



UNIVERSIDAD
DE GRANADA

Facultad de Ciencias

Escuela Técnica Superior de Ingenierías Informática y Telecomunicación

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

TRABAJO DE FIN DE GRADO

Aproximación de operadores de Lipschitz continuos entre espacios de Banach

Presentado por:
Violeta Ángeles Atienza Pereira

Curso académico 2023-2024

Aproximación de operadores de Lipschitz continuos entre espacios de Banach

Violeta Ángeles Atienza Pereira

Violeta Ángeles Atienza Pereira *Aproximación de operadores de Lipschitz continuos entre espacios de Banach.*

Trabajo de fin de Grado. Curso académico 2023-2024.

**Responsable de
tutorización**

Francisco Javier Merí de la Maza
Departamento de Análisis Matemático

Miguel Molina-Solana
*Departamento de Ciencias de la Computación e
Inteligencia Artificial*

Doble Grado en Ingeniería
Informática y Matemáticas

Facultad de Ciencias
Escuela Técnica Superior de
Ingenierías Informática y
Telecomunicación

Universidad de Granada

DECLARACIÓN DE ORIGINALIDAD

D./Dña. Violeta Ángeles Atienza Pereira

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2023-2024, es original, entendido esto en el sentido de que no he utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 14 de julio de 2024

Fdo: Violeta Ángeles Atienza Pereira

A mis padres, por su esfuerzo y apoyo durante estos años.

Índice general

Índice de figuras	VII
Summary	XI
Introducción	XIII
1 Plan del proyecto	1
1.1 Definición de las tareas a realizar	1
1.2 Planificación temporal del proyecto	2
1.3 Estimación de coste	2
2 Herramientas matemáticas	5
2.1 Algunas cuestiones básicas de Teoría de la Medida	5
2.2 Resultados auxiliares de Análisis Matemático	6
3 Espacios Vectoriales Topológicos	9
3.1 Presentación y construcción de EVT	9
3.2 Convergencia uniforme y complitud	11
3.3 Aplicaciones lineales entre EVT	13
3.4 Topologías iniciales	13
3.5 EVT Localmente Convexos	14
3.6 EVT Metrizables	15
3.7 Teoría de dualidad	16
3.8 Topología débil y débil-*	18
3.9 Espacio de funciones test	19
4 Teoría de Distribuciones	23
4.1 Distribuciones	23
4.1.1 Definición	23
4.1.2 Funciones vistas como distribuciones	24
4.2 Cálculo con distribuciones	24
4.2.1 Multiplicación por funciones	24
4.2.2 Derivadas de una distribución	25
4.2.3 Convolución	26
4.3 Transformada de Fourier. Distribuciones Temperadas	27
4.3.1 Motivación	27
4.3.2 Notación	28
4.3.3 Funciones de decrecimiento rápido	28

Índice general

4.3.4	Distribuciones Temperadas	31
5	Aprendizaje profundo. Fundamentos	33
5.1	Descripción de un modelo formal de Aprendizaje Automático	33
5.2	Redes Neuronales Artificiales	33
5.2.1	Tipos de redes neuronales	34
5.3	Redes Neuronales Prealimentadas	34
5.3.1	Funciones de Activación	35
5.3.2	Función de coste	38
5.3.3	Entrenamiento	39
6	Aproximación de operadores no lineales mediante redes neuronales	45
6.1	Notación y definiciones	45
6.2	Características de las funciones de activación	46
6.3	Aproximación de funcionales y operadores continuos no lineales	54
6.4	Conclusiones sobre los resultados teóricos presentados: Aplicación a Sistemas Dinámicos	58
7	Physics Informed Neural Networks: usos e implementación en <i>SciANN</i>	61
7.1	Introducción a PINNs	61
7.2	Introducción a <i>SciANN</i>	62
7.2.1	Arquitectura	62
7.2.2	Espacio de datos	64
7.2.3	Entrenamiento	64
8	Experimentación	67
8.1	Construcción del entorno de experimentación	67
8.2	Experimento 1: Problemas de Regresión	68
8.2.1	Ajuste de la función seno	68
8.2.2	Ajuste de la función exponencial	74
8.2.3	Ajuste bidimensional	80
8.3	Experimento 2: El problema de Burgers	84
8.3.1	Descripción del problema	84
8.3.2	Generación de datos sintéticos	87
8.3.3	Construcción del modelo	87
8.3.4	Resultados	88
8.4	Experimento 3: Aproximación de operadores	92
8.4.1	Motivación	92
8.4.2	Descripción del problema	93
8.4.3	Obtención de datos	94
8.4.4	Construcción del modelo	95
8.4.5	Interpretación de los resultados	98
9	Conclusiones y futuros trabajos	107
	Bibliografía	109

Índice de figuras

1.1	Diagrama de Gantt asociado al proyecto.	2
5.1	Arquitectura de Red Neuronal presentada en [SSBD14].	35
5.2	Función de activación sigmoidal.	36
5.3	Función de activación tangente hiperbólica.	37
5.4	Función de activación ReLU.	37
5.5	Función de activación Leaky ReLU.	38
5.6	Funciones de activación ELU y SELU.	38
5.7	Función de activación GELU.	39
5.8	Pasos de descenso de gradiente para una tasa de aprendizaje demasiado grande. Imagen extraída de [BIKP20].	40
5.9	Pasos de descenso de gradiente para una tasa de aprendizaje demasiado pequeña. Imagen extraída de [BIKP20].	41
5.10	Pasos de descenso de gradiente para una buena tasa de aprendizaje. Imagen extraída de [BIKP20].	41
5.11	Tasa de aprendizaje constante para una función de coste con cambios bruscos en la pendiente. Imagen extraída de [BIKP20].	41
6.1	Arquitectura de red neuronal para la aproximación de operadores no lineales extraída de [CC95].	60
7.1	Elementos abstractos de <i>SciANN</i> y su relación con capas de Keras y Tensorflow. Diagrama extraído de [HJ21].	63
7.2	Ejemplos de mallado para PINNs.	64
8.1	Esquema del modelo de <i>SciANN</i> para la curva $y = \sin(x)$.	70
8.2	Desglose del ajuste de hiperparámetros para para la curva $y = \sin(x)$ del experimento 8.1. con conjunto de entrenamiento ampliado bajo la métrica <i>loss</i> .	71
8.3	Desglose del ajuste de hiperparámetros para la curva $y = \sin(x)$ del experimento 8.1. con conjunto de entrenamiento ampliado bajo la métrica <i>val_loss</i> .	72
8.4	Ajuste de la curva $y = \sin(x)$ en el intervalo $[0, 2\pi]$, donde la función original aparece en negro y la predicción en rojo.	72
8.5	Predicción para la curva $y = \sin(x)$ en el intervalo $[-2\pi, 2\pi]$, donde la función original aparece en negro y la predicción en rojo.	73
8.6	Desglose del ajuste de hiperparámetros para para la curva $y = \sin(x)$ del experimento 8.1. con conjunto de entrenamiento reducido bajo la métrica <i>loss</i> .	73
8.7	Desglose del ajuste de hiperparámetros para la curva $y = \sin(x)$ del experimento 8.1. con conjunto de entrenamiento reducido bajo la métrica <i>val_loss</i> .	74

Índice de figuras

8.8 Desglose del ajuste de hiperparámetros para la curva $y = \sin(x)$ frente a ruido bajo la métrica <i>loss</i>	75
8.9 Desglose del ajuste de hiperparámetros para la curva $y = \sin(x)$ frente a ruido, bajo la métrica <i>val_loss</i>	75
8.10 Ranking de parámetros correlacionados con la métrica <i>time</i> para la curva $y = \sin(x)$ del experimento 8.1.	76
8.11 Desglose del ajuste de hiperparámetros para la curva $y = \sin(x)$ del experimento 8.1, con conjunto de entrenamiento reducido bajo la métrica <i>time</i>	76
8.12 Desglose del ajuste de hiperparámetros para la curva $y = \sin(x)$ del experimento 8.1, con conjunto de entrenamiento ampliado bajo la métrica <i>time</i>	77
8.13 Evolución de los términos de la función de pérdida durante el entrenamiento para la curva $y = \sin(x)$	77
8.14 Desglose del ajuste de hiperparámetros para la curva $y = \exp(x)$ del experimento 8.1, bajo la métrica <i>loss</i>	79
8.15 Desglose del ajuste de hiperparámetros para la curva $y = \exp(x)$ del experimento 8.1, bajo la métrica <i>val_loss</i>	79
8.16 Ranking de parámetros correlacionados con la métrica <i>loss</i> para la curva $y = \exp(x)$ del experimento 8.1.	80
8.17 Desglose del ajuste de hiperparámetros para la curva $y = \exp(x)$ del experimento 8.1, bajo la métrica <i>time</i>	81
8.18 Segundo desglose del ajuste de hiperparámetros para la curva $y = \exp(x)$ del experimento 8.1, bajo la métrica <i>time</i>	81
8.19 Ranking de parámetros correlacionados con la métrica <i>time</i> para la curva $y = \exp(x)$ del experimento 8.1.	82
8.20 Predicción para la curva $y = \exp(x)$ en el conjunto de validación, donde dibujamos en negro el valor real de la función y en rojo la predicción.	82
8.21 Evolución de los términos de la función de pérdida durante el entrenamiento del modelo especificado en el Listing 8.2 para la curva $y = \exp(x)$	83
8.22 Ajuste de la curva $z = \sin(x)e^y$ en el intervalo $[0, \pi] \times [0, \pi]$	83
8.23 Esquema de red neuronal para la curva $z = \sin(x)e^y$	85
8.24 Comparación de la función original con la predicción de <i>SciANN</i>	86
8.25 Evolución de los términos de la función de pérdida durante el entrenamiento para la curva $z = \sin(x)e^y$	86
8.26 Representación interna en <i>SciANN</i> para el problema de Burgers.	89
8.27 Desglose del ajuste de hiperparámetros para el problema de Burgers con parada anticipada bajo la métrica <i>loss</i>	90
8.28 Desglose del ajuste de hiperparámetros para el problema de Burgers con parada anticipada bajo la métrica <i>train_time</i>	90
8.29 Desglose del ajuste de hiperparámetros para el problema de Burgers sin parada anticipada bajo la métrica <i>train_time</i>	91
8.30 Desglose del ajuste de hiperparámetros para el problema de Burgers sin parada anticipada bajo la métrica <i>loss</i>	91
8.31 Evolución de la función de pérdida en el experimento 2.	92
8.32 Comparación de \hat{u} y u en el problema de Burgers extraída de [HJ21].	93
8.33 Arquitectura de las entradas necesarias para el experimento 3 propuesta en [LJK19].	94

Índice de figuras

8.34 Representación compacta de la arquitectura propuesta en el Capítulo 6 para $M = 1$ y $N = p$	96
8.35 Representación compacta de la arquitectura modificada propuesta en DeepONet.	96
8.36 Representación interna simplificada del experimento 3: primera aproximación.	98
8.37 Representación interna del experimento 3: segunda aproximación.	99
8.38 Desglose del ajuste de hiperparámetros para el experimento 8.3. con conjunto de datos reducido bajo la métrica <i>val_loss</i>	100
8.39 Desglose del ajuste de hiperparámetros para el experimento 8.3. con conjunto de datos reducido bajo la métrica <i>loss</i>	100
8.40 Ranking de parámetros correlacionados con la métrica <i>val_loss</i> para el experimento 8.3. con conjunto de datos reducido.	101
8.41 Comparación entre los modelos seleccionados para el conjunto de entrenamiento reducido. .	102
8.42 Desglose del ajuste de hiperparámetros para el experimento 8.3. con conjunto de datos ampliado bajo la métrica <i>val_loss</i>	103
8.43 Desglose del ajuste de hiperparámetros para el experimento 8.3. con conjunto de datos ampliado bajo la métrica <i>loss</i>	103
8.44 Ranking de parámetros correlacionados con la métrica <i>val_loss</i> para el experimento 8.3. con conjunto de datos ampliado.	104
8.45 Comparación entre los modelos seleccionados para el conjunto de entrenamiento ampliado. .	104

Summary

This work explores the key elements of Distribution Theory to understand and implement the findings in Chen and Chen's 1995 paper *Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its applications to dynamic systems*.

To address the results from Chen and Chen, the third chapter examines the fundamentals of Topological Vector Spaces (TVS), including their construction, convergence, and types. Emphasis is placed on test function spaces to introduce Distribution Theory. The fourth chapter explores the key results of Distribution Theory, presenting distributions as generalized functions compatible with classical differential calculus. It also covers rapidly decreasing functions to introduce tempered distributions, which are essential for some primary results in the mentioned paper.

Chapter five focuses on deep learning fundamentals. It outlines a formal model of machine learning, delves into the architecture and applications of feedforward neural networks, and discusses optimization techniques for this model.

With this theoretical groundwork laid, we tackle the results proposed by Chen and Chen. Chapter six provides proofs for these results, aiming first to relax conditions for a function to be considered an activation function, and then using these relaxed conditions to establish approximation theorems for functionals and operators. An example of applying the theorem's architecture to dynamical systems is given.

To apply these results, the chosen method is Physics-Informed Neural Networks (PINNs). The seventh chapter summarizes the key points from the literature review on PINNs, with a focus on the Scientific Computing with Artificial Neural Networks (*SciANN*) library. Here we demonstrate how PINNs operate using *SciANN*, evaluating the feasibility of approximating operators with this approach. The chapter includes an analysis of PINNs' features and *SciANN*'s specificities.

Finally, chapter eight presents experiments illustrating *SciANN*'s versatility in addressing various problems and its limitations. The first experiment aims to demonstrate the versatility of *SciANN*, which is not limited to the implementation of PINNs but also encompasses other tasks such as regression problems. The second experiment aims to demonstrate the acquired knowledge regarding the modeling of PINNs and to study how the complexity of the model affects the efficiency of the learning task in *SciANN*. For this purpose, we chose the Burgers problem.

The experimentation culminates by integrating the architecture proposed in the sixth chapter for operator approximation with *SciANN* and comparing the results with those obtained using *DeepONet*, the only PINN-oriented library currently supporting operator learning. We conclude that *SciANN* lacks the necessary abstractions to perform this task effectively.

Summary

Keywords: Topological Vector Space, Distributions Theory, Operator, Deep Learning, Machine Learning, Activation Function, SciANN, PINNs, DeepONet.

Introducción

En este trabajo se estudian los aspectos más importantes de la Teoría de Distribuciones con el objetivo de comprender, desarrollar e implementar los resultados propuestos en el artículo [CC95] para la aproximación mediante redes neuronales de operadores de Lipschitz continuos entre espacios de Banach.

El trabajo comienza por el estudio de los Espacios Vectoriales Topológicos (EVT), nombre que se le da a los espacios vectoriales dotados de una topología vectorial. En el [Capítulo 3](#), después de hacer una presentación constructiva de ellos, se demuestra el Teorema de Riesz para EVT separados. La importancia de este resultado reside en que demuestra la falta de compactos en espacios de dimensión infinita, lo que justificará la introducción de las topologías débil y débil-* más adelante. Tras esta introducción, se exponen las nociones principales de convergencia en EVT y se presentan las características de los tipos de EVT de los que se hará uso durante el trabajo. A continuación, se introduce el dual de un EVT y se proponen las topologías débil y débil-* para preservar la continuidad de funcionales del dual y garantizar la presencia de más conjuntos compactos que con la topología de la norma. El capítulo finaliza con la introducción del espacio de funciones test.

El [Capítulo 4](#) comprende los contenidos trabajados sobre la Teoría de Distribuciones. En él se introduce el concepto de distribución como función generalizada y su compatibilidad con el cálculo diferencial clásico. Posteriormente, se llega a una definición de transformada de Fourier compatible con las distribuciones, para lo que se estudian las funciones de decrecimiento rápido y las distribuciones temperadas.

Para la realización de los capítulos mencionados hasta ahora, las principales fuentes de información que se han consultado son:

- *Functional Analysis*, por Walter Rudin [[Rud91](#)].
- *Análisis Funcional*, por Miguel Martín Suárez [[MS11](#)].
- Material proporcionado para el curso Análisis Funcional Avanzado impartido en la titulación de máster FisyMat de la Universidad de Granada, curso 2023/2024.
- *Topological Vector Spaces I*, por G. Köthe [[K79](#)].
- *Fourier Transform of Measures*, por Toru Maruyama [[Mar18](#)].

En el [Capítulo 5](#) se realiza un estudio de los fundamentos del aprendizaje profundo. Para ello, se describe un modelo formal de aprendizaje automático y se profundiza en el modelo de red neuronal prealimentada y en su arquitectura, aplicaciones y técnicas de optimización. Para la realización de este

Introducción

capítulo, las principales fuentes de consulta son:

- *Deep Learning*, por Goodfellow, Bengio y Courville [**GBC16**].
- *Understanding Machine Learning: From Theory to Algorithms*, por Shai Shalev-Shwartz y Shai Ben-David [**SSBD14**].
- Material proporcionado para el curso de Aprendizaje Automático del Grado en Ingeniería Informática de la Universidad de Granada, curso 2023/2024.

Llegados a este punto, se han adquirido los conocimientos teóricos necesarios para abordar los resultados que se proponen en [**CC95**]. En el [Capítulo 6](#) se desarrolla la demostración de todos los resultados que aparecen en este artículo. Los primeros resultados tienen como objetivo debilitar las condiciones bajo las cuales una función puede ser considerada función de activación, mientras que los resultados posteriores aprovechan esta debilitación para establecer teoremas de aproximación tanto para funcionales como para operadores. Al final del capítulo se propone un ejemplo de la arquitectura dada por el teorema aplicado a sistemas dinámicos.

Los resultados del [Capítulo 6](#) no son constructivos, pero la arquitectura general propuesta para la aproximación de sistemas dinámicos recuerda al enfoque de las Physical-Informed Neural Networks (PINNs). Es por esto que otro objetivo del trabajo es mostrar cómo funcionan las PINNs a través de *SciANN* [**HJ21**], una de las librerías más utilizadas en el ámbito de las PINNs, y estudiar la viabilidad de aproximar operadores a través de ella. En el [Capítulo 7](#) se realiza un estudio sobre las características de las PINNs y se relaciona con las particularidades de la librería *SciANN*.

Por último, en el [Capítulo 8](#) se realizan varias experimentaciones que muestran tanto la versatilidad de *SciANN* para abordar distintos tipos de problemas como sus limitaciones, concluyendo que aún no dispone de las abstracciones necesarias para soportar el aprendizaje de operadores.

1 Plan del proyecto

1.1. Definición de las tareas a realizar

Este capítulo describe la planificación del Trabajo Fin de Grado realizado. Atendiendo a los objetivos mencionados en la introducción, se ha realizado el siguiente desglose en tareas:

- Tarea 1 (**T1**): En una primera toma de contacto con el artículo [CC95], esta tarea consiste en realizar una lectura identificando los puntos clave del mismo y revisando qué conceptos se conocen y cuáles no, con el objetivo de establecer un plan de estudio que nos permita llegar a comprender estos conceptos.
- Tarea 2 (**T2.1,T2.2**): Comprende la adquisición de los conocimientos identificados durante la tarea anterior. Para ello, se ha visto conveniente realizar un estudio individual de la Teoría de Distribuciones (**T2.1**) acompañado de la asistencia como oyente al curso de Análisis Funcional Avanzado impartido en la titulación de máster FisyMat de la Universidad de Granada (**T2.2**).
- Tarea 3 (**T3**): Consiste en el desarrollo de los resultados presentados en el artículo [CC95], detallando y completando sus demostraciones.
- Tarea 4 (**T4.1,T4.2**): En ella se engloban tanto la adquisición de conocimientos teóricos necesarios como la familiarización con las librerías de las que se va a hacer uso en el proyecto. Para ello, se prevé un acercamiento teórico (**T4.1**) al estado del arte en PINNs mediante el estudio de las publicaciones científicas más relevantes en el campo y de publicaciones más relevantes referentes a la librería SciANN, acompañado de unas pruebas iniciales (**T4.2**) con las que se afianzarán los conocimientos adquiridos.
- Tarea 5 (**T5**): Consiste en la definición y construcción del entorno experimental sobre el que va a realizarse la experimentación.
- Tarea 6 (**T6**): Se compone de las tareas de experimentación realizadas en el proyecto. Para cada tarea, se contempla el análisis del problema, el diseño del modelo de PINN asociado, la experimentación relativa a parámetros que definen el modelo y la evaluación de resultados.
- Tarea 7 (**T7**): Recoge el proceso de redacción de la memoria del proyecto.

1 Plan del proyecto

Desglose de tareas					
Tarea	Descripción	Duración	Inicio	Fin	Horas estimadas
T1	Identificación de conceptos	1 semana	21/08/2023	28/08/2023	8
T2.1	Estudio de Teoría de Distribuciones	21 semanas	04/09/2024	28/01/2024	130
T2.2	Asistencia a Análisis Funcional Avanzado	15 semanas	26/09/2023	16/01/2024	21
T3	Desarrollo de demostraciones en [CC95]	13 semanas	29/01/2024	28/04/2024	80
T4.1	Estudio bibliográfico de PINNs	3 semanas	01/04/2024	21/04/2024	50
T4.2	Pruebas iniciales complementarias	3 semanas	01/04/2024	21/04/2024	15
T5	Construcción del entorno experimental	1 semana	22/04/2024	28/04/2024	10
T6	Experimentación	7 semanas	29/04/2024	17/06/2024	100
T7	Redacción de la memoria	19 semanas	01/04/2024	10/07/2024	80

Tabla 1.1: Desglose del proyecto en tareas.

1.2. Planificación temporal del proyecto

En la [Tabla 1.1](#) se muestra la planificación temporal estimada para el proyecto. Debido a la gran carga teórica previa, necesaria para comprender los resultados que se proponen en [CC95], las tareas de implementación (**T4.1** en adelante) no comienzan hasta abril, una vez los conocimientos teóricos que la fundamentan están asentados. Además, la [Figura 1.1](#) muestra cómo las fases teórica y de implementación ideadas se han visto solapadas durante el mes de abril, debido a la carga mencionada.

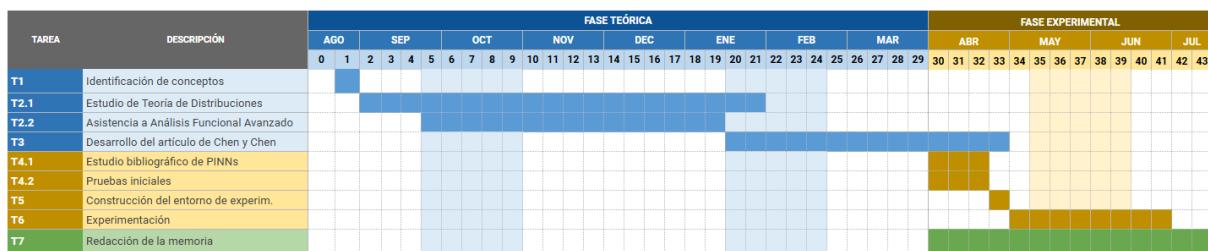


Figura 1.1: Diagrama de Gantt asociado al proyecto.

1.3. Estimación de coste

Para la estimación del coste del proyecto, se han consultado distintas fuentes que garanticen una estimación fiable del precio por cada recurso. De acuerdo con el Informe de Tendencias Salariales publicado por la agencia de recursos humanos Randstad para el año 2023 ¹, se estima que el salario medio de un Ingeniero de Software de rango medio en España es de 20€/hora. De acuerdo con la planificación temporal, se estima que se ha dedicado un total de 494 horas de trabajo. Esta cantidad de horas es superior a las que la normativa indica para el TFG de la titulación (18 ECTS).

Con respecto a los recursos necesarios para el desarrollo del proyecto, debe contemplarse la adquisi-

¹El informe puede consultarse en <https://www.randstadresearch.es/tendencias-salariales/>.

1.3 Estimación de coste

ción de ordenadores, con una estimación de 500€/unidad, así como de suscripciones a Google Colab Pro+ (51.12€/mes), respetando así el entorno de desarrollo del proyecto en este trabajo así como que se cuente con una capacidad de cómputo que garantice una buena experimentación co modelos de aprendizaje profundo. Por último, sería conveniente el alquiler de un espacio de co-working que garantice un ambiente adecuado para el desarrollo del proyecto. El precio medio de alquiler de este espacio en Granada es de 130€/mes por empleado.

Dado que vamos a contar con un único empleado para realizar todas las tareas, se necesitaría un único recurso de cada tipo especificado durante un periodo de 11 meses. En la [Tabla 1.2](#) se muestra el desglose del coste.

Recurso	Coste
Personal	9.880€
Ordenadores	500€
Subscripciones	562,32€
Espacio de co-working	1.430€
Total	12.372,32€

Tabla 1.2: Coste estimado del proyecto.

2 Herramientas matemáticas

En este capítulo se recogen los resultados trabajados durante el grado que han sido necesarios para la elaboración del trabajo.

2.1. Algunas cuestiones básicas de Teoría de la Medida

Definición 2.0.1. Dado un conjunto no vacío Ω , una σ -álgebra en Ω es una familia \mathcal{A} de partes de Ω que contenga a Ω y sea estable por complementación y por unión numerable, esto es, $\mathcal{A} \subseteq \mathcal{P}(\Omega)$ es una σ -álgebra en Ω si verifica las siguientes propiedades:

1. $\Omega \in \mathcal{A}$.
2. $E \in \mathcal{A} \Rightarrow \Omega \setminus E \in \mathcal{A}$.
3. Si $E_n \in \mathcal{A}$ para todo $n \in \mathbb{N}$, entonces $\bigcup_{n \in \mathbb{N}} E_n \in \mathcal{A}$.

Un espacio medible es un par (Ω, \mathcal{A}) donde Ω es un conjunto no vacío y \mathcal{A} es una σ -álgebra en Ω . Los elementos de \mathcal{A} se suelen llamar conjuntos medibles.

Definición 2.0.2. Si (X, \mathcal{T}) es un espacio topológico, la σ -álgebra engendrada por la topología \mathcal{T} recibe el nombre de σ -álgebra de Borel de X y sus elementos son los conjuntos Borel medibles (o boreelianos) de X .

Definición 2.0.3. Sean $(\Omega_1, \mathcal{A}_1), (\Omega_2, \mathcal{A}_2)$ espacios medibles. Decimos que una función $f : \Omega_1 \rightarrow \Omega_2$ es medible cuando la imagen inversa por f de cualquier conjunto medible en Ω_2 es medible en Ω_1 .

Definición 2.0.4. Dado un espacio medible (Ω, \mathcal{A}) , una medida en él es una función $\mu : \mathcal{A} \rightarrow [0, \infty]$ verificando:

- $\mu(\emptyset) = 0$.
- Si $E_n : n \in \mathbb{N}$ es una familia numerable de elementos disjuntos de \mathcal{A} dos a dos, se tiene

$$\mu(\bigcup_{n \in \mathbb{N}} E_n) = \sum_{n=1}^{\infty} \mu(E_n).$$

2 Herramientas matemáticas

Definición 2.0.5. Un espacio de medida es una terna $(\Omega, \mathcal{A}, \mu)$ donde Ω es un conjunto, \mathcal{A} una σ -álgebra en Ω y μ una medida definida en \mathcal{A} .

Teorema 2.1. Existe una σ -álgebra \mathcal{M} en \mathbb{R}^d y m una medida definida en \mathcal{M} con las siguientes propiedades:

- \mathcal{M} contiene a la σ -álgebra de Borel en \mathbb{R}^d .
- m es invariante por traslaciones.
- *Complitud: Si $E \in \mathcal{M}$, $m(E) = 0$ y $\mathcal{A} \subseteq E$, entonces $\mathcal{A} \in \mathcal{M}$.*
- *El par (\mathcal{M}, m) verificando estas propiedades es único.*

Los elementos de \mathcal{M} se llaman conjuntos de Lebesgue medibles en \mathbb{R}^d y m es la medida de Lebesgue en \mathbb{R}^d .

Definición 2.1.1. Llamamos función simple positiva a toda función $s : \Omega \rightarrow [0, \infty[$ medible cuya imagen sea un subconjunto finito de \mathbb{R}^+ . Notamos por \mathcal{S} al conjunto de las funciones simples positivas en Ω . Si $s(\Omega) = \alpha_1, \dots, \alpha_n$ es una enumeración de los valores que toma s y, para cada $k \in 1, \dots, n$, notamos $A_k = s^{-1}(\alpha_k)$, los conjuntos A_k son medibles, forman una partición de Ω y $s = \sum_{k=1}^n \alpha_k \chi_{\alpha_k}$. Esta expresión recibe el nombre de descomposición canónica de la función simple positiva s .

Definición 2.1.2. Sea $s \in \mathcal{S}$ y un conjunto medible E , llamamos integral de Lebesgue de s sobre E (con respecto a la medida μ), a

$$\int_E s \mu = \sum_{k=1}^n \alpha_k \mu(E \cap A_k) \in [0, \infty[,$$

donde $s = \sum_{k=1}^n \alpha_k \chi_{A_k}$ es la descomposición canónica de s . Dicha descomposición es única, salvo el orden de los sumandos, por lo que está bien definida.

2.2. Resultados auxiliares de Análisis Matemático

Teorema 2.2 (Teorema de Hausdorff). *Toda biyección continua entre dos espacios normados de dimensión finita es un isomorfismo.*

Teorema 2.3 (Teorema de derivación bajo el signo integral). *Sea U un subconjunto compacto de $\mathbb{R}^n \times \mathbb{R}^m$, y sea $f : U \rightarrow \mathbb{R}$ una función continua en todo U cuya derivada parcial $\frac{\partial f}{\partial y}$ existe y es continua en U . Entonces, si A y B son subconjuntos con volumen de \mathbb{R}^n y \mathbb{R}^m respectivamente, tales que B es abierto y $A \times B \subseteq U$, se tiene que la función $F : B \rightarrow \mathbb{R}$ definida por*

$$F(y) = \int_A f(x, y) dx$$

es diferenciable en B y

$$F'(y) = \int_A \frac{\partial f}{\partial y}(x, y) dx$$

para todo $y \in B$.

