PROPAGACIÓN DE ONDAS MEDIANTE APRENDIZAJE AUTOMÁTICO

Miguel Jesús Peñalver Carvajal

DOBLE GRADO EN MATEMÁTICAS E INGENIERÍA INFORMÁTICA
FACULTAD DE MATEMÁTICAS

UNIVERSIDAD COMPLUTENSE DE MADRID

Curso 2022/2023

Trabajo de Fin de Grado

Junio de 2023

Dirigido por:
Ana María Carpio Rodríguez
María Macarena Gómez Mármol

Resumen

El propósito de este trabajo es investigar en qué medida se pueden usar redes neuronales para aproximar resultados que, en principio, requerirían resolver numéricamente ecuaciones de ondas. Nos centraremos en geometrías unidimensionales prefijadas gobernadas por pocos parámetros e intentaremos aproximar problemas directos e inversos.

En el primer caso, tratamos de estimar mediante redes convolucionales aproximaciones de soluciones de ecuaciones de onda a partir de parámetros materiales. El problema inverso consiste en estimar mediante redes neuronales los parámetros de la ecuación de ondas que representan el material a partir de valores de la solución medidos en un extremo en una secuencia de tiempos.

Tras el estudio realizado, pudimos obtener unas estimaciones de las ondas mediante redes convolucionales con un error respecto al rango menor de un 5 % en todos los casos para el problema directo. Por otro lado, en el problema inverso, el modelo presentaba dificultades para estimar uno de los parámetros (la posición del objeto) de los tres estudiados. Finalmente, se comprobó que el problema residía en los conjuntos de datos generados para alimentar el modelo, pues no disponían de suficiente información para representar todo el espacio de parámetros.

Palabras clave

Ecuación de onda; Método de elementos finitos; Red convolucional 1D; Predicción de ondas; Problemas inversos; MATLAB; Python.

Abstract

The purpose of this work is to investigate the extent to which neural networks can be used to approximate results that would typically require numerically solving wave equations. We will focus on predetermined one-dimensional geometries governed by a few parameters and attempt to approximate both direct and inverse problems.

In the first case, we attempt to estimate wave equation solutions using convolutional networks based on material parameters. The inverse problem involves estimating the wave equation parameters that represent the material using neural networks, given measured solution values at one end over a sequence of time steps.

After the conducted study, we were able to obtain wave estimations using convolutional networks with a deviation of less than 5 % from the range of values in all cases for the direct problem. On the other hand, in the inverse problem, the model faced difficulties in estimating one of the parameters (object position) out of the three studied. Finally, it was determined that the issue resided in the generated data sets used to train the model, as they did not contain sufficient information to represent the entire parameter space.

Keywords

Wave equation; Finite element method; Convolutional network 1D; Wave prediction; Wave analysis; MATLAB; Python.

Índice general

Agradecimientos

En primer lugar, expresar mi profunda gratitud a mis tutoras, Ana y Macarena, por su orientación y dedicación durante todo el proceso de elaboración de este trabajo. Su experiencia y conocimiento en el campo de las Matemáticas y las ondas han sido de un valor mayúsculo para mí. Sin ellas, este trabajo no habría visto la luz de la forma en la que lo ha hecho.

A mi familia, por su presencia constante durante todo este proceso, por su consuelo incondicional en los momentos de duda y alegría compartida en los de celebración. A mis padres, quienes han sabido inculcarme desde que tengo recuerdo su amor por la Ciencia y cada una de sus ramas.

A mis compañeros de clase, ahora amigos, por su apoyo y colaboración a lo largo de esta travesía académica. Juntos hemos compartido momentos buenos y no tan buenos, retos educativos y personales, que nos han hecho mejores personas. Si los hemos sabido superar ha sido gracias a la ayuda mutua, que estoy seguro que se mantendrá en el tiempo.

A las Matemáticas, por hacerme crecer, esperar y luchar y por enseñarme a valorar el mundo que nos rodea. Por ser, a partir de ahora, mi fiel compañera de viaje.

Y, finalmente, a la Universidad Complutense de Madrid, por fomentarme la curiosidad y por su valiosa contribución en mi formación académica. Este proyecto de investigación no es sino el culmen de las oportunidades que me han brindado a lo largo de estos años.

Capítulo 1

Introducción

El propósito de este trabajo es investigar en qué medida se pueden usar redes neuronales para aproximar resultados que, en principio, requerirían resolver numéricamente ecuaciones de ondas. El problema se plantea en un contexto [1] en que 1) se prevé tener que resolver cantidades ingentes de ecuaciones de ondas en una geometría prefijada gobernada por pocos parámetros y 2) no se requiere el valor de la solución en todo punto, sino en unos pocos, o bien se precisa el cálculo de una magnitud derivada de la solución. En tal situación, se puede plantear la conveniencia de entrenar una red neuronal que proporcione rápidamente una aproximación a los valores requeridos, en lugar de enfrentar el coste de resolver la ecuación en derivadas parciales, especialmente, en dimensiones 2 y 3. Esto tiene sentido pues podemos garantizar la fiabilidad de la aproximación proporcionada por algún tipo de red.

Para testar la idea, vamos a considerar un caso práctico unidimensional. Fijamos una geometría en la que insertamos un material distinto en la región central de un hilo elástico finito. Generamos un impulso en un extremo, que se propaga por el hilo e interacciona con la inclusión, y medimos la señal recibida en el extremo opuesto durante un tiempo. La inclusión está caracterizada por tres parámetros: centro, longitud, y velocidad de onda. En testado no destructivo de materiales, se plantean situaciones en las que, conocido el impulso generado en un extremo y la señal recibida en el otro, se desea averiguar la ubicación, tamaño y naturaleza de la inclusión. Matemáticamente, se trata de un problema inverso. El problema directo asociado consiste en la resolución del problema de ondas para cada inclusión propuesta, dado el impulso generado. La solución numérica debería proporcionarnos una aproximación a los datos experimentalmente medidos. Las técnicas clásicas de resolución del problema inverso llevan aparejada la resolución de cantidades grandes de problemas directos con la misma estructura variando unos pocos parámetros.

El trabajo está estructurado en dos bloques principales. En el primer bloque, explicamos cómo resolver ecuaciones de onda sencillas en una dimensión, haciendo uso de la herramienta MATLAB. A partir de este código, en el cual se implementa el método

de elementos finitos, generaremos una serie de archivos con información de numerosas ondas y su propagación a lo largo del tiempo. Con estos cálculos intentamos averiguar cómo se propaga una onda en un medio en el cual hay un objeto con unas propiedades diferentes, las cuales vamos a hacer variar para hallar los conjuntos de información. Así pues, el tamaño, la posición y la velocidad de propagación de la onda en el objeto es lo que se va a hacer variar entre un problema y otro.

Por otro lado, en la segunda parte de este trabajo se han implementado dos redes neuronales en Python para trabajar con los datos de las ondas hallados anteriormente. En primer lugar, se ha diseñado una red neuronal convolucional que, a partir de los valores de centro, longitud y velocidad de la onda que caracterizan la inclusión del medio en el que se propaga la onda, sea capaz de hallar la imagen de esta para cierto intervalo temporal en cierto punto del medio. En segundo lugar, se ha diseñado otra red neuronal que trata de realizar la tarea inversa, en la que a partir de la onda sea capaz de estimar sus parámetros.

Con el estudio llevado a cabo se ha creado una red neuronal capaz de calcular las propiedades físicas de un objeto al ser atravesado por una onda determinada. Con esta información podríamos identificar materiales cuyas propiedades son previamente conocidas, al emitir una radiación sobre él sin llevar a cabo un estudio más exhaustivo del mismo. Esto podría llevar a un abaratamiento considerable de la maquinaria y hacer esta tecnología de identificación más accesible al público general.

Un caso de uso interesante sería la identificación de piedras preciosas al incidir la luz en las mismas, pues la velocidad de la onda en las mismas es distinta a la de otros materiales populares en las falsificaciones, como el cristal.

Los objetivos que se van a perseguir en este trabajo son los siguientes:

- Resolver ecuaciones de onda de una dimensión para generar los datos de los cuales se va alimentar la red neuronal. Estas ecuaciones de onda deben modelar la presencia de un objeto cuyas propiedades físicas van a variar.
- A partir de los datos anteriores, entrenar una red neuronal que nos permita, a partir de las propiedades del objeto, estimar la onda resultante.
- A partir de los datos del primer apartado, identificar gracias a una red neuronal las propiedades del objeto a partir de la onda. Esto es, el problema inverso del apartado anterior.

Capítulo 2

Ecuación de onda

En este capítulo trataremos la ecuación de onda que buscamos resolver para hallar los datos con los que entrenar la red neuronal. En primer lugar, se dará una explicación sobre el problema concreto desde el punto de vista matemático y se expondrán los distintos casos en los que nos vamos a centrar en el trabajo. Posteriormente, se expondrá cómo se han resuelto estas ecuaciones haciendo uso de la herramienta MATLAB, así como el guardado de los datos.

2.1. Descripción del problema

Consideremos una onda que se desplaza en un medio externo unidimensional de longitud L. Denotaremos por $x \in [0, L]$ la variable que describe dicho medio. En los extremos del medio suponemos que, no hay desplazamiento en $x = 0$ y el desplazamiento es libre en $x = L$. Además consideramos que la onda se encuentra sometida a una fuerza externa $F(x, t)$, donde t representa el instante de tiempo. Entonces las ecuaciones que modelan el desplazamiento se escriben como:

$$(c^2(x) \cdot u_x)_x + F(x, t) = u_{tt} \tag{2.1}$$

donde $u(x, t)$ representa el desplazamiento vertical y $c(x)$ la velocidad.

Como el desplazamiento en $x = 0$ es nulo y libre en $x = L$, tenemos condiciones de contorno de tipo Dirichlet y Neumann homogéneas, respectivamente:

$$u(0, t) = 0, \quad t > 0 \qquad u_x(L, t) = 0, \quad t > 0$$

El sistema se completa con la condición inicial:

$$u(x, 0) = 0, \quad x \in [0, L] \qquad u_t(x, 0) = 0, \quad x \in [0, L]$$

En esta geometría,

- El extremo inicial $x = 0$ será el extremo fijo, donde aplicaremos la fuerza $f(t)$ a la onda y por tanto iniciará el recorrido la misma, la cual se propagará por el medio (el intervalo de 0 a L). Con un cambio de variable, esta fuerza entra en la ecuación como $F(x, t)$, mientras que el valor en el extremo $x = 0$ de la variable u pasa a ser cero.
- El extremo final $x = L$ será el extremo donde nos centraremos y tomaremos la medición de la posición de la onda. Así, la imagen de la función de onda en este punto, como veremos más adelante, será la que guardaremos y usaremos como entrenamiento para la red neuronal. En este extremo libre, imponemos una condición de contorno $u_x = 0$.

El intervalo de definición de la onda $[0, L]$ será fijo para todas las ondas de este proyecto con $L = 10$ para asegurarnos que las perturbaciones sobre el extremo inicial lleguen al final del intervalo en un tiempo de, como mucho, 10 unidades.

En principio, la onda se desplazará a la velocidad c del medio externo, la cual puede ser considerada constante en todos los puntos. Para el estudio, colocaremos un objeto en el interior del medio con características distintas, que modificará dicha velocidad. El objeto estará centrado en el punto $x_0 \in (O, L)$, con un radio $l_0/2$, entonces la velocidad se escribe como:

$$c(x) = \begin{cases} c_0, & x \in \left(x_0 - \frac{l_0}{2}, x_0 + \frac{l_0}{2}\right) \\ c, & \text{caso contrario} \end{cases}$$

En consecuencia, el problema que queremos resolver tiene tres parámetros que irán cambiando en un determinado rango, x_0, l_0 y c_0 y estamos interesados en conocer el comportamiento de la onda para todas estas situaciones. Debido a la definición de la velocidad para el estudio, dividimos el dominio en tres partes, la primera de ella con velocidad de la onda $c(x) = 1$, la segunda, el objeto, con velocidad $c(x) = c_0$, y la tercera parte, el medio externo, con $c(x) = 1$.

En la figura 2.1 se han representado las condiciones exactas del medio externo en el cual vamos a resolver las ecuaciones de onda: la velocidad c será 1 en el medio externo, y la longitud del mismo, L, serán 10 unidades.

También fijaremos el tiempo para el cual estudiaremos las soluciones para la ecuación de onda, en este caso, 10 unidades temporales y tomando como tiempo inicial $t_0 = 0$. Además, fijaremos el paso de tiempo en el valor $dt = 1/60$, suficiente para resolver numéricamente con precisión en nuestro rango de parámetros. En la figura 2.2 presentamos cómo viaja la onda tomando ilustraciones cada 75 pasos de tiempo. Como vemos, la elección del tiempo total (10 unidades) no es arbitraria, sino que es el tiempo total que tarda la onda en llegar al extremo final, de manera que contemos con datos representativos para cada elección de parámetros, lo cual veremos más adelante cuando mostremos los datos recopilados por cada onda.

En la figura 2.2 observamos claramente la diferencia de velocidad de la onda en

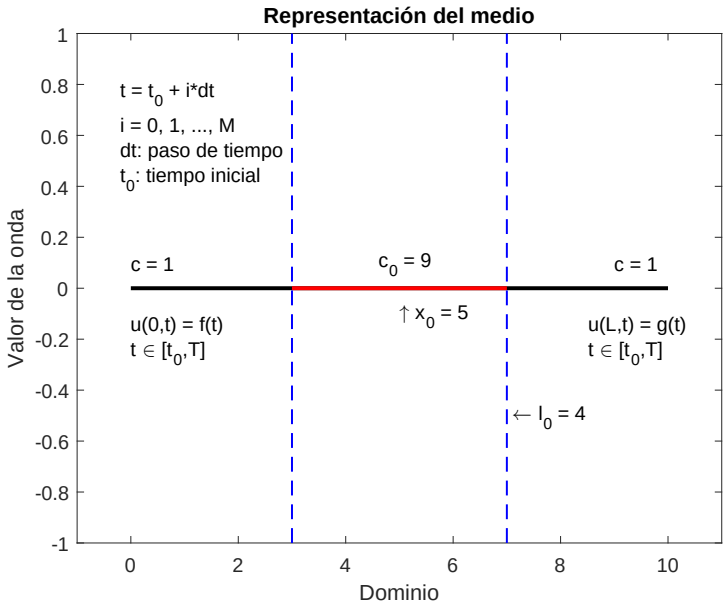


Figura 2.1: Representación gráfica del objeto (línea horizontal en rojo) situado en el medio externo (línea horizontal negra). Como vemos, la velocidad de la onda es distinta en este tramo, donde $c(x) = 9$ y en el medio externo $c(x) = 1$.

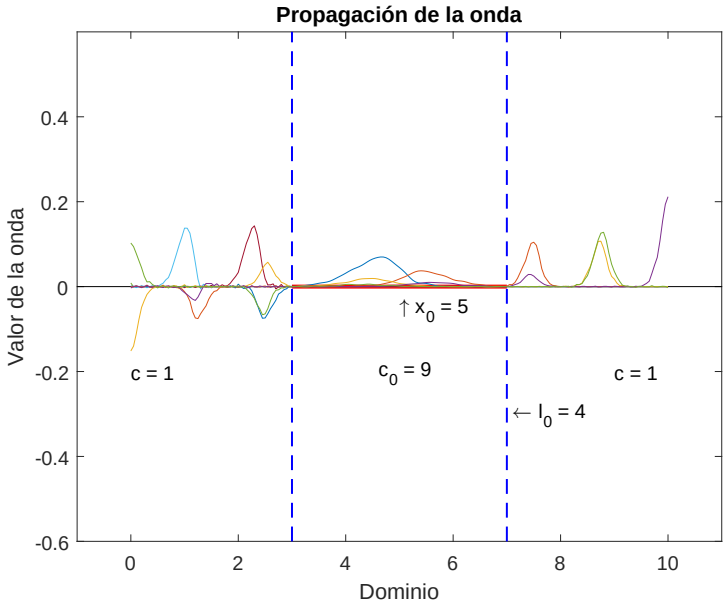


Figura 2.2: Varios perfiles de una misma onda para diferentes instantes temporales superpuestos sobre el dominio $[0, 10]$.

