red neuronal recurrente, se van a explorar solo el número de capas y las funciones de activación que mejor rendimiento han dado para la red *feedforward*. El número de celdas LSTM se ha establecido según el número de capas recurrentes de la siguiente manera:

- 1 capa: Formada por 25 celdas LSTM.
- 2 capas: Formadas por 20 y 30 celdas LSTM respectivamente.

El parámetro de devolución de seceuncias hace referencia a la posibilidad de que las capas recurrentes devuelvan toda la secuencia y el valor que predicen a partir de ella o solamente el valor.

Como observamos en la tabla 4.10 existen $1 \times 2 \times 2 \times 2 \times 2 = 16$ posibles combinaciones de hiperparámetros. En este caso una iteración del algoritmo *hyperband* nos permitirá explorar el $12/16 = 75\,\%$ combinaciones de hiperparámetros en el peor de los casos, por lo que se ha optado por no realizar más iteraciones.

Arquitectura final

En la tabla 4.11 se muestran las combinaciones de hiperparámetros que mejor resultado han obtenido en el algoritmo *hyperband*.

	F. de activación	**Dropout**	**Nº de capas LSTM**	**Secuencias**	**MSE**
Config. 1	*Leaky* ReLU	No	1	Sí	0.511
Config. 2	ReLU	No	2	Sí	0.523
Config. 3	*Leaky* ReLU	No	1	Sí	0.600

Tabla 4.11: *Valores de los hiperparámetros (solamente incluidos para los que se consideró más de una opción) de las tres mejores configuraciones obtenidas por el algoritmo Hyperband para la arquitectura recurrente, ordenadas ascendentemente por el valor de la función de coste (MSE) evaluada en el conjunto de validación.*

Siguiendo el procedimiento llevado a cabo para las otras alternativas, se va a entrenar con todos los datos durante 100 épocas la *configuración 1* de la tabla 4.11. El modelo completo correspondiente a esta configuración se puede apreciar en el apéndice A.3.

En la figura 4.5 en la que podemos observar el comportamiento de la función de coste a lo largo del entrenamiento, se puede apreciar un comportamiento similar al de la red *feedforward*, estabilizándose la curva del MSE en validación a partir de la época 30 en este caso. Los pesos del modelo utilizado finalmente para la red recurrente, han sido los generados en la época 93 ya que fueron los que mejor optimizaron la función de coste en validación con un valor de 0.418. En cambio, el mejor valor del MSE en el conjunto de entrenamiento se obtuvo más tarde en la época 98 con un valor de 0.224.

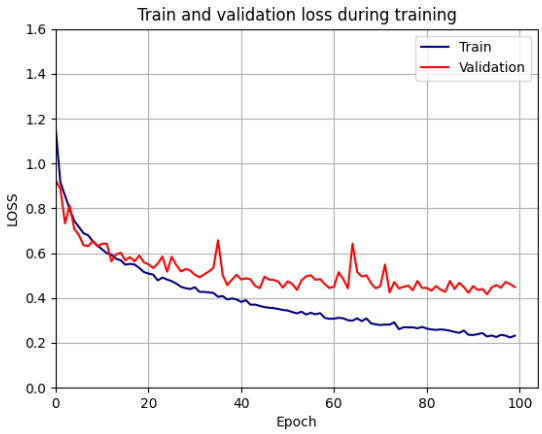

Figura 4.5: *Valores de la función de coste en el conjunto de entrenamiento y en el de validación durante el entrenamiento de la configuración 1 de la red neuronal recurrente.*