Definición 2.3.1. Un funcional sublineal en un espacio vectorial X es una función $p : X \rightarrow \mathbb{R}$ tal que:

1. $p(x + y) \leq p(x) + p(y)$ para cualesquiera $x, y \in X$.
2. $p(\alpha x) = \alpha p(x)$, $\forall \alpha \geq 0$ y $\forall x \in X$.

Teorema 2.4 (Teorema de extensión de Hahn-Banach). *Sea X un espacio vectorial y p un funcional sublineal en X . Si M es un subespacio de X y g es un funcional en M verificando*

$$\text{Reg}(m) \leq p(m), \quad (m \in M)$$

entonces existe un funcional lineal f en X cuya restricción a M coincide con g y que verifica

$$\text{Ref}(x) \leq p(x), \quad (x \in X)$$

Teorema 2.5 (Teorema de representación de Riesz). *Sea X un espacio de Hausdorff localmente compacto. Sea Λ un funcional lineal positivo en el espacio de funciones continuas en X de soporte compacto, que denotaremos $C_c(X)$. Entonces existe una σ -álgebra M en X que contiene a todos los conjuntos de Borel en X , y existe una medida positiva μ en M que satisface:*

1. Para todo $K \subseteq X$ compacto, $K \in M$ y $\mu(K) < \infty$.
2. Para todo $E \in M$, $\mu(E) = \inf\{\mu(V) : E \subset V, V$ abierto $\}$.
3. Para todo $E \in M$ tal que $\mu(E) < \infty$, $\mu(E) = \sup\{\mu(K) : K \subset E, K$ compacto $\}$.
4. Para toda $f \in C_c(X)$, $\Lambda(f) = \int_X f \mu$.

Teorema 2.6 (Aplicación del teorema de extensión de Tietze). *Sea X un espacio normado. Sea $f : A \rightarrow \mathbb{R}$ una función continua de un subconjunto cerrado A de X en \mathbb{R} con la topología estándar. Entonces, existe una extensión continua de f a X , esto es, existe una función $F : A \rightarrow \mathbb{R}$ continua en todo X con $F(a) = f(a)$ para todo $a \in A$. Además, se puede escoger F tal que $\sup\{|f(a)| : a \in A\} = \sup\{|F(x)| : x \in X\}$. Esto es, si f está acotada entonces F puede escogerse acotada.*

Lema 2.6.1. *Sea K un compacto en \mathbb{R}^n , $f \in C(K)$. Entonces, existe una función continua $E(f) \in C(\mathbb{R}^n)$ tal que:*

1. $f(x) = E(f)(x)$, $\forall x \in K$.
2. $\sup_{x \in \mathbb{R}^n} \{|E(f)(x)|\} \leq \sup_{x \in K} \{|f(x)|\}$.

2 Herramientas matemáticas

3. Existe una constante c tal que

$$\sup_{|x'-x''|<\delta} \{|E(f)(x') - E(f)(x'')|\} \leq c \sup_{|x'-x''|<\delta} \{|f(x') - f(x'')|\} \quad (x', x'' \in K).$$

Lema 2.6.2 (Teorema de Ascoli-Azela). *V es un conjunto compacto en $C(K)$ si, y solo si:*

1. *V es cerrado en $C(K)$.*
2. *Existe una constante M tal que $\|f(x)\|_{C(K)} \leq M, \forall f \in V$.*
3. *V es equicontinuo.*

Lema 2.6.3 (Lema de Riemann-Lebesgue). *Si $f \in L^1(\mathbb{R}^n)$, entonces*

$$\mathcal{F}(f)(\xi) = \int_{\mathbb{R}^n} f(x) e^{i\xi \cdot x} dx \rightarrow 0$$

cuando $|\xi| \rightarrow \infty$. Es decir, $\mathcal{F}(f) \in C_0(\mathbb{R}^n)$.

Teorema 2.7 (Teorema de la gráfica cerrada en espacios de Banach). *Sean X e Y espacios de Banach. Entonces, toda aplicación lineal f de X en Y tal que el conjunto*

$$G(f) = \{(x, y) \in X \times Y : y = f(x)\}$$

es cerrado en $X \times Y$, es continua.

Teorema 2.8 (Teorema de la convergencia dominada de Lebesgue). *Sea $\{f_n\}$ una sucesión de funciones medibles definidas en Ω que converge puntualmente a una función f medible. Si existe una función $g \in L_1(\Omega)$ tal que*

$$|f_n(x)| \leq g(x) \quad (\forall n \in \mathbb{N}, \forall x \in \Omega)$$

entonces f es integrable y, en particular,

$$\lim_{n \rightarrow \infty} \int_{\Omega} f_n = \int_{\Omega} f.$$

3 Espacios Vectoriales Topológicos

El camino hacia la Teoría de Distribuciones comienza con la presentación del tipo de espacios en el que viven estos objetos: los Espacios Vectoriales Topológicos. Como su nombre indica, estos espacios permiten la definición y estudio de propiedades tanto topológicas como algebraicas. Esta estructura resulta especialmente útil en Análisis Funcional, ya que todos los espacios de funciones tienen estructura de espacio vectorial y en ellos se introducen nociones de convergencia para los elementos del espacio. No es de extrañar, por tanto, que el lugar donde viven las “funciones generalizadas” posea la misma estructura.

3.1. Presentación y construcción de EVT

Definición 3.0.1. Sea τ una topología sobre un espacio vectorial X cumpliendo que

- Cada punto de X es un cerrado en (X, τ) .
- Las aplicaciones suma y producto por escalares son continuas con respecto a τ .

Entonces, decimos que τ es una topología vectorial en X y X con la topología τ es un Espacio Vectorial Topológico (EVT).

Definición 3.0.2. Diremos que un subconjunto U de un espacio vectorial X es absorbente si $X = \mathbb{R}^+ U$. Diremos que U es equilibrado si $\mathbb{D}U \subseteq U$, donde \mathbb{D} es la bola unitaria cerrada en \mathbb{R} o \mathbb{C} . Dado un conjunto cualquiera V de X , llamaremos envolvente equilibrada de V al mínimo subconjunto equilibrado de X que contiene a V , esto es, $\mathbb{D}V$.

Observación 3.0.1. Que un conjunto sea equilibrado no implica que sea absorbente, ya que le pueden faltar vectores linealmente independientes al resto para llegar a cubrir toda la esfera unidad de X . Recíprocamente, un conjunto absorbente no tiene por qué ser equilibrado (es decir, no tiene por qué ser invariante por rotaciones).

Definición 3.0.3. Sean X un espacio vectorial y $E \subseteq X$. Definimos el Funcional de Minkowski de E , $v_E : X \rightarrow [0, \infty[$ como

$$v_E(x) = \inf \{\rho > 0 : x \in \rho E\} \quad (x \in X).$$

Definición 3.0.4. Una familia \mathcal{B} de subconjuntos no vacíos de un conjunto X es una base de filtro si verifica que para cada $U, V \in \mathcal{B}$ existe algún $W \in \mathcal{B}$ tal que $W \subset U \cap V$.

3 Espacios Vectoriales Topológicos

Teorema 3.1 (Caracterización de las bases de entornos de cero en un EVT). *Todo entorno de cero U en un EVT es absorbente y contiene un entorno de cero equilibrado V tal que $V + V \subset U$. Recíprocamente, sea X un espacio vectorial y \mathcal{B} una base de filtro en X formada por conjuntos absorbentes y equilibrados, verificando que para cada $U \in \mathcal{B}$ existe $V \in \mathcal{B}$ tal que $V + V \subset U$. Entonces \mathcal{B} es base de entornos de cero para una (única) topología vectorial en X .*

Proposición 3.1.1. *Sea X un EVT. Entonces, los operadores traslación $T_a(x) = a + x$, $\forall a, x \in X$ y, para cada $\lambda \neq 0$, los operadores multiplicación $M_\lambda(x) = \lambda x$, $\forall x \in X$ son homeomorfismos de X . Esto es, todas las traslaciones, giros y homotecias son homeomorfismos en X .*

Como consecuencia de la proposición anterior, tomando una base de entornos de cero \mathcal{B} en X , $\{x + \mathcal{B}\}$ será una base de entornos para cada $x \in X$. Por tanto, una topología vectorial queda totalmente determinada por una base de entornos de cero.

Definición 3.1.1. *Sea X un EVT y A un subconjunto de X . Decimos que A es precompacto si para cada entorno de cero U , existe un subconjunto finito $F \subset X$ tal que $A \subseteq F + U$.*

Teorema 3.2 (Teorema de Tihonov). *Si X e Y son EVT separados de dimensión finita, toda biyección lineal de X sobre Y es un isomorfismo.*

Lema 3.2.1 (Lema de Riesz). *Sea X un EVT, M un subespacio cerrado y A un subconjunto acotado de X . Supongamos que existe un $\lambda \in \mathbb{K}$ con $|\lambda| < 1$ tal que $A \subset M + \lambda A$. Entonces $A \subset M$.*

Teorema 3.3 (Teorema de Riesz). *Sea X un EVT separado. Entonces equivalen:*

1. *X es localmente compacto.*
2. *Existe en X un entorno de cero compacto.*
3. *Existe en X un entorno de cero precompacto.*
4. *La dimensión de X es finita.*

Demostración.

- 4. \Rightarrow 1. Como X tiene dimensión finita, como consecuencia del **Teorema de Tihonov** sabemos que su topología es equivalente a la topología inducida por la norma euclídea en la misma dimensión de X . Como la topología inducida por la norma euclídea es localmente compacta, la topología de X también lo será.
- 1. \Rightarrow 2. \Rightarrow 3. Son evidentes.
- 3. \Rightarrow 4. Sea U un entorno de cero precompacto en X . Entonces, por definición, existirá un subconjunto finito F de X tal que, fijando $\lambda = \frac{1}{2}$, $U \subset F + \lambda U$. Si M es el subespacio de X generado por

3.2 Convergencia uniforme y complitud

F, M tiene dimensión finita y, por tanto, es cerrado en X . Como U está contenido en $M + \lambda U$ y es precompacto (por tanto, acotado), el **Lema de Riesz** nos dice que $U \subset M$. Como sabemos además que U es absorbente podemos deducir que $X = M$, por lo que X tiene dimensión finita.

□

Como hemos visto, el **Teorema de Tihonov** (1935) extiende al **Teorema de Hausdorff** (1932) para espacios normados de dimensión finita. Análogamente, el **Lema de Riesz** y el **Teorema de Riesz** son extensiones de los conocidos resultados para estos espacios. Como consecuencia de este último teorema, podemos deducir que todo subconjunto precompacto de un EVT de dimensión infinita tiene interior vacío. Esto tiene un gran impacto cuando trabajamos en espacios de funciones, por lo que volveremos a ello más adelante.

3.2. Convergencia uniforme y complitud

Las topologías vectoriales no son metrizables en general. Por tanto, las sucesiones pueden no ser suficientes para caracterizar la topología. Sin embargo, generalizando el concepto de sucesión, es posible manejar algunos aspectos de topologías generales como si fuesen metrizables.

Definición 3.3.1. Un conjunto dirigido es un conjunto no vacío Λ dotado de un preorden, \leqslant , verificando que para cualesquiera $\lambda_1, \lambda_2 \in \Lambda$, existe un $\lambda \in \Lambda$ tal que $\lambda_1 \leqslant \lambda$ y $\lambda_2 \leqslant \lambda$.

Si X es un conjunto no vacío, una red de elementos de X es una aplicación $\phi : \Lambda \rightarrow X$, donde Λ es un conjunto dirigido. De manera análoga a la notación para sucesiones, si $\phi : \Lambda \rightarrow X$ es una red, escribimos $x_\lambda = \phi(\lambda)$, $(x_\lambda)_{\lambda \in \Lambda} = \phi$.

Definición 3.3.2. Sea X un espacio topológico. Decimos que una red $(x_\lambda) \subset X$ converge a $x \in X$ y escribimos $(x_\lambda) \rightarrow x$ si para todo entorno U de x puede encontrarse un índice $\lambda_0 \in \Lambda$ tal que

$$\lambda \in \Lambda, \quad \lambda_0 \leqslant \lambda \quad \Rightarrow \quad x_\lambda \in U.$$

Definición 3.3.3. Decimos que x es un valor adherente a la red (x_λ) si para todo entorno U de x y para todo $\lambda \in \Lambda$ puede encontrarse $\mu \in \Lambda$ tal que $\lambda \leqslant \mu$ y $x_\mu \in U$. Es claro que si $(x_\lambda) \rightarrow x$, x es un valor adherente a la red (x_λ) , no siendo cierto el recíproco.

Sea $\phi = (x_\lambda)_{\lambda \in \Lambda}$ una red en X . Sea $A(\phi) = \{A_\lambda : \lambda \in \Lambda\}$ la familia de subconjuntos de X definida por

$$A_\lambda = \{x_\mu : \mu \in \Lambda, \lambda \leqslant \mu\} \quad (\lambda \in \Lambda).$$

Entonces A_λ cumple las siguientes condiciones:

3 Espacios Vectoriales Topológicos

1. $A_\lambda \neq \emptyset, \forall \lambda \in \Lambda.$
2. $\forall \lambda_1, \lambda_2 \in \Lambda$, existe $\lambda \in \Lambda$ tal que $A_\lambda \subset A_{\lambda_1} \cup A_{\lambda_2}$ (Bastará que $\lambda_1 \leq \lambda$ y $\lambda_2 \leq \lambda$).

Si X es además un espacio topológico, podemos caracterizar la posible convergencia de ϕ en términos de la familia $A(\phi)$:

1. Dado $x \in X$, ϕ converge a x si, y solo si, cada entorno de x contiene a un elemento de $A(\phi)$.
2. x es valor adherente a la red ϕ si, y solo si, cada entorno de x tiene intersección no vacía con cada elemento de $A(\phi)$.
3. El conjunto de los valores adherentes a ϕ coincide con $\bigcup_{A \in A(\phi)} \bar{A}$.

Proposición 3.3.1. Sea X un espacio topológico, $A \subset X$ y $x \in X$. Las siguientes afirmaciones son equivalentes:

1. $x \in \bar{A}$.
2. Existe una red en A que converge a x .
3. x es un valor adherente a una red en A .

Proposición 3.3.2. Sea X un espacio topológico Hausdorff y (x_λ) una red en X que converge a $x \in X$. Entonces, x es el único valor adherente a la red (x_λ) . Recíprocamente, si toda red en X converge, a lo sumo, a un punto de X , entonces X es un espacio de Hausdorff.

Proposición 3.3.3. Sean X e Y espacios topológicos, $f : X \rightarrow Y$ una función y $x \in X$. Las siguientes afirmaciones son equivalentes:

1. f es continua en x .
2. $(x_\lambda) \rightarrow x \Rightarrow f(x_\lambda) \rightarrow f(x)$.

Definición 3.3.4. Se dice que una red $(x_\lambda)_{\lambda \in \Lambda}$ de elementos de un EVT X es una red de Cauchy si para cada entorno de cero U puede encontrarse un índice $\lambda_0 \in \Lambda$ tal que,

$$\lambda, \mu \in \Lambda, \lambda_0 \leq \lambda, \lambda_0 \leq \mu \quad \Rightarrow \quad x_\lambda - x_\mu \in U$$

equivalentemente, $B_{\lambda_0} - B_{\lambda_0} \subset U$, donde $B_{\lambda_0} = \{x_\lambda : \lambda \in \Lambda, \lambda_0 \leq \lambda\}$.

Lema 3.3.1. Toda red de Cauchy en un EVT que posea un valor adherente, converge a dicho valor adherente.

Definición 3.3.5. Sea X un EVT y d una semidistancia. Decimos que d es invariante por traslaciones cuando cumple:

$$d(x+z, y+z) = d(x, y) \quad (x, y, z \in X).$$

Proposición 3.3.4. Sea X un EVT y d una semidistancia completa que genera una topología de X y es invariante por traslaciones. Entonces, toda red de Cauchy en X es convergente.

Definición 3.3.6. Se dice que un EVT X es completo cuando toda red de Cauchy en X es convergente. Si ocurre solamente que toda sucesión de Cauchy es convergente, decimos que X es secuencialmente completo. Análogamente se definen la complitud y la complitud secuencial de un subconjunto de un EVT.

Proposición 3.3.5. Sea X un EVT cuya topología proviene de una semidistancia d invariante por traslaciones. Entonces, son equivalentes:

1. X es completo.
2. X es secuencialmente completo.
3. La distancia d es completa.

3.3. Aplicaciones lineales entre EVT

Definición 3.3.7. Sean X e Y EVT y sea $F : X \rightarrow Y$ una aplicación. Decimos que F es uniformemente continua si para cada entorno de cero V en Y existe un entorno de cero U en X tal que $F(x) - F(y) \in V$, para todos $x, y \in X$ tales que $x - y \in U$.

Proposición 3.3.6. Sean X e Y dos EVT y $T : X \rightarrow Y$ una aplicación lineal. Son equivalentes:

- T es uniformemente continua.
- T es continua.
- T es continua en cero, esto es, $T^{-1}(V)$ es entorno de cero en X para cada entorno de cero V en Y .

3.4. Topologías iniciales

Definición 3.3.8. Sea $X \neq \emptyset$ un conjunto, $\{(X_i, \mathcal{T}_i), i \in I\}$ espacios topológicos, $\{f_i : i \in I\}$ una familia de aplicaciones definidas en X , cada una tomando valores en un espacio topológico X_i respectivamente. Llamamos topología inicial para $\{f_i : i \in I\}$ a la mínima topología en X que hace continuas a todas las aplicaciones f_i .

Proposición 3.3.7. Sea X un espacio vectorial, $\{X_i : i \in I\}$ una familia de EVT y, para cada $i \in I$, sea f_i una aplicación lineal de X en X_i . Entonces, la topología inicial en X para la familia $\{f_i : i \in I\}$ es una topología

3 Espacios Vectoriales Topológicos

vectorial en X . Si, para cada $i \in I$, \mathcal{B}_i es una base de entornos de cero en X_i , la familia

$$\mathcal{B} = \left\{ \bigcup_{j \in J} f_j^{-1}(U_j) : J \subset I, J \text{ finito}, U_j \in \mathcal{B}_j \forall j \in J \right\}$$

es base de entornos de cero en X para dicha topología inicial.

3.5. EVT Localmente Convexos

Definición 3.3.9. Sea X un espacio vectorial. Una topología localmente convexa en X es una topología vectorial que admite una base de entornos de cero convexos. Un espacio localmente convexo (ELC) es un par (X, \mathcal{T}) formado por un espacio vectorial X y una topología localmente convexa \mathcal{T} en X . Si no hay lugar a confusión, diremos que X es un ELC.

Definición 3.3.10. Sea X un espacio vectorial y $E \subset X$. Llamamos envolvente convexa de E a la intersección de todos los subconjuntos convexos de X que contienen a E . Esto es, al menor subconjunto convexo de X que contiene a E .

Teorema 3.4 (Caracterización de los entornos de cero y de las bases de entornos de cero en un ELC). *Todo ELC posee una base de entornos de cero formada por conjuntos equilibrados y convexos.*

Observación 3.4.1. Consecuencias directas:

- En un EVT, el cierre y el interior de un conjunto convexo son conjuntos convexos.
- En cualquier EVT, la envolvente convexa de un conjunto abierto es abierta.

Definición 3.4.1. Sea X espacio vectorial, decimos que una aplicación $\nu : X \rightarrow \mathbb{R}$ es una pseudonorma cuando cumple:

1. $\nu(x + y) \leq \nu(x) + \nu(y) \quad \forall x, y \in X$.
2. $\lambda \in \mathbb{D} \Rightarrow \nu(\lambda x) \leq |\lambda| \nu(x) \quad \forall x \in X$.
3. $\lim_{n \rightarrow \infty} \nu\left(\frac{x}{n}\right) = 0 \quad \forall x \in X$.

Por otro lado, una aplicación $p : X \rightarrow \mathbb{R}$ es una seminorma en X cuando:

1. $p(x + y) \leq p(x) + p(y) \quad \forall x, y \in X$.
2. $p(\lambda x) \leq |\lambda| p(x) \quad \forall \lambda \in \mathbb{K}, \forall x \in X$.

De la definición podemos ver cómo toda seminorma es una pseudonorma.

El siguiente resultado refleja lo relacionadas que están las seminormas con la convexidad local en los EVT:

Teorema 3.5 (Caracterización de las topologías localmente convexas). *Una topología \mathcal{T} en un espacio vectorial X es localmente convexa si, y solo si, es la topología asociada a una familia de seminormas en X . Más concretamente, sea X un ELC y sea \mathcal{B} una familia de entornos de cero convexos y equilibrados. Si para cada $U \in \mathcal{B}$, ϕ_U es el funcional de Minkowski de U , la topología de partida en X es la asociada a la familia de seminormas $\{\phi_U : U \in \mathcal{B}\}$.*

Corolario 3.5.1. *Todo ELC separado es isomorfo a un subespacio de un producto de espacios normados.*

3.6. EVT Metrizable

Decimos que un espacio topológico (X, \mathcal{T}) es metrizable si es homeomorfo a un espacio métrico, esto es, si existe una distancia en X que genere su topología. Por tanto, en el contexto de los EVT, es natural preguntarse en qué casos dicha topología es vectorial y, análogamente, cuándo podemos metrizar o asociar una distancia a una topología vectorial dada. En esta sección introduciremos formalmente el concepto de metrizabilidad en espacios topológicos y daremos respuesta a las dos cuestiones planteadas.

Definición 3.5.1. Sean X un espacio vectorial y ν una pseudonorma en X . Definiendo para cada $\varepsilon > 0$, $U_\varepsilon = \{x \in X : \nu(x) \leq \varepsilon\}$ tenemos que $\{U_\varepsilon : \varepsilon > 0\}$ es una base de entornos de cero para la topología vectorial en X asociada a la pseudonorma ν , con la que X es un EVT pseudonormal (seminormal cuando ν es una seminorma).

Definición 3.5.2. Un espacio topológico X es semimetrizable si existe una semidistancia d en X que genera su topología. Si la semidistancia d es una distancia, decimos que el espacio topológico X es metrizable.

Observación 3.5.1. La topología asociada a una pseudonorma es siempre semimetrizable, esto es, existe una semidistancia que genera la topología. Concretamente, si $\nu : X \rightarrow \mathbb{R}$ es una pseudonorma, la aplicación $d : X \times X \rightarrow \mathbb{R}$ dada por $d(x, y) = \nu(x - y)$, es una semidistancia que genera la topología.

La primera cuestión es agenciada por el resultado:

Teorema 3.6. *Sea X un espacio vectorial y sea d una semidistancia en X verificando:*

1. *d es invariante por traslaciones.*
2. *Si (λ_n) es una sucesión convergente a cero en \mathbb{K} , entonces $d(\lambda_n x, 0) \rightarrow 0$ para cada $x \in X$.*

3 Espacios Vectoriales Topológicos

3. Si (x_n) es una sucesión en X tal que $d(x_n, 0) \rightarrow 0$, entonces $d(\lambda x_n, 0) \rightarrow 0$ para cada $\lambda \in \mathbb{K}$.

Entonces la topología \mathcal{T} asociada a d es una topología vectorial. Además, (X, \mathcal{T}) es completo si, y sólo si, d es completa.

La segunda problemática planteada se resuelve en el siguiente resultado:

Teorema 3.7 (Criterio de metrizabilidad de Birkhoff-Kakutani). *Si X es un EVT, equivalen:*

1. X es pseudonormal.
2. X es semimetrizable.
3. X tiene una base numerable de entornos de cero.

Es decir, un EVT es metrizable si, y solo si, es separado y tiene una base numerable de entornos de cero.

Ya tenemos todas las herramientas necesarias para presentar los dos siguientes conceptos, de los que haremos uso durante el trabajo:

Definición 3.7.1. Llamamos F-espacio a un EVT completo metrizable. Si X tiene la topología asociada a una pseudonorma ν , X es un F-espacio cuando toda serie absolutamente convergente es convergente:

$$x_n \in X \quad \forall n \in \mathbb{N}, \quad \sum_{n=1}^{\infty} \nu(x_n) < \infty \Rightarrow \sum_{n \geq 1} x_n < \infty$$

Definición 3.7.2. Llamamos espacio de Fréchet a un F-espacio localmente convexo. Esto es, un espacio localmente convexo que es completo y metrizable.

3.7. Teoría de dualidad

Definición 3.7.3. Dado un espacio vectorial X , denotaremos como $X^\#$ a su dual algebraico, esto es, al espacio formado por los funcionales lineales en X . Si (X, τ) es un EVT, definimos su dual topológico $(X, \tau)^*$ como el subespacio de $X^\#$ formado por los funcionales lineales y τ -continuos en X . Si τ se sobrentiende, escribimos X^* .

Corolario 3.7.1. Sea X un espacio vectorial, U un subconjunto absorbente y convexo de X y $x_0 \in X$ tal que $x_0 \notin U$. Existe un funcional lineal f en X tal que

$$\operatorname{Re} f(x) \leq 1 \quad (x \in U) \quad y \quad \operatorname{Re} f(x_0) \geq 1.$$

Demostración. Sea μ el funcional de Minkowski en U . Entonces, μ es sublineal en X y se tiene que

$$\{x \in X : \mu(x) < 1\} \subset U \subset \{x \in X : \mu(x) \leq 1\}$$

por lo que $\mu(x_0) \geq 1$ y $\lambda\mu(x_0) \leq \mu(\lambda x_0)$ se cumple para todo $\lambda > 0$ por la homogeneidad de μ . Por tanto, tomando el subespacio $\mathbb{R}\{x_0\}$ de X y el funcional lineal que a cada $\lambda \in \mathbb{R}$ le asigna $g(\lambda) = \lambda\mu(x_0)$, definido en $\mathbb{R}\{x_0\}$, el **Teorema de Hahn-Banach** nos proporciona un funcional $f \in X^\#$ que cumple

$$f(x) \leq \mu(x) \quad (x \in X) \quad y \quad f(x_0) = \mu(x_0).$$

□

Corolario 3.7.2. *Sea X un EVT. Las siguientes afirmaciones son equivalentes:*

1. X^* separa los puntos de X .
2. Para cada $x_0 \in X \setminus \{0\}$ existe una seminorma continua ϕ en X tal que $\psi(x_0) \neq 0$.
3. La intersección de todos los entornos convexos de cero en X se reduce a $\{0\}$.

Demostración.

- 1. \Rightarrow 2. Si X^* separa los puntos de X , fijando un $x_0 \in X \setminus \{0\}$ y un $f \in X^*$ (el cual, por 1, cumplirá que $f(x_0) \neq 0$), podemos definir la seminorma continua $\varphi = |f|$, que verifica $\varphi(x_0) \neq 0$.
- 2. \Rightarrow 3. Sea ψ la seminorma definida en 2. Entonces, los entornos de cero convexos asociados a la topología de la seminorma vendrán dados por $U_{x_0} = \{x \in X : \psi(x) < \psi(x_0)\}$ y cumplirán que $x_0 \notin U_{x_0}$, para cada $x_0 \in X \setminus \{0\}$. Por tanto, $\bigcap_{x_0 \in X \setminus \{0\}} U_{x_0} = \{0\}$.
- 3. \Rightarrow 1. El corolario anterior nos proporciona, para cada $x_0 \in X \setminus \{0\}$, un funcional $f \in X^*$ cumpliendo que $f(x_0) = \mu(x_0) \neq 0$.

□

Corolario 3.7.3. *Sea X un ELC. Entonces equivalen:*

1. X^* separa los puntos de X .
2. X es separado.

Demostración. En un ELC todos los entornos de cero son localmente convexos, por lo que la equivalencia entre 1. y 3. del corolario anterior se traduce en este resultado. □

Teorema 3.8. *Sean A y B subconjuntos no vacíos, convexos, disjuntos, de un ELC X tales que A es cerrado y B*

3 Espacios Vectoriales Topológicos

es compacto. Entonces, existen $f \in X^*$ y $\alpha, \beta \in \mathbb{R}$ tales que

$$\text{Ref}(a) \leq \alpha < \beta \leq \text{Ref}(B) \quad (a \in A, b \in B).$$

Observación 3.8.1. Si tomamos $B = \{x\}$ donde $x \in X \setminus A$, el teorema anterior nos muestra que f separa puntos de X .

3.8. Topología débil y débil-*

La compacidad es una propiedad muy útil y deseable pero difícil de garantizar en EVT de dimensión infinita pues, como consecuencia del **Teorema de Riesz para EVT separados**, cualquier compacto en tales espacios tiene interior vacío. Esto se materializa, por ejemplo, en los espacios de Banach de dimensión infinita, donde la topología de la norma presenta pocos compactos.

Por esta razón, en dichos espacios conviene trabajar con topologías más pequeñas que, preservando la continuidad de funcionales del dual, presenten más conjuntos compactos. Estas topologías se conocen como la topología débil y débil-*.

Definición 3.8.1. Sea X un espacio vectorial, Y un subespacio del dual algebraico $X^\#$ que separa los puntos de X . Entonces (X, Y) es un par dual. Escribimos $\langle x, y \rangle$ para denotar la acción de $y \in Y \leq X^\#$ sobre $x \in X$. Como X separa los puntos de Y y $X \leq Y^\#$, (Y, X) es par dual.

Definición 3.8.2. Sea (X, Y) par dual. La topología inicial en X para los elementos de Y se denota por $\sigma(X, Y)$ y se denomina la topología débil en X asociada al par dual (X, Y) . Se trata de una topología localmente convexa y separada en X , asociada a la familia de seminormas

$$\varphi_y(x) = |\langle x, y \rangle| \quad (x \in X, y \in Y).$$

Los conjuntos de la forma

$$U(J, \varepsilon) = \{x \in X : |\langle x, y \rangle| \leq \varepsilon \quad \forall y \in J\}$$

donde J es un subconjunto finito de Y y $\varepsilon > 0$ forman una base de entornos de cero para $\sigma(X, Y)$.

Proposición 3.8.1. Sea (X, Y) un par dual.