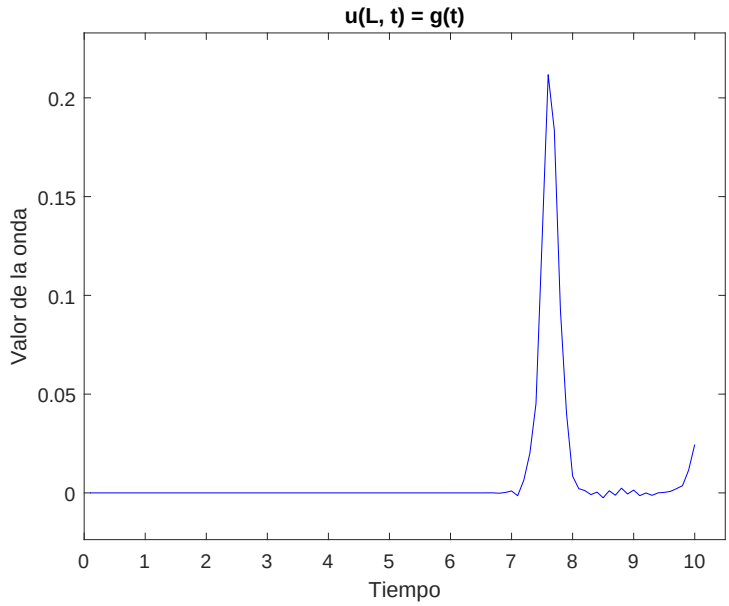


Figura 2.3: Representación de la función $g(t)$ obtenida al resolver la ecuación de onda de 2.2.

el medio externo, $c(x) = 1$, y en el objeto $c(x) = 9$. Podemos ver estas magnitudes reflejadas en la forma de propagarse de la onda, pues en el objeto esta "se aplana" dado que la velocidad de propagación es mayor en dicho medio.

2.2. Recopilación de datos

Los datos que vamos a guardar para cada onda son los valores que toma en el extremo derecho en un intervalo de tiempo, es decir, la función $u(L, t) = g(t)$, la cual representa la onda en el extremo final L, una vez ha atravesado el objeto colocado en la trayectoria. Con las configuraciones elegidas para dx y dt hemos generado un vector con 600 valores por cada una de las ondas fijando el tiempo de la simulación en $T = 10$. Sin embargo, por temas de espacio y de complejidad de la red neuronal se han tomado uno de cada dos puntos generados, por lo que acabamos guardando 300 puntos por cada onda al fijar el tiempo en $T = 10$.

El objetivo que perseguimos recopilando estos datos por cada onda es disponer de información suficiente para estimar dichas ondas a partir de los parámetros del objeto y viceversa mediante redes neuronales. Por ello, al generar los conjuntos de datos con la información de las ondas nos hemos asegurado de que todas ellas alcancen el final del intervalo.

En la figura 2.3 podemos ver la función $u(L, t)$ generada por la ecuación de onda representada anteriormente en 2.2. Como podemos identificar, el punto máximo de esta función se corresponde al momento exacto en el que la onda llega al extremo L, lo cual denominaremos como el primer rebote.

Los distintos parámetros de dicho objeto o intrusión definen el comportamiento de la onda resultante. En primer lugar, la velocidad de la onda en el objeto, c_0, define el momento en el que el primer rebote llega al final del intervalo y la forma de la misma, con mayor o menor amplitud debido a la aceleración sufrida a su paso por el objeto. La longitud del objeto, l_0, va a determinar cuánto tiempo va a ser acelerada la onda, por lo que influirá en el instante en la que el primer rebote llega al extremo L. Por último, la posición del objeto, x_0, no influirá de ninguna manera en el primer rebote de la onda en el extremo L, como veremos más adelante. Este parámetro solo influirá en los sucesivos rebotes de la onda en el extremo final, fruto de que estas golpeen el objeto y retornen al final de nuevo. Así, si el objeto se encuentra al inicio del intervalo, el segundo rebote tardará más tiempo en llegar al final que si estuviera situado más cerca del punto L.

En las figuras 2.1, 2.2 y 2.3 se ha empleado la siguiente configuración de parámetros para el objeto: $c_0 = 9$, $l_0 = 4$ y $x_0 = 5$. Esta configuración ha sido elegida por ser una de las más visuales al tener la onda una alta velocidad en el objeto y ser el tamaño del mismo lo suficientemente grande para que en 2.2 pueda verse representada la aceleración de la onda al atravesarlo. La elección de los parámetros c_0, l_0 y x_0 del objeto para generar los conjuntos de datos se trata en profundidad en la sección 2.4. En dicha sección se encontrarán los procedimientos y la metodología empleada para elegir los diferentes conjuntos.

Denominaremos entonces problema directo a hallar los valores medidos en el extremo libre a partir de los parámetros que definen al objeto, x_0, c_0 y l_0. Asimismo, denominaremos problema inverso a la búsqueda de los parámetros que definen el objeto a partir de los datos medidos.

2.3. Resolución numérica

Para resolver las ecuaciones de onda se ha empleado el método de elementos finitos. El primer paso de dicho método es discretizar el dominio en intervalos más pequeños. Como introducíamos anteriormente, se ha dividido el intervalo de definición de la onda $[0, L]$ en subintervalos usando $N = 100$ nodos con la idea de crear una malla. La solución de la ecuación de la onda queda aproximada de la siguiente manera:

$$u(x,t) = \sum_{i=1}^{N} \alpha_i(t)\phi_i(x) \tag{2.2}$$

de donde conocemos los $\phi_i(x), \quad i = 1, ..., N$, que son funciones de base del método de elementos finitos. Estas funciones son polinomios a trozos formados ensamblando polinomios de un grado prefijado en los extremos de cada subintervalo $[x_i, x_{i+1}]$, $i = 1, \ldots, N-1$. Se usan para aproximar la solución de la ecuación de onda dentro de cada intervalo evaluando en un número de nodos internos que se decide de antemano.

Los coeficientes de la aproximación (2.2) se determinan introduciendo esta expresión en una formulación variacional del problema (2.1) y aproximando las derivadas en

tiempo mediante el método de Newmark. Este método se basa en la aproximación de la aceleración de la estructura en función de la posición y la velocidad en cada instante de tiempo. Esto nos proporciona una relación de recurrencia vectorial que permite aproximar los pesos $\alpha_i(t_{k+1})$ conocidos los pesos $\alpha_i(t_k)$, $\alpha_i(t_{k-1})$, $i = 1, \ldots, N$ partiendo de $\alpha_i(t_1) = 0$, $\alpha_i(t_0) = 0$, $i = 1, \ldots, N$. Esta relación involucra coeficientes integrales definidos en términos de las funciones de base, que calculamos mediante cuadratura de Gauss-Legendre.

2.3.1. Resolución a nivel de código

Para resolver las ecuaciones de onda por el método de elementos finitos se ha modificado el código de MATLAB creado por Abdullah Wassem [2]. Para resolver las ecuaciones de onda con dicho código se realizan los siguientes pasos:

1. Descomponemos el dominio $[0, L]$ del problema en una serie de pequeños intervalos creando una malla x_i, $i = 1, \ldots, I$, $x_i = i\,dx$ como hemos visto anteriormente. Para ello empleamos la función `CreateMesh()`. Asimismo, mallamos el intervalo de tiempo $[0, T]$ con una secuencia de nodos t_k, $k = 0, \ldots, K$, $t_k = k\,dt$. Usamos como paso temporal $dt = 1/60$ y $dx = 1/10$, que cumplen la condición CFL de estabilidad $dt < \frac{1}{2} dx \cdot \text{mín}(c)$.

2. Aproximamos la solución, en cada uno de los intervalos temporales creados en el paso anterior, por el método de elementos finitos.

3. Calculamos las matrices de rigidez y de masa mediante el método de la cuadratura de Gauss-Legendre con 3 nodos. La matriz de masa (`Me`) representa la masa de cada intervalo en el sistema y la utilizamos para calcular la respuesta del sistema a una carga externa. La matriz de rigidez (`Ke`) representa la rigidez o resistencia de cada intervalo y la usamos para calcular las deformaciones en el sistema bajo carga externa. La matriz de fuerza (`Fe`) representa la fuerza aplicada a cada intervalo y la empleamos para calcular la carga externa en el sistema.

4. Llamamos a la función `Assembler()` para ensamblar las matrices de masa, rigidez y fuerza en una matriz global utilizando la información de conectividad entre los nodos de los intervalos finitos. La matriz resultante es una matriz tridimensional que contiene las matrices globales de masa (`barM`), rigidez (`barK`) y fuerza (`barF`) del sistema. Este paso es necesario para resolver la ecuación de onda en el sistema espacial completo, en vez de limitarnos a estudiar cada intervalo por separado.

   En este momento establecemos las condiciones de contorno del problema: el nodo-1 es el nodo Dirichlet y el último nodo es el nodo Neumann. El nodo Dirichlet es aquel que tiene una restricción de desplazamiento, de manera que queda fijo y es el origen del movimiento, donde se aplica la fuerza, mientras que el nodo Neumann se puede mover libremente. Este último nodo es el que vamos a estudiar y del cual vamos a guardar la posición en intervalos de tiempo fijos.

5. Se discretizan las derivadas temporales mediante el método de Newmark y se resuelven las recurrencias resultantes con paso dt en cada uno de los niveles hasta llegar al tiempo final T.

   En primer lugar, establecemos los parámetros del método de Newmark: gamma, parámetro que controla la amortiguación artificial introducida en el modelo numérico, y beta, el cual establece la precisión y estabilidad del método. Ambos son empleados para controlar la estabilidad de la solución.

   A continuación, se establece un vector de fuerza $f(x,t)$, el cual varía con cada instante de tiempo. Posteriormente, se utiliza el método de Newmark para resolver las ecuaciones de movimiento en cada instante de tiempo, calculando la posición de la onda a partir de la velocidad y la aceleración de los instantes anteriores y el actual. Para acelerar los cálculos, precalculamos una matriz dK, que se utiliza para procesar la matriz de rigidez y de masa del sistema. Esta matriz se combina con los términos de rigidez y de masa para formar una matriz del sistema que se puede resolver con un solo paso de la factorización LU.

2.3.2. Adaptación del código disponible

Como ya hemos mencionado, se ha modificado un código en Internet disponible con licencia MIT [2]. Esta modificación puede encontrarse en GitHub, donde se han ido incorporando todos los cambios de forma sucesiva al ir precisando conjuntos de datos con distintas características, pues el código del autor anterior únicamente obtenía los datos para una ecuación de onda sencilla [3].

Algunos de los cambios realizados al código para adaptarlo a las necesidades específicas del trabajo fueron:

1. Elaborar un código matriz para poder ejecutarlo en numerosas ocasiones y generar numerosas ondas de forma sucesiva.

2. Añadir el código necesario para guardar los datos en un archivo CSV, seleccionando el número de puntos a guardar por cada una de las ondas.

3. Modificar los parámetros para la resolución de las ecuaciones de onda y cómo estos eran interpretados en el código. Por ejemplo, las densidades de los materiales y las constantes de velocidad, pues no se correspondían con nuestro problema a modelar.

En los archivos CSV generados por MATLAB guardaremos la información relativa a las ondas. En las tres primeras columnas se guardarán los parámetros c_0, l_0 y x_0 del objeto y en las sucesivas se escribirá la imagen de la función $u(L,t)$, es decir, la imagen de la onda en el punto final del intervalo.

2.4. Elección de los parámetros del objeto c_0, l_0 y x_0

Para generar los conjuntos de datos con distintas ondas hemos variado los parámetros del objeto que situamos en el medio externo. Por ello, para calcular distintas ondas modificaremos la longitud del objeto, la posición del mismo y la velocidad de la onda en él. Con las redes neuronales intentaremos, en los capítulos sucesivos, estimar la onda resultante a partir de los parámetros del objeto y viceversa.

Los rangos en los que vamos a elegir los parámetros son los siguientes:

$$c_0 \in [1, 9] \quad l_0 \in [0'2, 5] \quad x_0 \in [3, 7]$$

Los tres parámetros del objeto han sido elegidos, dentro de su rango, mediante dos métodos distintos para comprobar si la elección de los datos afectaba al entrenamiento de la red neuronal. Con el primer método tomamos los parámetros de forma ordenada y equiespaciada, mientras que con el segundo se toman siguiendo un método casi aleatorio, el método del hipercubo latino.

2.4.1. Elección equiespaciada

Para obtener una configuración de parámetros equiespaciada tomamos el rango de cada uno de los parámetros $[a, b]$ y dividimos el intervalo en n subintervalos iguales. De esta manera, tomamos cada uno de los valores de los extremos de los subintervalos y realizamos todas las posibles combinaciones entre los extremos de los subintervalos de las distintas variables, obteniendo un total de $(n + 1)^3$ combinaciones distintas.

Esta forma de tomar valores para los tres parámetros es equivalente a realizar un mallado tridimensional de los parámetros y tomar aquellos valores en los que se cortan los tres valores. Dado que hemos tomado el mismo número de subintervalos para todas las variables y los valores de a y b son distintos para cada una de ellas, el valor de $h = \frac{b-a}{n}$ es distinto para los tres parámetros estudiados.

Mediante este método hemos ajustado el valor de n para conseguir conjuntos de alrededor de 1 000, 3 000, 10 000 y 30 000 ondas.

Con estas cantidades de datos hemos comprobado que la red neuronal alcanzaba una precisión suficientemente alta en sus estimaciones, por lo que no ha sido necesario obtener conjuntos de datos mayores.

2.4.2. Método del hipercubo latino (LHS)

Se han generado otros conjuntos de datos eligiendo los parámetros mediante el método del hipercubo latino (LHS por sus siglas en inglés). Los rangos para las variables son los mismos que en el caso de la malla equiespaciada. Por este método de mallado se han obtenido cinco conjuntos de datos distintos: uno con 1 000 ondas, otro con 3 000, otro con 10 000 y otro con 30 000 ondas; el último conjunto ha sido generado únicamente

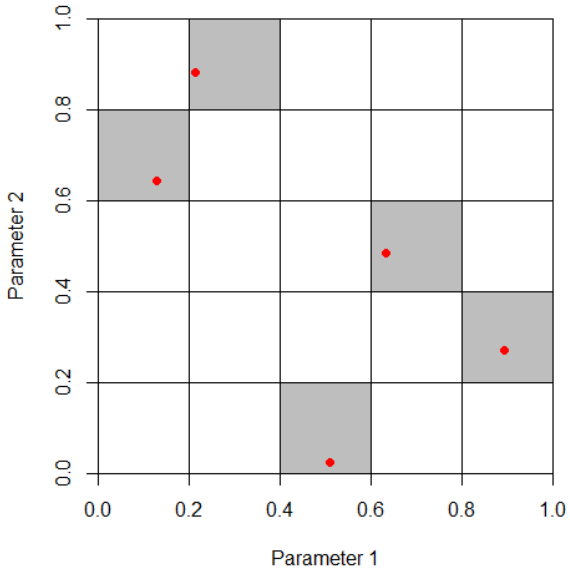


Figura 2.4: Elección de valores para el Método del Hipercubo Latino. Como vemos, por cada fila y cada columna solo se ha elegido una combinación de valores.

para probar las redes neuronales con un mayor número de ondas y contiene información de 100 000 ondas. La elección de este número de muestras ha sido para poder comparar los resultados del entrenamiento al elegir uno u otro método de mallado de datos.

El método LHS se basa en dividir el rango de valores de cada una de las tres variables de entrada en N intervalos iguales (si buscamos generar N combinaciones de valores de los parámetros) para después seleccionar un valor aleatorio dentro de cada uno de los intervalos, de forma que cada posible combinación de los valores de entrada tiene la misma posibilidad de ser seleccionada. Es decir, el método LHS garantiza una distribución uniforme de valores de entrada en todo el espacio de parámetros.

Como vemos en la figura 2.4, supongamos que tenemos dos variables de entrada, X e Y, que toman valores en el rango de 0 a 1. Para crear un conjunto de muestras de LHS de tamaño 5, primero dividimos el rango de valores de X y Y en 5 intervalos iguales. Luego, seleccionamos un valor aleatorio dentro de cada intervalo para cada variable de entrada. Finalmente, combinamos estos valores seleccionados de tal manera que cada posible combinación de valores tenga la misma probabilidad de ser seleccionada.

Capítulo 3

Redes convolucionales

Las redes neuronales convolucionales (CNN) se han convertido en una herramienta popular en el campo del aprendizaje profundo debido a su capacidad para procesar datos en dos y tres dimensiones. Sin embargo, hay aplicaciones donde los datos de entrada tienen una dimensión, como en el procesamiento de señales de audio o series de tiempo financieras. Para estos casos, las redes neuronales convolucionales 1D (CNN 1D) son una variante de las CNN que se han utilizado con éxito para extraer características de datos secuenciales y de una dimensión.

En este capítulo se proporciona una descripción general de las redes neuronales convolucionales 1D, incluida la base matemática detrás de su funcionamiento y su aplicación en diferentes campos.