4.4. Comparación de resultados y conclusiones

Se ha expresado a lo largo de la sección 3.2 que las redes neuronales con mayor potencial para resolver de forma rigurosa el problema son la red convolucional y la recurrente. Esto es debido a que estas están diseñadas para capturar relaciones espaciales y temporales, respectivamente, en los datos de entrada, y en nuestras matrices de entrada existen este tipo de relaciones. Por el contrario, la red neuronal *feedforward* debería tener más dificultades para predecir correctamente utilizando datos de esta naturaleza. Con esta hipótesis en mente, a continuación analizamos el rendimiento de cada red neuronal. Comparando los resultados en entrenamiento podemos ver que los tres tipos de redes neuronales han obtenido valores parecidos en el conjunto de entrenamiento. Sin embargo, la red neuronal recurrente parece la más prometedora ya que ha sido la que ha tenido mayor capacidad de ajustarse al conjunto de validación. Para analizar el rendimiento real de los modelos entrenados con los pesos escogidos de la manera indicada anteriormente, necesitamos evaluarlos en un conjunto que se haya encontrado aislado durante el proceso de entrenamiento, el conjunto de test. La tabla 4.12[8] muestra las diferentes métricas explicadas en la sección 3.1.2 evaluadas para cada

[8]El RMSE aporta información redundante al MSE ya que se calcula haciendo la raíz cuadrada del MSE. Sin embargo, aporta información útil ya que reduce el error a las unidades originales.

modelo en el conjunto de test. En ella se aprecia como el rendimiento de todas las redes es similar, aunque la red convolucional y la red recurrente ofrecen resultados ligeramente mejores. Teniendo en cuenta los rangos en los que se mueven los parámetros que se quieren predecir (mencionados en la sección 2.4), se puede observar que los errores no son demasiado altos, y por lo tanto, las predicciones son aceptables, aunque existe margen de mejora.

	F. Coste (MSE)	**RMSE**	**MAE**
feedforward	0.482	0.694	0.553
CNN	0.441	0.679	0.521
RNN	0.440	0.663	0.530

Tabla 4.12: *Comparación de los resultados de las métricas MSE, RMSE y MAE en el conjunto de test para cada arquitectura.*

En la tabla 4.13 podemos ver el mismo desglose de errores que en la tabla anterior, pero distinguiendo los dos parámetros más importantes a la hora de caracterizar la anomalía, ρ y μ. Teniendo en cuenta que el parámetro ρ sigue una distribución normal de centro 2,1 y desviación $\sqrt{2}$ un error cuadrático medio entre 0,7 y 0,8 se podría considerar relativamente aceptable aunque bastante susceptible de mejora. Lo mismo ocurre con el parámetro μ que sigue una distribución normal de centro 4,4 y y desviación $\sqrt{5}$, es decir, más amplia, pero su error cuadrático medio en las redes también es más grande, entre 1,39 y 1,55. La tabla también destaca la diferencia entre redes neuronales donde la red *feedfoward* es más precisa a la hora de predecir el parámetro ρ, mientras que la que mejor predice el parámetro μ es la red neuronal convolucional.

	MSE (ρ)	**RMSE (ρ)**	**MAE (ρ)**	**MSE (μ)**	**RMSE (μ)**	**MAE (μ)**
feedforward	0.518	0.720	0.450	2.382	1.543	1.175
CNN	0.624	0.790	0.529	1.941	1.393	1.057
RNN	0.531	0.728	0.444	2.228	1.493	1.114

Tabla 4.13: *Comparación de los resultados de las métricas MSE, RMSE y MAE de los parámetros a predecir ρ y μ en el conjunto de test para cada arquitectura.*

Por otro lado, el coste computacional que ha tenido entrenar cada modelo es un factor importante a tener en cuenta, ya que estamos tratando una gran cantidad de datos de

dimensiones considerablemente grandes. En la tabla 4.4 podemos observar los tiempos de entrenamiento y de predicción de los diferentes modelos ejecutados sobre una máquina T4 GPU de *Google Collab*[9]. Podemos ver como la complejidad estructural de las redes convolucional y recurrente se traduce en una mayor coste computacional durante el entrenamiento. Es importante tener en cuenta estos datos a la hora de juzgar si el aumento en el coste computacional compensa la mejora en el rendimiento de estos modelos. A priori parece que la mejora de la red neuronal recurrente y la red neuronal convolucional con respecto a la red *feedforward* en cuanto a precisión de las predicciones, no compensa el aumento en el coste computacional, pero este juicio dependerá del caso de uso específico y de la tolerancia al error que se desee mantener.