- Un funcional lineal f en X es $\sigma(X, Y)$ -continuo si, y sólo si, existe un $y_0 \in Y$ tal que

$$f(x) = \langle x, y_0 \rangle, \quad x \in X.$$

- La topología $\sigma(X, Y)$ es compatible con el par dual (X, Y) , esto es, $(X, \sigma(X, Y))^* = Y$; y esta es la mínima topología en X con esa propiedad.

Definición 3.8.3. Sea X un ELC separado, (X, X^*) par dual.

- $\sigma(X, X^*)$ es la topología débil de X .
- $\sigma(X^*, X)$ es la topología débil-* de X^* asociada al par dual (X^*, X) .

Observación 3.8.2. Se trata de la menor topología en X^* que hace continuos a los elementos de X . En particular, $(X^*, \sigma(X^*, X))^* = X$.

Observación 3.8.3. Se trata de la topología en X^* de la convergencia puntual sobre los elementos de X . Esto es, la topología en X^* de la convergencia uniforme sobre los subconjuntos finitos de X .

Observación 3.8.4. Sea X ELC. En general, es posible que no tengamos en X^* ninguna topología destacable. En ese caso, sabemos que al menos dispondremos de la topología débil-*.

3.9. Espacio de funciones test

En esta sección haremos una presentación del espacio de funciones test. Por un lado, esta presentación nos permitirá particularizar y afianzar los conceptos que se han expuesto para EVT. Por otro lado, como veremos más adelante, la presentación de estos espacios es esencial dentro del estudio de la Teoría de Distribuciones.

Definición 3.8.4. Sea Ω un conjunto abierto no vacío de \mathbb{R}^d . Llamamos multi-índice a la d-tupla $\alpha = (\alpha_1, \dots, \alpha_d)$ de enteros no negativos. A cada multi-índice α , podemos asociar un operador diferencial

$$D^\alpha = \left(\frac{\partial}{\partial x_1} \right)^{\alpha_1} \cdots \left(\frac{\partial}{\partial x_d} \right)^{\alpha_d}$$

cuyo orden es $|\alpha| = \alpha_1 + \cdots + \alpha_d$.

Observación 3.8.5. Cuando $|\alpha| = 0$, tenemos que $D^\alpha f = f$.

Definición 3.8.5. Una función f definida en $\Omega \subset \mathbb{R}^d$ pertenece a $C^\infty(\Omega)$ si cumple que $D^\alpha f \in C(\Omega)$ para todo multi-índice α .

Definición 3.8.6. Sea K un compacto en \mathbb{R}^d . Llamamos $\mathcal{D}(K)$ o \mathcal{D}_K al espacio de todas las $f \in C^\infty(\mathbb{R}^d)$ cuyo soporte queda contenido en K . Si $K \subset \Omega$, entonces $\mathcal{D}(K)$ se puede identificar con un subespacio de $C^\infty(\Omega)$.

A continuación veremos como, dotado de la topología adecuada, $C^\infty(\Omega)$ es un espacio de Fréchet con la propiedad de Heine-Borel y, por tanto, $\mathcal{D}(K)$ es un subespacio cerrado suyo, siempre que $K \subset \Omega$.

3 Espacios Vectoriales Topológicos

Para cada $N \in \mathbb{N}$, definimos la seminorma

$$p_N(f) = \max\{|D^\alpha f(x)| : x \in K_N, |\alpha| \leq N\}$$

siendo K_i conjuntos compactos que cumplen:

- $K_i \subset \text{int}(K_{i+1})$.
- $\Omega = \bigcup K_i$.

Lema 3.8.1. La familia de seminormas $\{p_N\}_{N \in \mathbb{N}}$ define una topología metrizable y localmente convexa en $C^\infty(\Omega)$. Para esta topología, los conjuntos

$$V_N = \left\{ f \in C^\infty(\Omega) : p_N(f) < \frac{1}{N} \right\}$$

con $N \in \mathbb{N}$ forman una base local.

Observación 3.8.6. Las normas

$$\|\phi\|_N = \max\{|D^\alpha f(x)| : x \in \Omega, |\alpha| \leq N\}$$

con $N \in \mathbb{N}$ y $\phi \in \mathcal{D}(\Omega)$, cuando son restringidas a cada \mathcal{D}_K fijo, inducen la misma topología que las seminormas p_N .

Observación 3.8.7. Para cada $x \in \Omega$, el funcional $f \rightarrow f(x)$ es continuo para dicha topología.

Lema 3.8.2. $C^\infty(\Omega)$ es un espacio de Fréchet.

Lema 3.8.3. $C^\infty(\Omega)$ tiene la propiedad de Heine-Borel.

Definición 3.8.7. Consideramos ahora el espacio vectorial $\mathcal{D}(\Omega) = \bigcup \mathcal{D}_K$. Entonces, $\phi \in \mathcal{D}(\Omega)$ si, y solo si, $\phi \in C^\infty(\Omega)$ y $\text{supp}(\phi) = K$, para algún subconjunto compacto K de Ω .

Definición 3.8.8. Para cada compacto $K \subset \Omega$, τ_K denota la topología heredada del espacio de Fréchet $C^\infty(\Omega)$.

Para cada \mathcal{D}_K , τ_K no es completa y por tanto tampoco lo será en $\mathcal{D}(\Omega)$. Es por esto que buscaremos definir una topología τ en $\mathcal{D}(\Omega)$ que sí lo sea.

Definición 3.8.9.

- Llamaremos β al conjunto de los $W \subset \mathcal{D}(\Omega)$ convexos y equilibrados tales que $\mathcal{D}_K \cap W \in \tau_K$ para todo compacto $K \in \Omega$.
- $\tau = \bigcup \phi + W$ tales que $\phi \in \mathcal{D}(\Omega)$ y $W \in \beta$.

Teorema 3.9.

1. τ es una topología en $\mathcal{D}(\Omega)$ y β es una base local para esta topología.
2. $\mathcal{D}(\Omega)$ dotado de la topología τ es un EVT localmente convexo.

Demostración.

1. Supongamos que $V_1, V_2 \in \tau$, $\phi \in V_1 \cap V_2$. Entonces, atendiendo a la definición de β , para demostrar
1. bastará con probar que existe $W \in \beta$ tal que $\phi + W \subseteq V_1 \cap V_2$.

Como $V_1, V_2 \in \tau$ y atendiendo a la construcción de τ , sabemos que existen $\phi_i \in \mathcal{D}(\Omega)$ y $W_i \in \beta$ tales que $\phi \in \phi_i + W_i \subset V_i$, para $i = 1, 2$.

Tomando un K lo suficientemente grande como para que $\phi_1, \phi_2 \in \mathcal{D}_K$, tendremos que $\mathcal{D}_K \cap W_i$ es abierto para la topología τ_K y, por tanto, $\phi - \phi_i \in (1 - \delta_i)W_i$ para algún $\delta_i > 0$. Como escogimos W_i convexo, tenemos que $\phi - \phi_i + \delta_i W_i \subset (1 - \delta_i)W_i + \delta_i W_i = W_i$ y por tanto $\phi + \delta_i W_i \subset \phi_i + W_i \subset V_i$, para $i = 1, 2$.

Por tanto, hemos encontrado el $W = (\delta_1 W_1) \cap (\delta_2 W_2) \in \beta$ que necesitábamos.

2. Tomemos ahora $\phi_1, \phi_2 \in \mathcal{D}(\Omega)$, y sea

$$W = \{\phi \in \mathcal{D}(\Omega) : \|\phi\|_0 < \|\phi_1 - \phi_2\|_0\}$$

donde $\|\cdot\|_0$ es la norma definida en esta sección. Entonces $W \in \beta$ y $\phi_1 \notin \phi_2 + W$. Por tanto, $\{\phi_1\}$ es un conjunto cerrado para la topología τ .

Como cada $W \in \beta$ es convexo, tenemos que

$$(\psi_1 + \frac{1}{2}W) + (\psi_2 + \frac{1}{2}W) = (\psi_1 + \psi_2) + W$$

para todas $\psi_1, \psi_2 \in \mathcal{D}(\Omega)$ siempre que $\psi_1 \neq \psi_2$. Por tanto, la suma es τ -continua.

Tomemos ahora un escalar α_0 y una $\phi_0 \in \mathcal{D}(\Omega)$. Entonces, para toda $\phi \in \mathcal{D}(\Omega)$ y todo escalar α se cumplirá:

$$\alpha\phi - \alpha_0\phi_0 = \alpha(\phi - \phi_0) + (\alpha - \alpha_0)\phi_0.$$

Además, si $W \in \beta$ existirá un $\delta > 0$ tal que $\delta\phi_0 \in \frac{1}{2}W$ y podremos escoger una constante c que

3 Espacios Vectoriales Topológicos

cumpla $2c(|\alpha_0| + \delta) = 1$. Como W es equilibrado y convexo, tenemos que

$$\alpha\phi - \alpha_0\phi_0 \in W$$

siempre que $\phi - \phi_0 \in cW$ y $|\alpha - \alpha_0| < \delta$.

□

Teorema 3.10.

1. Un conjunto convexo y equilibrado en $\mathcal{D}(\Omega)$ es abierto si, y solo si, pertenece a β .
2. La topología τ_K de cada $\mathcal{D}_K \subset \mathcal{D}(\Omega)$ coincide con la topología τ restringida a \mathcal{D}_K .
3. Sea E un subconjunto acotado de $\mathcal{D}(\Omega)$. Entonces existe un $K \in \Omega$ tal que $E \subset \mathcal{D}_K$. Además, existen constantes $M_N < \infty$ tales que $\|\phi\|_N \leq M_N$, $\forall \phi \in E$, $\forall N \in \mathbb{N}$.
4. $\mathcal{D}(\Omega)$ tiene la propiedad de Heine-Borel.
5. En $\mathcal{D}(\Omega)$, toda sucesión de Cauchy es convergente.

4 Teoría de Distribuciones

4.1. Distribuciones

La Teoría de Distribuciones, desarrollada por L. Schwartz durante las décadas de 1940 y 1950, surgió como respuesta a la necesidad de manejar “funciones generalizadas” desde varias ramas de la Física, donde el concepto clásico de función como observable físico presentaba limitaciones de precisión. En aquel momento ya se utilizaban formalismos como la δ de Dirac que funcionaban y daban respuesta a esta problemática sin haber sido aún integrados dentro de una teoría matemática que les diese respaldo. Planteada esta necesidad, se inició la búsqueda de una clase de objetos que extendiesen la noción de función y sobre la que se pudiese generalizar el cálculo diferencial clásico.

Dicha clase de objetos tendrá como objetivo albergar a todas las funciones continuas, permitir que cada objeto perteneciente a ella tenga derivadas parciales y que estas de nuevo sean un objeto de la misma clase. Además, se espera que dicho concepto de derivada case con el concepto clásico de derivada y con las propiedades existentes para funciones diferenciables.

Una primera aproximación intuitiva puede exponerse tomando una función $f : \mathbb{R} \rightarrow \mathbb{K}$ a la que únicamente le exigimos ser localmente integrable. El salto a la visión distribucional consiste en ver esta función como un operador $\phi \rightarrow \int f\phi$, que asocia a cada “función test” ϕ de soporte compacto el valor $\int f\phi$. Esto es, que nos permita asomarnos a los valores que toma f en el soporte de ϕ .

Durante las siguientes secciones expondremos los contenidos necesarios para llegar a una definición rigurosa del concepto de distribución y posteriormente presentaremos los resultados principales de esta teoría.

4.1.1. Definición

Definición 4.0.1. Una distribución es un funcional lineal en $\mathcal{D}(\Omega)$ que es continuo con respecto a la topología τ . Denotaremos $\mathcal{D}'(\Omega)$ al espacio de las distribuciones.

Teorema 4.1. *Sea Λ un funcional lineal sobre $\mathcal{D}(\Omega)$. Equivalen:*

1. $\Lambda \in \mathcal{D}'(\Omega)$.
2. *Para cada compacto $K \subset \Omega$, podemos encontrar $N \in \mathbb{N}$, $C < \infty$ tales que $|\langle \Lambda, \phi \rangle| \leq C \|\phi\|_N$, $\forall \phi \in \mathcal{D}_K$.*

4 Teoría de Distribuciones

Observación 4.1.1. Cada $x \in \Omega$ determina una distribución $\delta_x \in \mathcal{D}'(\Omega)$ dada por $\delta_x(\phi) = \phi(x)$.

4.1.2. Funciones vistas como distribuciones

Dada $f : \Omega \rightarrow \mathbb{K}$ localmente integrable, fijemos un compacto $K \subset \Omega$. Definimos

$$\langle \Lambda_f, \phi \rangle = \int_{\Omega} \phi(x) f(x) dx \quad (\phi \in \mathcal{D}(\Omega)).$$

Entonces, se cumple que

$$|\langle \Lambda_f, \phi \rangle| = \left| \int_{\Omega} \phi(x) f(x) dx \right| \leq \left(\int_K |f| \right) \cdot \|\phi\|_0$$

siempre que $\phi \in \mathcal{D}_K$. Como esto es cierto para cada compacto $K \subset \Omega$, por el [Teorema 4.1](#) sabemos que $\Lambda_f \in \mathcal{D}'(\Omega)$. A la distribución Λ_f la llamaremos distribución asociada a f y diremos que este tipo de distribuciones son “funciones”.

De manera similar, si μ es una medida de Borel en Ω con $\mu(K) < \infty$ para cada compacto $K \subset \Omega$, la igualdad

$$\Lambda_{\mu}(\phi) = \int_{\Omega} \phi d\mu \quad (\phi \in \mathcal{D}(\Omega))$$

define una distribución Λ_{μ} a la que llamaremos distribución asociada a μ .

4.2. Cálculo con distribuciones

En esta sección introduciremos algunos de los resultados que definen y demuestran la compatibilidad de las distribuciones con el cálculo diferencial clásico.

4.2.1. Multiplicación por funciones

Definición 4.1.1. Sean $\Lambda \in \mathcal{D}'(\Omega)$ y $f \in C^{\infty}(\Omega)$. Definimos la distribución $f\Lambda \in \mathcal{D}'(\Omega)$ dada por

$$\langle f\Lambda, \phi \rangle = \langle \Lambda, f\phi \rangle$$

para cada $\phi \in \mathcal{D}(\Omega)$.

4.2.2. Derivadas de una distribución

Definición 4.1.2. Para cada multi-índice $\alpha = (\alpha_1, \dots, \alpha_n)$, de aquí en adelante denotaremos:

$$D^\alpha = \left(\frac{\partial}{\partial x_1} \right)^{\alpha_1} \cdots \left(\frac{\partial}{\partial x_n} \right)^{\alpha_n}.$$

Definición 4.1.3. Sea α un multi-índice y sea $\Lambda \in \mathcal{D}'(\Omega)$. Para cada $\phi \in \mathcal{D}(\Omega)$ definimos la distribución derivada α -ésima de Λ mediante

$$\langle D^\alpha \Lambda, \phi \rangle = (-1)^{|\alpha|} \langle \Lambda, D^\alpha \phi \rangle.$$

Definición 4.1.4. Si f es una función localmente integrable en Ω , se define la distribución derivada α -ésima de f mediante $D^\alpha f = D^\alpha \Lambda_f$, esto es,

$$\langle D^\alpha f, \phi \rangle = (-1)^{|\alpha|} \int_{\Omega} f D^\alpha \phi(x) dx \quad (\phi \in \mathcal{D}(\Omega)).$$

Observación 4.1.2. Si la función f tiene derivadas parciales hasta orden k en Ω , la derivada distribucional $D^\alpha f$ coincide con la distribución asociada a la derivada parcial clásica de orden α de f , para cada multi-índice α con $|\alpha| \leq k$. Si $f^{(k)}$ es continua, se llega a la igualdad como consecuencia del Teorema de Fubini y de la fórmula de integración por partes.

Proposición 4.1.1. Para cada multi-índice α y β :

1. $D^\alpha : \mathcal{D}(\Omega) \rightarrow \mathcal{D}(\Omega)$ es un operador lineal y continuo.
2. $D^\alpha D^\beta \Lambda = D^{\alpha+\beta} \Lambda = D^\beta D^\alpha \Lambda, \forall \Lambda \in \mathcal{D}'(\Omega)$.

Demostración.

1. Dada una distribución Λ , para cada compacto K existen $n \in \mathbb{N}$ y $C > 0$ tales que

$$|\langle \Lambda, \phi \rangle| \leq C \|\phi\|_n \quad (\phi \in \mathcal{D}(K)).$$

Entonces, tenemos:

$$|\langle D^\alpha \Lambda, \phi \rangle| = |\langle \Lambda, D^\alpha \phi \rangle| \leq C \|\phi\|_{n+|\alpha|} \quad (\phi \in \mathcal{D}(K))$$

y, por tanto, $D^\alpha \Lambda$ es un funcional continuo sobre todo $\mathcal{D}(K)$ y como consecuencia sobre $\mathcal{D}(\Omega)$.

2.

$$\begin{aligned}\langle D^\alpha D^\beta \Lambda, \phi \rangle &= \langle (-1)^{|\alpha|} D^\beta \Lambda, D^\alpha \phi \rangle = \langle (-1)^{|\alpha|+|\beta|} \Lambda, D^\beta D^\alpha \phi \rangle \\ &= \langle (-1)^{|\alpha|+|\beta|} \Lambda, D^{\alpha+\beta} \phi \rangle = \langle D^{\alpha+\beta} \Lambda, \phi \rangle.\end{aligned}$$

□

Teorema 4.2. Sean $\Lambda \in \mathcal{D}'(\Omega)$ y K un compacto de Ω . Entonces, existen una función continua f definida en Ω y un multi-índice α tales que

$$\langle \Lambda, \phi \rangle = (-1)^{|\alpha|} \int_{\Omega} f(x) (D^\alpha \phi)(x) dx$$

para cada $\phi \in \mathcal{D}(K)$.

Definición 4.2.1. Dada una distribución $\Lambda \in \mathcal{D}'(\Omega)$ definimos su restricción a U como la distribución $\Lambda|_U \in \mathcal{D}'(U)$ definida por

$$\langle \Lambda|_U, \phi \rangle = \langle \Lambda, \phi \rangle \quad (\phi \in \mathcal{D}(U)).$$

Diremos que Λ es cero en U si $\Lambda|_U = 0$.

Definición 4.2.2. Sea $\Lambda \in \mathcal{D}'(\Omega)$ y sea U el mayor subconjunto abierto de Ω donde Λ es cero. Entonces, llamaremos soporte de Λ y denotaremos por $\text{supp}(\Lambda)$ al conjunto $\Omega \setminus U$.

Teorema 4.3. Sea $\Lambda \in \mathcal{D}'(\Omega)$ de orden N y $p \in \Omega$ tal que $\text{supp}(\Lambda) = \{p\}$. Entonces, existen constantes c_α tales que

$$\Lambda = \sum_{|\alpha| \leq N} c_\alpha D^\alpha \delta_p.$$

4.2.3. Convolución

Definición 4.3.1. Sean $u : \mathbb{R}^d \rightarrow \mathbb{K}$, $x \in \mathbb{R}^d$. Se definen las funciones $\tau_x u, \check{u} : \mathbb{R}^d \rightarrow \mathbb{K}$ por

$$\begin{aligned}[\tau_x u](y) &= u(y - x) \\ \check{u}(y) &= u(-y)\end{aligned}$$

para cada $y \in \mathbb{R}^d$.

Definición 4.3.2. Sean $u, v : \mathbb{R}^d \rightarrow \mathbb{K}$. Definimos su convolución como la función $u * v : \mathbb{R}^d \rightarrow \mathbb{K}$ dada

por

$$\begin{aligned}(u * v)(x) &= \int_{\mathbb{R}^d} u(y)v(x-y)dy = \int_{\mathbb{R}^d} u(y)[\tau_x \check{v}](y)dy \\ &= \langle \Lambda_u, \tau_x \check{v} \rangle\end{aligned}$$

para todo $x \in \mathbb{R}^d$, suponiendo que dicha integral existe. Por tanto, para $\Lambda \in \mathcal{D}'(\mathbb{R}^d)$ y $\phi \in \mathcal{D}(\mathbb{R}^d)$ podemos definir la convolución de Λ y ϕ como la distribución dada por

$$[\Lambda * \phi](x) = \langle \Lambda, \tau_x \check{\phi} \rangle$$

para cada $x \in \mathbb{R}^d$.

4.3. Transformada de Fourier. Distribuciones Temperadas

4.3.1. Motivación

Definición 4.3.3. Dada $f \in L_1(\mathbb{R}^d)$, se define su transformada de Fourier como la función $\mathcal{F} : \mathbb{R}^d \rightarrow \mathbb{K}$ dada por

$$[\mathcal{F}(f)](\xi) = (2\pi)^{-\frac{d}{2}} \int_{\mathbb{R}^d} e^{-ix \cdot \xi} f(x) dx, \quad (\xi \in \mathbb{R}^d)$$

donde $x \cdot \xi = x_1 \xi_1 + \dots + x_d \xi_d$. Con esta definición, también es natural ver la transformada de Fourier como un operador que lleva cada función $f \in L_1(\mathbb{R}^d)$ en la función $\mathcal{F}(f)$.

Sabemos, por el lema de Riemann-Lebesgue, que $\mathcal{F}(f)$ pertenece a $C_0(\mathbb{R}^d)$ y es, por tanto, localmente integrable. Así, $\mathcal{F}(f)$ se identifica con una distribución $\Lambda_{\mathcal{F}(f)}$ dada por :

$$\begin{aligned}\langle \Lambda_{\mathcal{F}(f)}, \phi \rangle &= \int_{\mathbb{R}^d} [\mathcal{F}(f)](\xi) \phi(\xi) d\xi \\ &= \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} e^{-ix \cdot \xi} f(x) \phi(\xi) dx d\xi \\ &= \int_{\mathbb{R}^d} [\mathcal{F}(\phi)](x) f(x) dx = \langle \Lambda_f, \mathcal{F}(\phi) \rangle\end{aligned}$$

para cada $\phi \in \mathcal{D}(\mathbb{R}^d)$.

Esto nos puede llevar a querer generalizar la Transformada de Fourier de una distribución Λ de la siguiente manera:

$$\langle \mathcal{F}(\Lambda), \phi \rangle = \langle \Lambda, \mathcal{F}(\phi) \rangle \quad (\phi \in \mathcal{D}(\mathbb{R}^d)).$$

4 Teoría de Distribuciones

Sin embargo, hay dos cuestiones que impiden dicha generalización. Por un lado, no es posible asegurar que $\mathcal{F}(\phi) \in \mathcal{D}(\mathbb{R}^d)$ siempre que $\phi \in \mathcal{D}(\mathbb{R}^d)$. Esto es, $\mathcal{D}(\mathbb{R}^d)$ es demasiado pequeño como espacio de funciones test para extender la transformada de Fourier.

Por otro lado, $C^\infty(\mathbb{R}^d)$ no está contenido en $L_1(\mathbb{R}^d)$, por lo que la transformada de Fourier de una función de $C^\infty(\mathbb{R}^d)$ puede no existir. Esto es, $C^\infty(\mathbb{R}^d)$ es demasiado grande como espacio de funciones test.

4.3.2. Notación

Definición 4.3.4. Definimos la medida normalizada de Lebesgue en \mathbb{R}^d como

$$dm_d(x) = (2\pi)^{-\frac{d}{2}} dx.$$

Definición 4.3.5. Para cada $t \in \mathbb{R}^d$, e_t denotará:

$$e_t(x) = e^{it \cdot x} \quad (x \in \mathbb{R}^d).$$

Es claro que para cada $t \in \mathbb{R}^d$, e_t satisface la propiedad $e_t(x+y) = e_t(x)e_t(y)$. Además, podemos reescribir la definición de transformada de Fourier en términos de e_t :

$$\mathcal{F}(f)(t) = (f * e_t)(0) = \int_{\mathbb{R}^d} f e_{-t} dm_d \quad (t \in \mathbb{R}^d).$$

Definición 4.3.6. Sea α un multi-índice. De aquí en adelante vamos a llamar D_α al operador

$$D_\alpha = i^{-|\alpha|} D^\alpha = \left(\frac{1}{i} \frac{\partial}{\partial x_1} \right)^{\alpha_1} \cdots \left(\frac{1}{i} \frac{\partial}{\partial x_d} \right)^{\alpha_d}.$$

Sea P un polinomio de d variables y coeficientes complejos c_α . Definimos los operadores diferenciales

$$P(D) = \sum_{\alpha} c_\alpha D_\alpha \quad \text{y} \quad P(-D) = \sum_{\alpha} (-1)^{|\alpha|} c_\alpha D_\alpha.$$

4.3.3. Funciones de decrecimiento rápido

Definición 4.3.7 (Funciones de decrecimiento rápido). Sea $\varphi \in C^\infty(\mathbb{R}^d)$ cumpliendo que

$$\sup_{|\alpha| \leq N} \sup_{x \in \mathbb{R}^d} \left\{ (1 + |x|^2)^N |(D^\alpha \varphi)(x)| \right\} < \infty \quad (\forall N \in \mathbb{N}).$$

4.3 Transformada de Fourier. Distribuciones Temperadas

Esto es, $PD^\alpha \varphi$ es una función acotada en \mathbb{R}^d , para todo polinomio P . Entonces, decimos que φ es una función de decrecimiento rápido en infinito. Estas funciones forman un espacio vectorial, denotado por $\mathcal{S}(\mathbb{R}^d)$, al que llamaremos la clase de Schwartz en \mathbb{R}^d . En $\mathcal{S}(\mathbb{R}^d)$ consideraremos la topología localmente convexa y metrizable asociada a la familia de seminormas

$$s_{k,N}(\varphi) = \sup_{|\alpha| \leq N} \sup_{x \in \mathbb{R}^d} \left\{ (1 + |x|^2)^N |(D^\alpha \varphi)(x)| \right\} < \infty \quad (\varphi \in \mathcal{S}(\mathbb{R}^d) \quad k, N \in \mathbb{N}).$$

Observación 4.3.1. $\mathcal{D}(\mathbb{R}^d) \subset \mathcal{S}(\mathbb{R}^d) \subset L_1(\mathbb{R}^d)$ y $\mathcal{S}(\mathbb{R}^d)$ es denso en $L_1(\mathbb{R}^d)$ (pues lo es $\mathcal{D}(\mathbb{R}^d)$).

Proposición 4.3.1. $\mathcal{D}(\mathbb{R}^d)$ es denso en $\mathcal{S}(\mathbb{R}^d)$ y la inclusión $I : \mathcal{D}(\mathbb{R}^d) \rightarrow \mathcal{S}(\mathbb{R}^d)$ es continua.

Teorema 4.4.

1. $\mathcal{S}(\mathbb{R}^d)$ es un espacio de Fréchet.
2. Sean P un polinomio, $\varphi, \psi \in \mathcal{S}(\mathbb{R}^d)$ y α un multi-índice. Entonces las aplicaciones

$$\varphi \rightarrow P\varphi, \quad \varphi \rightarrow \psi\varphi, \quad \varphi \rightarrow D^\alpha \varphi \quad (\varphi \in \mathcal{S}(\mathbb{R}^d))$$

son aplicaciones continuas de $\mathcal{S}(\mathbb{R}^d)$ en $\mathcal{S}(\mathbb{R}^d)$.

3. Sea $\varphi \in \mathcal{S}(\mathbb{R}^d)$ y sea α un multi-índice. Entonces, $\mathcal{F}(D^\alpha \varphi) = i^{|\alpha|} \xi^\alpha \mathcal{F}(\varphi)$ y $\mathcal{F}(x^\alpha \varphi) = i^{|\alpha|} D^\alpha \mathcal{F}(\varphi)$.
4. Sean $\varphi \in \mathcal{S}(\mathbb{R}^d)$ y P un polinomio. Entonces $\mathcal{F}(P(D)\varphi) = P\mathcal{F}(\varphi)$ y $\mathcal{F}(P\varphi) = P(-D)\mathcal{F}(\varphi)$.
5. La transformada de Fourier es un operador lineal y continuo de $\mathcal{S}(\mathbb{R}^d)$ en sí mismo.

Demostración.

1. Sea $\{\varphi_i\}$ una sucesión de Cauchy en $\mathcal{S}(\mathbb{R}^d)$. Por ser las funciones φ_i de decrecimiento rápido para todo $i \in \mathbb{N}$, para cada par de multi-índices α y β , $\{x^\beta D^\alpha \varphi_i(x)\} \rightarrow g_{\alpha\beta}(x)$ uniformemente, donde $g_{\alpha\beta} = x^\beta D^\alpha g$ es una función acotada. Por tanto, $\{\varphi_i\} \rightarrow g \in \mathcal{S}(\mathbb{R}^d)$, lo que demuestra que $\mathcal{S}(\mathbb{R}^d)$ es completo.
2. Si $\varphi \in \mathcal{S}(\mathbb{R}^d)$ sabemos que $D^\alpha \varphi \in \mathcal{S}(\mathbb{R}^d)$. En el caso de $\varphi\psi$, sabemos que $D^\alpha(\varphi\psi) = (D^\alpha \varphi)\psi + \varphi D^\alpha \psi \in \mathcal{S}(\mathbb{R}^d)$. Análogamente, $D^\alpha(P\varphi) = (D^\alpha P)\varphi + PD^\alpha \varphi$, siendo ambos sumandos de crecimiento lento por serlo φ . Hemos demostrado que las aplicaciones anteriores están bien definidas. Para demostrar su continuidad, basta con utilizar el **Teorema de la gráfica cerrada**.
3. Fijemos un multiíndice $\alpha_k = (0, \dots, 0, 1, 0, \dots, 0)$ y una función $\varphi \in \mathcal{S}(\mathbb{R}^d)$. Entonces, tenemos que

$$[\mathcal{F}(D^{\alpha_k} \varphi)](\xi) = \int_{\mathbb{R}^d} e^{-ix \cdot \xi} D^{\alpha_k} \varphi(x) dm_d(x).$$

4 Teoría de Distribuciones

Integrando por partes, esto equivale a

$$i\xi_k \int_{\mathbb{R}^d} e^{-ix \cdot \xi} \varphi(x) dm_d(x) = i\xi_k [\mathcal{F}(\varphi)](\xi).$$

Por otro lado,

$$\begin{aligned} D^{\alpha_k} \mathcal{F}(\varphi)(\xi) &= D^{\alpha_k} \left[\int_{\mathbb{R}^d} e^{-ix \cdot \xi} \varphi(x) dm_d(x) \right] \\ &= \int_{\mathbb{R}^d} D^{\alpha_k} e^{-ix \cdot \xi} \varphi(x) dm_d(x) \\ &= (-i) \int_{\mathbb{R}^d} e^{-ix \cdot \xi} x^{\alpha_k} \varphi(x) dm_d(x) \\ &= -i \mathcal{F}(x^{\alpha_k} \varphi). \end{aligned}$$

Como todo multi-índice α puede expresarse como suma de multi-índices α_k , hemos demostrado el enunciado.