3.1. Descripción general de las redes 1D

Estas redes son una variante de las redes neuronales convolucionales que se utilizan para procesar datos de entrada secuenciales y de una dimensión. En las redes CNN 1D diferenciamos las siguientes capas:

- Capa convolucional: esta capa es la que da nombre a este tipo de redes y utiliza filtros convolucionales de una dimensión para extraer características de los datos de entrada. Cada filtro convolucional es una pequeña ventana deslizante que se mueve a lo largo de los datos de entrada y aplica un producto escalar entre dos matrices, siendo una de ellas el conjunto de parámetros, los cuales son modificados a lo largo del aprendizaje. El resultado de esta operación es una nueva matriz de pesos que representa las características de los datos de entrada que son relevantes para el aprendizaje.
- Capa de agrupamiento o *pooling*: esta capa se emplea para reducir el tamaño espacial de la matriz de pesos, disminuyendo el cómputo requerido en los cálculos, pero conservando las características importantes. Se suele colocar después de las capas convolucionales. Esta capa emplea una ventana deslizante para tomar el máximo

o el promedio de los valores de características dentro de la ventana, aunque se suele emplear el máximo para reconocer patrones en los datos y conservarlos tras la operación. El resultado es una nueva matriz de pesos con un tamaño espacial reducido. La capa de agrupamiento también puede ayudar a reducir el riesgo de sobreajuste al reducir el número de parámetros de la red y aumentar la tolerancia a la variación de la posición de las características en la entrada.

- Capa de activación: esta capa se utiliza para introducir no linealidad en la red y permite el análisis de datos complejos. Se suele colocar después de capas convolucionales y de capas totalmente conectadas. La función de activación más común es la función ReLU (Rectified Linear Unit), que ha sido la utilizada en este proyecto y aplica la función $f(x) = \text{máx}(0, x)$ a la matriz de características. La importancia de esta capa reside en que permite a la red aprender funciones no lineales y mejorar la capacidad de la misma para modelar datos complejos.
- Capa de aplanamiento: esta capa se utiliza para transformar la matriz de parámetros o características generada por las capas convolucionales y de agrupamiento en un vector de una dimensión que puede alimentar a una capa densa. La capa de aplanamiento simplemente toma la matriz de características y la transforma en un vector 1D colocando todos los elementos de la matriz en una sola dimensión. De esta manera, se puede utilizar una capa densa para realizar una clasificación o regresión en función de las características extraídas por las capas convolucionales y de agrupamiento.
- Capa densa o totalmente conectada: esta capa se utiliza para buscar una relación entre las capas de entrada y de salida, en función de las características extraídas por las capas convolucionales y de agrupamiento. Es similar a una red neuronal multicapa tradicional, donde cada neurona está conectada a todas las neuronas de la capa anterior y a todas las neuronas de la capa posterior. Las capas densas son las que realmente clasifican los datos de salida y generan las predicciones.

3.2. Operaciones en las capas

Las redes neuronales convolucionales se basan en el concepto de la convolución discreta, basado en un proceso secuencial de multiplicación de matrices. A la señal de entrada se le aplica una operación de convolución con un filtro de pesos, que es el encargado de extraer las características de la entrada y que se desplaza a lo largo de la señal. Cada filtro de convolución produce una nueva señal de salida, que se convierte en la entrada de la siguiente capa de la red. Este filtro de pesos es el que modificamos con cada iteración en el aprendizaje de la red.

La operación de convolución discreta se puede representar matemáticamente como:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k] \tag{3.1}$$

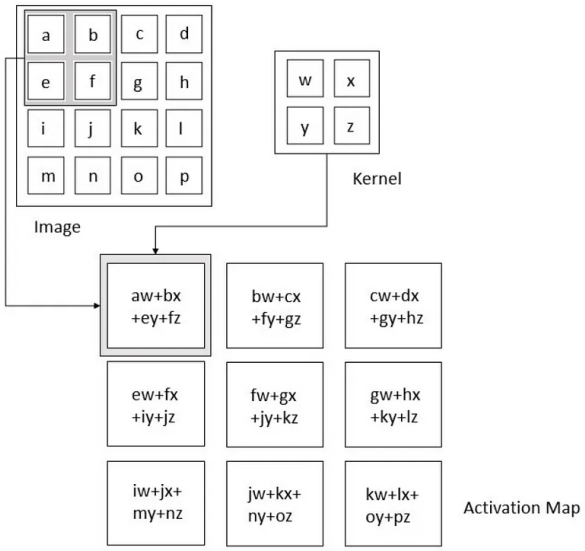


Figura 3.1: Operación de convolución [4].

donde $x[k]$ es la señal de entrada, $h[n-k]$ es el filtro de convolución (que modificamos con cada iteración de aprendizaje de la red) e $y[n]$ es la señal de salida.

Además de la convolución discreta, las CNN 1D también utilizan capas de agrupamiento para reducir la dimensionalidad de los datos de entrada y evitar el sobreajuste de la red, como vimos anteriormente. La operación se puede representar matemáticamente como:

$$y[i] = f(x[j]) \tag{3.2}$$

donde $y[i]$ es el valor de salida de la capa de agrupamiento, $x[j]$ es el valor de entrada para la región j y f es la función de agrupamiento, que suele ser la función de máximo o la de promedio, junto con otras menos utilizadas. Debemos tener en cuenta que de esta manera reducimos la dimensión de la matriz $y[i]$, tantas veces como el tamaño de la región j que estemos considerando en la operación.

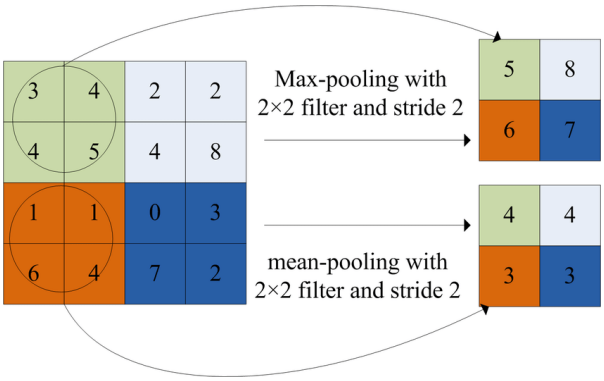


Figura 3.2: Operación de agrupamiento con una matriz de filtro de tamaño 2x2 [5].

Las capas convolucionales y de agrupamiento suelen aparecer juntas en una CNN. En cada capa convolucional se pueden utilizar varios filtros de convolución para extraer

varias características de los datos de entrada. La salida de cada capa convolucional se pasa a una capa de agrupamiento, reduciendo así la dimensionalidad de los datos y el coste de procesamiento en las capas más profundas de la red.

En general, la operación de una red neuronal convolucional de una dimensión puede representarse matemáticamente como:

$$y = f_L(f_{L-1}(...f_2(f_1(x * h_1) \oplus p_1) * h_2) \oplus p_2) * ... * h_L \oplus p_L \tag{3.3}$$

donde y es la salida de la red, x es la señal de entrada, h_i son los filtros de convolución en la capa i, p_i son las capas de agrupamiento en la capa i y f_i es la función de activación en la capa i.

3.3. Aplicaciones

Las redes neuronales convolucionales 1D se utilizan comúnmente en una amplia variedad de aplicaciones, especialmente en procesamiento de señales. El uso generalizado de las CNN viene motivado principalmente por su capacidad inherente para unir la extracción de características y la clasificación en un mismo modelo de aprendizaje. Algunos de los usos más comunes de estas redes son [6]:

- Reconocimiento del lenguaje natural hablado: el propósito de esta aplicación es el procesado en tiempo real y la transcripción del habla en palabras escritas. Sin embargo, esta tarea aumenta considerablemente de dificultad si hay cierto nivel de ruido ambiental o se tienen en cuenta los distintos dialectos dentro de un mismo lenguaje. En este ámbito hay varios avances, donde se han empleado operaciones de convolución sobre fragmentos de audio para clasificar si estos provienen de un teléfono, un altavoz, o incluso el género del emisor. De esta forma, tras esta primera clasificación, otras redes neuronales específicas pueden ser empleadas para el reconocimiento.

- Monitorización de un electrocardiograma en tiempo real: el electrocardiograma es una prueba no invasiva que permite reconocer ciertos problemas del corazón, como las arritmias, que pueden ser el inicio de otros problemas más graves. El estudio de un electrocardiograma debe ser realizado por un médico especializado, siendo su interpretación parcialmente subjetiva y la cual consume una gran cantidad de tiempo. La monitorización de arritmias gracias a las CNN es posible. Además, estas son entrenadas con datos específicos de cada paciente. Con estas técnicas se han conseguido identificar las arritmias en pacientes con más de un 97 % de precisión.

- Detección de daños estructurales por medio de vibraciones en infraestructuras: estos métodos de detección de daños emplean una serie de vibraciones controladas en las infraestructuras y, a partir de la respuesta, permiten identificar y localizar los daños estructurales. Recientemente, se han comenzado a aplicar novedosos algoritmos para detectar dichos daños. Sin embargo, algunos estudios han comprobado

que el empleo de redes convolucionales 1D y 2D facilitan el trabajo enormemente, aunque lo habitual es emplear las redes de una dimensión por su menor coste computacional.

- Algunas otras aplicaciones incluyen el reconocimiento de tareas en personas por medio de acelerómetros. Un ejemplo de estas tareas serían caminar, subir escaleras y correr. Las CNN también permiten detectar fallos en maquinaria rotatoria, monitorizando la degradación de los rodamientos por medio del análisis de los ruidos y termografía infrarroja.

3.4. Preprocesado de los datos para entrenar la red

Tras cargar los datos para el entrenamiento del modelo, debemos normalizar las cantidades antes de ser procesadas por la red. Estas normalizaciones son habitualmente utilizadas para evitar que en el proceso de entrenamiento las redes no se queden atascadas en mínimos locales. Además, si los datos de entrada están en escalas distintas, estos pueden tener un mayor efecto en el modelo al realizar operaciones de multiplicación de pesos. Esto puede afectar negativamente a la capacidad del modelo para aprender patrones en los datos y tener un rendimiento deficiente, pues no tiene en cuenta por igual a todos los valores al generar los resultados, y son necesarios más ciclos de entrenamiento para un resultado óptimo [7].

En nuestro caso, la normalización elegida es la *z-score* o estandarización, que implica restar la media de los datos y dividir por la desviación estándar, llevando los datos a una distribución normal de media cero y desviación estándar de uno.

Para los datos relativos a las imágenes de las funciones $u(L,t)$ se ha aplicado una normalización a todos los datos de todas las ondas al mismo tiempo. Sin embargo, para los parámetros del objeto, hemos aplicado una normalización por columnas; es decir, se han tratado los datos independientemente para c_0, l_0 y x_0, de manera que los datos se distribuyan en el mismo intervalo a la hora de entrenar la red neuronal. Este paso es importante para los tres parámetros, pues si uno de ellos tiene valores superiores a los demás corremos el riesgo de que la red lo pondere positivamente y lo considere más importante para los resultados.

Los conjuntos de datos han sido mezclados de forma aleatoria y divididos en tres conjuntos para el entrenamiento de las redes convolucionales. Se ha separado un primer conjunto con el 60 % de los datos, que es el que va a ser usado para entrenar la red; el siguiente 28 % va a ser el conjunto de validación, que nos servirá para controlar la función de pérdida durante el entrenamiento; mientras que el 12 % restante será el conjunto de prueba para comprobar el rendimiento de la red con datos con los que no ha sido entrenada.

3.5. Cálculo de errores de la red

Para medir la precisión de las redes neuronales se ha calculado, además del valor de la función de pérdida, el error absoluto y el error respecto al rango de los canales de salida de las redes. Para el cálculo del error absoluto se han transformado los datos a su escala original, de manera que podamos visualizar correctamente los errores a simple vista sin cálculos adicionales y podamos compararlos para distintas redes neuronales y conjuntos de datos a lo largo del proceso de aprendizaje.

El error absoluto ϵ ha sido empleado con una finalidad informativa en el entrenamiento y mide la diferencia absoluta entre el valor predicho, $\hat{y}_i$, y el valor real, y_i, de los parámetros de salida. En este caso, calculamos la diferencia entre el valor real y el valor predicho y dividimos la suma de estas diferencias entre el número de predicciones n, mediante la fórmula:

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

Por otro lado, el error respecto al rango ϵ_r nos aporta una medida más eficaz a la hora de comparar distintas estimaciones de la red neuronal al ser entrenada con distintos conjuntos de datos, pues en este caso tenemos en cuenta el rango en el que se encuentran los datos de entrada, de manera que medimos la diferencia entre el valor predicho y el valor real en relación con el rango de los datos. La fórmula utilizada para calcular el error respecto al rango es:

$$\epsilon_r = \frac{1}{n}\sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{abs(max(y_i) - min(y_i))}$$

donde n es el número total de valores de salida, y_i es el valor real correspondiente al i-ésimo valor de salida e $\hat{y}_i$ es el valor predicho por el modelo correspondiente al i-ésimo valor de salida.

Capítulo 4

Red convolucional para el problema directo

En este capítulo vamos a explicar el diseño de la red convolucional empleada para hallar las imágenes de la onda a partir de los parámetros del objeto. Es decir, la red convolucional recibirá los tres valores del objeto, por lo que será una red neuronal con 3 canales de entrada, y generará los 300 puntos que forman la imagen de la función $u(L, t)$ de la onda recibida, por lo que tendrá 300 canales de salida.

El código referenciado en este capítulo está disponible en GitHub [8] en la carpeta `Models`. Ahí podemos encontrar la red neuronal desarrollada para el problema directo. Las tareas de programación han sido desarrolladas en Python [9], haciendo uso del IDE PyCharm y empleando la librería PyTorch [10] para la creación de las redes convolucionales.

4.1. Diseño de la red

Para estimar las ondas generadas a partir de unos parámetros del objeto que situamos en el medio externo, se ha diseñado una red neuronal con 3 canales de entrada y 300 canales de salida. El error relativo de este modelo es bastante reducido (menor al 1 %) y, como veremos en la sección 4.3, permite identificar el momento en el que la onda llega al final del intervalo donde la estamos estudiando.

Esta red está compuesta por las siguientes capas:

1. Una capa totalmente conectada con 3 canales de entrada y 200 canales de salida. La función de esta capa es la de extender los datos de entrada para poder seleccionar aquellos que tengan más peso en los resultados.
2. Una capa de activación ReLU para romper la linealidad.
3. Una capa convolucional de una dimensión con tamaño de ventana 3, obteniendo los datos de entrada más relevantes en el proceso de aprendizaje.

4. Una capa de agrupamiento en la que reducimos los canales a 100 y tomamos el máximo de los valores dentro de la ventana, en este caso de tamaño 2.
5. Una capa de activación ReLU.
6. Una capa de aplanamiento para transformar la matriz de entrada en un vector de una dimensión con la finalidad de reducir el trabajo de la red. Sin este paso no es posible disponer del modelo de entrenamiento en la memoria gráfica del ordenador.
7. Una capa totalmente conectada con 100 canales de entrada y 200 canales de salida.
8. Una capa de activación ReLU.
9. Una capa totalmente conectada con 200 canales de entrada y 300 canales de salida.

Esta red es relativamente sencilla y nos permite entrenar con todo el conjunto de datos en una misma iteración. A la luz de los resultados, se ha considerado que no era necesario cargar los datos en bloques para alimentar una red más compleja con una precisión mayor.

Para diseñar las capas de esta red convolucional se ha tenido en cuenta el diseño de otras redes convolucionales populares empleadas en el reconocimiento de imágenes 2D como AlexNet y LeNet [11]. Algunas de las ideas extraídas de estas redes para el diseño de nuestro modelo han sido, entre otras, el colocar al inicio del modelo las convoluciones y capas de agrupamiento, así como el hecho de situar las capas totalmente conectadas al final del mismo. En este caso hemos usado dos capas completamente conectadas al final de la red para mejorar el resultado obtenido, como también se hace en otras redes como AlexNet.

4.2. Ajuste de los hiperparámetros

Los hiperparámetros son una serie de variables de configuración de las redes neuronales que administran el entrenamiento de los modelos de aprendizaje automático. Se configuran de forma manual antes de entrenar el modelo y no debemos confundirlos con los parámetros, que son los elementos internos de la red como los pesos de las variables.

Los hiperparámetros controlan directamente la estructura y el funcionamiento de los modelos. Su ajuste nos permite modificar el rendimiento y la eficacia de la red para lograr resultados óptimos. El proceso de ajuste de los hiperparámetros es iterativo y se deben probar distintas combinaciones para entender cómo afectan a la precisión del modelo [12].

En nuestro caso, vamos a ajustarlos manualmente, fijando en 3 000 el número de épocas (recorrido completo de la red por el conjunto de entrenamiento) y variando tanto la tasa de aprendizaje como la inercia, dos de los parámetros más comunes. El propósito de este ajuste es minimizar el valor devuelto por la función de pérdida de la red, que nos indica cuánto nos hemos equivocado con nuestras predicciones y cuyo valor mostraremos

en las figuras de esta sección. Los valores para los cuales pondremos a prueba la red con los hiperparámetros son los recomendados en la bibliografía especializada [4].

La tasa de aprendizaje controla la velocidad a la que el modelo aprende. Específicamente, controla el ritmo con el que el algoritmo modifica los pesos de cada variable al finalizar cada ciclo de entrenamiento, aprendiendo según su desempeño.

Generalmente, una alta tasa de aprendizaje permitirá al modelo aprender más rápidamente, pero con la contrapartida de tener, al final del proceso de entrenamiento, un conjunto de pesos subóptimo. Una tasa de aprendizaje reducida permite al modelo llegar a una configuración de los pesos más óptima o, incluso, globalmente óptima, pero va a requerir un mayor tiempo de entrenamiento [4, 13].