	T. de entrenamiento (100 épocas)	T. de predicción del conjunto de test
feedforward	9-15 s/epoch $\longrightarrow \approx$ 1000s	2s
CNN	52-65 s/epoch $\longrightarrow \approx$ 5400s	5s
RNN	46-53 s/epoch $\longrightarrow \approx$ 4800s	3s

Tabla 4.14: *Comparación del coste computacional para cada arquitectura teniendo en cuenta que el conjunto de entrenamiento es de 11275 pares matriz-parámetros y el conjunto de de validación y de test está formado por 1409 pares matriz-parámetros. Tiempos de entrenamiento y predicción sobre una máquina T4 GPU en Google Collab.*

[9]Es importante remarcar que se puede llevar a cabo esta comparación gracias a que implementación en *Keras* de las capas utilizadas en estas arquitecturas están optimizadas para GPU.

Capítulo 5

Conclusiones y trabajo futuro

El trabajo muestra el flujo completo de estudio del problema planteado por las ecuaciones (1.1) y (1.2), abordado tanto como problema directo como problema inverso. En primer lugar, se presenta la resolución del problema directo utilizando e implementando el método de elementos finitos. Por otro lado, se enfoca la mayor parte del trabajo en el desarrollo de métodos de aprendizaje automático como propuesta para resolver el problema inverso.

La aproximación planteada en este trabajo para abordar la resolución del problema inverso ha demostrado ser una alternativa viable a otros métodos, como las soluciones basadas en métodos bayesianos planteadas en [2]. Los resultados presentados en el capítulo 4 muestran errores relativamente pequeños, lo que sugiere un gran potencial en algunas de las arquitecturas de redes neuronales propuestas para inferir los parámetros de las anomalías. En la tabla 5.1 se muestra un ejemplo concreto de los parámetros que caracterizan una de las anomalías del conjunto de datos y la predicción de cada red neuronal para ese caso específico. Se puede observar como todas las redes neuronales aproximan la solución de este problema inverso concreto de manera satisfactoria.

Una vez asegurada la viabilidad de este enfoque, queda como trabajo futuro profundizar en la capacidad real de estas arquitecturas para ajustarse a los resultados esperados de una manera más rigurosa. Como se observa en la sección 4.3, se plantean espacios de hiperparámetros reducidos debido al alto coste computacional de explorar más alternativas, especialmente en el caso de las redes neuronales convolucionales y recurrentes, como se deta-

	cx	**cy**	**a**	**b**	θ	ρ	μ
Resultado esperado	0.831	3.210	1.841	0.429	0.291	2.626	3.651
feedforward	0.662	3.118	1.708	0.481	0.154	2.468	2.792
CNN	0.797	3.100	1.847	0.601	0.440	2.671	3.465
RNN	0.653	3.088	1.820	0.480	0.201	2.448	3.537

Tabla 5.1: *Ejemplo de las predicciones hechas por cada red neuronal para un resultado esperado dado.*

lla en la sección 4.4. Una forma de explorar la capacidad de estos modelos para aproximarse más rigurosamente a la solución sería realizar una exploración más amplia y exhaustiva de las arquitecturas propuestas. Concretamente, hay hiperparámetros importantes en la configuración de una red neuronal que no se han considerado en esta exploración y cuya inclusión podría mejorar los modelos finales seleccionados, como la tasa de aprendizaje o el tipo de regularización. Además, en cuanto a los hiperparámetros explorados, se podrían valorar más opciones como diferentes configuraciones de las capas densas u otros tipos de funciones de activación como SELU o Tanh.

Debido a la limitada disponibilidad de recursos computacionales, se consideró utilizar el algoritmo de exploración *hyperband* con los parámetros mencionados en el capítulo 4.3.1. En este sentido, la exploración podría beneficiarse de la utilización de métodos de exploración más exhaustivos como *GridSearch* y de un entrenamiento de más épocas durante dicha exploración, aunque ambas propuestas requerirían de más recursos computacionales.

Por otro lado queda pendiente resolver uno de los principales problemas de la arquitectura convolucional. Como vemos durante el entrenamiento de la red neuronal convolucional en la tabla 4.4, se aprecian una picos bruscos en la función de coste medida en el conjunto de validación a lo largo del entrenamiento. Esto ocurre a pesar de ser la arquitectura para la que se ha considerado un espacio de hiperparámetros más amplio. Este problema normalmente se soluciona eligiendo un optimizador[1] más adecuado, y explorando diferentes tasas de aprendizaje y métodos de regularización. Este hecho, recalca la necesidad de hacer una

[1]En este trabajo se utiliza el optimizador Adam en todos los modelos con una tasa de aprendizaje de 0.001.

exploración de hiperparámetros más extensa y exhaustiva.