4. Si $\varphi \in \mathcal{S}(\mathbb{R}^d)$, por 2. sabemos que $P(D)\varphi \in \mathcal{S}(\mathbb{R}^d)$ y, además, se cumple

$$\begin{aligned} [(P(D)\varphi) * e_t](y) &= \int_{\mathbb{R}^d} (P(D)\varphi)(x) e_t(y-x) dm_d = \int_{\mathbb{R}^d} \left(\sum c_\alpha(i)^{-|\alpha|} D^\alpha \varphi(x) \right) e^{it \cdot (y-x)} dm_d \\ &= \int_{\mathbb{R}^d} \varphi(x) \left(\sum c_\alpha(i)^{-|\alpha|} D^\alpha e_t(y-x) \right) dm_d = \int_{\mathbb{R}^d} \varphi(x) P(t) e_t(y-x) dm_d \\ &= P(t)[\varphi * e_t](y). \end{aligned}$$

Evaluando en el origen, concluimos que $\mathcal{F}(P(D)\varphi) = P\mathcal{F}(\varphi)$. Para demostrar la segunda igualdad, tomemos $t = (t_1, \dots, t_d)$ y $t' = (t_1 + \varepsilon, \dots, t_d)$, con $\varepsilon \neq 0$. Entonces, fijando $\varphi \in \mathcal{S}(\mathbb{R}^d)$, tenemos:

$$\frac{\mathcal{F}(\varphi)(t) - \mathcal{F}(\varphi)(t')}{i\varepsilon} = \int_{\mathbb{R}^d} x_1 \varphi(x) \frac{e^{-ix_1\varepsilon} - 1}{ix_1\varepsilon} e^{-ix \cdot t} dm_d.$$

Como $x_1 \varphi \in \mathcal{S}(\mathbb{R}^d) \subset L_1(\mathbb{R}^d)$, podemos aplicar el **Teorema de convergencia dominada**, obteniendo así:

$$\int_{\mathbb{R}^d} x_1 \varphi(x) e^{-ix \cdot t} dm_d = -\frac{1}{i} \frac{\partial}{\partial t_1} \mathcal{F}(\varphi).$$

Con esto, hemos probado la igualdad para el caso en el que $P = x_1$. Iterando este proceso, se llega a la igualdad para cualquier polinomio P .

4.3 Transformada de Fourier. Distribuciones Temperadas

5. Sean $\varphi \in \mathcal{S}(\mathbb{R}^d)$, α un multi-índice y $g(x) = (-1)^{|\alpha|} x^\alpha \varphi(x)$. Entonces es claro que $g \in \mathcal{S}(\mathbb{R}^d)$. Además, por 4. sabemos que $\mathcal{F}(g) = D_\alpha \mathcal{F}(\varphi)$ y que $P D_\alpha \mathcal{F}(\varphi) = P \mathcal{F}(g) = \mathcal{F}(P(D)g)$. Como la transformada de Fourier está definida en $L^1(\mathbb{R}^d)$, $(P(D)g)$ vive en este espacio y su transformada de Fourier es, por tanto, una función acotada. Como $P D_\alpha \mathcal{F}(\varphi)$ está acotada para cualesquiera α y P , concluimos que $\mathcal{F}(\varphi) \in \mathcal{S}(\mathbb{R}^d)$. Esto es, la transformada de Fourier está bien definida en $\mathcal{S}(\mathbb{R}^d)$. Para probar su continuidad, bastará con aplicar el **Teorema de la gráfica cerrada**.

□

Teorema 4.5 (Teorema de inversión). *La transformada de Fourier es un isomorfismo de $\mathcal{S}(\mathbb{R}^d)$ en sí mismo. Si $\varphi \in \mathcal{S}(\mathbb{R}^d)$, entonces*

$$\varphi(x) = (2\pi)^{-d/2} \int_{\mathbb{R}^d} e^{ix \cdot \xi} [\mathcal{F}(\varphi)](\xi) d\xi \quad (x \in \mathbb{R}^d)$$

esto es, $\mathcal{F}^{-1} \circ \mathcal{F} = Id_{\mathcal{S}(\mathbb{R}^d)}$. Además, este isomorfismo cumple que $\mathcal{F}^4(\varphi) = \varphi$.

Teorema 4.6. *Sean $\varphi, \psi \in \mathcal{S}(\mathbb{R}^d)$. Entonces, se cumple:*

1. $\varphi * \psi \in \mathcal{S}(\mathbb{R}^d)$.
2. $\mathcal{F}(\varphi * \psi) = \mathcal{F}(\varphi) \cdot \mathcal{F}(\psi)$.

4.3.4. Distribuciones Temperadas

Definición 4.6.1. Una distribución temperada o función generalizada de crecimiento lento es un funcional lineal y continuo en $\mathcal{S}(\mathbb{R}^d)$. Se denota por $\mathcal{S}'(\mathbb{R}^d)$ al espacio de las distribuciones temperadas, esto es, al dual de $\mathcal{S}(\mathbb{R}^d)$. La topología que consideramos en $\mathcal{S}'(\mathbb{R}^d)$ es la débil-*, esto es, $\sigma(\mathcal{S}'(\mathbb{R}^d), \mathcal{S}(\mathbb{R}^d))$.

En el **Teorema 4.4** veímos cómo la transformada de Fourier está bien definida en el espacio $\mathcal{S}(\mathbb{R}^d)$. Hemos encontrado, por tanto, el espacio de “funciones test” de tamaño adecuado que nos permite, siguiendo el espíritu natural de las distribuciones, definir la transformada de Fourier en $\mathcal{S}'(\mathbb{R}^d)$.

Definición 4.6.2. Para cada $\Lambda \in \mathcal{S}'(\mathbb{R}^d)$ definimos su transformada de Fourier:

$$\langle \mathcal{F}(\Lambda), \varphi \rangle = \langle \Lambda, \mathcal{F}(\varphi) \rangle.$$

Como la transformada de Fourier es un isomorfismo de $\mathcal{S}(\mathbb{R}^d)$ en sí mismo, $\mathcal{F}(\Lambda)$ está bien definida y es una distribución temperada.

Teorema 4.7. *Sean $\Lambda \in \mathcal{S}'(\mathbb{R}^d)$ y P un polinomio. Entonces $\mathcal{F}(P(D)\Lambda) = P\mathcal{F}(\Lambda)$ y $\mathcal{F}(P\Lambda) = P(-D)\mathcal{F}(\Lambda)$ para los operadores $P(D)$ y $P(-D)$ definidos en términos de D_α .*

Demostración.

$$\begin{aligned}\langle \mathcal{F}(P(D)\Lambda), \varphi \rangle &= \langle P(D)\Lambda, \mathcal{F}(\varphi) \rangle = \langle \Lambda, P(-D)\mathcal{F}(\varphi) \rangle \\ &= \langle \Lambda, \mathcal{F}(P\varphi) \rangle = \langle \mathcal{F}(\Lambda), P\varphi \rangle \\ &= \langle P\mathcal{F}(\Lambda), \varphi \rangle.\end{aligned}$$

$$\begin{aligned}\langle \mathcal{F}(P\Lambda), \varphi \rangle &= \langle P\Lambda, \mathcal{F}(\varphi) \rangle = \langle \Lambda, P\mathcal{F}(\varphi) \rangle \\ &= \langle \Lambda, \mathcal{F}(P(D)\varphi) \rangle = \langle \mathcal{F}(\Lambda), P(D)\varphi \rangle \\ &= \langle P(-D)\mathcal{F}(\Lambda), \varphi \rangle.\end{aligned}$$

□

Lema 4.7.1. Si una distribución tiene como transformada de Fourier una combinación lineal de deltas de Dirac y sus derivadas, esa distribución vista como función es un polinomio.

Demostración. Sea Λ una distribución tal que $\mathcal{F}(\Lambda) = \sum_{k=0}^N c_k D^k \delta$. Ya sabemos que los polinomios son distribuciones temperadas. Podemos por tanto calcular sus transformadas de Fourier.

- Supongamos que $N = 0$. Entonces $\mathcal{F}(\Lambda) = c_0 \delta$. Utilizando el **Teorema de inversión**, deducimos que

$$\begin{aligned}\langle \mathcal{F}(\Lambda), \varphi \rangle &= \int_{\mathbb{R}^d} c_0 \delta(\varphi) = c_0 \varphi(0) = \int_{\mathbb{R}^d} c_0 (2\pi)^{-\frac{d}{2}} e^{i0 \cdot \xi} [\mathcal{F}(\varphi)](\xi) d\xi \\ &= \int_{\mathbb{R}^d} P_0 (2\pi)^{-\frac{d}{2}} e^{i0 \cdot \xi} [\mathcal{F}(\varphi)](\xi) d\xi = \langle \Lambda_{P_0}, \mathcal{F}(\varphi) \rangle\end{aligned}$$

para cualquier función $\varphi \in S(\mathbb{R}^d)$. $P_0(x) = c_0$ es, por tanto, el polinomio que buscamos. En particular, tenemos que $\mathcal{F}(\Lambda_1) = \delta$. Análogamente, es fácil comprobar que

$$\langle \mathcal{F}(\delta), \varphi \rangle = \langle \delta, \mathcal{F}(\varphi) \rangle = \mathcal{F}(\varphi)(0) = \int_{\mathbb{R}^d} \varphi dm_d = \langle \Lambda_1, \varphi \rangle.$$

- Supongamos ahora $N \geq 1$. Por **Teorema 4.7** sabemos que

$$\mathcal{F}(P(D)\delta) = P\mathcal{F}(\delta) = P.$$

□

5 Aprendizaje profundo. Fundamentos

5.1. Descripción de un modelo formal de Aprendizaje Automático

El Aprendizaje Automático comprende una serie de prácticas y algoritmos que comparten como objetivo común la tarea de aprender información a partir de unos datos de entrada. La descripción formal para un modelo de Aprendizaje Automático suele tener los siguientes elementos:

- Dominio del problema: se trata de un conjunto X que contendrá todos los objetos que tenemos a nuestra disposición para realizar la tarea de aprendizaje.
- Conjunto de entrenamiento: Un conjunto finito de datos $S \subset X$, donde X es el espacio del dominio del problema. En caso de tratarse de un problema de aprendizaje supervisado, $S = \{(x_1, y_1), \dots, (x_n, y_n)\} \subset X \times Y$, donde X e Y son respectivamente el espacio del dominio del problema y su conjunto de etiquetas asociado.
- Una hipótesis de aprendizaje que el algoritmo A construye tras haber recibido el conjunto de entrenamiento S . La denotaremos por $A(S)$.
- Un predictor $h : X \rightarrow Y$ devuelto por el algoritmo tras la tarea de aprendizaje que describe la hipótesis $A(S)$.
- Una medida de bondad de la predicción realizada por el algoritmo. Esta medida tiene el nombre de función de coste y está asociada a la probabilidad de que, dado un elemento arbitrario $x \in X$, el algoritmo cometa un error en la predicción.

Este modelo tan general se ha concretado en varios modelos de aprendizaje. En este trabajo nos centraremos en describir las principales características del Aprendizaje Profundo, cuyo objeto de interés principal son las redes neuronales.

5.2. Redes Neuronales Artificiales

Una Red Neuronal Artificial (RNA) es un modelo de Aprendizaje Automático basado en la idea de que muchas unidades de cómputo pequeñas pueden unirse y comunicarse para construir modelos complejos. Para ello, imitan estructura y el comportamiento de las neuronas biológicas. Esta idea surge en 1943 [MP43] de la mano del neurofisiólogo Warren McCulloch y el matemático Walter Pitts, quienes diseñaron el primer modelo de neurona artificial. En las décadas posteriores a su descubrimiento se

fueron explorando nuevas arquitecturas, como el perceptrón multicapa. Sin embargo, en este momento su capacidad de aprendizaje era muy limitada. En los años 80, la popularización del algoritmo de retropropagación dio nuevas esperanzas a las redes neuronales, pero sus altos requerimientos computacionales, como el uso de GPU, hicieron que el interés general fuese a otras técnicas de aprendizaje automático que habían surgido en la época, como SVM. Sin embargo, en los últimos años las redes neuronales profundas han demostrado ser capaces de abordar tareas complicadas que otras técnicas no son capaces de abordar, y se han convertido en el objeto de muchas tareas de investigación.

5.2.1. Tipos de redes neuronales

Una de las grandes ventajas de las redes neuronales es su flexibilidad para aprender en tareas con objetivos muy dispares. Es por esto que a partir de los años 90, aparecen distintas arquitecturas que facilitan tareas de aprendizaje con distintos objetivos. Entre ellas destacan:

- Redes neuronales prealimentadas (MLP): Están compuestas por una capa de entrada, una capa de salida y, entre ellas, una o varias capas llamadas capas ocultas. Cada neurona está conectada a todas las neuronas de su capa siguiente y, análogamente, cada neurona de la capa de salida está conectada a todas las neuronas de la capa anterior. Son utilizadas para tareas de clasificación, regresión y aproximación de funciones.
- Redes neuronales convolucionales (CNN): Compuestas por capas de convolución en las que se aplican filtros para extraer características locales de los datos que la capa siguiente toma como entrada. Se utilizan principalmente para reconocimiento de imágenes.
- Redes neuronales recurrentes (RNN): Las conexiones entre neuronas forman ciclos, lo que les permite mantener una memoria interna de las secuencias de datos que entran a la red. Pueden utilizarse para procesamiento del lenguaje natural y para predicción de secuencias.

5.3. Redes Neuronales Prealimentadas

En [SSBD14] se describe una red neuronal prealimentada como un grafo dirigido acíclico $G = (V, E)$ organizado por capas, $V = \{V_0, V_1, \dots, V_n\}$, con una función de pesos $w : E \rightarrow \mathbb{R}$ asociada. Como puede verse en la Figura 5.1, los vértices representan neuronas artificiales (unidades de cómputo simple) y las aristas representan las conexiones entre ellas. Supongamos que la capa i tiene n neuronas. Entonces, denotaremos $V_i = \{v_{i,1}, \dots, v_{i,n}\}$ a las neuronas en esa capa. Si tenemos una red neuronal con $T + 1$ capas, V_0, \dots, V_T , llamamos a V_0 capa de entrada, a V_T capa de salida y a las capas $\{V_1, \dots, V_{T-1}\}$ capas ocultas. Una red neuronal profunda es aquella que tiene un número considerable de capas ocultas.

Aunque hayamos considerado todas las neuronas como unidades de cómputo simple, es importante remarcar que en la capa de entrada V_0 no se realiza ningún cálculo: únicamente se procesan los datos de entrada. Esta capa tiene tantas neuronas como dimensiones tenga un dato de entrada además de una neurona extra, que recibe una entrada constante y que sirve como término de sesgo.

Para el resto de capas V_1, \dots, V_T , una neurona sí conforma una unidad de cómputo. Así, en cada neurona $v_{i,j}$ de la capa i , con $i = 1, \dots, T$, se realiza una suma ponderada de las entradas que le llegan de la capa anterior, esto es, se realiza la operación $z = w^T x + b$ donde el vector w viene dado por la función de pesos asociada a la red neuronal, x representa las salidas de las neuronas de la capa $i - 1$ conectadas a $v_{i,j}$ y b es, de nuevo, un término de sesgo. A esta suma ponderada se aplica una función real, a la que llamaremos función de activación.

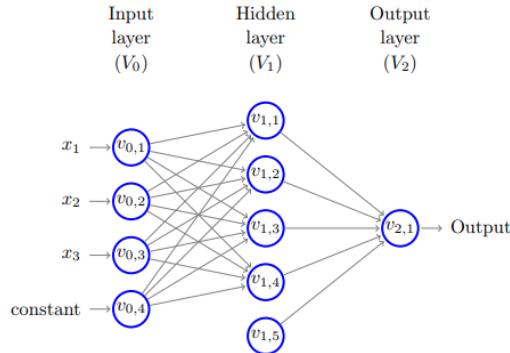


Figura 5.1: Arquitectura de Red Neuronal presentada en [SSBD14].

Las redes neuronales profundas han supuesto un antes y un después en el Aprendizaje Automático ya que su arquitectura les permite aprender patrones complejos y abstractos: las capas inferiores (las más próximas a la capa de entrada) aprenden relaciones simples entre los datos, las capas siguientes, partiendo de esas relaciones simples, consiguen encontrar relaciones más abstractas sobre ellas, dando lugar a un aprendizaje más complejo y general conforme vamos llegando a las capas superiores (las capas cercanas a la de salida).

Con esta estructura, una red neuronal prealimentada define una aplicación $f(x, w)$ y aprende los valores de w que llevan a la mejor aproximación de y . En una red prealimentada, la información únicamente fluye hacia delante antes de ser evaluada, en contraposición a lo que ocurre con las redes neuronales recurrentes mencionadas en el apartado anterior.

5.3.1. Funciones de Activación

Una de las cuestiones más importantes a la hora de definir una red neuronal es la elección de una función de activación adecuada para cada capa de neuronas. De hecho, la elección de funciones de activación demasiado simples fue, históricamente, una de las principales razones por las que las redes neuronales fueron consideradas poco capaces en sus inicios. La correcta elección de funciones de activación depende principalmente de tres factores:

5 Aprendizaje profundo. Fundamentos

- Objetivo de la tarea de aprendizaje: en la capa de salida, es posible que necesitemos funciones de activación que describan una distribución de probabilidad (clasificación binaria), probabilidad conjunta (clasificación multiclase) o que respondan a otro tipo de comportamiento, lineal o no (regresión).
- Complejidad de la tarea de aprendizaje: para tareas complejas, algunas funciones de activación pueden no ser suficientes.
- Compatibilidad de la función de activación con el algoritmo de entrenamiento de la red neuronal: los algoritmos que definen y optimizan el entrenamiento de una red neuronal, además de los que regularizan el modelo final, pueden comportarse mejor si las funciones de activación cumplen unos requisitos de derivabilidad y preservación de la varianza. Entraremos en estos detalles más adelante.

En el contexto del Aprendizaje Profundo, algunas de las funciones de activación más utilizadas son:

- Sigmoidal: Es una de las funciones de activación más utilizadas. Sin embargo, se ha demostrado que no preserva bien la varianza de los datos de entrada, lo que puede dar problemas de gradientes inestables durante el entrenamiento.

$$\sigma(z) = \frac{1}{1 + e^{-(z)}}$$

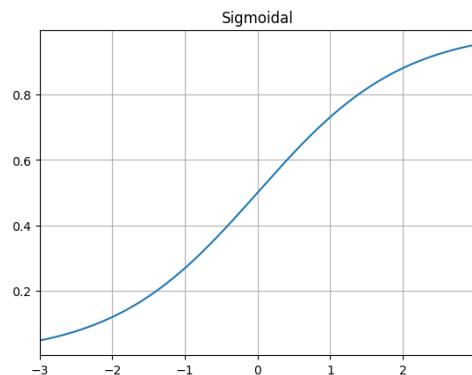


Figura 5.2: Función de activación sigmoidal.

- Tangente hiperbólica: Como se puede ver en la Figura 5.3, tiene una media cercana a 0, lo que ayuda a preservar la varianza más que la función sigmoidal.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

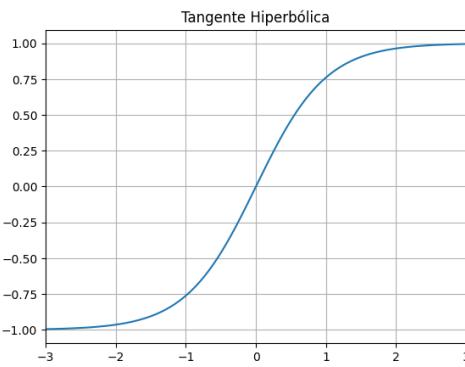


Figura 5.3: Función de activación tangente hiperbólica.

- Lineal rectificada (ReLU): Entre sus ventajas destacan su simplicidad de cómputo. Suele funcionar muy bien en redes neuronales profundas.

$$\text{ReLU}(x) = \max(0, x)$$

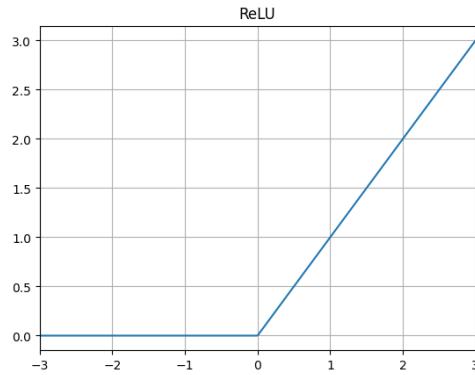


Figura 5.4: Función de activación ReLU.

- Leaky ReLU: Conocida como ReLU con fugas, garantiza un gradiente positivo para todo $z < 0$ mediante el hiperparámetro de grado de fuga α . Esto garantiza que para salidas $z < 0$ las neuronas nunca mueran.
- ELU y SELU: Toma valores negativos, lo que permite que sus valores medios de salida sean más cercanos a 0. Además es suave para $\alpha = 1$, lo que ayuda a acelerar el descenso de gradiente para valores cercanos a 0. SELU [KUMH17] es una versión escalada de ELU con $\alpha = 1.67$ que ayuda a

5 Aprendizaje profundo. Fundamentos

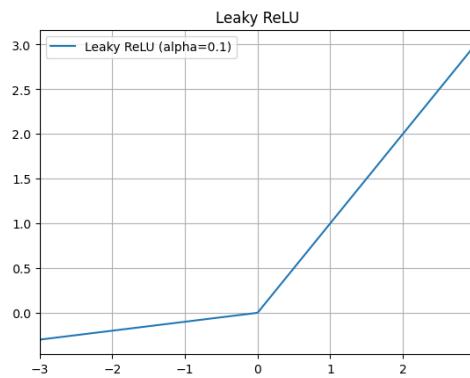


Figura 5.5: Función de activación Leaky ReLU.

la autonormalización de redes neuronales densas.

$$ELU_\alpha(z) = \begin{cases} \alpha(e^z - 1) & \text{si } z < 0 \\ z & \text{si } z \geq 0 \end{cases}$$

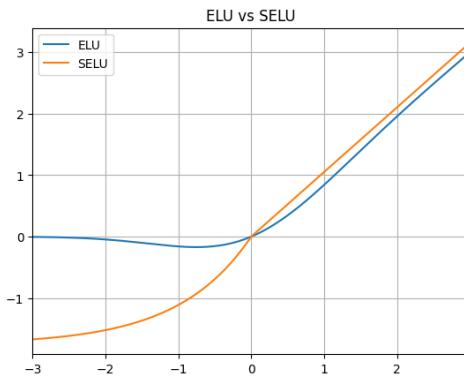


Figura 5.6: Funciones de activación ELU y SELU.

- GELU: Presentada en [HG16], es parecida a las funciones ReLU y ELU pero, a diferencia de ellas, no es convexa ni monótona. Su buen funcionamiento se atribuye a tener una curvatura distinta en cada punto.

5.3.2. Función de coste

Aunque es posible orientar el aprendizaje de una red neuronal hacia varios propósitos, en este trabajo vamos a centrarnos en el uso de redes neuronales para aprender funciones, funcionales y operadores.

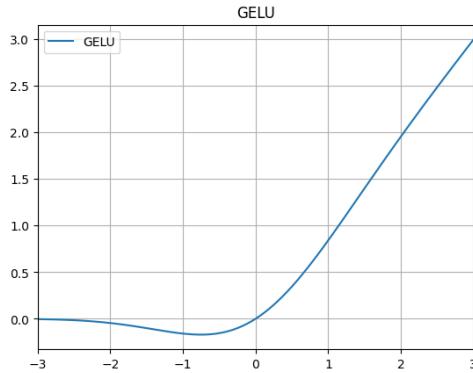


Figura 5.7: Función de activación GELU.

Para esta tarea, los elementos formales descritos en la [Sección 5.1](#) se asemejan a los de un problema de regresión.

El objetivo del aprendizaje será, por tanto, encontrar el conjunto de parámetros que minimiza una métrica de error (generalmente, el error cuadrático medio) con respecto a una función, funcional u operador. Recordemos que los parámetros asociados a una red neuronal son el conjunto de pesos $\theta = \{w(E)\}$. La función de coste asociada a θ , $J(\theta)$, estará relacionada con la métrica de error.

5.3.3. Entrenamiento

5.3.3.1. Optimización

Como buscamos minimizar $J(\theta)$ en función de θ , no es de extrañar que el algoritmo de descenso de gradiente forme parte del proceso de entrenamiento en redes neuronales prealimentadas. Este algoritmo de optimización genérico es capaz de encontrar soluciones óptimas a una amplia gama de problemas. La idea general del descenso de gradiente es ajustar los parámetros de forma iterativa para minimizar la función de coste. Para ello, cada vez que un lote de información se propaga hasta la capa de salida, se realiza un paso de descenso de gradiente. Esto es, sobre el conjunto de pesos θ , se computa

$$\theta^{siguiente} = \theta - \eta \nabla_{\theta} J(\theta)$$

donde η es la tasa de aprendizaje, parámetro que indica cómo de brusco será el paso de descenso de gradiente.

En el contexto de entrenamiento de redes neuronales para la predicción de funciones complejas, el descenso de gradiente tiene varios problemas como su elevado coste computacional, su sensibilidad a cambios bruscos en la pendiente de $J(\theta)$ y su riesgo de caer en mínimos locales sin alcanzar el mínimo global del problema. Esto hace que para este tipo de problemas se escojan versiones alternativas del

algoritmo que favorecen la tarea de aprendizaje.

5.3.3.2. Propagación hacia atrás

Aunque obtener la expresión analítica de un paso de descenso de gradiente es, como hemos visto, sencillo, evaluar numéricamente esa expresión puede llegar a ser muy costoso. En 1970, el investigador Seppo Linnainmaa introdujo en su tesis de máster [Lin70] una técnica para calcular los gradientes de forma automática y eficiente mediante la regla de la cadena. Este algoritmo se denomina ahora diferenciación automática en modo inverso. El algoritmo de propagación hacia atrás (back propagation), que combina esta técnica con el descenso de gradiente, mitiga los problemas de coste de este último y es el algoritmo estándar utilizado para ajustar θ tras el cálculo del error (es decir, en el paso hacia atrás).

5.3.3.3. Tasa de Aprendizaje

Uno de los grandes retos del correcto entrenamiento de una red neuronal es encontrar el valor adecuado para la tasa de aprendizaje. Como vemos en la Figura 5.8, cuando usamos una tasa demasiado grande, los pasos de descenso de gradiente escapan el mínimo local, haciendo que el algoritmo diverja.

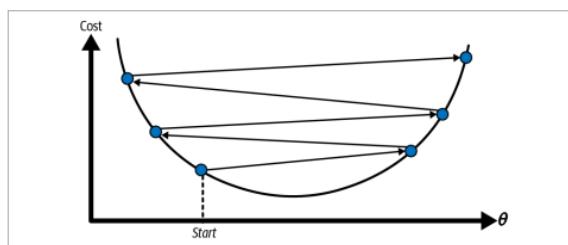


Figura 5.8: Pasos de descenso de gradiente para una tasa de aprendizaje demasiado grande. Imagen extraída de [BIKP20].

Por otro lado, si la tasa de aprendizaje es demasiado pequeña, como en la Figura 5.9, necesitaremos demasiadas etapas de entrenamiento para llegar al mínimo local. Es posible que el entrenamiento se detenga antes de alcanzar ese mínimo.

El valor óptimo para la tasa de aprendizaje es aquel que minimiza el número de pasos de descenso de gradiente a realizar antes de alcanzar el mínimo local de $J(\theta)$. Para este valor, los pasos de descenso de gradiente se comportan como en la Figura 5.10.

En la Subsubsección 5.3.3.1 comentábamos que el algoritmo de descenso de gradiente es sensible a cambios bruscos de pendiente. Esto se debe a que no todas las funciones de coste son convexas como en las gráficas anteriores. En muchas ocasiones la función de coste presenta tramos con pendientes muy distintas, lo que hace que, como se puede observar en la Figura 5.11, una tasa de aprendizaje que funcione bien en un tramo, pueda no ser adecuada en otro.

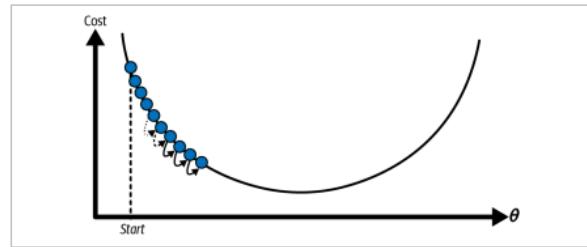


Figura 5.9: Pasos de descenso de gradiente para una tasa de aprendizaje demasiado pequeña. Imagen extraída de [BIKP20].

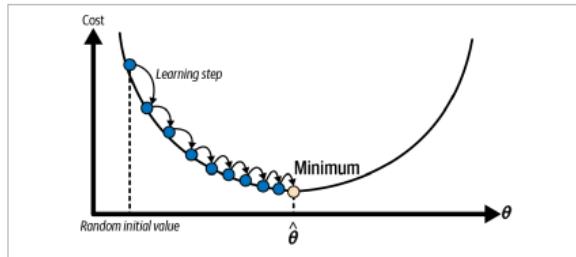


Figura 5.10: Pasos de descenso de gradiente para una buena tasa de aprendizaje. Imagen extraída de [BIKP20].