Para fijar la tasa de aprendizaje en nuestro entrenamiento, se ha calculado la pérdida con los conjuntos de validación a lo largo del entrenamiento. En este caso, se ha entrenado el modelo variando el mencionado hiperparámetro, empleando como conjunto de entrenamiento y de validación aquellos que contienen información de 30 000 ondas, tanto en el que elegimos los parámetros de forma equiespaciada como en el que usamos el método del hipercubo latino. Por otro lado, los distintos valores para la tasa de aprendizaje se han seleccionado con escala logarítmica, como se menciona en la bibliografía [4].

Los resultados de este estudio pueden ser observados en las figuras 4.1 y 4.2. Como vemos, apenas hay diferencias en la pérdida entre los conjuntos de validación en ambos casos. Sin embargo, sí hemos podido comprobar que la mejor elección en cuanto a tasa de aprendizaje sería el valor $lr = 0{,}5$ en ambos casos. Aunque omitimos las pruebas realizadas para otros conjuntos de datos con un menor número de ondas, los resultados obtenidos eran similares, por lo que las conclusiones extraídas para este hiperparámetro son las mismas.

La irregularidad presente en las gráficas indica que el modelo aprende pasando por distintas colinas y valles en su aprendizaje, buscando mínimos globales. En este proceso, que en algunas ocasiones puede parecer errático, podemos aportar algo de inercia al aprendizaje por si el modelo encontrase una dirección en la que reduzca su función de pérdida y tenga una tendencia a seguir por dicho camino. De esta forma aceleraremos el proceso. Esta inercia puede conseguirse modificando el otro hiperparámetro anteriormente mencionado, α, llamado inercia o *momentum*.

Este hiperparámetro puede acelerar el proceso de aprendizaje en aquellos problemas en los que el "espacio de pesos" que está siendo navegado por el proceso de optimización tiene estructuras que engañan al algoritmo de descenso de gradiente, como regiones planas o de curvatura pronunciada. En nuestro caso nos evita que nuestro modelo se estanque y deje de oscilar, además de mejorar el aprendizaje.

Los valores que hemos elegido para comprobar el aprendizaje de nuestro modelo con inercia son los más empleados en la práctica: $\alpha = 0{,}5$, $\alpha = 0{,}9$ y $\alpha = 0{,}99$ [4]. En

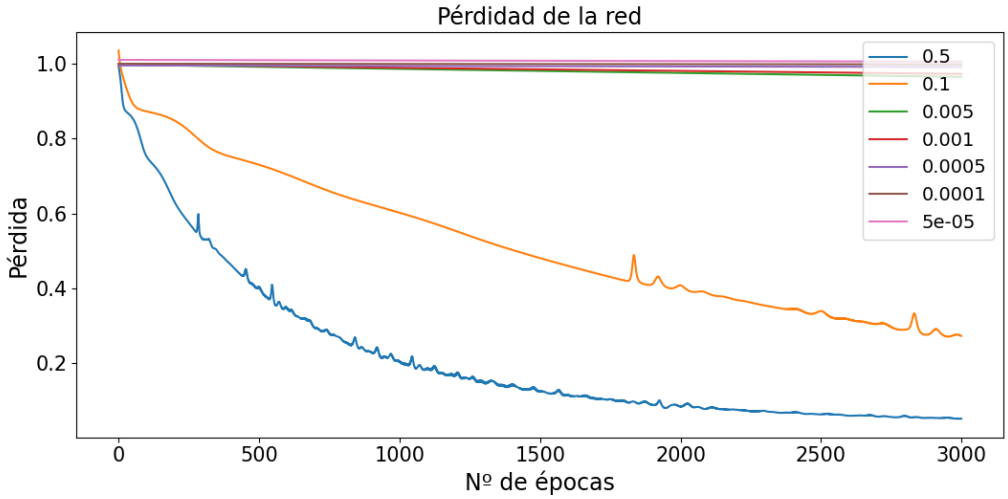


Figura 4.1: Pérdida de la red con el conjunto de entrenamiento equiespaciado y variando la tasa de aprendizaje.

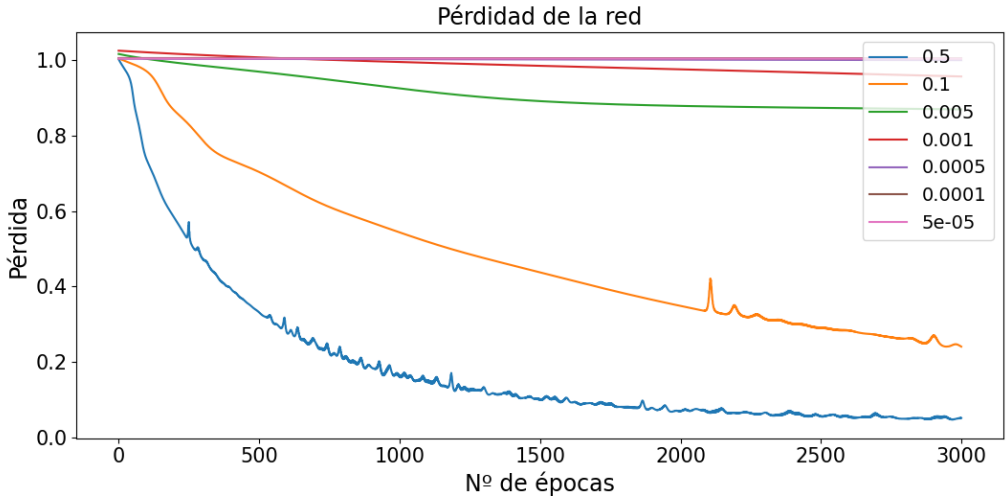


Figura 4.2: Pérdida de la red con el conjunto de entrenamiento elegido por método LHS y variando la tasa de aprendizaje.

la figura 4.3 podemos observar el valor de la función de pérdida para distintas tasas de aprendizaje tomando los valores anteriores para la inercia. Como ya se ha mencionado, hemos decidido entrenar el modelo con 3 000 épocas pues, en vista de los resultados expuestos en las figuras, es suficiente para obtener unos buenos resultados sin caer en un sobreaprendizaje del modelo.

A la vista de los resultados de la pérdida de la red, se decidió emplear la configuración con tasa de aprendizaje $lr = 0{,}1$ e inercia $\alpha = 0{,}99$, a pesar de no ser la curva más pronunciada del modelo (esta sería aquella con $lr = 0{,}5$ y $\alpha = 0{,}9$), pues la pérdida con $\alpha = 0{,}9$ presenta algunas irregularidades con las tasas de aprendizaje que estamos empleando que pueden perjudicarnos con otros conjuntos de datos.

4.3. Resultados de la red convolucional

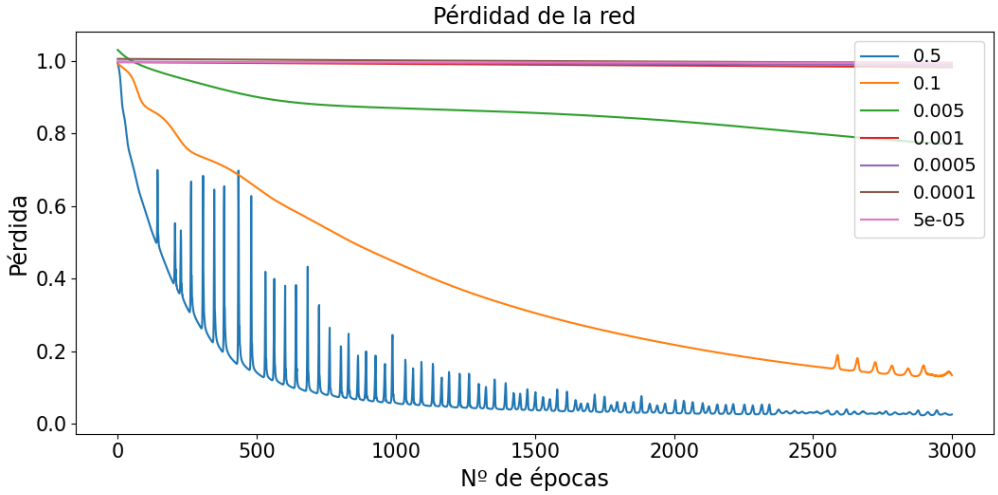
En este capítulo vamos a visualizar los resultados producidos por la red convolucional anterior para todos los conjuntos de datos, tanto los hallados por el método equiespaciado, como los generados con el método del hipercubo latino.

Para generar las figuras 4.4 y 4.5 se ha entrenado la red neuronal con el conjunto equiespaciado de 10 000 ondas y se han generado las gráficas poniendo a prueba dicha red con el conjunto de 1 000 ondas generadas por LHS (únicamente con el 12 % empleado como conjunto de prueba). En ambas figuras podemos ver representada la función $u(L, t)$ para cada una de las 120 ondas utilizadas al probar el modelo, de forma que cada fila se corresponde con una onda y el color de cada uno de los 300 puntos que conforman cada fila representa el valor de la onda en dicho punto.

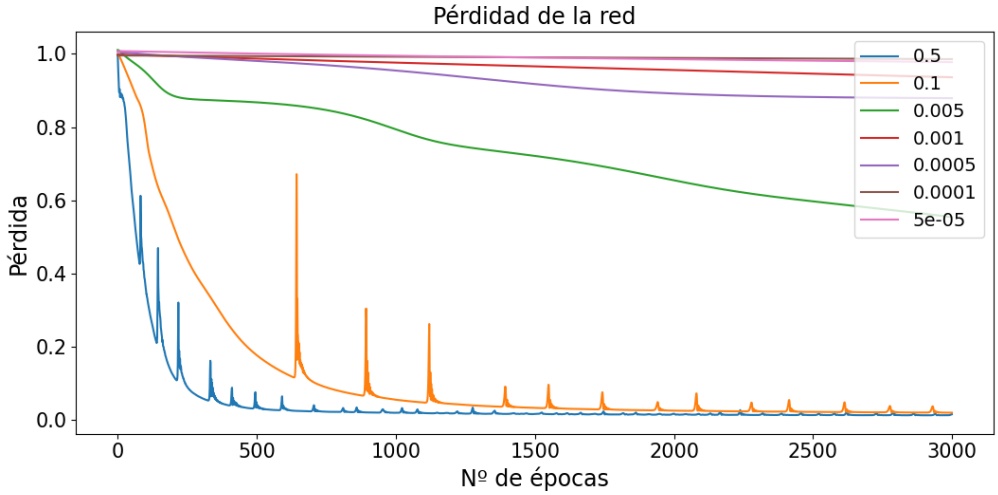
En la figura 4.4 podemos visualizar los resultados calculados mediante el método de elementos finitos en el programa MATLAB, a la izquierda, frente a los resultados de la red neuronal, a la derecha. La escala de color en ambas imágenes es la misma y, como podemos comprobar a simple vista, el momento en el que la onda llega al final del intervalo, lo que consideramos como primer rebote, está perfectamente estimado por la red.

Para visualizar correctamente el error cometido por la red neuronal en los mapas de calor 4.4, mostramos en la figura 4.5 la diferencia entre los dos mapas anteriores, reduciendo la escala del error a $[-0{,}5, 0{,}5]$ para mejorar la visualización. En esta nueva figura podemos observar que la red comete errores al estimar la amplitud del primer rebote de la onda, pero estima correctamente cuándo este va a ocurrir. Para complementar este razonamiento mostraremos más adelante algunas ondas estimadas por los modelos, comparando individualmente su valor real con el estimado gráficamente.

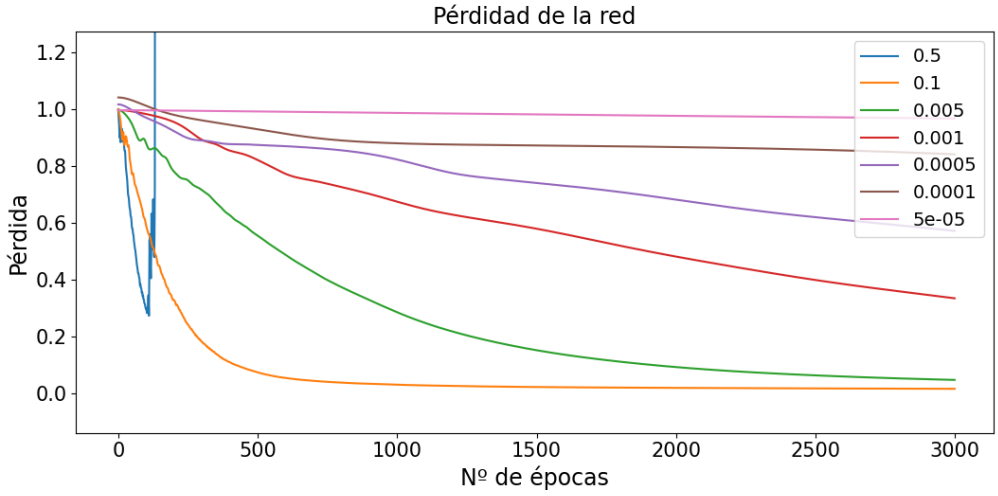
Para ver en términos numéricos la efectividad del modelo, se ha calculado cuántos puntos de la predicción se han desviado más de un 5 % respecto al valor real. El porcentaje de desviación se ha obtenido respecto al rango de todas las ondas en el conjunto de



(a) $\alpha = 0{,}5$.



(b) $\alpha = 0{,}9$.



(c) $\alpha = 0{,}99$.

Figura 4.3: Comparación de la pérdida del modelo para distintos valores del hiperparámetro α, variando la tasa de aprendizaje para cada uno de los valores de α.

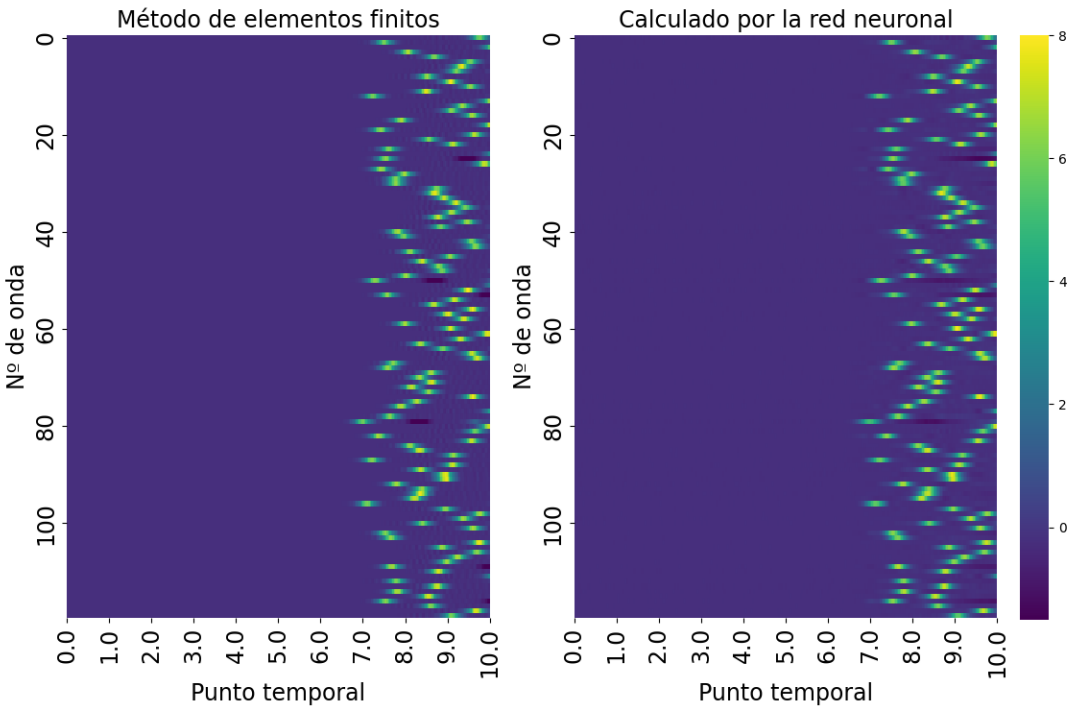


Figura 4.4: Conjunto calculado por el método de elementos finitos frente a los resultados de la red neuronal.