En general, parece que la red neuronal *feedforward* tiene menor capacidad para resolver problemas donde existen dependencias espaciales y temporales entre los datos de entrada como este. Sin embargo, las redes neuronales convolucionales y recurrentes sí tienen esta capacidad. En este estudio no se muestran grandes diferencias ya que se han considerado pocas alternativas para cada arquitectura. Cabe destacar que, en el caso de las redes neuronales *feedforward*, no se puede experimentar mucho más con su arquitectura, lo que limita su margen de mejora. Analizar las razones por las cuales una red obtiene resultados ligeramente mejores para ciertos parámetros mientras que para otros ocurre lo contrario (como se muestra en la tabla 4.13), podría proporcionar entendimiento útil a la hora de diseñar arquitecturas más óptimas.

En conclusión, este trabajo plantea de manera exhaustiva la resolución del problema definido por las ecuaciones (1.1) y (1.2). La resolución del problema directo presentada es particular de la ecuación diferencial planteada. Sin embargo, los métodos propuestos para aproximar la solución del problema inverso son extrapolables a la inferencia de parámetros de otros modelos matemáticos. Aunque aún queda pendiente profundizar en la rigurosidad de este método, el estudio plantea un enfoque innovador y actual para resolver problemas inversos de distinta naturaleza.

Bibliografía

[1] C. Abugattas, A. Carpio, E. Cebrian, and G. Oleaga. Quantifying uncertainty in inverse scattering problems set in layered environments. *preprint*, 2024.

[2] A. Carpio, E. Cebrián, and A. Gutiérrez. Object based bayesian full-waveform inversion for shear elastography. *Inverse Problems*, 39(7):075007, 2023.

[3] F. Chollet et al. Keras. https://keras.io, 2015.

[4] D. P. García. Universidad computense de madrid, notas de clase: Elementos de ecuaciones diferenciales ordinarias, 2020. Notas tomadas por Adrián Sanjuán Espejo.

[5] A. Huet. ahstat.github.io. https://ahstat.github.io/RNN-Keras-understanding-computations/, 2018. Accessed: May 2024.

[6] A. Jadon, A. Patil, and S. Jadon. A comprehensive survey of regression based loss functions for time series forecasting, 2022. URL https://arxiv.org/abs/2211.02989.

[7] C. Leo. Towards data science. https://towardsdatascience.com/the-math-behind-convolutional-neural-networks-6aed775df076#8daf, 2024. Accessed: May 2024.

[8] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[9] J.-L. Lions and E. Magenes. *Problèmes aux limites non homgènes et application*, volume 1. Dunod, Paris, 1968.

[10] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, et al. Kerastuner. https://github.com/keras-team/keras-tuner, 2019.

[11] K. O'Shea and R. Nash. An introduction to convolutional neural networks, 2015. URL https://arxiv.org/abs/1511.08458.

[12] P.-A. Raviart and J. M. Thomas. *Introduction à l'analyse numérique des équations aux dérivées partielles*. Ed. Masson, Paris, 1983.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[14] R. M. Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview, 2019. URL https://arxiv.org/abs/1912.05911.

[15] K. Sechidis, G. Tsoumakas, and I. Vlahavas. On the stratification of multi-label data. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part III 22*, pages 145–158. Springer, 2011.

[16] S. Sharma, S. Sharma, and A. Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[18] N. D. Toro. *La pseudoinversa en el proceso de aprendizaje del asociador lineal*. Tesis de maestría, Universidad del Valle, Santiago de Cali, Colombia, 2014. Sección 4.4.

[19] F. R. Vicente. Las matemáticas del machine learning: Redes neuronales (ii). https://telefonicatech.com/blog/las-matematicas-del-machine-learning-redes-neuronales-ii, 2020. Accessed: June 2024.