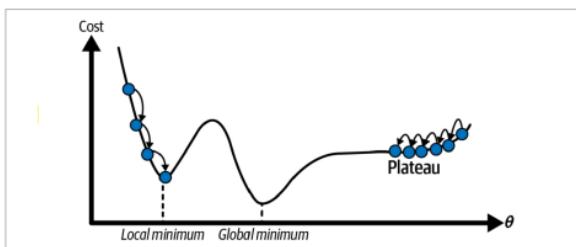


Figura 5.11: Tasa de aprendizaje constante para una función de coste con cambios bruscos en la pendiente. Imagen extraída de [BIKP20].

Es por esto que, especialmente en tareas complejas, se opta por optimizadores que adaptan la tasa de aprendizaje a la etapa de entrenamiento o por programar el ritmo de aprendizaje. Entre las programaciones más utilizadas destacan:

- Power Scheduling: Modifica la tasa de aprendizaje en función del número de iteración t . El comportamiento viene descrito por: $\eta(t) = \frac{\eta_0}{(1+\frac{t}{s})^c}$, donde η_0 es la tasa de aprendizaje inicial, s el paso de decay y c suele fijarse a 1. Esta técnica requiere afinar los hiperparámetros mencionados.
- Performance Scheduling: Reduce la tasa de aprendizaje en un factor λ cuando el error de validación deja de decaer.
- Programación exponencial: Reduce la tasa de aprendizaje en un factor de 10 cada s pasos. Este comportamiento viene dado por $\eta(t) = \eta_0 0.1^{\frac{t}{s}}$.
- Programación constante a trozos: Consiste en asociar valores de tasa de aprendizaje a un número de épocas determinado. Esta aproximación es la más sensible a ajuste de parámetros.

5.3.3.4. Optimizadores más rápidos

Además de una buena elección de funciones de activación y de un valor adecuado para la tasa de aprendizaje, otro aspecto clave para la eficiencia en el aprendizaje viene dado por el algoritmo de optimización escogido para la función de coste. En la Subsección 5.3.3.1 presentábamos cómo se usa el algoritmo de descenso de gradiente en redes neuronales prealimentadas. Como pudimos ver, este algoritmo realiza pasos constantes y actualiza el gradiente de la función de coste atendiendo únicamente al gradiente local, esto es, sin considerar los valores de gradiente anteriores. Esto se traduce en que no es capaz de coger velocidad cuando nos encontramos cerca del mínimo, lo que ralentiza el proceso de aprendizaje.

La primera respuesta a esta problemática surgió en 1964 de la mano de Boris Polyak, quien propuso la optimización basada en el momentum, la cual introduciremos a continuación. En esta sección introduciremos algunas variantes del descenso de gradiente inspiradas en esta idea.

- Momentum: tiene en cuenta el gradiente local, pero lo utiliza como aceleración, no como velocidad. Para ello, el vector de gradientes se actualiza de forma:

$$1. m \leftarrow \beta m - \eta \nabla_{\theta} J(\theta)$$

$$2. \theta \leftarrow \theta + m$$

donde m es el vector de momentos y el parámetro β , denominado *momentum*, simula una fricción que evita que m crezca demasiado. Suele escogerse $\beta = 0.9$.

- Gradiente acelerado de Nesterov: Es una variante de Momentum introducida por Yurii Nesterov en 1983. Propone evaluar la función de coste en $\theta + \beta m$ en vez de en θ . Este cambio se debe a que, en general, el vector m toma direcciones que apuntan más al óptimo que θ . Este empuje en

la dirección correcta se implementa como sigue:

$$1. \ m \leftarrow \beta m - \eta \nabla_{\theta} J(\theta + \beta m)$$

$$2. \ \theta \leftarrow \theta + m.$$

- AdaGrad: Corrige la tendencia natural del descenso de gradiente por la pendiente más pronunciada, que no tiene por qué apuntar al óptimo global. Esto se logra escalando el vector de gradiente. De este modo, el algoritmo hace decaer la tasa de aprendizaje más rápido para direcciones empinadas y más despacio en direcciones de gradiente más suave. Este algoritmo funciona bien para problemas sencillos, pero funciona mal en modelos complejos de aprendizaje profundo, donde la tasa de aprendizaje puede reducirse tanto que no se llegue a alcanzar el óptimo global. Para ello, en cada etapa se calcula:

$$1. \ s \leftarrow s + \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta)$$

$$2. \ \theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta) \oslash \sqrt{s + \epsilon}$$

donde \otimes representa el producto elemento a elemento, por lo que s acumula el cuadrado de los gradientes.

- RMSProp: Soluciona el riesgo que corre Adagrad de acabar el entrenamiento demasiado pronto acumulando los gradientes de las iteraciones más recientes. Así, obtenemos:

$$1. \ s \leftarrow \rho s + (1 - \rho) \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta)$$

$$2. \ \theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta) \oslash \sqrt{s + \epsilon}$$

donde el parámetro ρ , que suele fijarse en 0.9, representa el decaimiento exponencial que introduce el algoritmo para mitigar el problema.

- Estimación adaptativa por momentos (Adam): Combina las idea de Momentum y RMSProp. Para ello, se realizan los siguientes pasos:

$$1. \ m \leftarrow \beta_1 m - (1 - \beta_1) \nabla_{\theta} J(\theta)$$

$$2. \ s \leftarrow \beta_2 s + (1 - \beta_2) \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta)$$

$$3. \ \hat{m} \leftarrow \frac{m}{1 - \beta_1^t}$$

$$4. \ \hat{s} \leftarrow \frac{s}{1 - \beta_2^t}$$

$$5. \ \theta \leftarrow \theta + \eta \hat{m} \oslash \sqrt{\hat{s} + \epsilon}.$$

Es un algoritmo ampliamente utilizado. Al contar con una tasa de aprendizaje adaptativa, requiere menos ajustes y es cómodo de utilizar.

- Nesterov-accelerated Adaptive Moment (Nadam): Es una variante de Adam que aplica la idea de

5 Aprendizaje profundo. Fundamentos

aceleración de Nesterov. Suele converger más rápido que Adam.

6 Aproximación de operadores no lineales mediante redes neuronales

En las últimas décadas, la mayoría de avances en el campo del aprendizaje profundo han estado inspirados en el Teorema de Aproximación Universal [Cyd89] publicado por George Cybenko en el año 1989. En él, se demuestra que las funciones de tipo $\sum_{j=1}^n \alpha_j \sigma(w_j^T x + \theta_j)$, que definen una red neuronal con una única capa oculta y activación sigmoidal, son densas en el espacio de funciones continuas $C(I^n)$, con $I^n = [0, 1]$. Unos años más tarde, en 1995, Tianping Chen y Robert Chen publican un artículo [CC95] en el que se revisan las limitaciones de este teorema. En concreto, señalan las restricciones impuestas a las funciones de activación sigmoidales, que se presuponen continuas, y la incapacidad de este tipo de redes neuronales para hacer frente a problemas regidos por funcionales u operadores no lineales. En esta sección entraremos en detalle en los avances propuestos en estas dos líneas.

6.1. Notación y definiciones

Comenzaremos aclarando algunos conceptos clave para seguir las demostraciones de esta sección:

Definición 6.0.1. Una función $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ es sigmoidal si cumple

$$\sigma(x) = \begin{cases} \lim_{x \rightarrow -\infty} \sigma(x) = 0 \\ \lim_{x \rightarrow \infty} \sigma(x) = 1. \end{cases}$$

Observación 6.0.1. A diferencia de otras definiciones, cuando nos refiramos a una función sigmoidal no vamos a presuponer continuidad ni monotonía.

Definición 6.0.2. Una función $g : \mathbb{R} \rightarrow \mathbb{R}$ se llama de Tauber-Wiener (TW) si satisface que todas las combinaciones lineales $\sum_{i=1}^N c_i g(\lambda_i x + \theta_i)$, $\lambda_i, \theta_i, c_i \in \mathbb{R}$, $i = 1, \dots, N$ son densas en cualquier espacio $C[a, b]$.

Definición 6.0.3. Denotaremos por $C_p[-1, 1]^n$ al espacio de las funciones periódicas con respecto a cada variable x_1, \dots, x_n cuyo periodo es 2.

6.2. Características de las funciones de activación

En esta sección trabajaremos en debilitar las condiciones bajo las cuales una función puede ser considerada función de activación, lo que nos dará más libertad a la hora de trabajar con operadores no lineales en la próxima sección. El resultado más importante es el [Teorema 6.2](#), que nos dice que pertenecer a la clase (TW) es condición suficiente para que una función sea de activación. Como las funciones (TW) toman valores en \mathbb{R} , vamos a poder trabajar con problemas simplificados. La convergencia uniforme que nos da este teorema va a ser muy importante para los resultados de la sección siguiente.

Teorema 6.1. *Supongamos que g es una función continua y $g \in S'(\mathbb{R})$ vista como distribución. Entonces $g \in (TW)$ si g no es un polinomio.*

Demostración. Si las combinaciones $\sum_{i=1}^N c_i g(\lambda_i x + \theta_i)$, vistas como subespacio, no son densas en $C[a, b]$ (es decir, si su cierre forma un subespacio cerrado propio en $C[a, b]$) entonces por el [Teorema de Extensión de Hahn-Banach](#) existe un funcional lineal y continuo F cuyo núcleo se traga al subespacio. Por otra parte, el [Teorema de Representación de Riesz](#) asegura que hay una medida de Borel $d\mu$ con $\text{supp}(d\mu) \subseteq [a, b]$ cumpliendo que $F(f) = \int_{\mathbb{R}} f d\mu$ para toda $f \in C[a, b]$.

Como $g(\lambda x + \theta)$ está en dicho subespacio para todo $\lambda \neq 0$ y $\theta \in \mathbb{R}$, por los dos resultados anteriores, sabemos que:

$$F(g(\lambda x + \theta)) = \int_{\mathbb{R}} g(\lambda x + \theta) d\mu = 0$$

para todo $\lambda \neq 0$ y $\theta \in \mathbb{R}$. Tomando ahora cualquier $w \in S(\mathbb{R})$, tenemos que

$$\int_{\mathbb{R}} w(\theta) d\theta \int_{\mathbb{R}} g(\lambda x + \theta) d\mu = 0$$

para todo $\lambda \neq 0$ y $\theta \in \mathbb{R}$.

Fijados un $\lambda \neq 0$ y un $\theta \in \mathbb{R}$, definimos la medida $\hat{d}\mu(A) = d\mu(\frac{A-\theta}{\lambda})$ y efectuamos el cambio de variable $y = \lambda x + \theta$. Esto nos va a permitir ver a g como distribución:

$$\int_{\mathbb{R}} g(y) \int_{\mathbb{R}} w(\theta) d\mu\left(\frac{y-\theta}{\lambda}\right) = \langle \Lambda_g, w(\cdot) * d\mu(\lambda \cdot) \rangle = 0. \quad (6.1)$$

Por el [Teorema de inversión](#), tenemos:

$$\begin{aligned} \langle \Lambda_g, w(\cdot) * d\mu(\lambda \cdot) \rangle &= \langle \Lambda_g, \mathcal{F}(\mathcal{F}(w(\cdot) * d\mu(\lambda \cdot))) \rangle = \langle \mathcal{F}(\Lambda_g), \mathcal{F}(w(\cdot) * d\mu(\lambda \cdot)) \rangle \\ &= \langle \mathcal{F}(\Lambda_g), \mathcal{F}(w(\cdot)) \cdot \mathcal{F}(d\mu(\lambda \cdot)) \rangle = 0. \end{aligned} \quad (6.2)$$

6.2 Características de las funciones de activación

Para que (6.1) tenga sentido, tenemos que probar que $[\mathcal{F}(w)](t) \cdot [\mathcal{F}(d\mu)](\lambda t) \in \mathcal{S}(\mathbb{R}^d)$. Como $\text{supp}(d\mu) \subseteq [a, b]$, podemos escribir

$$\begin{aligned}\langle \mathcal{F}(d\mu), \phi \rangle &= \langle d\mu, \mathcal{F}(\phi) \rangle = \int_{\mathbb{R}} d\mu(t) \mathcal{F}(\phi)(t) dt = \int_{\mathbb{R}} \left[\int_{\mathbb{R}} \phi(x) e^{-ixt} dx \right] d\mu(t) dt \\ &= \int_{\mathbb{R}} \phi(x) \left[\int_{\mathbb{R}} e^{-ixt} d\mu(t) dt \right] dx = \int_{\mathbb{R}} \phi(x) h(x) dx = \langle \Lambda_h, \phi \rangle\end{aligned}$$

donde $h(x) = \int_{\mathbb{R}} e^{-ixt} d\mu(t)$.

Además, por el **Teorema de derivación bajo el signo integral** sabemos que $h \in C^\infty(\mathbb{R})$, lo cual nos dice que $\mathcal{F}(d\mu)$ es, vista como una función, de clase $C^\infty(\mathbb{R})$.

Veamos ahora que para cada $k = 1, 2, \dots$, existe una constante c_k tal que:

$$|\frac{\partial^k}{\partial t^k} \mathcal{F}(d\mu)(t)| \leq c_k.$$

Efectivamente, tenemos que

$$|h(x)| = \left| \int_{\mathbb{R}} e^{-ixt} d\mu(t) \right| \leq \int_{\mathbb{R}} |e^{-ixt}| |d\mu(t)| \leq \int_{\mathbb{R}} |d\mu(t)| \leq 1, \quad \forall x \in \mathbb{R}.$$

De forma análoga, podemos ver que

$$\begin{aligned}\langle \frac{\partial^k}{\partial t^k} \mathcal{F}(d\mu), \phi \rangle &= \langle \mathcal{F}(t^k d\mu), \phi \rangle = \langle t^k d\mu, \mathcal{F}(\phi) \rangle = \int_{\mathbb{R}} \mathcal{F}(\phi)(t) t^k d\mu(t) dt \\ &= \int_{\mathbb{R}} \left[\int_{\mathbb{R}} \phi(x) e^{-ixt} t^k d\mu(t) dx \right] dt = \int_{\mathbb{R}} \phi(x) \left[\int_{\mathbb{R}} e^{-ixt} t^k d\mu(t) dt \right] dx \\ &= \langle \Lambda_{h_k}, \phi \rangle\end{aligned}$$

donde estamos considerando $\Lambda_{h_k} = \mathcal{F}(t^k d\mu)$, lo cual nos permite ver a h_k como función.

Desde aquí, teniendo en cuenta que $\text{supp}(d\mu) \subset [a, b]$, es fácil ver que

$$|h_k(x)| \leq \int_{\mathbb{R}} |t^k| |d\mu(t)| = c_k$$

para alguna constante $c_k \in \mathbb{R}$.

6 Aproximación de operadores no lineales mediante redes neuronales

Como consecuencia, por ser $\mathcal{F}(w)$ de crecimiento lento y tener $\mathcal{F}(d\mu)$ todas sus derivadas acotadas, $\mathcal{F}(w) \cdot \mathcal{F}(d\mu) \in \mathcal{S}(\mathbb{R})$. Por el **Teorema de Representación de Riesz** sabíamos que $d\mu \neq 0$ y, además, hemos visto que $\mathcal{F}(d\mu) \in C_c^\infty(\mathbb{R})$. Por tanto, podemos afirmar que existe algún $t_0 \neq 0$ con un entorno $(t_0 - \delta, t_0 + \delta)$ tal que $\mathcal{F}(d\mu)(t) \neq 0$ para todo $t \in (t_0 - \delta, t_0 + \delta)$. Si tomamos $\lambda = \frac{t_0}{t_1}$, con $t_1 \neq 0$, entonces $\mathcal{F}(d\mu)(\lambda t) \neq 0$, $\forall t \in (t_1 - \frac{\delta}{\lambda}, t_1 + \frac{\delta}{\lambda})$. Tomando un $\mathcal{F}(w) \in C_c^\infty(t_0 - \frac{\delta}{2\lambda}, t_0 + \frac{\delta}{2\lambda})$ cualquiera, entonces $\frac{\mathcal{F}(w)}{\mathcal{F}(d\mu)(\lambda t)} \in \mathcal{S}(\mathbb{R})$ y, por (6.2):

$$\mathcal{F}(g)(\mathcal{F}(w)(\cdot)) = \mathcal{F}(g) \left(\frac{\mathcal{F}(w)(\cdot)}{\mathcal{F}(d\mu)(\lambda \cdot)} \mathcal{F}(d\mu)(\lambda \cdot) \right) = 0.$$

Acabamos de ver que para cualquier punto $t^* \in \mathbb{R}$ fijo, existe un entorno $[t^* - \eta, t^* + \eta]$ tal que $\mathcal{F}(g)(\mathcal{F}(w)(\cdot)) = 0$ se cumple para todo $\mathcal{F}(w)$ con soporte compacto en $[t^* - \eta, t^* + \eta]$, esto es, $\text{supp}(\mathcal{F}(g)) \subseteq \{0\}$. Por el **Teorema 4.3**, $\mathcal{F}(g)$ es una combinación de deltas de Dirac y sus derivadas. Lo que equivale, como vimos en el **Lema 4.7.1**, a que g sea un polinomio. Con esto, el **Teorema 6.1** queda probado. \square

Teorema 6.2. *Supongamos que K es un conjunto compacto en \mathbb{R}^n , U es un conjunto compacto en $C(K)$ y $g \in (TW)$. Entonces, para cada $\varepsilon > 0$, existe un entero positivo N , números reales θ_i , vectores $w_i \in \mathbb{R}^n$, $i = 1, \dots, N$, con independencia de $f \in U$ y constantes $c_i(f)$, $i = 1, \dots, N$ dependientes de f tal que*

$$|f(x) - \sum_{i=1}^N c_i(f)g(w_i x + \theta_i)| < \varepsilon$$

se cumple para todo $x \in K$ y para toda $f \in U$.

Antes de demostrar el **Teorema 6.2**, veamos un resultado previo que nos va a ser necesario:

Lema 6.2.1. *Supongamos que K es un conjunto compacto en $I^n = [0, 1]^n$ y V es un conjunto compacto en $C(K)$. Entonces V puede extenderse a un conjunto compacto en $C_p[-1, 1]^n$.*

Demostración. Para toda $f \in V$ sabemos, por el **Lema 2.6.1**, que existe una extensión continua $E(f) \in C([0, 1]^n)$ tal que $\sup_{x \in [0, 1]^n} \{|E(f)(x)|\} \leq \sup_{x \in K} \{|f(x)|\}$. Por el **Teorema de Ascoli-Azela**, sabemos también que existe una constante M tal que $\|f(x)\|_{C(K)} \leq M$, por lo que $\sup_{x \in [0, 1]^n} \{|E(f)(x)|\} \leq \sup_{x \in K} \{|f(x)|\} < M$. Llamemos V_1 al conjunto de las extensiones $E(f)$ tales que $f \in V$. Concluimos que V_1 es compacto en $C([0, 1])$, pues $V_1 = E(V)$ y E es un operador continuo.

Para cada $f \in V_1$, podemos construir una extensión par de f :

$$f^*(x_1, \dots, x_k, \dots, x_n) = \begin{cases} f(x_1, \dots, -x_k, \dots, x_n) & \text{si } x_k \in [-1, 0[, \\ f(x_1, \dots, x_k, \dots, x_n) & \text{si } x \in [0, 1]^n. \end{cases}$$

6.2 Características de las funciones de activación

Entonces para cada $f \in V_1$, $f^* \in C_p([-1, 1]^n)$ y por tanto $U = \{f^* : f \in V_1\}$ es el compacto en $C_p[-1, 1]^n$ que buscamos. \square

Lema 6.2.2. Supongamos que U es un compacto en $C_p([-1, 1]^n)$.

$$B_R(f; x) = \sum_{\|m\| \leq R} (1 - \frac{\|m\|^2}{|R|^2})^\alpha c_m(f) e^{i\pi m \cdot x}$$

es la medida de Bochner-Riesz de la serie de Fourier de f , donde $m = (m_1, \dots, m_n)$, $\|m\|^2 = \sum_{i=1}^n |m_i|^2$ y $c_m(f)$ son los coeficientes de Fourier de f . Entonces, para todo $\varepsilon > 0$ existe un $R > 0$ tal que

$$|B_R(f; x) - f(x)| < \varepsilon$$

para toda $f \in U$ y $x \in [-1, 1]^n$, dado un $\alpha > \frac{n-1}{2}$.

Demostración del Teorema 6.2. Sin perder generalidad, podemos asumir que $K \subseteq [0, 1]^n$. Por el Lema 6.2.1, sabemos que podemos extender U a un $U' \subseteq C_p[-1, 1]^n$. Además, por el Lema 6.2.2, sabemos que para todo $\varepsilon > 0$ existe un $R > 0$ tal que se cumple

$$|B_R(f^*; x) - f^*(x)| = \left| \sum_{i=1}^n (1 - \frac{\|m\|^2}{|R|^2}) c_m(f^*) e^{i\pi(m \cdot x)} - f^*(x) \right| < \frac{\varepsilon}{2} \quad (6.3)$$

para $x \in [-1, 1]^n$ y $f \in U$, donde f^* sigue la misma construcción expuesta en el Lema 6.2.1.

Por la paridad de f^* , simplificando los coeficientes de Fourier podemos reescribir (6.3)

$$\left| \sum_{\|m\| \leq R} d_m \cos(\pi(m \cdot x)) - f^*(x) \right| < \frac{\varepsilon}{2} \quad (6.4)$$

donde $d_m = (1 - \frac{\|m\|^2}{|R|^2}) c_m(f^*)$.

Podemos ahora realizar el cambio de variable $u = \pi m \cdot x$ para $x \in [-1, 1]^n$, por lo que $u \in [-\sqrt{n}\pi R, \sqrt{n}\pi R]$, ya que $\|x\| \leq \sqrt{n}$, $\|m\| \leq R$. Como $\cos(u)$ es una función continua en $[-\sqrt{n}\pi R, \sqrt{n}\pi R]$ y $g \in (TW)$, existen un entero M y números reales s_j, η_j, ξ_j con $j = 1, \dots, M$ tales que

$$\left| \sum_{j=1}^M s_j g(u\xi_j + \eta_j) - \cos(u) \right| < \frac{\varepsilon}{2L}$$

6 Aproximación de operadores no lineales mediante redes neuronales

se cumple para todo $u \in [-\sqrt{n}\pi R, \sqrt{n}\pi R]$, donde $L = \sum_{\|m\| \leq R} |d_m|$. Esto equivale a que

$$\left| \sum_{j=1}^M s_j g(\xi_j \pi(m \cdot x) + \eta_j) - \cos(\pi m \cdot x) \right| < \frac{\varepsilon}{2L} \quad (6.5)$$

se cumpla para todo $x \in [-1, 1]^n$.

Sean $A = \{m \in \mathbb{N}^n : \|m\| \leq R\}$, $A = \{m^1, \dots, m^N\}$ una indexación de A y sea $N = |A|M$, donde $|A|$ es el cardinal de A . Considerando ahora el índice $i = 1, \dots, N$, que despliega $|A| M$ veces, denotamos $i \bmod |A|$ al resto de dividir i por $|A|$ ($i \leq N$). Así, conforme vayamos avanzando en i , $m^{i \bmod |A|}$ denotará al elemento correspondiente de A dentro de uno de los M despliegues sobre el conjunto y podemos definir:

- $c_i(f) = z_i d_{m^i \bmod |A|} = z_i (1 - \frac{\|m^i \bmod |A|\|^2}{|R|^2}) c_{m^i \bmod |A|}(f^*)$.

- $z_i = \begin{cases} s_1 & \text{si } 1 \leq i \leq |A| \\ s_2 & \text{si } |A| < i \leq 2|A| \\ \dots \\ s_M & \text{si } (M-1)|A| < i \leq M|A|. \end{cases}$

- $\omega_i = \begin{cases} \xi_1 \pi m^i \bmod |A| & \text{si } 1 \leq i \leq |A| \\ \xi_2 \pi m^i \bmod |A| & \text{si } |A| < i \leq 2|A| \\ \dots \\ \xi_M \pi m^i \bmod |A| & \text{si } (M-1)|A| < i \leq M|A|. \end{cases}$

- $\theta_i = \begin{cases} \eta_1 & \text{si } 1 \leq i \leq |A| \\ \eta_2 & \text{si } |A| < i \leq 2|A| \\ \dots \\ \eta_M & \text{si } (M-1)|A| < i \leq M|A|. \end{cases}$

Usando (6.4) y (6.5), podemos concluir que los elementos $N \in \mathbb{N}$, $\theta_i \in \mathbb{R}$, $w_i \in \mathbb{R}^n$, $i = 1, \dots, N$

definidos cumplen:

$$\begin{aligned}
 |f^*(x) - \sum_{i=1}^N c_i(f^*)g(w_i \cdot x + \theta_i)| &= |f^*(x) - \sum_{\|m\| \leq R} d_m \cos(\pi(m \cdot x)) \\
 &\quad + \sum_{\|m\| \leq R} d_m \cos(\pi(m \cdot x)) - \sum_{i=1}^N c_i(f^*)g(w_i \cdot x + \theta_i)| \\
 &\leq |f^*(x) - \sum_{\|m\| \leq R} d_m \cos(\pi(m \cdot x))| + \left| \sum_{\|m\| \leq R} d_m \cos(\pi(m \cdot x)) - \sum_{i=1}^N c_i(f^*)g(w_i \cdot x + \theta_i) \right| \\
 &< \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon
 \end{aligned}$$

para todo $x \in [-1, 1]^n$. Restringiéndonos ahora a $[0, 1]^n$ tenemos que

$$|f(x) - \sum_{i=1}^N c_i(f)g(w_i \cdot x + \theta_i)| < \varepsilon$$

se cumple para todo $x \in [0, 1]^n$ y $f \in U$.

Con respecto al paso de $c_i(f^*)$ a $c_i(f)$, para cada $m = (m_1, \dots, m_n)$, el coeficiente de Fourier

$$c_m(f^*) = \int_{-1}^1 \cdots \int_{-1}^1 e^{i\pi(m \cdot x)} f^*(x) dx$$

es un funcional continuo definido en U' . Por la definición de f^* y cambiando los límites de integración, $c_i(f)$ está bien definido en U . Así, el [Teorema 6.2](#) queda demostrado. \square

Observación 6.2.1. Por ser los $c_i(f)$ múltiplos de los coeficientes de Fourier de f^* con la restricción a U mencionada, cada $c_i(f)$, $i = 1, \dots, N$ actúa de forma continua sobre U .

Teniendo ya en mente, por el teorema anterior, que basta que una función sea de clase (TW) para considerarla función de activación, el siguiente resultado nos permite relajar las condiciones clásicas de continuidad o monotonía exigidas a una función de activación sigmoidal:

Teorema 6.3. *Si σ es una función sigmoidal acotada, entonces $\sigma \in (\text{TW})$.*

Demostración. El enunciado de este teorema es equivalente a afirmar que si $\sigma(x)$ es una función sigmoidal acotada y tomamos $f(x) \in C(\mathbb{R})$ tal que $\lim_{x \rightarrow -\infty} f(x) = A$ y $\lim_{x \rightarrow \infty} f(x) = B$, con $A, B \in \mathbb{R}$, entonces para cada $\varepsilon > 0$ existirán $N \in \mathbb{N}$, $c_i, y_i, \theta_i \in \mathbb{R}$ tales que

6 Aproximación de operadores no lineales mediante redes neuronales

$$|f(x) - \sum_{i=1}^N c_i \sigma(y_i x + \theta_i)| < \varepsilon$$

se cumple para todo $x \in (-\infty, \infty)$.

Partiendo de la función f que hemos fijado, para cada $\varepsilon > 0$ podemos encontrar una constante real $M > 0$ tal que

1. $|f(x) - A| < \frac{\varepsilon}{4}$ si $x < -M$.
2. $|f(x) - B| < \frac{\varepsilon}{4}$ si $x > M$.
3. $|f(x') - f(x'')| < \frac{\varepsilon}{4}$ si $|x'| \leq M, |x''| \leq M$ y $|x' - x''| \leq \frac{1}{M}$.

Tomemos una partición de $[-M, M]$ en $2M^2$ segmentos iguales, cada uno de longitud $\frac{1}{M}$.

$$-M = x_0 < x_1 < \dots < x_M = 0 < x_{M+1} < \dots < x_{2M} = M.$$

Por ser σ una función sigmoidal, existe un $W > 0$ tal que

1. $|\sigma(u) - 1| < \frac{1}{M^2}$ si $u > W$.
2. $|\sigma(u)| < \frac{1}{M^2}$ si $u < -W$.

Tomando una constante $K > 0$ que cumpla $K \frac{1}{2M} > W$ podemos entonces construir una función

$$g(x) = f(-M) + \sum_{i=1}^N [f(x_i) - f(x_{i-1})] \sigma(K(x - t_{i-1})) \quad (6.6)$$

siendo $t_i = \frac{1}{2}(x_i + x_{i+1})$, $|x - t_i| \leq \frac{1}{2M}$, $\forall x \in (x_i, x_{i+1})$ y $N = 2M^2$. Veamos que se cumple que $|f(x) - g(x)| < \varepsilon$ para cada $x \in \mathbb{R}$.

■ Si $x < -M$, entonces

$$|f(x) - f(-M)| = |f(x) - A + A - f(-M)| < |f(x) - A| + |A - f(-M)| < \frac{\varepsilon}{4} + \frac{\varepsilon}{4} = \frac{\varepsilon}{2}.$$

Además, $(x - t_{i-1}) < (-M - (-M + \frac{1}{2M})) = -\frac{1}{2M}$, por lo que usando (6.6) obtenemos:

$$|g(x) - f(-M)| \leq \sum_{i=1}^N |f(x_i) - f(x_{i-1})| |\sigma(K(x - t_{i-1}))| < \sum_{i=1}^N \frac{\varepsilon}{4} \frac{1}{M^2} = \frac{\varepsilon}{2}$$

por ser $K(x - t_{i-1}) < -\frac{K}{2M} < -W$.