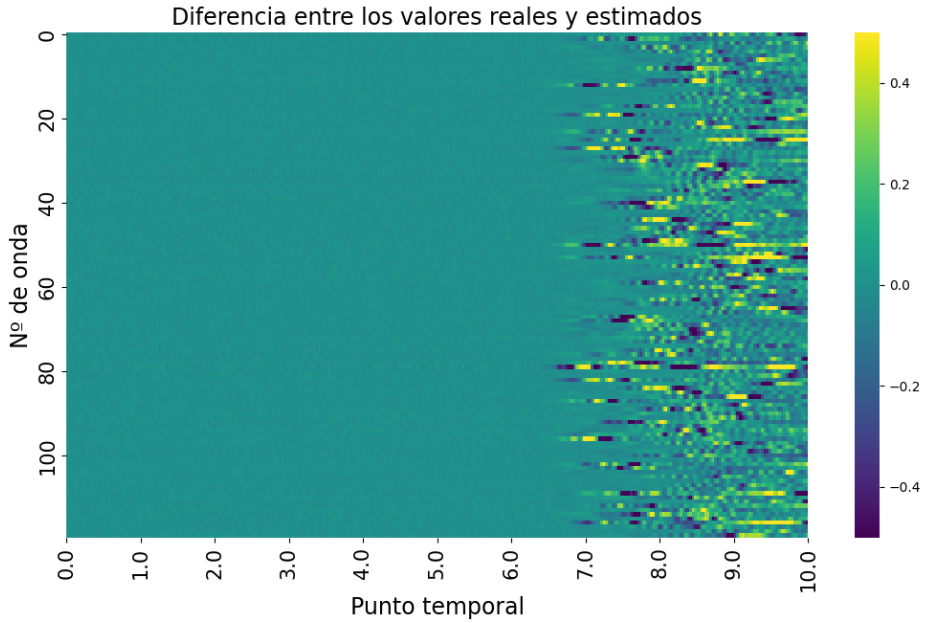


Figura 4.5: Mapa de calor con las diferencias entre los dos conjuntos de datos de la figura 4.4, reduciendo la escala a $[-0{,}5, 0{,}5]$.

prueba, es decir, se ha calculado el valor máximo y el mínimo del conjunto y, a partir de la diferencia de estos valores, se admite como mucho una tolerancia de un 5 % respecto a dicha diferencia. Con este cálculo, podemos concluir que únicamente el 2,54 % de los valores se han desviado más de un 5 % en el ejemplo anterior.

Tomando todos los conjuntos de ondas de los que disponemos y entrenando la red neuronal para todos ellos se han obtenido los datos del error de la red para los distintos conjuntos de entrenamiento. En los cuadros 4.1 y 4.2 representamos los errores resultantes de entrenar la red con los distintos conjuntos de datos. En la mitad superior mostramos los conjuntos empleados para entrenar la red, de manera que cada columna representa el error de una red neuronal entrenada con un determinado conjunto de datos (indicado en el encabezado). En la fila indicamos el conjunto de datos utilizado para probar el modelo entrenado, en este caso únicamente hemos utilizado el conjunto generado por LHS con 100 000 ondas, el cual usamos en su totalidad para probar las redes. Además, en el entrenamiento hemos dividido los conjuntos utilizados, de manera que empleamos un 60 % de los datos para entrenar la red, un 28 % para validar los datos y el 12 % restante es el que empleamos para probar las redes. El conjunto de prueba de 100 000 ondas no se ha dividido, pues ninguna de sus ondas ha sido empleada para entrenar la red neuronal, pero recorren todo el espacio de parámetros. De esta forma, se ha conseguido una visión muy acertada de la capacidad de aprendizaje y la precisión de los resultados.

Los valores mostrados en los cuadros 4.1 y 4.2 representan el porcentaje de los puntos estimados por la red con un error respecto al rango total de dichos valores superior al 5 %. Como podemos observar, con el mismo número de ondas, al entrenar la red con un conjunto de datos generado por el método del hipercubo latino obtenemos una precisión superior a la hora de estimar las ondas respecto a los conjuntos equiespaciados.

Desviación >5 %	Entrenamiento	Equiespaciado			
Prueba	Nº de ondas	1 000	3 000	10 000	30 000
LHS	100 000	4.07 %	1.38 %	0.74 %	0.74 %

Cuadro 4.1: Porcentaje de los puntos con un error superior al 5 % al comparar la red neuronal entrenada con los conjuntos de entrenamiento equiespaciados y el conjunto de 100 000 ondas generado por LHS.

Desviación >5 %	Entrenamiento	LHS			
Prueba	Nº de ondas	1 000	3 000	10 000	30 000
LHS	100 000	1.50 %	0.75 %	0.74 %	0.64 %

Cuadro 4.2: Porcentaje de los puntos con un error superior al 5 % al comparar la red neuronal entrenada con los conjuntos de entrenamiento generados por LHS y el conjunto de 100 000 ondas generado por este mismo método.

El conjunto de ondas utilizado en el entrenamiento de la red no es el único que interviene en estos resultados, sino que debemos elegir correctamente los hiperparámetros

para garantizar un buen funcionamiento y aprendizaje del modelo, como ya hemos visto en la sección 4.2. Para todos estos casos hemos empleado la misma configuración con una tasa de aprendizaje $lr = 0{,}1$ y una inercia $\alpha = 0{,}9$, pues hemos observado que ofrecía unos resultados óptimos al entrenar todos los conjuntos. Además, dado que en la elección de los pesos iniciales en las capas convolucionales de la red intervienen algunos procesos aleatorios, en algunos casos hemos tenido que volver a entrenar el modelo, pues en el conjunto de validación teníamos una pérdida inusualmente grande, lo que nos indicaba que el entrenamiento se había estancado en un mínimo local.

A continuación mostraremos una serie de comparaciones entre las ondas estimadas por las redes neuronales al ser entrenadas con los distintos conjuntos de datos y al ser probadas con el conjunto LHS con 100 000 ondas. El valor calculado para discernir entre una buena y una mala estimación ha sido el error absoluto medio cometido en la estimación, calculado por cada onda. Así, en todos los casos hemos considerado la onda para la cual se ha cometido un menor error y aquella en la que el error era mayor.

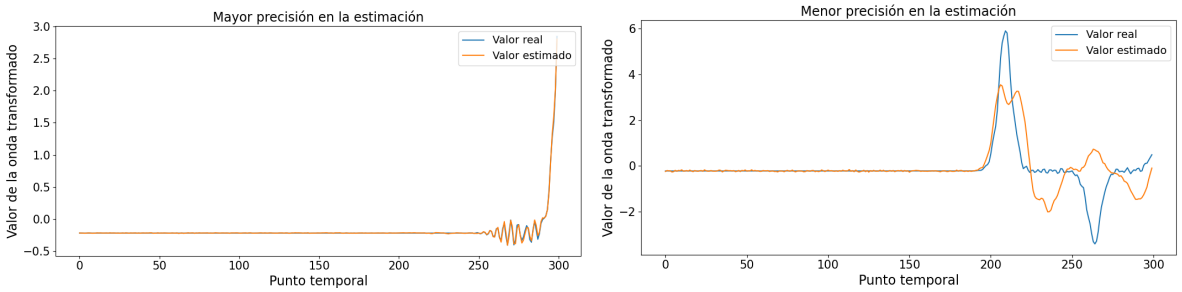


Figura 4.6: Ondas estimadas con mayor y menor precisión por la red entrenada con el conjunto equiespaciado de 1 000 ondas.

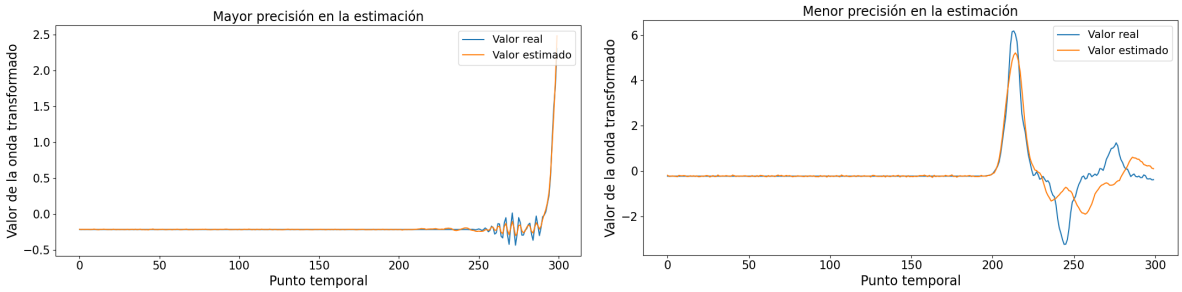


Figura 4.7: Ondas estimadas con mayor y menor precisión por la red entrenada con el conjunto equiespaciado de 3 000 ondas.

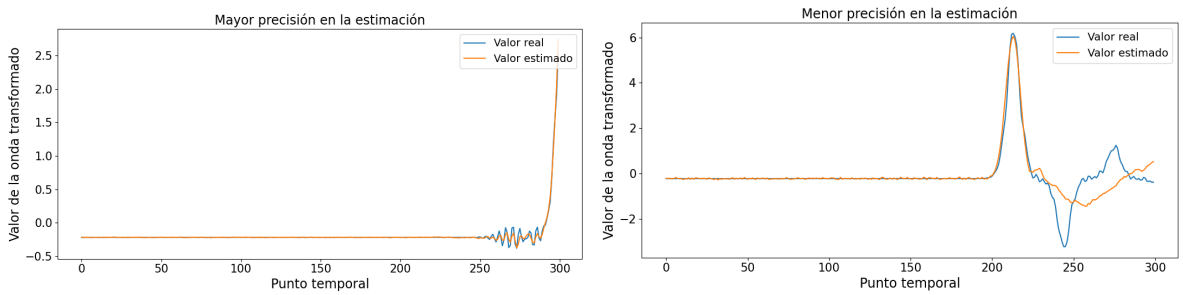


Figura 4.8: Ondas estimadas con mayor y menor precisión por la red entrenada con el conjunto equiespaciado de 10 000 ondas.

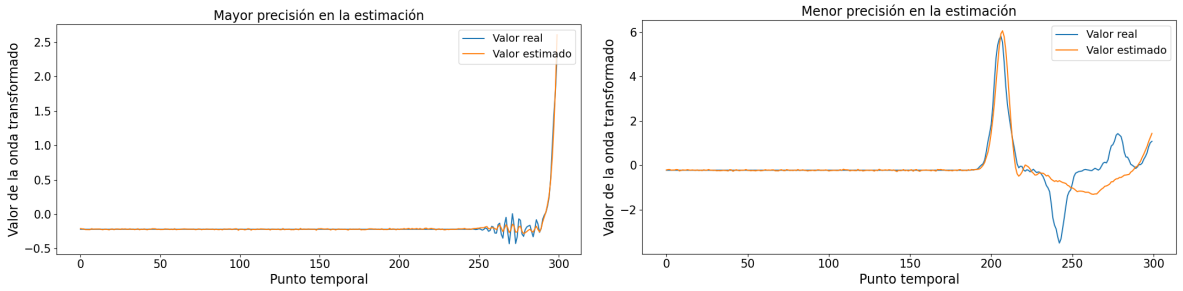


Figura 4.9: Ondas estimadas con mayor y menor precisión por la red entrenada con el conjunto equiespaciado de 30 000 ondas.

En las figuras 4.6, 4.7, 4.8 y 4.9 vemos el desempeño del modelo entrenado con conjuntos equiespaciados. Como podemos observar, incluso en el peor caso se estima correctamente el primer rebote, que es el que empleamos para identificar la velocidad y el tamaño de la inclusión en el medio. En estas figuras vemos una progresión bastante clara en cuanto a calidad del aprendizaje. Así, en el peor caso de 4.6 el primer rebote no está claro, pues hay dos muy seguidos y se ha cometido un error de más de 2 puntos al estimar la amplitud del mismo. Sin embargo, como podemos observar al entrenar con un conjunto de datos más grande en 4.7, 4.8 y 4.9, la estimación del primer rebote es mucho más precisa, siendo en el último caso una predicción casi exacta. La falta de precisión de la red se debe a un segundo e incluso tercer rebote, como vemos en 4.8.

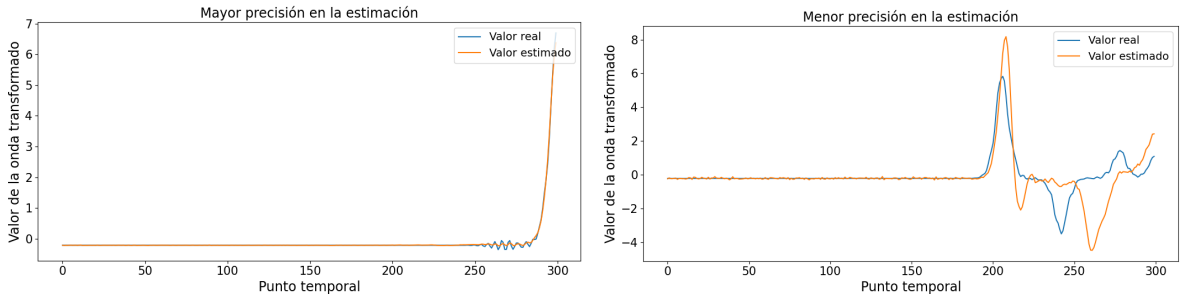


Figura 4.10: Ondas estimadas con mayor y menor precisión por la red entrenada con el conjunto generado por el método del hipercubo latino de 1 000 ondas.

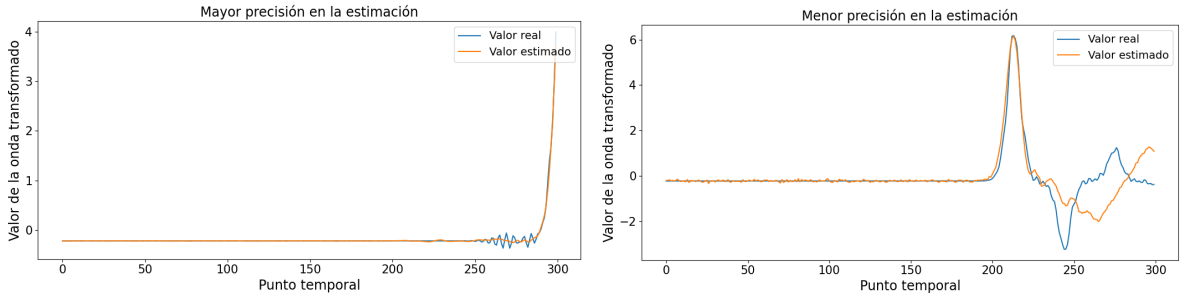


Figura 4.11: Ondas estimadas con mayor y menor precisión por la red entrenada con el conjunto generado por el método del hipercubo latino de 3 000 ondas.

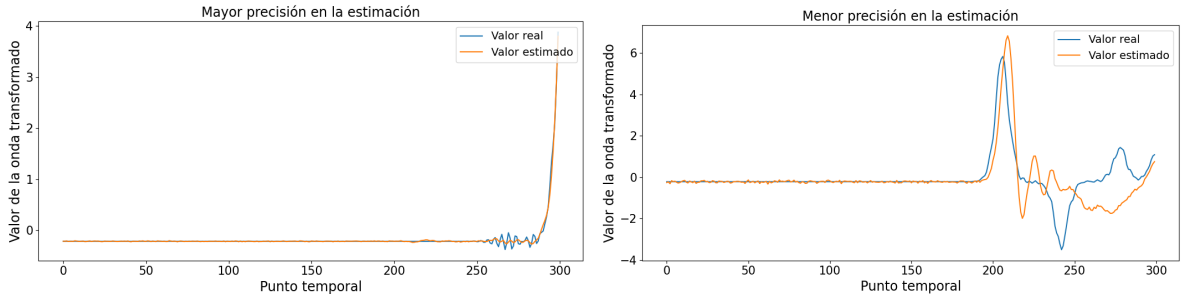


Figura 4.12: Ondas estimadas con mayor y menor precisión por la red entrenada con el conjunto generado por el método del hipercubo latino de 10 000 ondas.

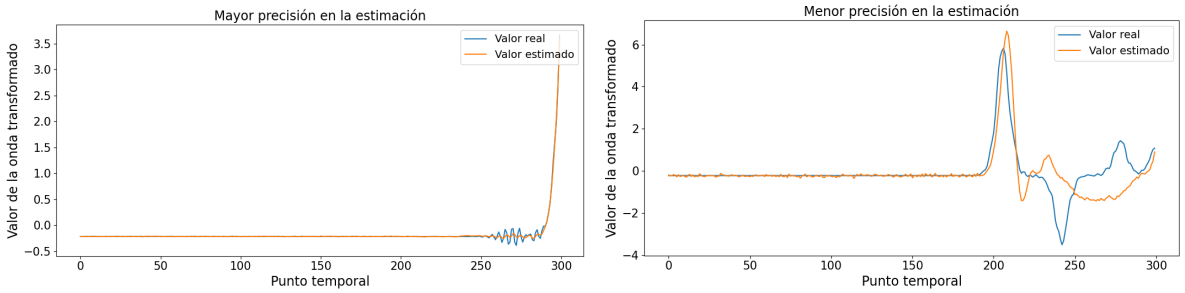


Figura 4.13: Ondas estimadas con mayor y menor precisión por la red entrenada con el conjunto generado por el método del hipercubo latino de 30 000 ondas.