Apéndice A

Arquitecturas

A.1. Arquitectura de la red neuronal *feedforward*

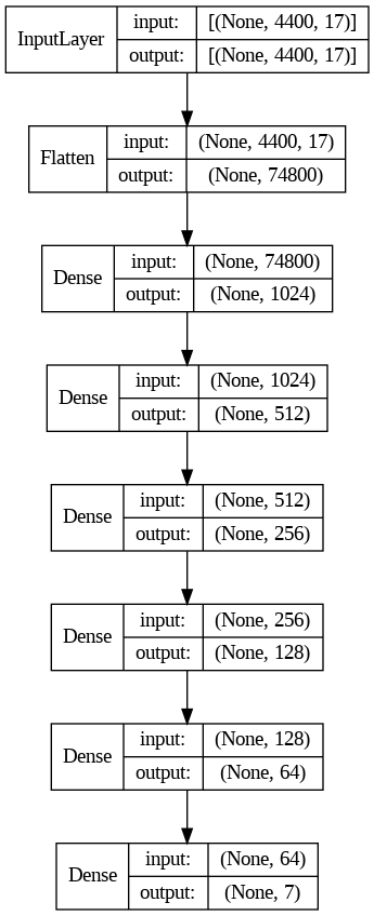

Figura A.1: *Arquitectura de la configuración 1 de la red neuronal feedforward.*

A.2. Arquitectura de la red neuronal convolucional

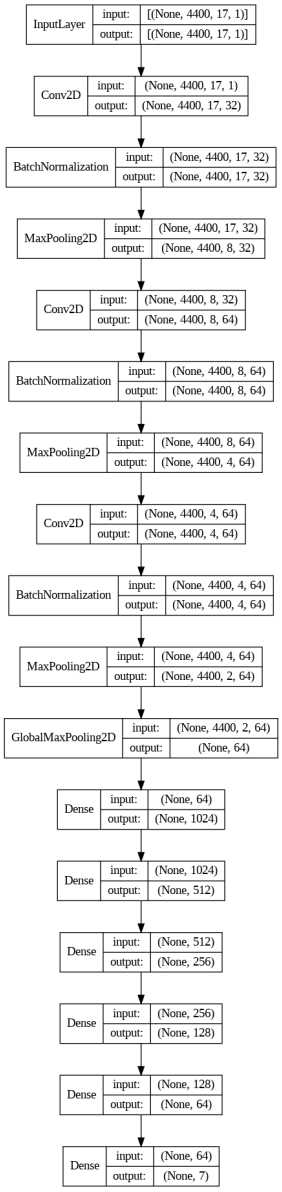

Figura A.2: *Arquitectura de la configuración 1 de la red neuronal convolucional.*

A.3. Arquitectura de la red neuronal recurrente

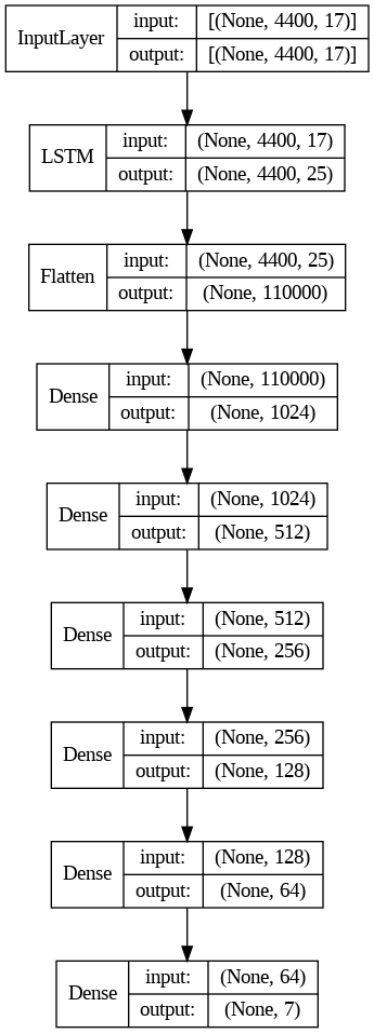

Figura A.3: *Arquitectura de la configuración 1 de la red neuronal recurrente.*