Por tanto,

$$|f(x) - g(x)| \leq |f(x) - f(-M)| + |f(-M) - g(x)| < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon.$$

- La demostración es análoga para $x > M$.
- Consideremos ahora el caso en el que $x \in [-M, M]$. Entonces, para algún $k = 0, \dots, 2M$, tendremos que $x \in [x_{k-1}, x_k]$. Por cómo hemos definido la partición sabemos que $|x - t_{i-1}| \leq \frac{1}{2M}$ si $i = k$ y $|x - t_{i-1}| > \frac{1}{2M}$ si $i \neq k$. Además, si $i < k$, $K(x - t_{i-1}) > W$ y por tanto $|\sigma(K(x - t_{i-1})) - 1| < \frac{1}{M^2}$. Análogamente, si $i > k$ entonces $K(x - t_{i-1}) < -W$ y por tanto $|\sigma(K(x - t_{i-1}))| < \frac{1}{M^2}$. En este caso, (6.6) nos permite escribir

$$\begin{aligned} & |g(x) - f(-M) - [f(x_k) - f(x_{k-1})]\sigma(K(x - t_{k-1})) - \sum_{i=1}^{k-1} [f(x_i) - f(x_{i-1})]| \\ & \leq \sum_{i=1}^{k-1} |f(x_i) - f(x_{i-1})||\sigma(K(x - t_{i-1})) - 1| + \sum_{i=k+1}^{2M} |f(x_i) - f(x_{i-1})||\sigma(K(x - t_{i-1}))| \\ & \leq \sum_{i=1}^{k-1} \frac{\varepsilon}{4} \frac{1}{M^2} + \sum_{i=k+1}^{2M} \frac{\varepsilon}{4} \frac{1}{M^2} \leq \frac{\varepsilon}{2}. \end{aligned}$$

Por otro lado,

$$\begin{aligned} & f(-M) + [f(x_k) - f(x_{k-1})]\sigma(K(x - t_{k-1})) - \sum_{i=1}^{k-1} [f(x_i) - f(x_{i-1})] \\ & = f(x_{k-1}) + [f(x_k) - f(x_{k-1})]\sigma(K(x - t_{k-1})). \end{aligned}$$

De las desigualdades anteriores, deducimos que

$$|g(x) - f(x)| \leq \frac{\varepsilon}{2} + |f(x) - f(x_{k-1})| + |f(x_k) - f(x_{k-1})||\sigma(K(x - t_{k-1}))| < \frac{\varepsilon}{2} + \frac{\varepsilon}{4} + \frac{\varepsilon}{4} = \varepsilon$$

lo que concluye la prueba.

□

6.3. Aproximación de funcionales y operadores continuos no lineales

Teorema 6.4. Supongamos que $g \in (TW)$, X es un espacio de Banach, $K \subseteq X$ es compacto, V es un conjunto compacto de $C(K)$ y f es un funcional definido en V . Entonces, para cada $\varepsilon > 0$ existen un entero positivo N , m puntos $x_1, \dots, x_m \in K$ y constantes $c_i, \xi_{ij}, \theta_i \in \mathbb{R}$ tales que

$$|f(u) - \sum_{i=1}^N c_i g\left(\sum_{j=1}^m \xi_{ij} u(x_j) + \theta_i\right)| < \varepsilon \quad (\forall u \in V).$$

Lema 6.4.1. Sea X un espacio de Banach y $K \subseteq X$, entonces K es compacto si y solo si se cumplen simultáneamente:

1. K es cerrado en X .
2. Para todo $\delta > 0$, existe una red $N(\delta) = \{x_1, \dots, x_{n(\delta)}\}$. Esto es, para cada $x \in K$, existe $x_k \in N(\delta)$ tal que $\|x - x_k\|_X < \delta$.

Observación 6.4.1. Para cada δ , el número $n(\delta)$ de elementos de $N(\delta)$ es finito, lo que equivale a decir que K está totalmente acotado.

Lema 6.4.2 (Otra lectura del Teorema de Ascoli-Azela). Si $V \subseteq C(K)$ es compacto en $C(K)$, entonces V es uniformemente acotado y equicontinuo. Esto es:

1. Existe $A > 0$ tal que $\|u(x)\|_{C(K)} \leq A, \forall u \in V$.
2. $\forall \varepsilon > 0, \exists \delta > 0$ tal que $|u(x') - u(x'')| < \varepsilon, \forall u \in V$, siempre que $\|x' - x''\|_X < \delta$.

Definición 6.4.1. Construiremos ahora varios elementos que vamos a utilizar en los resultados siguientes.

Sea f un funcional continuo definido en un compacto $V \subseteq C(K)$. Entonces, escogiendo una sucesión $\varepsilon_1 > \varepsilon_2 > \dots > \varepsilon_n \rightarrow 0$, podemos encontrar otra sucesión $\delta_1 > \delta_2 > \dots > \delta_n \rightarrow 0$ tal que $|f(u) - f(v)| < \varepsilon_k$ siempre que los $u, v \in V$ escogidos cumplan $\|u - v\|_{C(K)} < 2\delta_k$. Por el lema Lema 6.4.2, V es acotado y equicontinuo, por lo que además podemos encontrar una sucesión $\eta_1 > \eta_2 > \dots > \eta_n \rightarrow 0$ tal que $|u(x') - u(x'')| < \delta_k$ para todo $u \in V$ y $\|x' - x''\|_K < \eta_k$.

Usando el Lema 6.4.1, podemos reordenar los elementos de K para conseguir una sucesión $\{x_i\}_{i=1}^\infty$, con $x_i \in K$, y una sucesión de enteros positivos $n(\eta_1) < n(\eta_2) < \dots < n(\eta_k) \rightarrow \infty$ tal que los $n(\eta_k)$ primeros elementos $N(\eta_k) = \{x_1, \dots, x_{n(\eta_k)}\}$ formen una η_k -red en K .

6.3 Aproximación de funcionales y operadores continuos no lineales

Para cada η_k -red, definimos

$$T_{\eta_k,j}^*(x) = \begin{cases} 1 - \frac{\|x - x_j\|_X}{\eta_k} & \text{si } \|x - x_j\|_X \leq \eta_k \\ 0 & \text{en otro caso} \end{cases} \quad \text{y} \quad T_{n(\eta_k),j} = \frac{T_{\eta_k,j}^*(x)}{\sum_{j=1}^{n(\eta_k)} T_{\eta_k,j}^*(x)}$$

para $j = 1, \dots, n(\eta_k)$.

Con esta definición, se puede ver que $\{T_{n(\eta_k),j}\}$ es una partición de la unidad, ya que

$$0 \leq T_{n(\eta_k),j}(x) \leq 1, \quad \sum_{j=1}^{n(\eta_k)} T_{n(\eta_k),j}(x) = 1 \quad \text{y} \quad T_{n(\eta_k),j}(x) = 0 \quad \text{si } \|x - x_j\|_X > \eta_k.$$

Para cada $u \in V$, podemos definir ahora funciones

$$u_{\eta_k}(x) = \sum_{j=1}^{n(\eta_k)} u(x_j) T_{n(\eta_k),j}(x)$$

y conjuntos $V_{\eta_k} = \{u_{\eta_k} : u \in V\}$, $V^* = V \bigcup (\bigcup_{k=1}^{\infty} V_{\eta_k})$.

Lema 6.4.3.

1. Para cada k fijo, V_{η_k} es compacto en un subespacio de dimensión $n(\eta_k)$ en $C(K)$.

2. Para cada $u \in V$, se cumple

$$\|u - u_{\eta_k}\|_{C(K)} < \delta_k.$$

3. V^* es un conjunto compacto en $C(K)$.

Demostración del Teorema 6.4. Por el **Teorema de extensión de Tietze**, podemos definir un funcional continuo en V^* tal que

$$f^*(x) = f(x) \quad \forall x \in V.$$

Como f^* es un funcional continuo definido en el compacto V^* , por el **Lema 6.4.2** sabemos que f^* es equicontinua; esto es, para cada $\varepsilon > 0$ podemos encontrar un $\delta > 0$ tal que $|f^*(u) - f^*(v)| < \frac{\varepsilon}{2}$, dados cualesquiera $u, v \in V^*$ tales que $\|u - v\|_{C(K)} < \delta$. Fijando k tal que $\delta_k < \delta$, entonces para cada $u \in V$, tenemos que

$$\|u - u_{\eta_k}\|_{C(K)} < \delta_k.$$

6 Aproximación de operadores no lineales mediante redes neuronales

Lo cual implica que

$$|f^*(u) - f^*(u_{\eta_k})| < \frac{\varepsilon}{2} \quad \forall u \in V. \quad (6.7)$$

Por el [Lema 6.4.3](#) V_{η_k} es compacto en un subespacio de dimensión finita de $C(K)$, lo cual equivale, por el [Teorema de Hausdorff](#), a ser isométricamente isomorfo a un compacto del espacio $\mathbb{R}^{n(\eta_k)}$, al que vamos a denotar como $U_{\eta_k} \subset \mathbb{R}^{n(\eta_k)}$. Puesto que f^* es continua en V^* lo será, en concreto, en $V_{\eta_k} \subset V^*$ y, por el isomorfismo indicado, también podremos verla como una función continua $f^* : U_{\eta_k} \rightarrow \mathbb{R}$. Por el [Teorema 6.2](#), tomando $K = U_{\eta_k}$ y teniendo en mente para f^* la preservación de la compacidad por funciones continuas, podemos encontrar $N, c_i, \xi_{ij}, \theta_i \in \mathbb{R}$, $i = 1, \dots, N$, $j = 1, \dots, n(\eta_k)$ tales que

$$|f^*(u_{\eta_k}) - \sum_{i=1}^N c_i g\left(\sum_{j=1}^{n(\eta_k)} \xi_{ij} u(x_j) + \theta_i\right)| < \frac{\varepsilon}{2}.$$

Combinando esto con (6.7) concluimos que, para toda $u \in V$,

$$\begin{aligned} |f(u) - \sum_{i=1}^N c_i g\left(\sum_{j=1}^{n(\eta_k)} \xi_{ij} u(x_j) + \theta_i\right)| &= |f^*(u) - \sum_{i=1}^N c_i g\left(\sum_{j=1}^{n(\eta_k)} \xi_{ij} u(x_j) + \theta_i\right)| \\ &= |f^*(u) - f^*(u_{\eta_k}) + f^*(u_{\eta_k}) - \sum_{i=1}^N c_i g\left(\sum_{j=1}^{n(\eta_k)} \xi_{ij} u(x_j) + \theta_i\right)| \\ &\leq |f^*(u) - f^*(u_{\eta_k})| + \left|f^*(u_{\eta_k}) - \sum_{i=1}^N c_i g\left(\sum_{j=1}^{n(\eta_k)} \xi_{ij} u(x_j) + \theta_i\right)\right| < \varepsilon. \end{aligned}$$

Llamando m a $n(\eta_k)$ queda demostrado el [Teorema 6.4](#). □

Teorema 6.5. Sean $g \in (TW)$, $X = C(K_1)$ un espacio de Banach, $V \subseteq X$, $K_1, K_2 \subseteq \mathbb{R}^n$ compactos en X y \mathbb{R}^n respectivamente. Sea G un operador continuo no necesariamente lineal que lleva V en $C(K_2)$. Entonces, para cada $\varepsilon > 0$ existen enteros $M, N, m > 0$, constantes $c_i^k, \zeta_k, \xi_{ij}^k \in \mathbb{R}$ y puntos $w_k \in \mathbb{R}^n$, $x_j \in K_1$, con $i = 1, \dots, M$, $k = 1, \dots, N$, $j = 1, \dots, m$ tales que

$$|G(u)(y) - \sum_{k=1}^N \sum_{i=1}^M c_i^k g\left(\sum_{j=1}^m \xi_{ij}^k u(x_j) + \theta_i^k\right) g(w_k \cdot y + \zeta_k)| < \varepsilon, \quad \forall u \in V, \forall y \in K_2.$$

Demostración. De la hipótesis de que G es un operador continuo que lleva a $V \subset C(K_1)$ en $C(K_2)$, se comprueba directamente que la imagen $G(V) = \{G(u) : u \in V\}$ también es un conjunto compacto en $C(K_2)$. Por el [Teorema 6.2](#), para cada $\varepsilon > 0$ existen $N \in \mathbb{N}$, $c_k(G(u)), \zeta_k \in \mathbb{R}$ y vectores $w_k \in \mathbb{R}^n$, $k = 1, \dots, N$ tales que

6.3 Aproximación de funcionales y operadores continuos no lineales

$$|G(u)(y) - \sum_{k=1}^N c_k(G(u))g(w_k \cdot y + \zeta_k)| < \frac{\varepsilon}{2} \quad (6.8)$$

se cumple para cualesquiera $y \in K_2$ y $u \in V$.

El teorema también nos dice que para cada $k = 1, \dots, N$, $c_k(G(u))$ es un funcional que actúa de forma continua en V . Aplicando repetidamente el **Teorema 6.4**, para cada $k = 1, \dots, N$, podemos encontrar $N_k, m_k \in \mathbb{N}$, constantes $c_i^k, \xi_{ij}^k, \theta_i^k \in \mathbb{R}$ y $x_j \in K_1$, con $i = 1, \dots, N_k, j = 1, \dots, m_k$, tales que

$$|c_k(G(u)) - \sum_{i=1}^{N_k} c_i^k g\left(\sum_{j=1}^{m_k} \xi_{ij}^k u(x_j) + \theta_i^k\right)| < \frac{\varepsilon}{2L} \quad (6.9)$$

se cumple para cualquier $k = 1, \dots, N$ y $u \in V$, donde vamos a tomar

$$L = \sum_{k=1}^N \sup |g(w_k \cdot y + \zeta_k)|.$$

Usando (6.9) y (6.8), obtenemos que

$$\begin{aligned} & |G(u)(y) - \sum_{k=1}^N \sum_{i=1}^{N_k} c_i^k g\left(\sum_{j=1}^{m_k} \xi_{ij}^k u(x_j) + \theta_i^k\right) g(w_k \cdot y + \zeta_k)| = \\ & |G(u)(y) - \sum_{k=1}^N c_k(G(u))g(w_k \cdot y + \zeta_k) + \sum_{k=1}^N c_k(G(u))g(w_k \cdot y + \zeta_k) \\ & - \sum_{k=1}^N \sum_{i=1}^{N_k} c_i^k g\left(\sum_{j=1}^{m_k} \xi_{ij}^k u(x_j) + \theta_i^k\right) g(w_k \cdot y + \zeta_k)| \leqslant \\ & |G(u)(y) - \sum_{k=1}^N c_k(G(u))g(w_k \cdot y + \zeta_k)| + |\sum_{k=1}^N c_k(G(u))g(w_k \cdot y + \zeta_k) \\ & - \sum_{k=1}^N \sum_{i=1}^{N_k} c_i^k g\left(\sum_{j=1}^{m_k} \xi_{ij}^k u(x_j) + \theta_i^k\right) g(w_k \cdot y + \zeta_k)| < \varepsilon \end{aligned} \quad (6.10)$$

se cumple para cualesquiera $u \in V$ e $y \in K_2$.

Tomando ahora $M = \max_k\{N_k\}$, $m = \max_k\{m_k\}$, tales que para cada k , para todo i que cumpla $N_k < i \leq M$ se tenga $c_i^k = 0$ y para todo j que cumpla $m_k < j \leq m$, se tenga $\xi_{ij}^k = 0$, (6.10) se puede

6 Aproximación de operadores no lineales mediante redes neuronales

reescribir como

$$|G(u)(y) - \sum_{k=1}^N \sum_{i=1}^M c_i^k g\left(\sum_{j=1}^m \xi_{ij}^k u(x_j) + \theta_i^k\right) g(w_k \cdot y + \zeta_k)| < \varepsilon$$

$\forall u \in V, \forall y \in K_2$. Esto completa la prueba del [Teorema 6.5](#).

□

6.4. Conclusiones sobre los resultados teóricos presentados: Aplicación a Sistemas Dinámicos

Como consecuencia directa del [Teorema 6.5](#), podemos utilizar redes neuronales para aproximar la salida de un sistema dinámico no lineal (esto es, regido por un operador no lineal).

Sea un sistema $V = KU$, donde U es la entrada, V es la salida y K es sistema que queremos identificar. Supongamos que utilizando sensores hemos podido extraer varias relaciones de entrada y salida $V_1 = KU_1, \dots, V_n = KU_n$, que podemos expresar como $\{u_s(x_j), s = 1, \dots, n, j = 1, \dots, m\}, \{v_s(y_l), s = 1, \dots, n, l = 1, \dots, L\}$. Usando estos puntos, por el [Teorema 6.5](#) sabemos que podemos construir un funcional

$$E = \sum_{l=1}^L \sum_{s=1}^n |V_s(y_l) - \sum_{k=1}^N \sum_{i=1}^M C_i^k g\left(\sum_{j=1}^m \xi_{ij}^k u_s(x_j) + \theta_i^k\right) g(w_k \cdot y_l + \zeta_k)|^2$$

donde los parámetros $C_i^k, \xi_{ij}^k, \theta_i^k, w_k, \zeta$ pueden elegirse de modo que minimicen E . De este modo, podemos ver

$$v(y) = \sum_{k=1}^N \sum_{i=1}^M C_i^k g\left(\sum_{j=1}^m \xi_{ij}^k u(x_j) + \theta_i^k\right) g(w_k \cdot y + \zeta_k)$$

como un aproximante de $V(y) = (KU)(y)$ y, por tanto, se identifica con el sistema K .

Si el sistema es lineal, entonces E y $V(y)$ se pueden simplificar como:

$$E = \sum_{l=1}^L \sum_{s=1}^n |V_s(y_l) - \sum_{k=1}^N \sum_{i=1}^M \sum_{j=1}^m \xi_{ij}^k u_s(x_j) g(w_k \cdot y_l + \zeta_k)|^2$$

$$v(y) = \sum_{k=1}^N \sum_{i=1}^M \sum_{j=1}^m \xi_{ij}^k u(x_j) g(w_k \cdot y + \zeta_k).$$

6.4 Conclusiones sobre los resultados teóricos presentados: Aplicación a Sistemas Dinámicos

Cuanto mayores sean los valores de n , L y m , más precisión tendrá esta aproximación.

Por tanto, tenemos una forma de construir modelos de redes neuronales que se identifiquen con sistemas dinámicos. En la [Figura 6.1](#) se muestra la arquitectura de la red neuronal que se propone.

6 Aproximación de operadores no lineales mediante redes neuronales

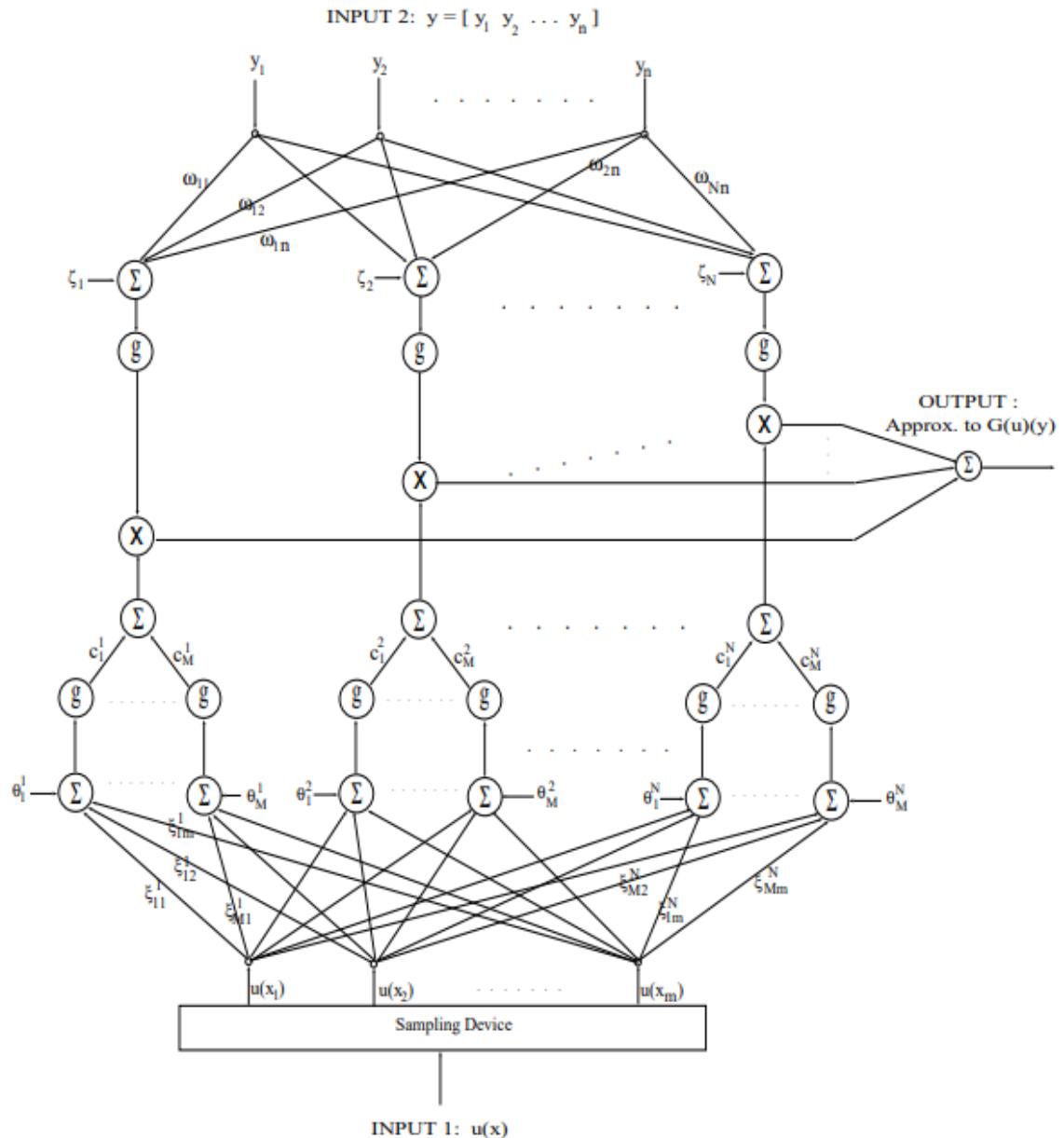


Figura 6.1: Arquitectura de red neuronal para la aproximación de operadores no lineales extraída de [CC95].

7 Physics Informed Neural Networks: usos e implementación en *SciANN*

El capítulo anterior finaliza presentando una arquitectura de red neuronal válida para la aproximación de operadores. En la [Figura 6.1](#) podemos ver que una de las entradas para esta arquitectura son muestras de datos recogidas por un dispositivo. Analizando esta arquitectura y teniendo en cuenta que en [\[CC95\]](#) no se menciona ninguna técnica concreta para conseguir un entrenamiento eficiente, en este trabajo proponemos un acercamiento a través de PINNs.

7.1. Introducción a PINNs

Las Physics Informed Neural Networks (PINNs), presentadas en [\[RPK19\]](#), conforman un novedoso paradigma que permite combinar las bondades del Aprendizaje Profundo con las restricciones matemáticas y físicas que requiere el modelado de distintos fenómenos físicos.

Históricamente, la labor de investigación principal en el campo del Aprendizaje Profundo ha sido encontrar estructuras y técnicas que sean capaces de abordar problemas cada vez más ambiciosos. En estas complejas arquitecturas, es esencial disponer de un gran volumen de datos para realizar correctamente la tarea de aprendizaje pues, en caso de no disponer de los suficientes datos, una red neuronal de este calibre puede sobre-ajustarse a un conjunto de datos pequeño o dejar a medias la tarea de aprendizaje.

En una era gobernada por el dato, las PINNs responden a la problemática de que, en muchos fenómenos físicos, es posible que nuestra capacidad para tomar mediciones (es decir, extraer datos) de un fenómeno sea limitada y que, por tanto, no dispongamos de los suficientes datos. En ellas se explota la capacidad de regularización que tienen varios elementos inherentes a un sistema físico, como pueden ser las leyes físicas que lo rigen u otros comportamientos validados empíricamente.

Para lograr esta regularización y que la capacidad de las redes neuronales profundas para construir aproximadores globales [\[HSW89\]](#) no conduzca a sobreajustes, las PINNs hacen uso de algunos avances recientes en el campo de la Diferenciación Automática [\[BPRS18\]](#), la cual engloba una familia de técnicas que siguen el espíritu de la propagación hacia atrás y tiene aplicaciones en distintos ámbitos como las ciencias atmosféricas, la optimización en diseños de ingeniería o la dinámica de fluidos computacional. Este enfoque permite añadir restricciones a la función de pérdida del modelo para evitar que los datos describan escenarios que son físicamente imposibles. Así, quedan restringidos los grados de libertad del modelo, permitiendo un ajuste adecuado a cada problema.

Con este enfoque, las PINNs ofrecen soluciones a dos tipos de problemas: encontrar sistemas a partir de datos y encontrar las ecuaciones en derivadas parciales que describen los datos.

7.2. Introducción a *SciANN*

SciANN [HJ21] es un paquete de python implementado sobre Tensorflow y Keras que permite la implementación de PINNs mediante una interfaz de alto nivel. Aunque actualmente únicamente permite la implementación de redes neuronales densamente conectadas, el proyecto pretende dar soporte a más tipos de arquitecturas.

7.2.1. Arquitectura

Una vez entendido el propósito general de la librería, procedemos a ver más detalladamente cómo se lleva a cabo esta abstracción. En primer lugar, comenzaremos exponiendo sus componentes principales:

- Clase Functional: Conceptualmente representa a una función cuyo comportamiento queremos aprender, que en implementación se corresponde con una red neuronal profunda. Un objeto Functional básico es representado internamente como una red neuronal formada por varias capas densas a especificar por el usuario. Sin embargo, como veremos más adelante, *SciANN* da soporte a arquitecturas más complejas permitiendo atar las entradas o salidas de objetos *Functional* a través de operaciones matemáticas (estas operaciones equivalen a una capa Lambda de Keras). El resultado de combinar objetos *Functional* mediante operaciones matemáticas también será un objeto de esta clase. Esto, retomando el significado conceptual de la clase, es equivalente a decir que podemos modelar funciones a través de redes neuronales con arquitecturas más complejas.
- Clase MLPFunctional (Multi Layer Perceptron Functional): Modela una red neuronal densa. A efectos prácticos cumple la misma función que *Functional*, pero *SciANN* la utiliza como superclase para definir distintos objetos de la interfaz —como las variables de entrada— que pueden beneficiarse de la estructura MLP, más sencilla y delimitada.
- Clase Variable: Es una clase heredada de *MLPFunctional* que configura las entradas de la red neuronal. Estas entradas se identifican con las variables de una función —que habrá sido modelada a través de la clase *Functional*—. La creación de una nueva variable conlleva de forma interna crear una capa *InputLayer* de Keras, que se pasa como única capa al constructor de la superclase *MLPFunctional*. Al encapsularse dentro de *MLPFunctional*, a la capa de entrada se le asigna un nombre único que será utilizado para realizar operaciones matemáticas a las funciones —como cálculo de derivadas parciales— durante el proceso de entrenamiento. *SciANN* requiere que cada variable sea un objeto unidimensional, por lo que si estamos trabajando con una función que toma m variables de dimensión n , tendremos que definir un total de $m \times n$ variables.
- Clase Field: Heredada de la clase *Dense* de Keras, conserva su estructura y funcionalidades. A nivel lógico, en *SciANN* se utiliza para definir las salidas de cada función a modelar. Si bien su

uso no es obligatorio, ya que para modelos simples se construye implícitamente, es recomendable definir objetos de la clase *Field* cuando queremos tener varias salidas en nuestra red neuronal para asegurar una correcta asociación entre cada objeto *Functional* y sus correspondientes salidas.

- Clase Constraint: Se utiliza para asociar restricciones a funciones y una de las clases claves para comprender cómo se materializa la implementación de PINNs en alto nivel en *SciANN*. Para hacer efectiva la asociación entre una función y una restricción, únicamente necesitamos instanciar esta clase pasando como parámetros el nombre del objeto *Functional* que se identifica con la función y el “llamable” que define la restricción.
- Utils: Esta parte de la interfaz implementa las operaciones matemáticas que podemos necesitar en el modelo a distintos niveles conceptuales. Por un lado, permite para definir las distintas restricciones que aplicaremos al modelo y que se añadirán a la función de pérdida. Por otro lado, también ofrece operaciones matemáticas que se pueden aplicar en la salida de los objetos *Functional* para modificar la estructura de la red neuronal que describe al modelo. Estas operaciones están implementadas a través de capas *Lambda* en Keras.
- Clase SciModels: Aunque en la práctica es similar a la clase *Model* de Keras, es una representación a más alto nivel que integra ese modelo clásico con las restricciones definidas por las clases mencionadas anteriormente.

Ahora que hemos entendido cómo funcionan cada uno de los elementos principales, veamos un ejemplo de cómo se relacionan entre ellos:

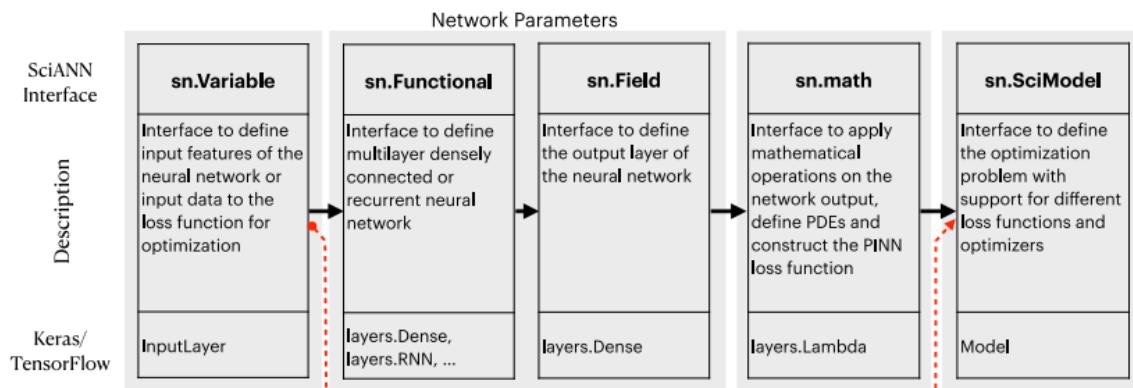


Figura 7.1: Elementos abstractos de *SciANN* y su relación con capas de Keras y Tensorflow. Diagrama extraído de [HJ21].