En las figuras 4.10, 4.11, 4.12 y 4.13 podemos apreciar el desempeño del modelo al ser entrenado con los conjuntos de datos generados por el método del hipercubo latino. Como en el caso equiespaciado, la peor estimación cuando entrenamos la red con el conjunto de 1 000 ondas es bastante imprecisa en cuanto a la amplitud del primer rebote. Sin embargo, vemos una clara mejoría al entrenar la red con conjuntos de datos más grandes y, como en el caso equiespaciado, la posición y amplitud del primer rebote están bien estimadas incluso en el caso peor. A pesar de que el rendimiento de la red entrenada con 3 000 ondas pueda parecer mayor en el caso peor, sabemos que el desempeño al tener en cuenta todo el conjunto de ondas es bastante similar en términos de error al de conjuntos de entrenamiento más grandes, como ya vimos en el cuadro 4.2.

Capítulo 5

Red convolucional para el problema inverso

En este capítulo explicaremos el diseño de la red convolucional empleada para hallar los parámetros c_0, l_0 y x_0 (velocidad de la onda, longitud y posición del objeto, respectivamente) a partir de la imagen de la función $u(L,t)$ de la onda. Análogamente a la red expuesta en capítulos anteriores, este modelo dispondrá de 300 canales de entrada, recibirá los 300 puntos de la función $u(L,t)$ y dará una estimación de los tres parámetros del objeto, por lo que dispondrá de 3 canales de salida.

De manera adicional y para intentar estimar mejor algunos de los parámetros de salida se ha diseñado una red convolucional con 600 canales de entrada y 3 canales de salida. Para entrenar esta red hemos generado un nuevo conjunto de datos con 600 puntos por onda, en vez de los 300 anteriores, y con 50 000 ondas, el cual trataremos más adelante.

Como en el caso de la red anteriormente explicada, el código se encuentra disponible en GitHub [8], en la carpeta `invertedModels`. Ahí podemos encontrar el código para entrenar los modelos para el problema inverso.

5.1. Diseño de la redes neuronales

5.1.1. CNN con 300 canales de entrada

Para estimar los parámetros del objeto se ha diseñado una red neuronal con 300 canales de entrada y 3 canales de salida. En este caso, dado que estamos estimando tres parámetros distintos, se ha calculado el error respecto al rango por cada uno de ellos, obteniendo un error respecto al rango de más de un 25 % para el parámetro x_0 y menor de un 8 % para los parámetros l_0 y c_0. En la sección 5.3 se estudiará el problema de estimación del parámetro x_0.

Esta red está compuesta por las siguientes capas:

1. Una capa totalmente conectada con 300 canales de entrada y 300 canales de salida. Esta capa es la encargada de seleccionar los aspectos de la entrada con mayor influencia en los resultados, asignándoles un mayor peso.
2. Una capa de activación ReLU, de forma que rompamos la linealidad entre capas.
3. Una capa convolucional de una dimensión con tamaño de ventana 3. Con esta capa intentamos extraer las características más importantes de los datos de entrada.
4. Una capa de agrupamiento en la que reducimos los canales de entrada de 300 a 150 canales. En este caso, tomamos el máximo de los valores dentro de la ventana, la cual hemos seleccionado de tamaño 3. Hemos comprobado que el empleo del máximo en vez de la media en esta capa ayudaba a reducir el ruido en los resultados.
5. Una capa de activación ReLU.
6. Una capa de aplanamiento de los valores para transformar la matriz de entrada en un vector de una dimensión. Esta acción, como en la red anterior, reduce el trabajo de la red así como la memoria gráfica necesaria para llevar a cabo el proceso de entrenamiento.
7. Una capa totalmente conectada con 150 canales de entrada y 150 de salida, para mejorar el aprendizaje de la red.
8. Una capa de activación ReLU para romper la linealidad entre las dos capas completamente conectadas.
9. Una capa totalmente conectada con 150 entradas y 66 salidas. La elección de estos parámetros de salida se ha realizado siguiendo las proporciones de otras redes neuronales convolucionales que han destacado por su efectividad, como AlexNet [11].
10. Una capa de activación ReLU, una vez más para introducir no linealidad.
11. Una capa totalmente conectada con 66 canales de entrada y 3 de salida.

En este caso nos hemos basado en la configuración utilizada en AlexNet y en LeNet para las últimas capas de la red, donde empleamos varias capas totalmente conectadas separadas por funciones de activación ReLU. En el caso de AlexNet y LeNet también se intercalan capas de *dropout*, empleadas normalmente para reducir el sobreajuste de la red eliminando algunos canales de salida de la capa anterior; sin embargo, no hemos tenido dicho problema en el entrenamiento, por lo que se decidió no incluirlas para ahorrar memoria gráfica y poder entrenar con todo el conjunto de datos en una misma iteración.

Como veremos más adelante, la estimación tanto de la longitud del objeto l_0 como la velocidad de la onda c_0 ha sido óptima, mientras que para la posición del objeto, el término x_0, las estimaciones de la red no han llegado a la precisión de los otros dos parámetros.

5.1.2. CNN con 600 canales de entrada

Adicionalmente y con el mismo propósito que la red anterior, hemos diseñado otra red convolucional con 600 canales de entrada y 3 canales de salida que serán los 3 parámetros del objeto c_0, x_0 y l_0. También hemos calculado el error respecto al rango para cada uno de los parámetros, obteniendo unos resultados muy parecidos a los de la red anterior.

La finalidad de esta red era comprobar si la falta de precisión para el parámetro x_0 se debía a un bajo aprendizaje por el modelo o por el propio conjunto de datos. Además, para aumentar el número de puntos de entrada, se aumentó el intervalo de tiempo para el cual calculamos los valores de la ecuación de onda hasta $[0, 20]$, guardando, por cada onda, 600 puntos. De esta forma, la densidad de puntos por unidad de tiempo es la misma que en los conjuntos anteriores.

Como vimos anteriormente en el capítulo 4 en la sección de resultados, el primer rebote nos da bastante información de una onda y es lo que mejor estima la red neuronal. Sin embargo, el segundo y tercer rebote son los que realmente aportan información sobre el parámetro x_0, como veremos más adelante. Es por esto que se ha aumentado el tiempo de cálculo de la onda hasta $T = 20$ al generar el conjunto de datos, de forma que para todas las ondas dispongamos de información de, al menos, un segundo rebote de la misma.

Para entrenar esta red se ha generado un nuevo conjunto de ondas con las características anteriores, $T = 20$ y con los parámetros en el mismo rango que los demás conjuntos. El método empleado para la generación es el método del hipercubo latino y se han generado un total de 50 000 ondas para este conjunto.

No vamos a detallar las capas de esta red neuronal, pues la arquitectura es similar a las anteriores y puede encontrarse en el código fuente del programa en Python del proyecto [8]. Esta red dispone de un mayor número de capas que las anteriores y una mayor complejidad, pues se han añadido más capas convolucionales con varios canales de salida para mejorar la precisión del resultado.

La estimación de los parámetros ha sido similar a la red anterior, no siendo posible lograr un cálculo correcto para x_0. Sin embargo, con los dos otros parámetros continuamos teniendo unos resultados satisfactorios.

5.2. Ajuste de los hiperparámetros

Para fijar la tasa de aprendizaje al entrenar las redes neuronales anteriores hemos seguido un procedimiento similar al visto en la sección 4.2 y basándonos en las técnicas de la bibliografía específica [4, 13].

Con esta finalidad, hemos entrenado los modelos durante 3 000 épocas con distintas

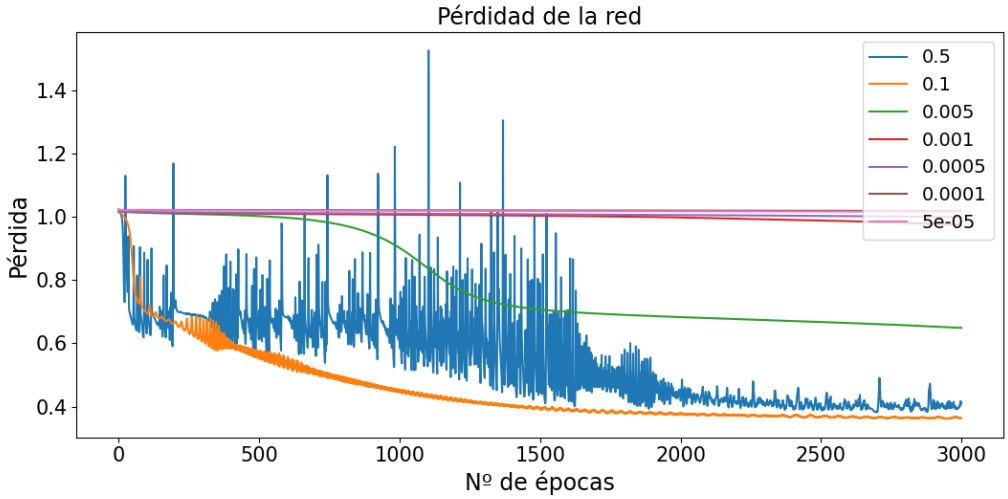


Figura 5.1: Pérdida de la red convolucional para el problema inverso con el conjunto de entrenamiento equiespaciado y variando la tasa de aprendizaje.

tasas de aprendizaje y valores de inercia, estudiando los valores devueltos por la función de pérdida, intentando minimizar dichos valores, al igual que hacíamos para la red del problema directo.

5.2.1. CNN con 300 canales de entrada

Los resultados obtenidos pueden ser observados en la figura 5.1 para el conjunto de datos equiespaciado con 10 000 ondas y en 5.2 para los datos hallados por el método del hipercubo latino con 10 000 ondas.

Como podemos observar en las figuras 5.1 y 5.2, en estos casos los modelos presentan dificultades para aprender con tasas altas de aprendizaje en ambos casos. Como ocurría con la red anterior, el aprendizaje con uno u otro conjunto de datos presenta pocas diferencias en cuanto al valor de la función de pérdida. Para evitar las colinas y los valles en el aprendizaje y lograr unos resultados más homogéneos repetiremos las pruebas para el conjunto hallado por el método LHS variando el hiperparámetro α, que representa la inercia. De esta forma lograremos suavizar el aprendizaje y hacerlo más eficaz.

Aunque omitiremos las gráficas para otros conjuntos de datos, los resultados son similares y las mismas conclusiones son aplicables en dichos casos.

Los valores elegidos para la inercia al comprobar la pérdida de nuestra red neuronal son los más empleados en la práctica según la bibliografía: $\alpha = 0{,}5$, $\alpha = 0{,}9$ y $\alpha = 0{,}99$ [4]. Los resultados de este entrenamiento pueden ser apreciados en la figura 5.3.

Tras estudiar los valores de la función de pérdida para distintas configuraciones de los hiperparámetros de la red neuronal, se ha decidido configurar la misma para el entrenamiento con una tasa de aprendizaje $lr = 0{,}005$ y una inercia de $\alpha = 0{,}99$, pues

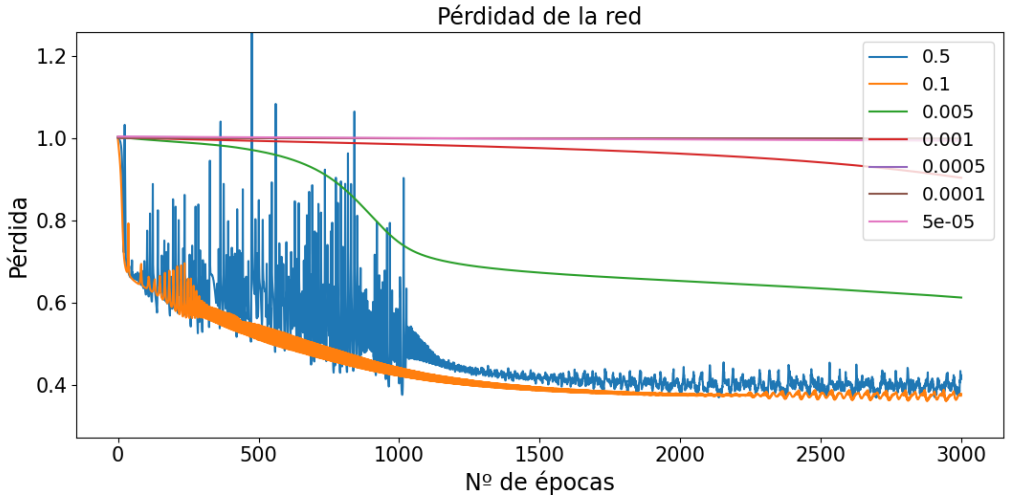


Figura 5.2: Pérdida de la red convolucional para el problema inverso con el conjunto de entrenamiento elegido por método LHS y variando la tasa de aprendizaje.

son los valores para los cuales la pérdida converge con un menor número de épocas y el entrenamiento es más consistente.

5.2.2. CNN con 600 canales de entrada

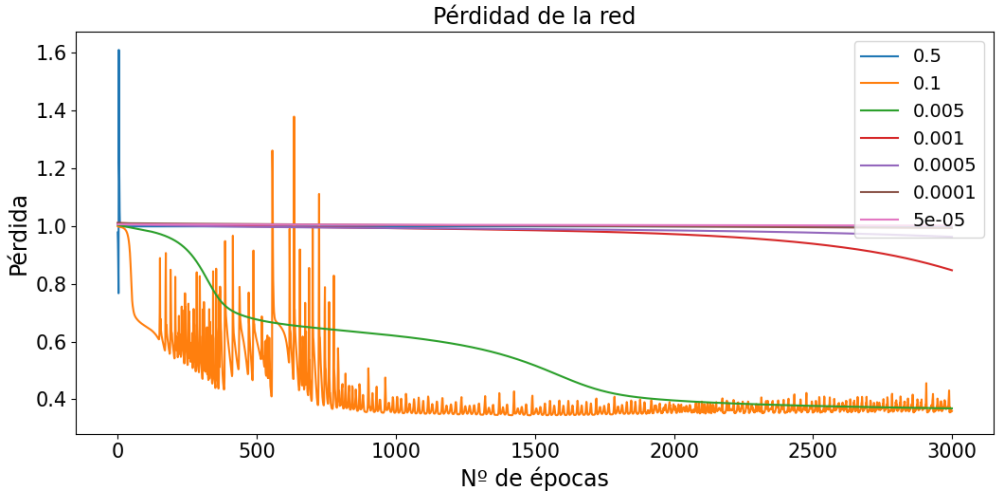
Para esta red convolucional hemos realizado un estudio similar a las anteriores, donde se han llevado a cabo distintos entrenamientos con 3 000 épocas y se ha calculado la pérdida para cada una de ellas, variando tanto la tasa de aprendizaje como la inercia de la red neuronal. En la figura 5.4 vemos la configuración con la que lográbamos una mayor reducción de la función de pérdida con los valores de los hiperparámetros $lr = 0{,}001$ y $\alpha = 0{,}99$.

5.3. Resultados de las redes convolucionales

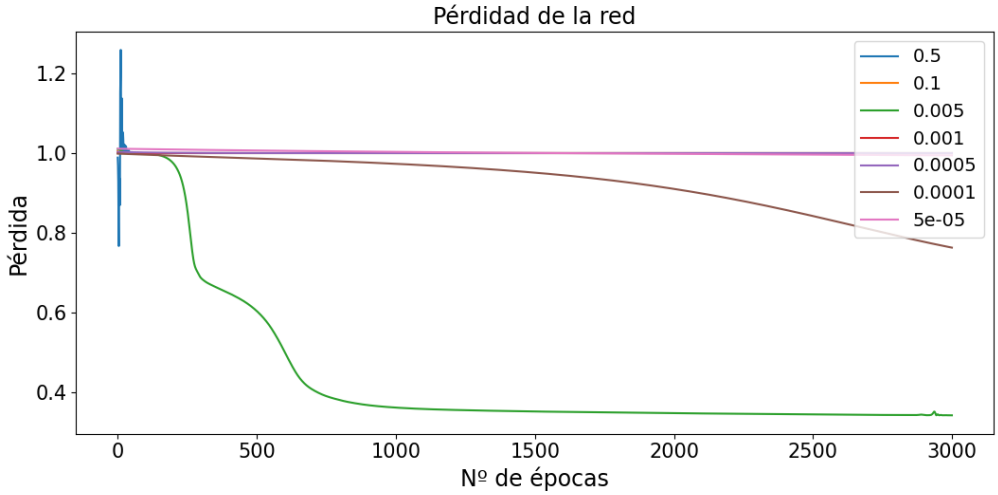
En este capítulo vamos a visualizar los resultados de las redes convolucionales anteriores al entrenarlas con los distintos conjuntos de datos de los que disponemos y por qué fallan al estimar ciertos parámetros. Veremos estos resultados tanto para las ondas halladas por el método equiespaciado como para las generadas por el método del hipercubo latino.

Para visualizar los resultados de estas redes convolucionales se ha visto conveniente hallar el error respecto al rango de cada uno de los parámetros, pues estos se encuentran uniformemente distribuidos en su propio rango. Para ello se han realizado los cálculos expuestos en la sección 3.5.

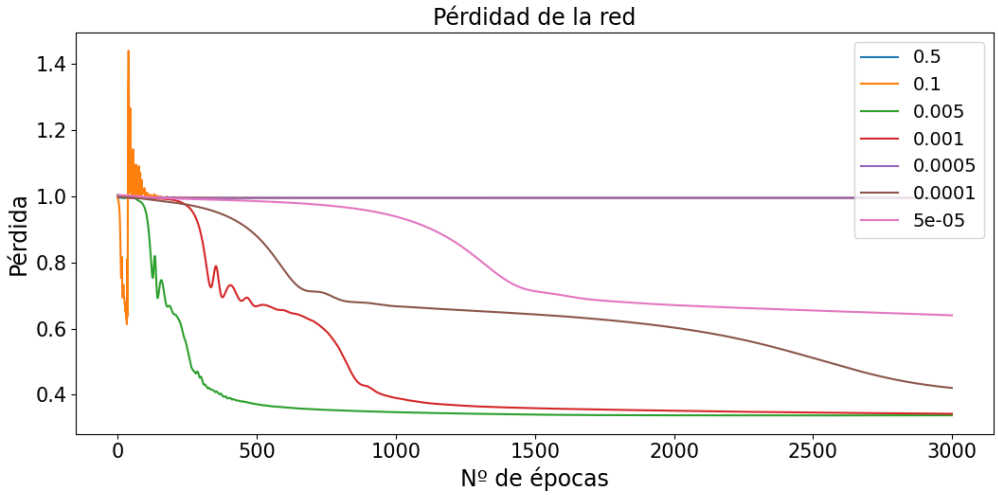
A continuación, veremos para cada una de las dos redes convolucionales los resultados de sus estimaciones.



(a) $\alpha = 0{,}5$.



(b) $\alpha = 0{,}9$.



(c) $\alpha = 0{,}99$.

Figura 5.3: Comparación de la pérdida del modelo para distintos valores del hiperparámetro α en la red convolucional para el problema inverso.

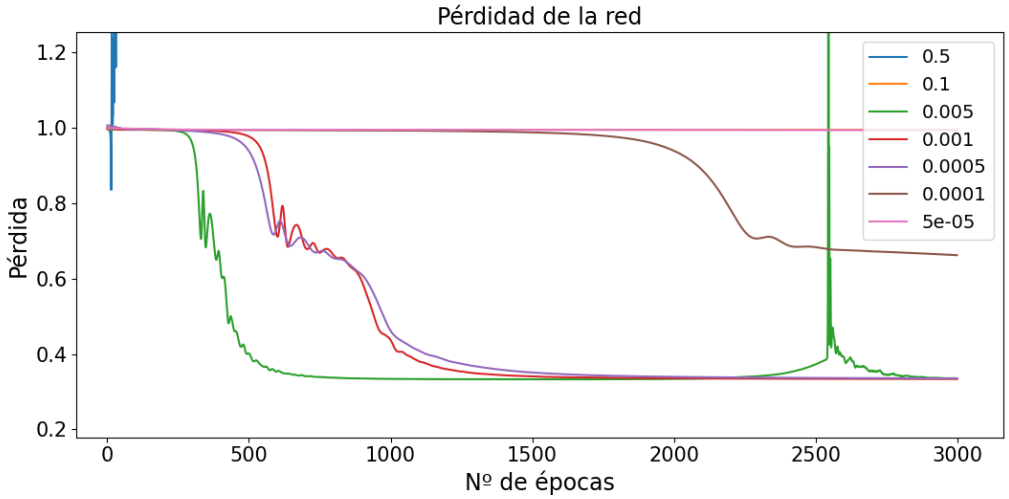


Figura 5.4: Pérdida de la red convolucional con inercia $\alpha = 0{,}99$ para el problema inverso. El conjunto de entrenamiento ha sido elegido por el método LHS con intervalo temporal $[0, 20]$. Para calcular la gráfica se ha variado la tasa de aprendizaje.

5.3.1. CNN con 300 canales de entrada

En los cuadros 5.1 y 5.2 se muestra el error respecto al rango de los parámetros l_0, c_0 y x_0 cometido por la red convolucional. Para cada columna del cuadro se indica, en la fila superior, el conjunto de entrenamiento utilizado para entrenar el modelo en cada columna. En la información del propio cuadro se muestra el porcentaje de error de los parámetros respecto al rango, al utilizar como conjunto de prueba el archivo con 100 000 ondas generado por el método LHS. Esto lo podemos realizar debido a que los conjuntos de datos usados para el entrenamiento del modelo contienen ondas distintas a aquellas que hemos generado para el archivo de 100 000 ondas con el que probamos la red. Para entrenar las redes se ha empleado siempre el 60 % del conjunto, mientras que un 28 % se dedicaba a la validación y el 12 % restante se reserva para probar la red (aunque en este caso no ha sido usado). Los datos mostrados en el cuadro son el resultado de probar la red con la totalidad del conjunto LHS con 100 000 ondas.

Entrenamiento	Equiespaciado			
Nº de ondas	1 000	3 000	10 000	30 000
Parámetro l_0	3.43 %	1.75 %	0.86 %	0.76 %
Parámetro c_0	7.31 %	5.04 %	1.56 %	1.66 %
Parámetro x_0	27.33 %	29.94 %	25.03 %	25.02 %

Cuadro 5.1: Error respecto al rango cometido por las redes neuronales entrenadas con los conjuntos de datos generados por el método equiespaciado al resolver el problema inverso y puestas a prueba con el conjunto LHS con 100 000 ondas.

Entrenamiento	LHS			
Nº de ondas	1 000	3 000	10 000	30 000
Parámetro l_0	1.72 %	1.38 %	1.04 %	0.78 %
Parámetro c_0	2.63 %	2.95 %	1.32 %	0.88 %
Parámetro x_0	34.19 %	31.26 %	25.56 %	25.04 %

Cuadro 5.2: Error respecto al rango cometido por las redes neuronales entrenadas con los conjuntos de datos generados por el método LHS al resolver el problema inverso y puestas a prueba con el conjunto LHS con 100 000 ondas.

A la luz de los resultados, podemos comprobar que la red neuronal estima satisfactoriamente los parámetros l_0 y c_0, no cometiendo un error superior al 8 % para ninguno de los conjuntos de datos. Sin embargo, en lo referente al parámetro x_0, el error respecto al rango cometido por la red es muy superior y no mejora más allá de un 25 %. Además, observamos diferencias de rendimiento entre los conjuntos equiespaciados y los LHS con el mismo número de ondas, siendo mucho más efectivo el entrenamiento con estos últimos.

En las figuras 5.5 y 5.6 se han dibujado los resultados estimados por la red neuronal tras ser entrenada con los conjuntos LHS con 1 000 y 30 000 ondas respectivamente. Para generar los resultados, se ha comprobado la precisión de la red neuronal con la totalidad del conjunto de prueba de 100 000 ondas, de la misma forma que en los cuadros anteriores. En el eje horizontal colocamos el número de cada onda que estamos mostrando y en el eje vertical se muestra el valor para el parámetro en cuestión.

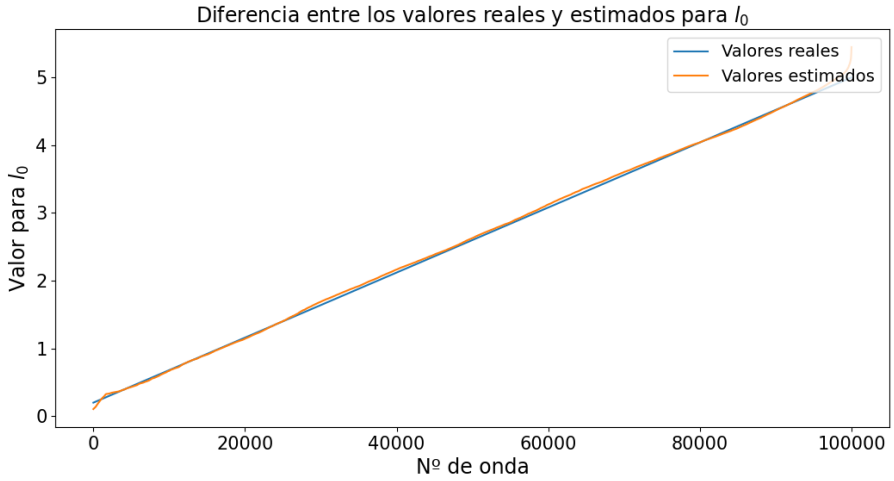
Para generar estas figuras se han estimado todos los datos del conjunto de 100 000 ondas con la red neuronal y se han ordenado atendiendo al valor real del parámetro. Es por ello que aparece como una función creciente.

A la luz de los resultados, podemos concluir que, salvo en los puntos de los extremos, los valores para los parámetros de velocidad de la onda c_0 y longitud del objeto l_0 son estimados correctamente por la red, incluso para los conjuntos con un menor número de ondas. Sin embargo, el parámetro x_0 parece tomar valores aleatorios y no mejorar con el tiempo, por lo que la red le acaba asignando su valor medio, un valor de 5, para reducir el error lo máximo posible, pues recordemos que x_0 se distribuye entre 3 y 7.

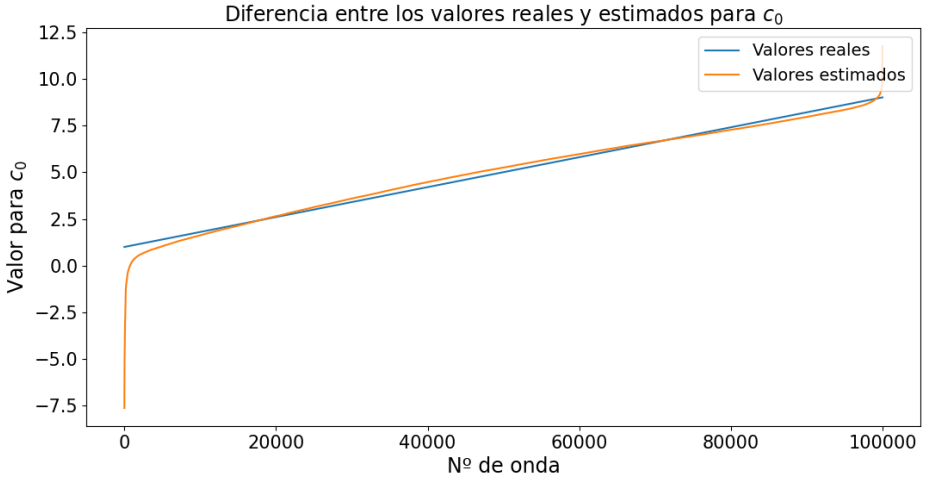
Para comprobar si este error para x_0 se debía a un fallo de diseño de la red neuronal, se decidió programar la red con 600 canales de entrada y 3 de salida, que veremos a continuación.

5.3.2. CNN con 600 canales de entrada

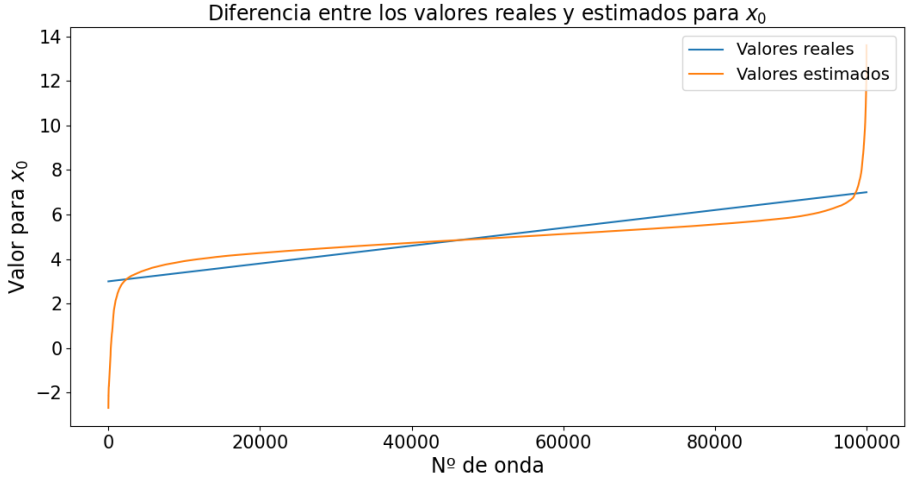
Para entrenar y probar esta red neuronal se generó un conjunto de datos específico con 50 000 ondas y un intervalo temporal de $[0, 20]$, pues como vimos en anteriores capítulos, con este tiempo nos aseguramos la presencia de un segundo rebote, que determina



(a) Estimación del parámetro l_0.

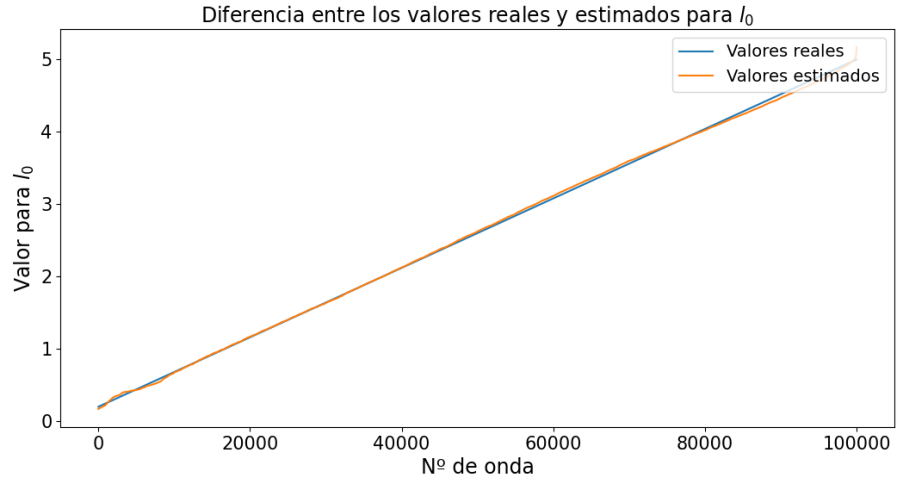


(b) Estimación del parámetro c_0.

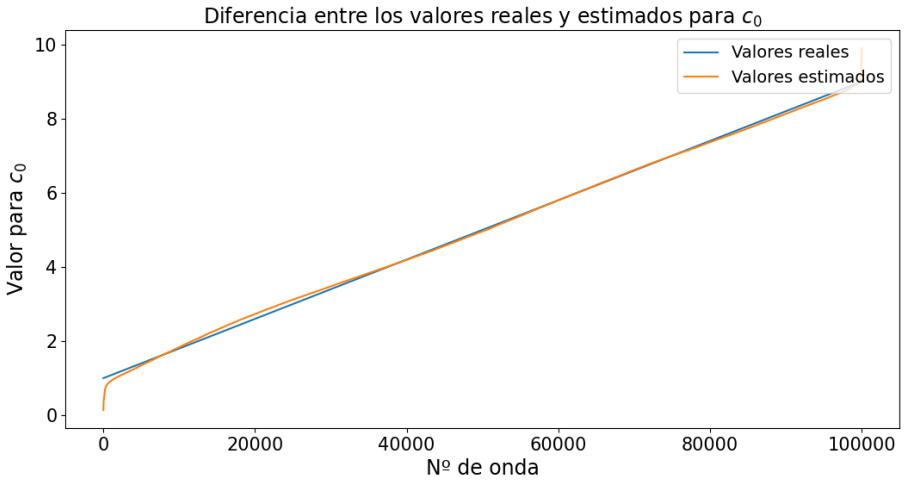


(c) Estimación del parámetro x_0.

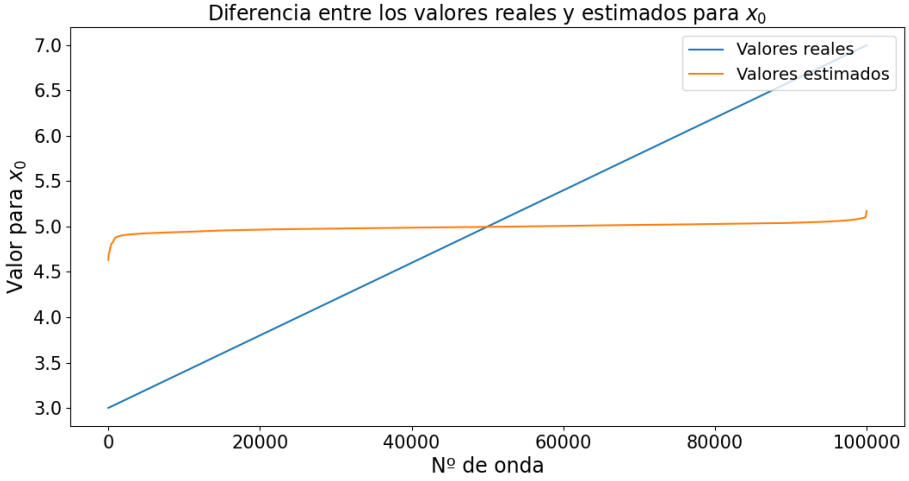
Figura 5.5: Comparación de los valores reales y estimados para cada uno de los parámetros al entrenar la red con el conjunto LHS con 1 000 ondas.