En el diagrama de la Figura 7.1, se muestra el flujo de creación de una PINN a través de cinco bloques. Los primeros tres bloques muestran cómo se mantiene una estructura secuencial entre la capa de variables de entrada, las capas de la función y la capa de salida. Entre las salidas podemos definir restricciones, relaciones y operaciones matemáticas, tal y como se muestra en el cuarto bloque. Son únicamente estas restricciones junto con los datos de entrada lo que debemos especificar para construir

el modelo —último bloque del procedimiento—, ya que los primeros tres bloques del diagrama quedan “ligados” a partir de las restricciones definidas en el cuarto bloque.

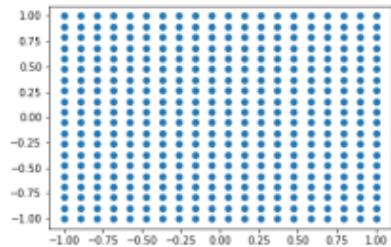
Uno de los puntos clave de *SciANN* es que al añadir capas de operaciones matemáticas a las salidas, estamos haciendo que estas operaciones sean la última capa de la red, esto es, las nuevas salidas. Así, la función de pérdida tomará esas operaciones en vez de las salidas de un elemento *Functional*.

7.2.2. Espacio de datos

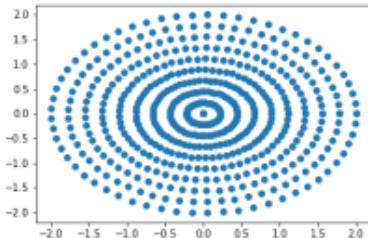
Como mencionábamos en el apartado anterior, las entradas de una PINN en *SciANN* definen las entradas de una función a aprender y se modelan mediante la clase *Variable*. En esta sección, vamos a entender qué tipo de entradas requiere una PINN.

A la hora de simular un fenómeno físico, debemos identificar los elementos que lo componen para estudiar cuál es el espacio que permite definir el fenómeno en su totalidad. Por ejemplo, los elementos a tener en cuenta cuando simulamos el calor transferido al ambiente en un disipador no serán los mismos que cuando queremos simular la distribución de cargas en una viga.

De forma general, definimos el espacio de entrada de una PINN, denotado por $S = (X, T)$, y en él las entradas como una distribución $S^* = (X^*, T^*) \subset S$ de puntos $(x^*, t^*) \in S^*$ distribuidos por el espacio a simular. Esto es, un mallado en el espacio cuya dimensionalidad, densidad, formato y otras cuestiones dependerán del fenómeno concreto a simular. En la Figura 7.2 se muestran algunos ejemplos concretos de mallados válidos para PINNs.



(a) Mallado uniforme centrado en $(0,0)$.



(b) Mallado radial centrado en $(0,0)$.

Figura 7.2: Ejemplos de mallado para PINNs.

7.2.3. Entrenamiento

Mientras que las PINNs originales proponían algoritmos de entrenamiento diferentes a la propagación hacia atrás, donde distintos conjuntos de datos (valores de entrada, funciones a modelar y sus restricciones) contribuían de forma independiente a la función de pérdida, la arquitectura de los modelos en

SciANN añade una capa *Lambda* que describe esas restricciones simbólicas, lo que nos permite dar una equivalencia secuencial y unificada del problema y lo hace interpretable por Keras.

A nivel formal, si queremos aprender una función u cuyas entradas pertenecen a un espacio S , estas restricciones se pueden representar mediante relaciones de tipo

$$G(u) = F(u)$$

donde G y F son funcionales de tipo diferencial que actúan sobre u . Como el objetivo es representar estas relaciones como términos de la función de pérdida a minimizar, suelen reescribirse de forma

$$H(u) = G(u) - F(u) = 0.$$

Así, la función de pérdida asociada al problema de aprendizaje resulta:

$$\mathcal{L}(\theta) = \sum_{i=1}^N \left(\sum_{S^*} \| H_i(u_\theta(x^*, t^*)) \| \right) + \sum_{S^*} \| u(x^*, t^*) - u_\theta(x^*, t^*) \| .$$

Por tanto, obtenemos una función de pérdida adecuada según los requerimientos de las PINNs mientras que aprovechamos las ventajas del entrenamiento de redes neuronales en Keras, entre ellas la paralelización y el mini-batch.

8 Experimentación

Una vez presentada la librería *SciANN*, en este capítulo se recoge el proceso de selección, diseño, implementación y discusión de problemas relacionados con fenómenos físicos a través de *SciANN*. El capítulo culmina con el desarrollo de un aproximador de operadores, cumpliendo así el último de los objetivos de este trabajo.

Para ello se realizarán tres experimentos que pretenden cubrir las principales tareas de aprendizaje a las que *SciANN* puede hacer frente. En el primer experimento vamos a trabajar directamente con funciones. El objetivo de este experimento es estudiar cómo responde la librería a representaciones matemáticas más o menos complejas y cómo de escalable es en términos de complejidad de arquitectura y de tamaño del conjunto de entrenamiento. En el segundo experimento vamos a hacer frente a un problema físico propiamente dicho, estudiando las restricciones que se generan y cómo influyen cada una de ellas en el proceso de aprendizaje. Por último, el tercer experimento consiste en la implementación de un aproximador para operadores. Este último experimento se propone como un estudio comparativo con la librería *DeepONet* [LJK19], la única librería que hasta el momento soporta esta tarea de aprendizaje. El código relativo a los experimentos puede consultarse en <https://github.com/vaatiper/TFG>.

8.1. Construcción del entorno de experimentación

La experimentación se ha realizado en Google Colab, un servicio en la nube que permite la ejecución de código Python en el entorno Jupyter Notebook. Entre los distintos recursos hardware ofrecidos por Google Colab, se ha utilizado la GPU de alto rendimiento NVIDIA Tesla T4, recomendada en la documentación de Google Cloud para tareas de Machine Learning y Ciencia de Datos. Entre sus características principales, mencionar que cuenta con:

- Memoria GPU GDDR6 de 16 GB con un ancho de banda de 320 GBps.
- 2560 núcleos CUDA para la paralelización del procesamiento.
- 320 Tensor Cores diseñados específicamente para acelerar operaciones de aprendizaje profundo.
- Consumo de energía de 70 W TDP (Thermal Design Power), bajo en comparación con otras GPUs de alto rendimiento.

Aunque *SciANN* esté implementado sobre Keras, durante el desarrollo de este proyecto se han detectado varias dificultades a la hora de integrar ambas librerías. En concreto, en lo que concierne a esta sección, se ha probado que no es posible integrar *SciANN* con Keras Tuner, herramienta indispensable

8 Experimentación

para el ajuste de hiperparámetros en Keras que se planificaba usar en el entorno de experimentación.

Por este motivo, para la experimentación se ha utilizado *Weights and Biases (W&B)*, una plataforma de software que facilita y automatiza el seguimiento, la gestión y la colaboración en proyectos de aprendizaje automático. Entre sus funcionalidades principales se encuentran el seguimiento de experimentos, la visualización de datos y la gestión de modelos. *W&B* es un software como servicio (SaaS) que ofrece integraciones con librerías de machine learning como TensorFlow, PyTorch y Keras.

En *W&B* el ajuste de hiperparámetros se realiza a través de *Sweeps*. Un *Sweep* viene definido por un archivo de configuración YAML en el que se define el espacio de búsqueda de hiperparámetros, la métrica objetivo y el método de búsqueda. Para su ejecución, una configuración *Sweep* se asocia a un proyecto, lo que permite acceder de forma sencilla a todas las experimentaciones realizadas para cada proyecto. Los proyectos creados en *W&B* para este trabajo pueden consultarse en <https://wandb.ai/vaatiper/projects>.

Como *SciANN* está implementado sobre Keras, al principio se probó a utilizar la integración de *W&B* y Keras mediante *Callbacks* pero, una vez más, *SciANN* no soportaba dicha integración, por lo que se optó por definir las métricas manualmente y loggear los resultados de cada ejecución. Aunque para esta experimentación se considera que los datos recogidos son suficientes, es necesario mencionar que *W&B* ofrece análisis mucho más completos para las librerías con integraciones soportadas.

8.2. Experimento 1: Problemas de Regresión

Este experimento tiene como objetivo mostrar la versatilidad de *SciANN*, que no se reduce a la implementación de PINNs si no que también abarca otras tareas como los problemas de regresión. Realizaremos ajustes en tres funciones distintas: seno, exponencial y un ejemplo de función bidimensional.

8.2.1. Ajuste de la función seno

Este experimento está inspirado en el ejemplo de ajuste de curvas que aparece en [HJ21]. En él, pretendemos aproximar la función seno en el intervalo $[0, 2\pi]$ mediante una red neuronal densa con un número variable de capas ocultas. Partiremos de la configuración de parámetros propuesta en [HJ21] y realizaremos modificaciones tanto en el conjunto de entrenamiento como en la arquitectura de la red, para estudiar cómo escala *SciANN* frente a modelos complejos o grandes volúmenes de datos.

8.2.1.1. Generación de datos sintéticos

Hemos utilizado la librería Numpy de Python para generar una partición del intervalo de definición del problema y para evaluar en esta partición tanto la función seno como su derivada. De acuerdo con los objetivos de este experimento, el conjunto de entrenamiento va a estar definido en $[0, 2\pi]$, mientras que el conjunto de validación lo estará en $[-2\pi, 2\pi]$. Además, contaremos con dos conjuntos

de entrenamiento, uno con una granularidad de 10.000 datos dentro de la partición y otro con 100 datos.

8.2.1.2. Construcción del modelo

```

1 #Definimos la estructura de la función que queremos aprender.
2 x = Variable('x')
3 y = Functional('y', x, [10, 10, 10], activation=['tanh', 'g-cos', 'l-sin'])
4
5 #Imponemos restricciones al modelo.
6 dy_dx = sn.diff(y, x)
7 c1 = Data(y)
8
9 #Definimos el modelo.
10 model = SciModel(x, [y, dy_dx], optimizer='adam')
```

Listing 8.1: Modelo en *SciANN* para el ajuste de la función seno.

Mostramos en el Listing 8.1 la estructura general que *SciANN* crea para abordar el problema de ajuste a una curva. Tal y como se explicó en el capítulo anterior, los elementos principales del modelo son:

- Variables de entrada: $\sin(x)$ es una función de una variable unidimensional, por lo que únicamente necesitaremos crear un objeto de tipo *Variable*.
- Funciones: Como queremos construir un aproximador para una única función, basta con utilizar un único objeto *Functional*, que definirá una red neuronal densa. Para ello, indicamos el nombre único que asignaremos a la función, sus entradas, el número de neuronas en cada capa profunda, y la función de activación que queremos considerar en cada capa.
- Restricciones: Cada una de las restricciones marca un objetivo a tener en cuenta durante el entrenamiento. Por un lado, en la línea 7 especificamos que el objetivo del modelo es aprender la función guardada en la variable *y*, y que vamos a hacerlo proporcionando datos. Por otro lado, en la línea 6, estamos indicando que el segundo objetivo del entrenamiento es respetar unos valores objetivo para la derivada de la función a aprender.
- Parámetros importantes del modelo: La elección parámetros para este experimento se ha realizado teniendo en cuenta los valores de referencia en del modelo proporcionado. Entre ellos destacan el número de épocas, tamaño de lote, la programación exponencial de la tasa de aprendizaje y las funciones de activación. Con respecto a las funciones de activación, en el esquema del modelo podemos ver que para la segunda y tercera capa hemos empleado unas funciones de activación especiales implementadas a través de la clase *SciRowdyActivation* de *SciANN*. Esta clase permite la suma de varias funciones de activación distintas, así como la incorporación de un parámetro α aplicable a la entrada antes de pasarla a la función. Si la función de activación especificada tiene el prefijo 'l-', α actúa de forma local y es independiente para cada neurona. En caso de tener el prefijo 'g-', α actúa de forma global y se comparte entre todas las neuronas de una misma capa.

Cuando las restricciones vienen únicamente impuestas por datos el modelo mantiene una estructura secuencial, como puede verse en la Figura 8.1.

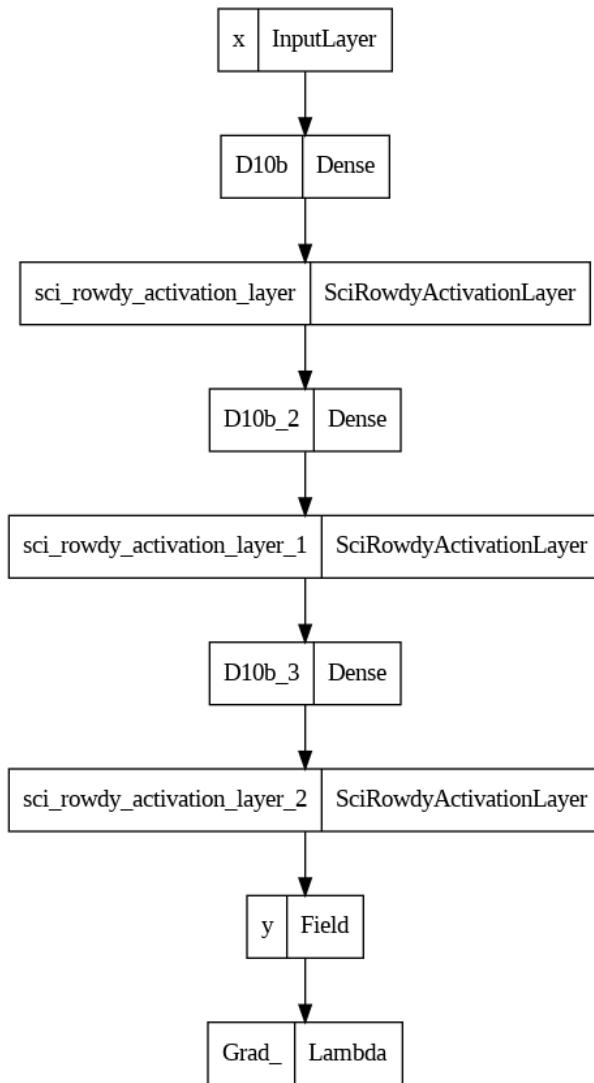


Figura 8.1: Esquema del modelo de *SciANN* para la curva $y = \sin(x)$.

8.2.1.3. Resultados

Uno de los objetivos de este experimento es medir cómo responden los predictores creados frente a distintos tamaños de conjunto de entrenamiento. Para ello, vamos a comenzar analizando los resultados obtenidos para un conjunto de entrenamiento con 10.000 elementos en el intervalo $[0, 2\pi]$. Posteriormente, se realizará un análisis similar escogiendo 100 datos en el mismo intervalo. La sección finalizará haciendo una comparación de los resultados obtenidos.

Para el conjunto de entrenamiento con 10.000 datos, tal y como se puede observar en la [Figura 8.2](#), todos los modelos consiguen un error de predicción en el conjunto de entrenamiento cercano a cero, dando mejores resultados cuando los modelos no son demasiados simples. Estos resultados no se mantienen en validación. Aun así, comparando la [Figura 8.2](#) con la [Figura 8.3](#), se puede observar una correspondencia entre los modelos que dan mejores resultados en *loss* y los que los dan en *val_loss*. No es una correspondencia en magnitud, pero sí lo es a la hora de establecer una jerarquía entre los mejores modelos.

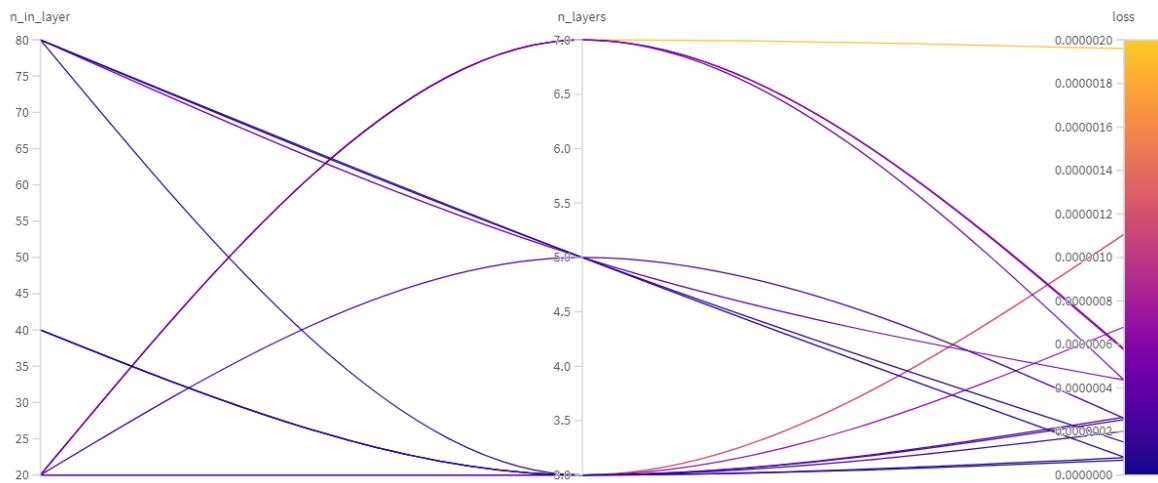


Figura 8.2: Desglose del ajuste de hiperparámetros para para la curva $y = \sin(x)$ del experimento 8.1 con conjunto de entrenamiento ampliado bajo la métrica *loss*.

Al hacer una representación gráfica de las predicciones para el modelo con los hiperparámetros especificados en el [Listing 8.1](#), podemos dar explicación a los resultados obtenidos. En la [Figura 8.24](#) se muestra el ajuste en entrenamiento mientras que en [Figura 8.5](#) se muestra la predicción para el conjunto de test. En ambos casos, se muestra en negro la función original y en rojo la predicción. Podemos ver cómo las gráficas originales se solapan con la predicción en el intervalo $[0, 2\pi]$ y el error de validación viene, por tanto, de que el ajuste realizado es local: el modelo no es capaz de generalizar bien a datos de entrada fuera del intervalo para el que hemos aprendido a predecir.

El experimento continúa haciendo una búsqueda en el mismo espacio de parámetros, esta vez entrenando con un conjunto de 100 datos en la partición de $[0, 2\pi]$. En la [Figura 8.6](#) podemos ver como las predicciones en entrenamiento han empeorado considerablemente. Al realizar una comparación con los

8 Experimentación

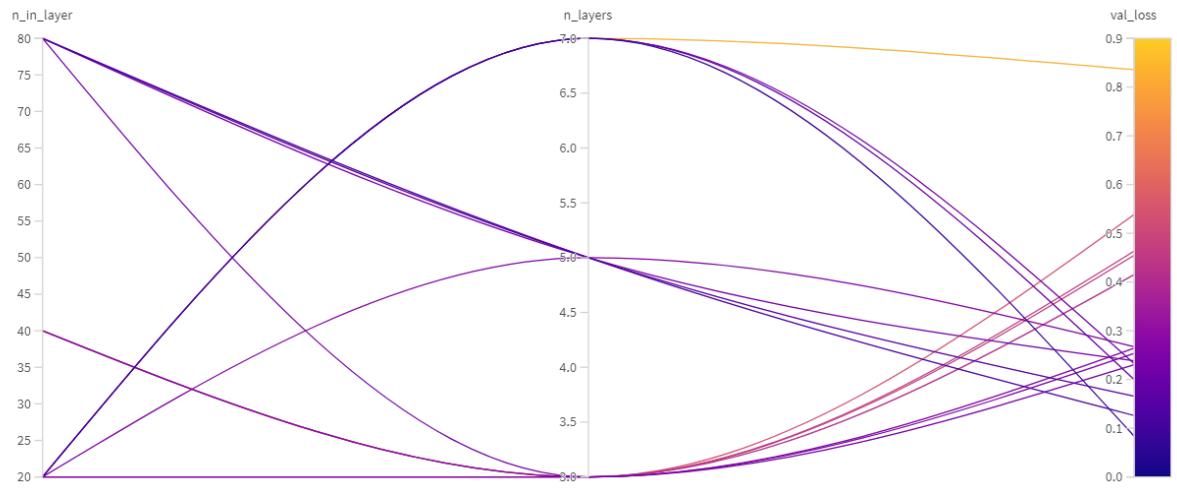


Figura 8.3: Desglose del ajuste de hiperparámetros para la curva $y = \sin(x)$ del experimento 8.1. con conjunto de entrenamiento ampliado bajo la métrica *val_loss*.

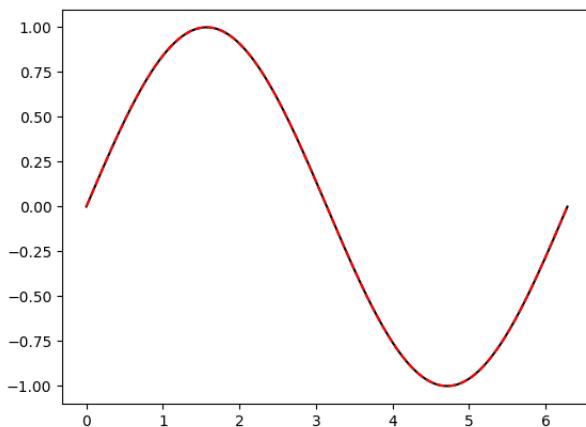


Figura 8.4: Ajuste de la curva $y = \sin(x)$ en el intervalo $[0, 2\pi]$, donde la función original aparece en negro y la predicción en rojo.

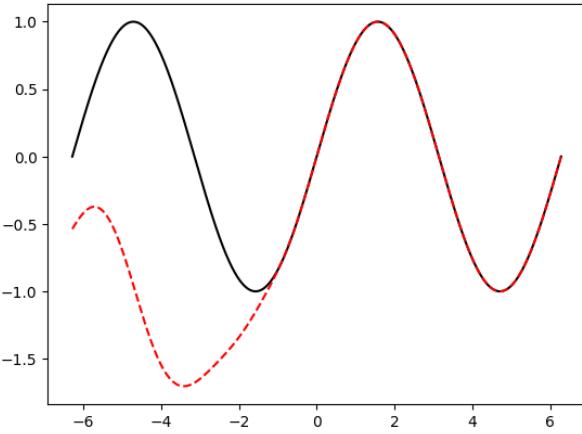


Figura 8.5: Predicción para la curva $y = \sin(x)$ en el intervalo $[-2\pi, 2\pi]$, donde la función original aparece en negro y la predicción en rojo.

resultados de validación, en la Figura 8.7 vemos que, en contraste con los resultados en entrenamiento, el modelo generaliza igual de bien que el entrenado con más datos. Podríamos deducir, por tanto, que al dar al modelo tantos datos parecidos, creamos en el primer caso una situación de sobre-ajuste a dicho conjunto de entrenamiento.

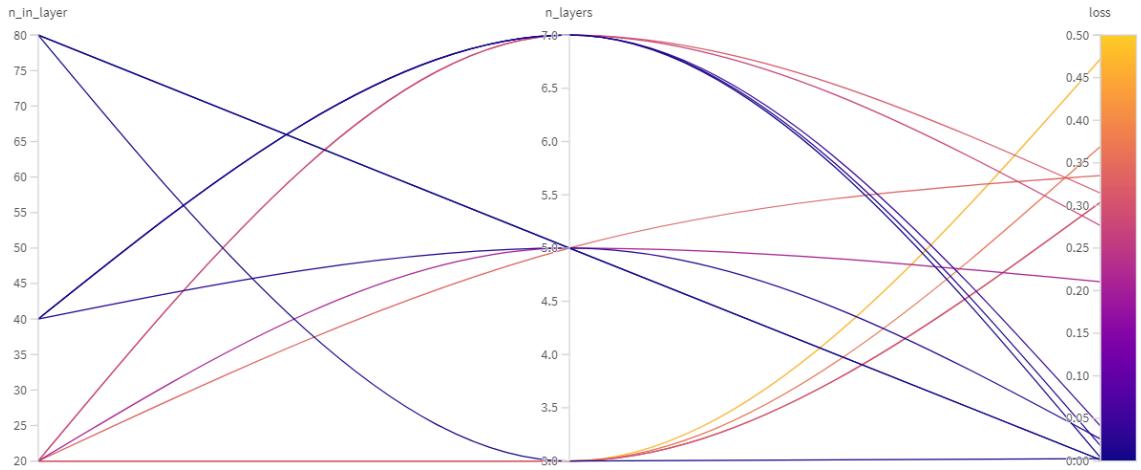


Figura 8.6: Desglose del ajuste de hiperparámetros para para la curva $y = \sin(x)$ del experimento 8.1. con conjunto de entrenamiento reducido bajo la métrica *loss*.

Continuamos realizando un estudio de la robustez de estos modelos frente a ruido. Para ello, se ha introducido ruido en el conjunto de entrenamiento de 10.000 datos. En la Figura 8.8 y la Figura 8.9 podemos apreciar que los valores de error en *loss* y en *val_loss* son muy similares a los obtenidos para el entrenamiento sin ruido. Esto indica que el modelo ajusta igual de bien en ambos casos y corrobora que, en ambos modelos, el error en validación se debe al dominio de definición de la función para cada

8 Experimentación

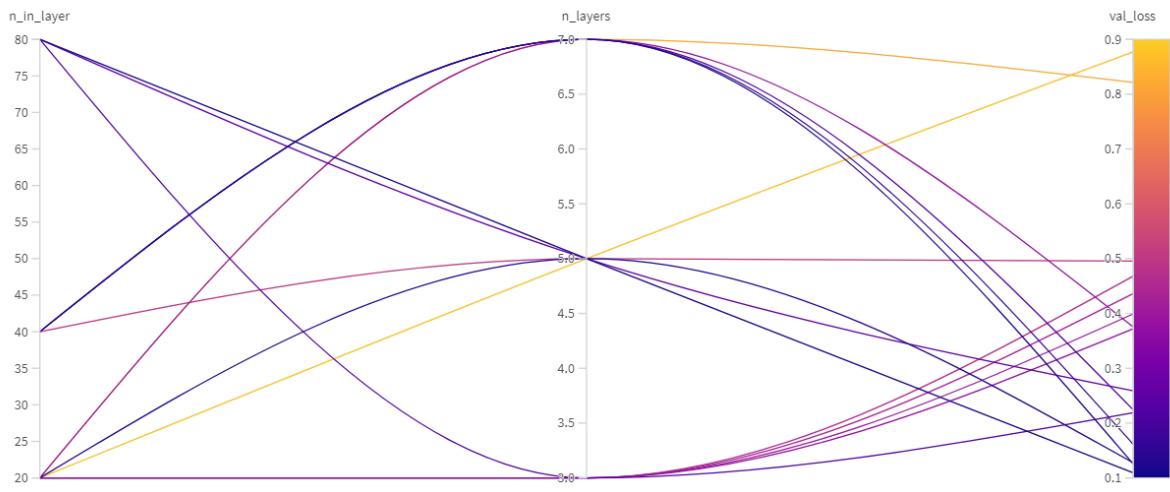


Figura 8.7: Desglose del ajuste de hiperparámetros para la curva $y = \sin(x)$ del experimento 8.1. con conjunto de entrenamiento reducido bajo la métrica val_loss .

conjunto de entrenamiento.

Para evaluar cómo escalan los modelos vamos a realizar un estudio de las experimentaciones propuestas con respecto a su tiempo de ejecución. Los experimentos lanzados para el mayor conjunto de entrenamiento han tenido unos tiempos de ejecución en el rango de [350, 1050] segundos. Dentro de este rango, en la Figura 8.10 podemos ver cómo el hiperparámetro más penalizado ha sido contar con un número de capas ocultas alto. Los resultados para la ejecución con ambos conjuntos de entrenamiento, mostrados en la Figura 8.11 y la Figura 8.12, corroboran esta hipótesis.

Por último, en la Figura 8.13 mostramos un desglose de la evolución del error por cada término de la función de pérdida. Podemos observar que el modelo aprende más rápido de los datos propios de regresión que de los datos inferidos a través de la relación de diferenciación.

Con todo esto, podemos concluir que *SciANN* es capaz de completar satisfactoriamente tareas sencillas de regresión dentro del intervalo de definición estipulado en el entrenamiento. Además, estas tareas son sensibles tanto al número de capas profundas del modelo seleccionado como al tamaño del conjunto de entrenamiento.

8.2.2. Ajuste de la función exponencial

Este segundo experimento pretende reflejar la capacidad de *SciANN* para ajustar funciones en un caso para el que no disponemos de tanta información sobre los parámetros óptimos para el aprendizaje. Puesto que el objetivo es el mismo que en el experimento anterior, la estructura del modelo de *SciANN* se conserva, como puede observarse en el Listing 8.2.

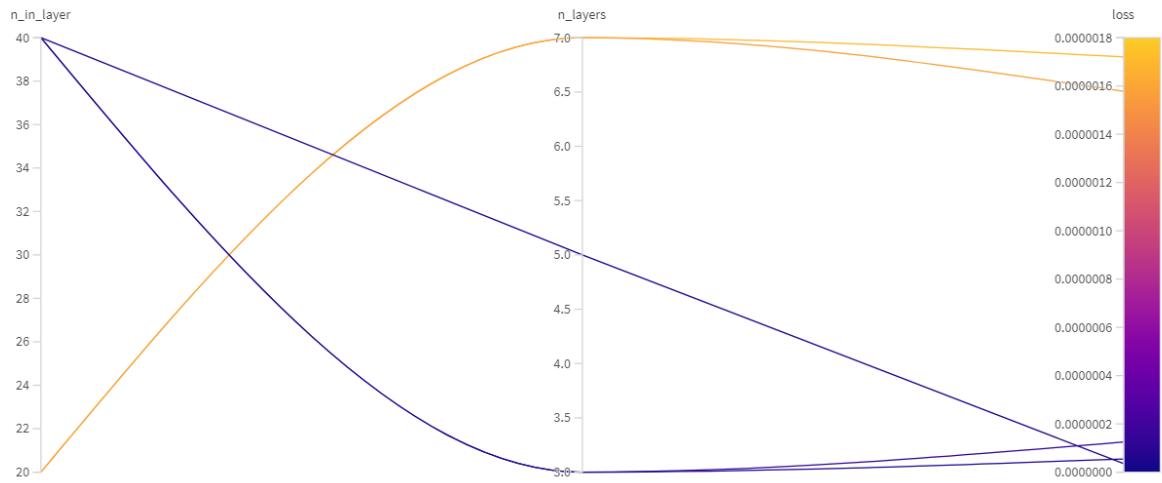


Figura 8.8: Desglose del ajuste de hiperparámetros para la curva $y = \sin(x)$ frente a ruido bajo la métrica $loss$.