(a) Estimación del parámetro l_0.



(b) Estimación del parámetro c_0.



(c) Estimación del parámetro x_0.

Figura 5.6: Comparación de los valores reales y estimados para cada uno de los parámetros al entrenar la red con el conjunto LHS con 30 000 ondas.

el parámetro x_0 de las ondas. Además, dado que no se modificó la frecuencia con la que tomábamos datos, el conjunto dispone de 600 puntos por cada onda representada.

Tras realizar el entrenamiento de la red con este conjunto, obtuvimos unos errores relativos bastante similares al modelo anterior: 0.72 % para el parámetro l_0, 0.98 % para el parámetro c_0 y 25.25 % para el parámetro x_0. En la figura 5.7 podemos contemplar las estimaciones de la red neuronal para el parámetro x_0. En este caso, hemos usado el 60 % del conjunto para entrenar la red, el 28 % para validar los datos y el 12 % restante para probar las predicciones de la misma, pues no podíamos emplear el conjunto LHS de 100 000 ondas al haber sido generado con $T = 10$ y no con $T = 20$.

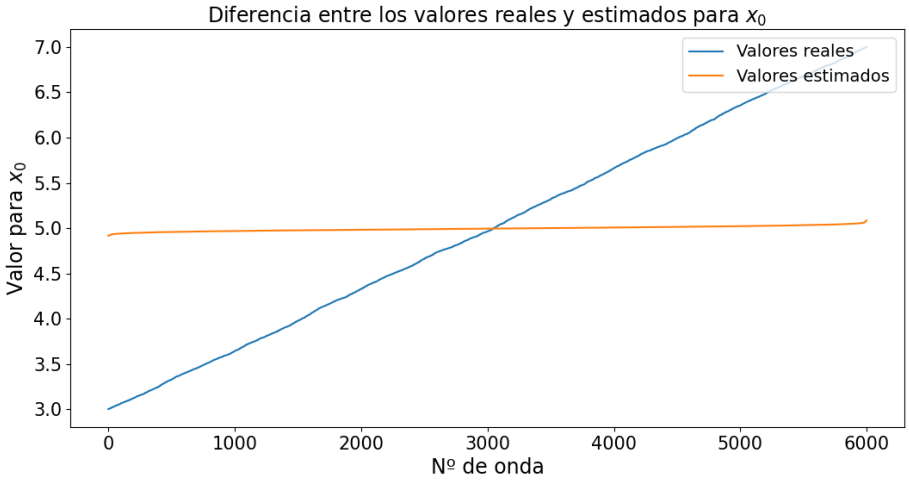


Figura 5.7: Estimación del parámetro x_0 por la red con 600 puntos.

Por tanto, se decidió inspeccionar el conjunto de datos para comprobar si al variar este parámetro se modifica la función $u(L, t)$ de la onda.

5.3.3. Estudio de los conjuntos de entrenamiento respecto a la posición del objeto x_0

Para estudiar los conjuntos de entrenamiento se han clasificado las ondas por sus parámetros de manera que, fijando l_0 y c_0, hemos variado x_0. De esta forma, hemos obtenido conjuntos con varias ondas. En las figuras siguientes hemos representado la función $u(L, t)$ de las mismas, situando en el eje horizontal el valor del tiempo y en el eje vertical el valor de la función $u(L, t)$ en dicho instante.

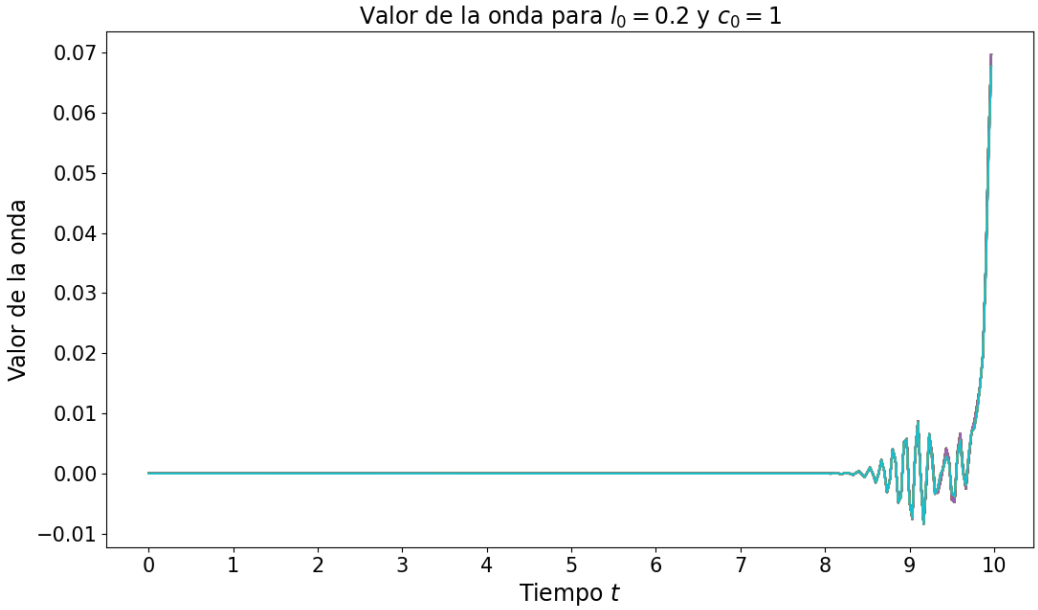


Figura 5.8: Representación gráfica de $u(L,t)$ para 40 ondas con $T = 10$, fijando $l_0 = 0{,}2$ y $c_0 = 1$, dejando libre el parámetro x_0.

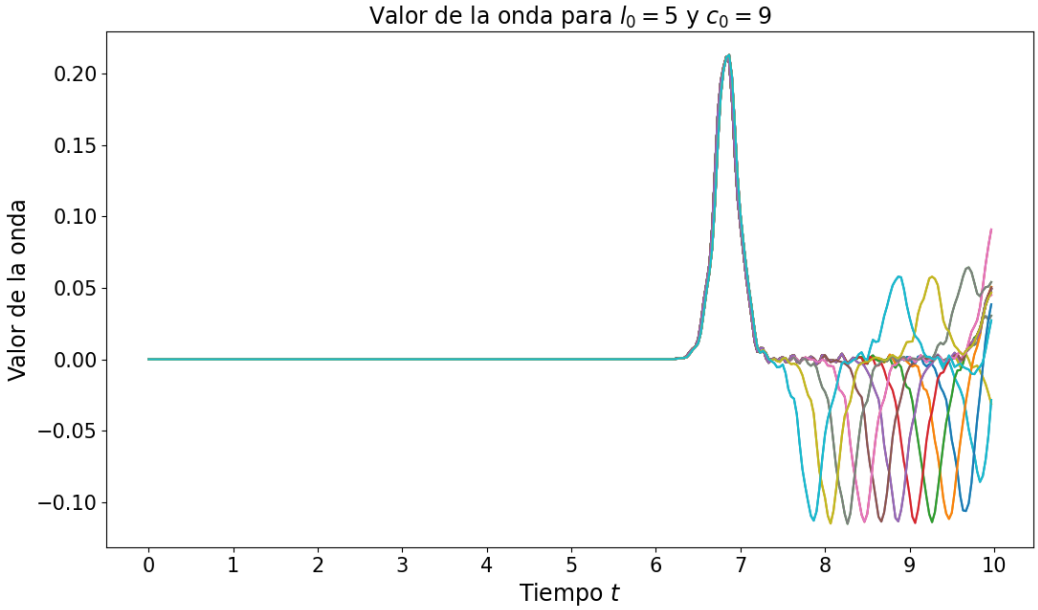


Figura 5.9: Representación gráfica de $u(L,t)$ para 40 ondas con $T = 10$, fijando $l_0 = 5$ y $c_0 = 9$, dejando libre el parámetro x_0.

Como vemos en las gráficas 5.8 y 5.9, las ondas apenas varían la imagen de $u(L,t)$ en el primer rebote, haciendo indistinguibles algunas de ellas al fijar el tiempo en $T = 10$, como vemos en 5.8. Para valores altos de tiempo, la única variación sería el segundo rebote con la pared, el cual podemos ver representado para algunas de las ondas en la figura 5.9.

Sin embargo, esto únicamente permite identificar un pequeño número de ondas a partir de este segundo rebote, pues no siempre queda representado en el conjunto de datos. De hecho, en la figura 5.9 es donde un mayor número de ellas veremos representadas con el segundo rebote para una configuración de l_0 y c_0, pues es cuando el objeto tiene un tamaño mayor y la onda tiene una velocidad superior en él, por lo que llega antes al extremo $T = 10$. A pesar de ello, únicamente 11 de las 40 ondas presentan diferencias en el intervalo de representación gracias al segundo rebote. Por este motivo, el aprendizaje de la red neuronal para el parámetro x_0 es bastante difícil con este conjunto de datos y el error cometido es elevado en comparación con otros parámetros.

Para intentar solventarlo hemos generado el conjunto de datos con 600 puntos fijando el tiempo en $T = 20$. En la configuración anterior con $l_0 = 0{,}2$ y $c_0 = 1$, como vimos en 5.8, no había representación del segundo rebote, pero como podemos apreciar en la figura 5.10 aquí sí tenemos un segundo rebote por cada una de las ondas y el valor $T = 20$ es el menor valor para el cual este rebote queda representado. Sin embargo, la red neuronal sigue sin ser capaz de estimar el parámetro x_0, lo cual nos confirma que la red no puede identificar correctamente este segundo rebote.

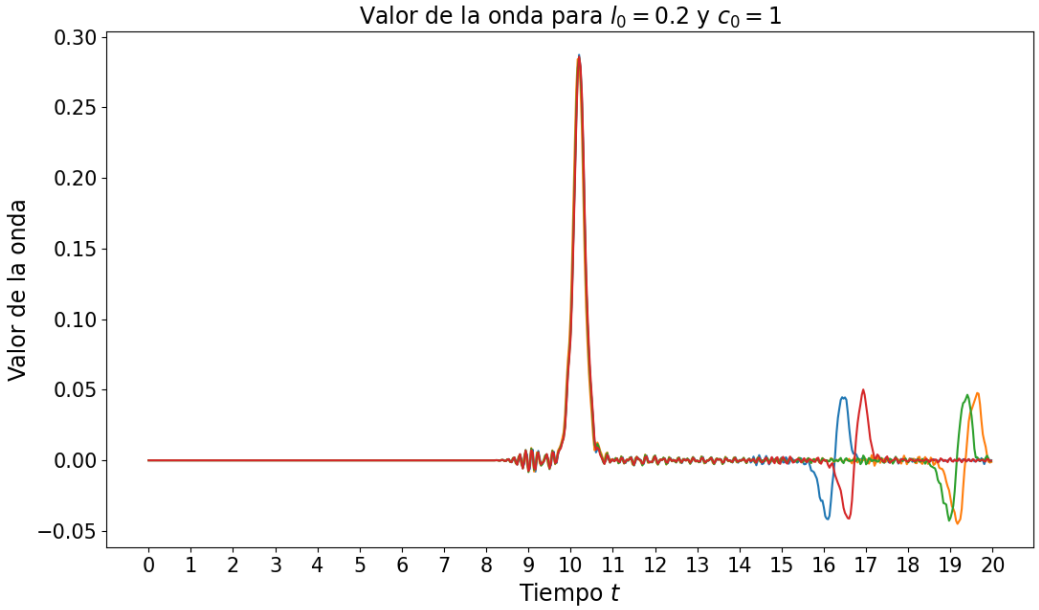


Figura 5.10: Representación gráfica de $u(L, t)$ para 4 ondas con $T = 20$, fijando $l_0 = 0{,}2$ y $c_0 = 1$, dejando libre el parámetro x_0.

Capítulo 6

Conclusiones

En vista de los resultados para el problema directo vemos que la elección del conjunto de datos influye en la capacidad de aprendizaje de la red. Optar por los conjuntos generados por el método LHS hace que el entrenamiento de la red sea más efectivo respecto del conjunto equiespaciado con el mismo número de ondas. Dado que la intención inicial era la de estimar el primer rebote de la onda, este objetivo ha sido alcanzado adecuadamente, pues incluso para las redes con conjuntos de entrenamiento más básicos se lograba una muy buena estimación tanto de la amplitud como de la localización de dicha característica.

Cuando generamos los conjuntos de datos con 300 puntos solo tuvimos en cuenta el hecho de que el primer rebote llegase al final de intervalo, puesto que era lo que nos interesaba estimar. Esto se debía a que el resto de información podía ser interpretada como ruido por la red neuronal. Sin embargo, al realizar el problema inverso hemos podido comprobar que los siguientes rebotes de la onda representan la situación del objeto en el medio; es decir, desvelan la posición x_0 del mismo.

Al estimar los parámetros en el problema inverso, hemos visto que la red tenía ciertos problemas con el parámetro x_0, pues este únicamente dependía de los rebotes secundarios. Por este motivo, tuvimos que desarrollar un nuevo conjunto de datos para asegurarnos la representación de estos segundos rebotes. Sin embargo, este conjunto presenta algunas limitaciones que eran el motivo por el que en un inicio se había descartado su creación:

- La presencia de ruido entre rebotes es muy alta, y en muchos casos puede ser falsamente interpretada por la red, lo que además dificulta el aprendizaje.
- La red requiere de más parámetros de entrada para poder procesar los 600 puntos, lo que aumenta la complejidad y el tiempo de procesamiento.
- El segundo rebote puede llegar al final del intervalo para cualquier valor del tiempo en $t \in [8, 20]$, según nuestros estudios del conjunto de datos. Esto dificulta bastante calcular el número de capas de la red neuronal para 600 puntos.

El problema que estamos intentando solucionar, de estimar el valor de x_0, es algo inherente a la forma de procesar la onda y los datos que calculamos de ella. En este caso, tomamos únicamente los valores de la ecuación en el punto final del intervalo, por lo que solo disponemos de información del momento en el que los rebotes llegan a este extremo, pero la posición del objeto no influye en el primero de los rebotes. Esto es debido a que la onda se acelera al atravesar el objeto, pero su rebote final no varía respecto al momento en el que ha sufrido dicha aceleración. Además, en algunos casos la presencia de ruido dificulta la identificación de los rebotes secundarios, lo que supone un problema adicional.

Asimismo, la onda apenas sufre alteraciones al atravesar el objeto para ciertas configuraciones de los parámetros. Por ejemplo, cuando la velocidad de la onda en él es muy similar a la del medio. Esto dificulta aún más el cálculo de su posición.

Una de las soluciones que se podría considerar en estos casos sería fijar el valor de x_0 en los conjuntos de datos, aunque esto reduce enormemente la utilidad práctica de la red neuronal.

Otra solución posible sería la de no intentar calcular el valor exacto de la posición, sino que la red aporte un intervalo donde el objeto se podría encontrar con una probabilidad determinada, transformando el problema de x_0 en uno de clasificación. De esta forma, cuando no se disponga de información suficiente para estimar la posición del objeto, la red podría estimar que el valor de la posición es una variable uniformemente distribuida en el intervalo.

Referencias

[1] Ana Carpio, Elena Cebrián, and Andrea Gutiérrez. Object based bayesian full-waveform inversion for shear elastography. *Inverse Problems*, 39:075007, 2023. ISSN 0266-5611. doi: https://doi.org/10.1088/1361-6420/acd5f8. URL https://iopscience.iop.org/article/10.1088/1361-6420/acd5f8.

[2] Abdullah Waseem. A simple matlab implementation of 1d finite elements, 2020. URL https://github.com/AbdullahWaseem/1D-Finite-Element-Codes-Matlab.

[3] Miguel Peñalver. Repositorio de github con el código de matlab para el fem, 2023. URL https://github.com/MiguelJPen/wave_analysis_tfg.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2015.

[5] Keming Mao, Duo Lu, Dazhi E, and Zhenhua Tan. A case study on attribute recognition of heated metal mark image using deep convolutional neural networks. *Sensors*, 18:1871, 06 2018. doi: 10.3390/s18061871.

[6] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 151:107398, 2021. ISSN 0888-3270. doi: https://doi.org/10.1016/j.ymssp.2020.107398. URL https://www.sciencedirect.com/science/article/pii/S0888327020307846.

[7] Timo Stöttner. Por qué los datos deben ser normalizados antes de entrenar una red neuronal. URL https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d.

[8] Miguel Peñalver. Repositorio de github con el código de python y las redes neuronales, 2023. URL https://github.com/MiguelJPen/wave_analysis_python.

[9] Python. Página web del lenguaje de programación python. URL https://www.python.org/.

[10] PYTorch. Página web de la librería pytorch. URL https://pytorch.org/.

[11] Wikipedia. Alexnet. URL https://en.wikipedia.org/wiki/AlexNet.

[12] Amazon AWS. What is hyperparameter tuning. URL https://aws.amazon.com/what-is/hyperparameter-tuning/.

[13] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. URL https://arxiv.org/abs/1206.5533.