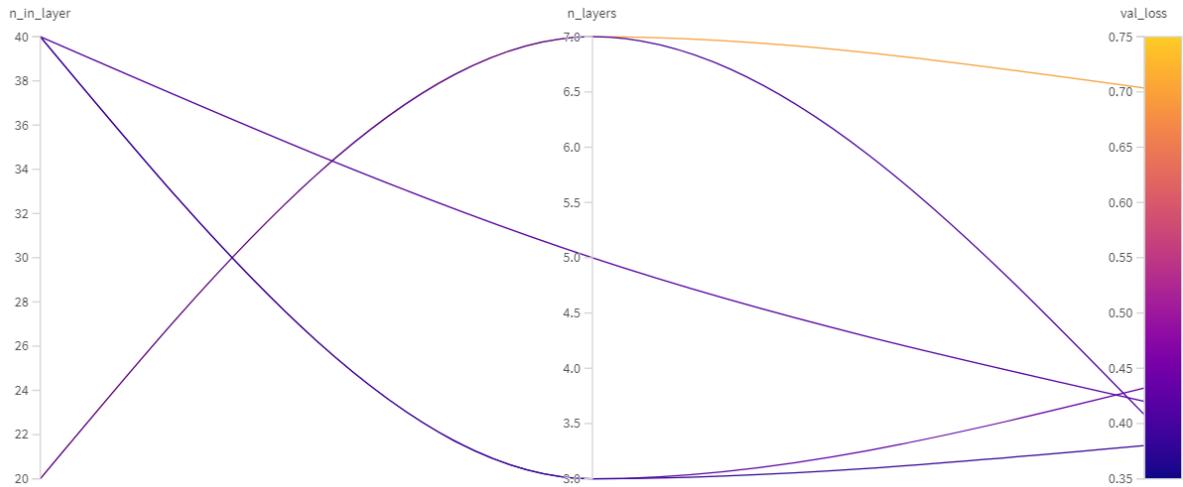


Figura 8.9: Desglose del ajuste de hiperparámetros para la curva $y = \sin(x)$ frente a ruido. bajo la métrica val_loss .

8 Experimentación

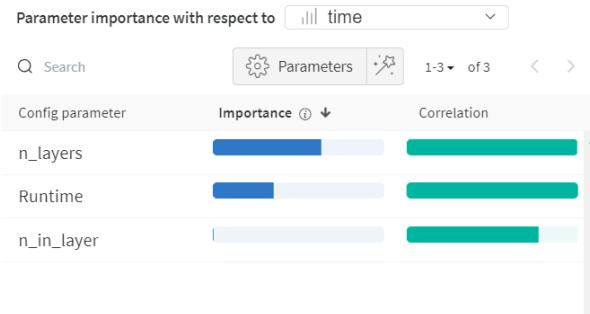


Figura 8.10: Ranking de parámetros correlacionados con la métrica $time$ para la curva $y = \sin(x)$ del experimento 8.1.

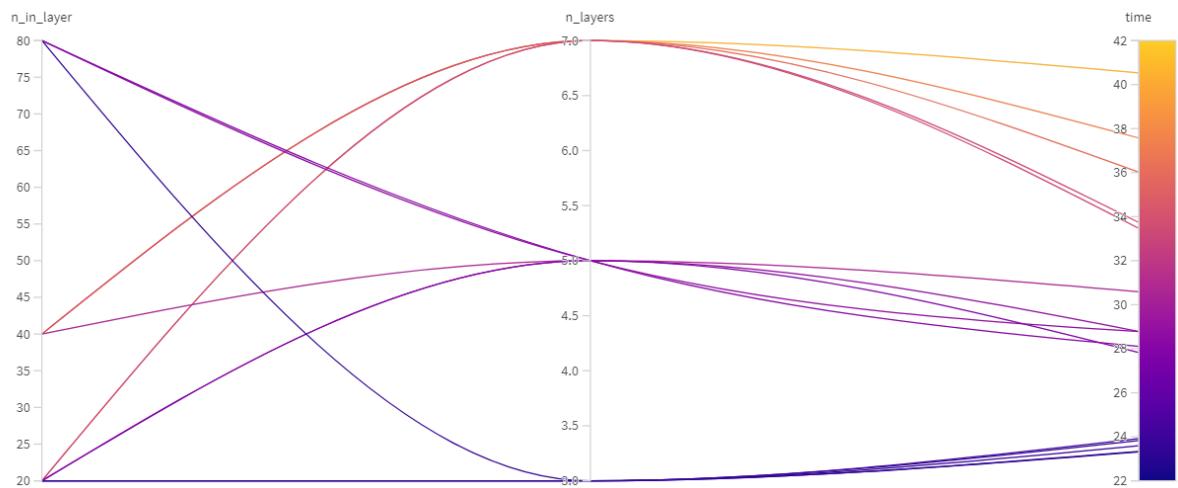


Figura 8.11: Desglose del ajuste de hiperparámetros para la curva $y = \sin(x)$ del experimento 8.1. con conjunto de entrenamiento reducido bajo la métrica $time$.

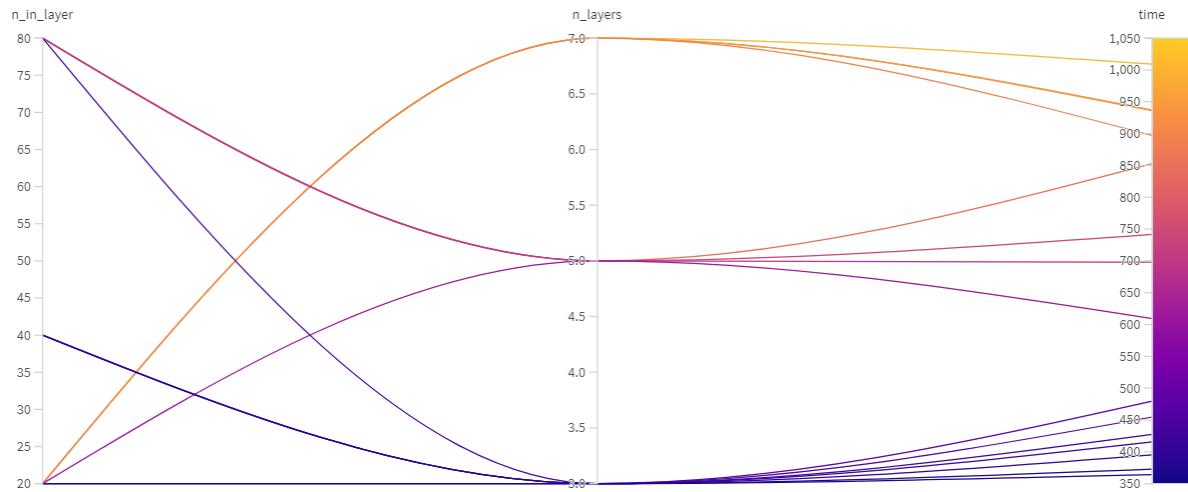


Figura 8.12: Desglose del ajuste de hiperparámetros para la curva $y = \sin(x)$ del experimento 8.1. con conjunto de entrenamiento ampliado bajo la métrica *time*.

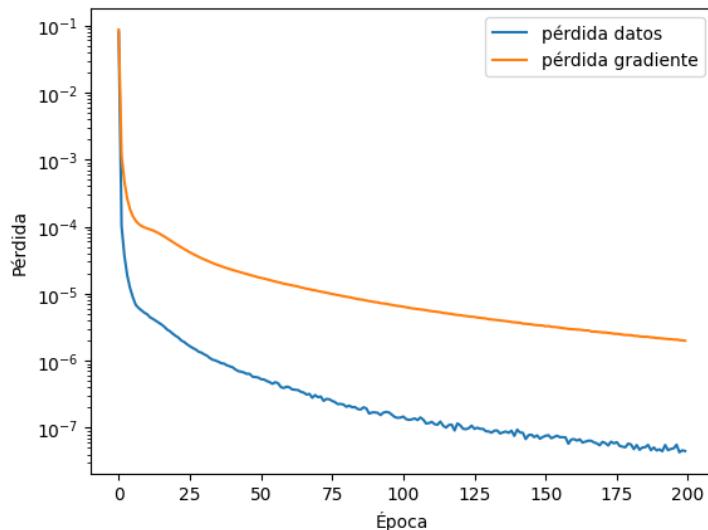


Figura 8.13: Evolución de los términos de la función de pérdida durante el entrenamiento para la curva $y = \sin(x)$.

8 Experimentación

Como desconocemos la complejidad del modelo necesario para hacer un buen ajuste, se ha experimentado con la densidad de la red neuronal asociada a la función exponencial, así como con sus funciones de activación. Además de esto, se ha hecho un ajuste de los hiperparámetros más relevantes de toda tarea de entrenamiento en aprendizaje profundo: la tasa de aprendizaje y el tamaño de lote. Con respecto a la evolución del entrenamiento, se ha optado por utilizar el procedimiento de parada anticipada para garantizar un número de épocas adecuado.

```
1 #Definimos la estructura de la función que queremos aprender.
2 x = Variable('x')
3 y = Functional('y', x, 5*[20], activation=5*['tanh'])
4
5 #Imponemos restricciones al modelo.
6 dy_dx = sn.diff(y, x)
7 c1 = Data(y)
8
9 #Definimos el modelo.
10 model = SciModel(x, [y, dy_dx], optimizer='adam')
11
12 start_time = time.time()
13
14 #Entrenamiento.
15 entrenamiento_exp = model.train(x_true,
16                                 [y_true, dy_true],
17                                 epochs=10000,
18                                 learning_rate=0.01,
19                                 batch_size=128,
20                                 callbacks=[keras.callbacks.EarlyStopping(monitor="loss",
21                                                               min_delta = 0, patience=20, verbose=1)])
22 )
```

Listing 8.2: Ejemplo de modelo en *SciANN* para el ajuste de la función exponencial.

8.2.2.1. Resultados

En la [Figura 8.15](#) podemos observar cómo el modelo que minimiza la pérdida en validación cuenta con cinco capas ocultas con 20 neuronas y activación de tangente hiperbólica. En general, la experimentación refleja cómo modelos más complejos (con muchas capas ocultas o muchas neuronas por capa) son más propensos a responder mal ante la tarea. Además, el ranking proporcionado por *W&B* de la [Figura 8.16](#) muestra que el único hiperparámetro que parece estar correlacionado con menores valores de función de pérdida es la función de activación sigmoidal. En la [Figura 8.14](#) podemos ver cómo el modelo seleccionado en validación también es uno de los que mejor se ajusta al conjunto de entrenamiento.

En cuanto a los tiempos de ejecución del entrenamiento, en la [Figura 8.17](#) podemos ver que quedan dentro del intervalo de [7, 100] segundos. A simple vista es difícil encontrar una correlación directa entre alguno de los hiperparámetros y el tiempo de entrenamiento, pero a través del ranking proporcionado por *W&B*, que se muestra en la [Figura 8.19](#), se ha detectado que la función de activación sigmoidal está correlacionada con mayores tiempos de entrenamiento. Sin embargo, una visualización más clara de la correlación entre las funciones de activación y la métrica *time*, que se muestra en la [Figura 8.18](#), deja ver que la relación encontrada se debe a que, en la búsqueda aleatoria de configuraciones, se han

8.2 Experimento 1: Problemas de Regresión

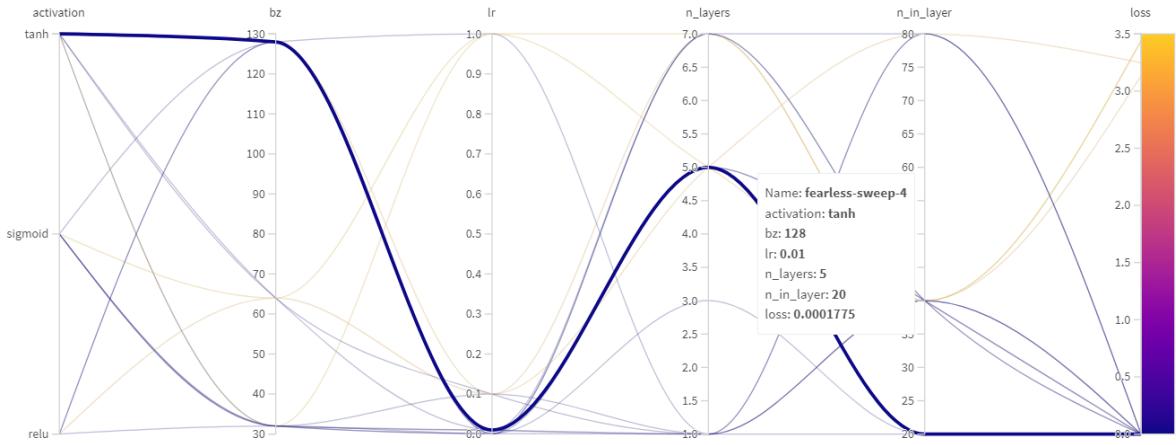


Figura 8.14: Desglose del ajuste de hiperparámetros para la curva $y = \exp(x)$ del experimento 8.1. bajo la métrica *loss*.

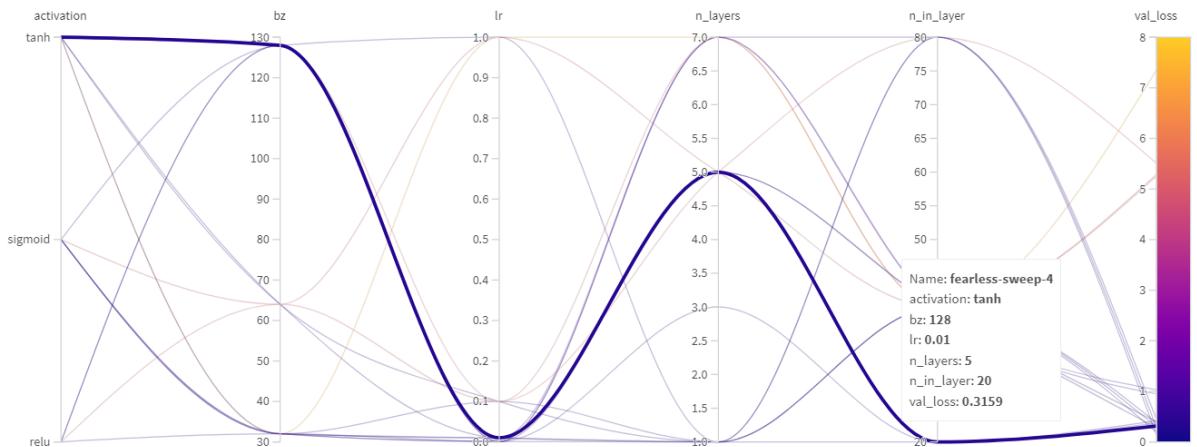


Figura 8.15: Desglose del ajuste de hiperparámetros para la curva $y = \exp(x)$ del experimento 8.1. bajo la métrica *val_loss*.

8 Experimentación

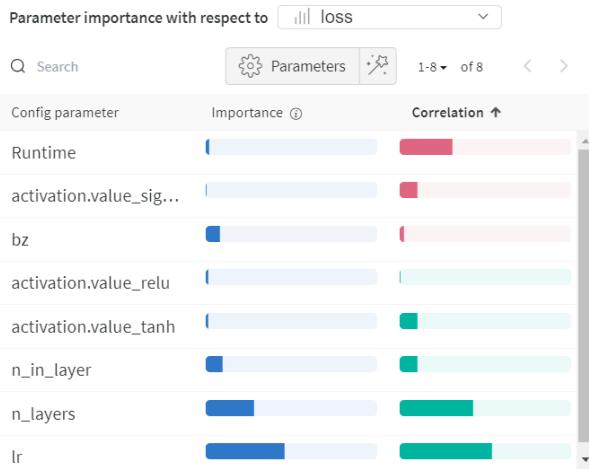


Figura 8.16: Ranking de parámetros correlacionados con la métrica $loss$ para la curva $y = \exp(x)$ del experimento 8.1.

dado varias configuraciones en las que la función sigmoidal iba asociada a tasas de aprendizaje muy pequeñas junto con modelos complejos, con muchas capas densas. Es por esto que el ranking nos da esa información.

Procedemos por tanto a realizar un análisis más detallado de la evolución del entrenamiento para el modelo que ha minimizado la pérdida en validación. De manera análoga al experimento anterior, en la Figura 8.20 podemos ver cómo la predicción es buena únicamente dentro del intervalo de entrenamiento. Sin embargo, en contraposición con el mismo, la Figura 8.21 muestra que en este caso el entrenamiento se ha beneficiado por igual de la pérdida inducida por los datos de la función que de la inducida por los datos del gradiente.

8.2.3. Ajuste bidimensional

8.2.3.1. Generación de datos sintéticos

Para este problema se ha creado un mallado en Numpy con una granularidad de 100×100 datos. En la figura Figura 8.22 se muestra la función evaluada en el dominio $[0, \pi] \times [0, \pi]$.

8.2.3.2. Construcción del modelo

El modelo, que queda detallado en el Listing 8.3, es similar a los descritos anteriormente. Como tenemos dos variables de entrada, ambas unidimensionales, hemos definido un total de dos objetos *Variable* para el modelo. Además, al tratarse de un problema algo más complejo, hemos aumentado el número de

8.2 Experimento 1: Problemas de Regresión

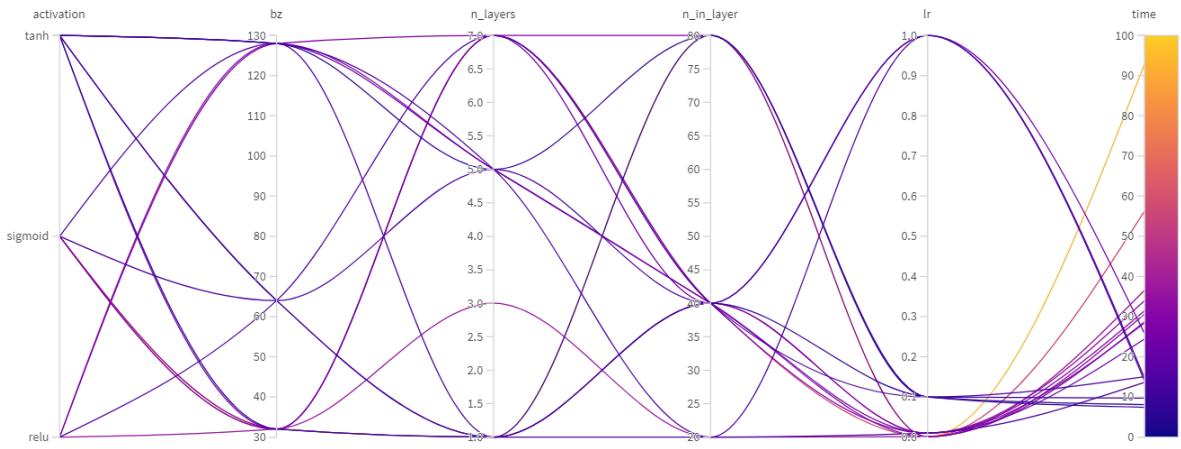


Figura 8.17: Desglose del ajuste de hiperparámetros para la curva $y = \exp(x)$ del experimento 8.1. bajo la métrica *time*.

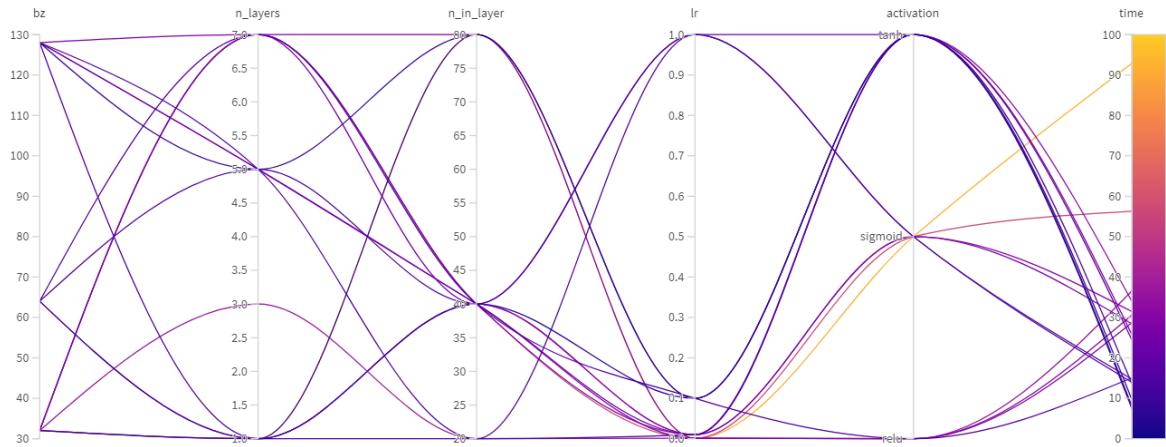


Figura 8.18: Segundo desglose del ajuste de hiperparámetros para la curva $y = \exp(x)$ del experimento 8.1. bajo la métrica *time*.

8 Experimentación

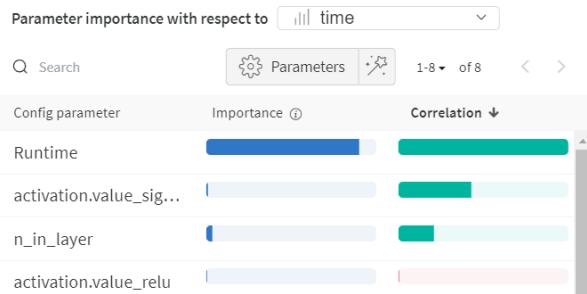


Figura 8.19: Ranking de parámetros correlacionados con la métrica $time$ para la curva $y = \exp(x)$ del experimento 8.1.

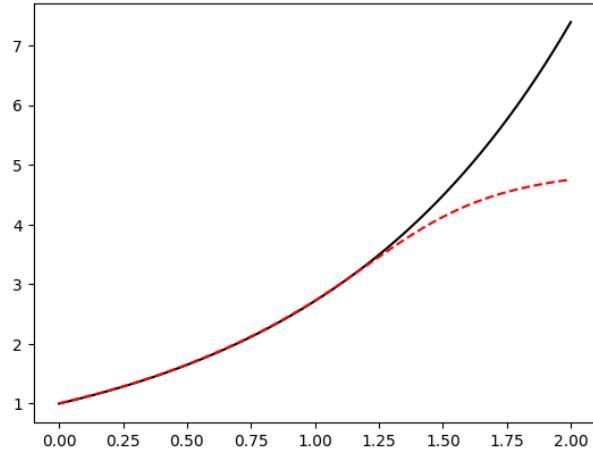


Figura 8.20: Predicción para la curva $y = \exp(x)$ en el conjunto de validación, donde dibujamos en negro el valor real de la función y en rojo la predicción.

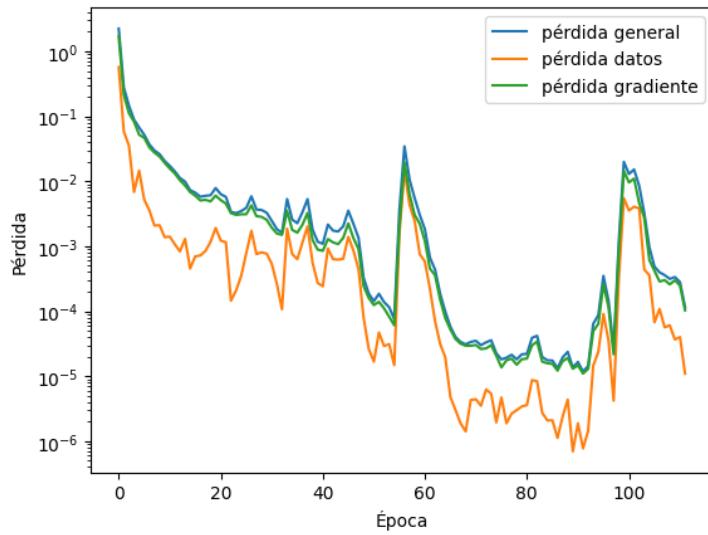


Figura 8.21: Evolución de los términos de la función de pérdida durante el entrenamiento del modelo especificado en el Listing 8.2 para la curva $y = \exp(x)$.

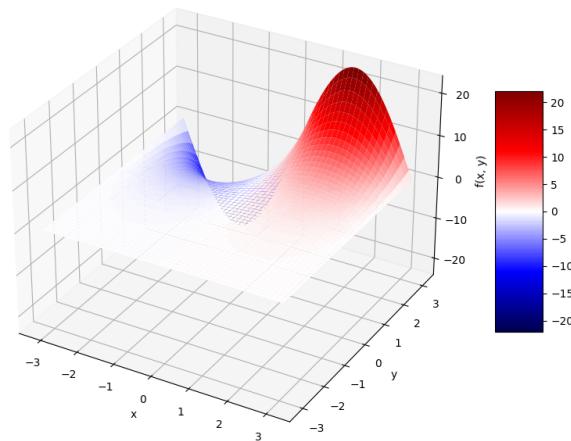


Figura 8.22: Ajuste de la curva $z = \sin(x)e^y$ en el intervalo $[0, \pi] \times [0, \pi]$.

8 Experimentación

capas densas.

```
1 #Definición de las entradas.
2 x = sn.Variable('x')
3 y = sn.Variable('y')
4
5 #Definición de la función.
6 f = sn.Functional('f', [x, y], [10, 10, 10, 10], 'tanh')
7
8 #Imposición de restricciones.
9 df_dx = sn.diff(f, x)
10 df_dy = sn.diff(f, y)
11
12 d1 = sn.Data(f)
13
14 #Definición del modelo.
15 modelo_2d = sn.SciModel([x,y],[f, df_dx,df_dy])
```

Listing 8.3: Modelo en *SciANN* para ajuste bidimensional.

En la [Figura 8.23](#) podemos ver cómo se mantiene la estructura secuencial debido a que, una vez más, únicamente hemos impuesto restricciones que vengan únicamente a partir de datos explícitos.

8.2.3.3. Resultados

Como podemos ver en la [Figura 8.24](#), la aproximación solo es buena localmente, en el dominio de entrenamiento. En la evolución del entrenamiento (recogida en [Figura 8.25](#)) se puede observar como, al igual que en el primer experimento, el aprendizaje viene sobre todo de los datos explícitos de la función. En cuanto al tiempo de entrenamiento, la media ha sido de 532,631 segundos.

8.3. Experimento 2: El problema de Burgers

Este experimento tiene como objetivos exponer los conocimientos adquiridos en referencia al modelado de PINNs y estudiar cómo afecta la complejidad del modelo a la eficiencia de la tarea de aprendizaje en *SciANN*.

8.3.1. Descripción del problema

El problema de Burgers describe los fenómenos de advección y difusión que modelan, entre otros, la evolución de la velocidad en un fluido o la densidad del tráfico. Se utiliza ampliamente en la dinámica de fluidos y la teoría del tráfico. En 2018 se abrió la puerta a su modelado mediante el uso de redes neuronales profundas [\[Rai18\]](#). El modelo se rige por la ecuación:

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$

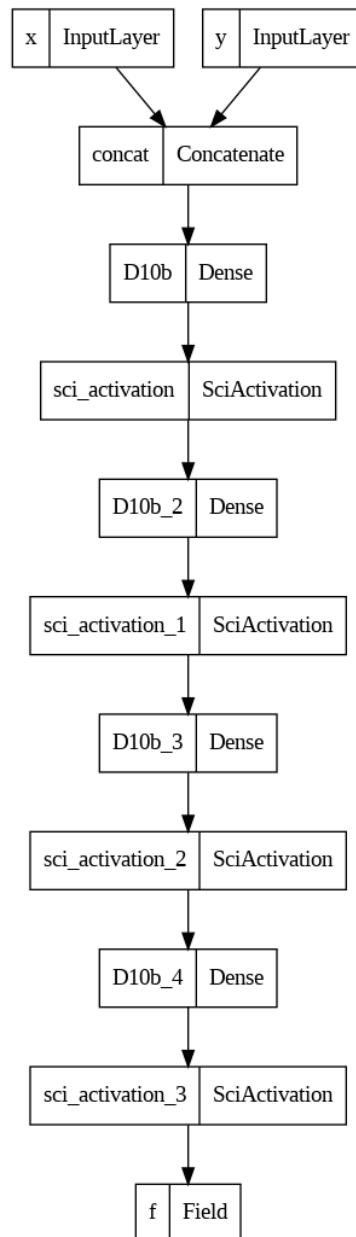


Figura 8.23: Esquema de red neuronal para la curva $z = \sin(x)e^y$.

8 Experimentación

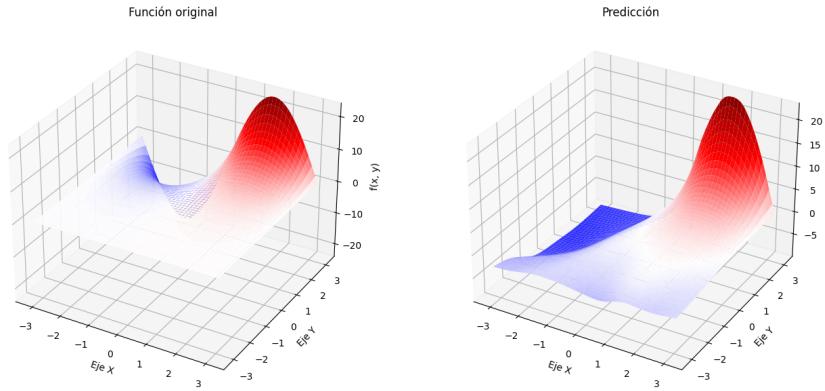


Figura 8.24: Comparación de la función original con la predicción de *SciANN*.

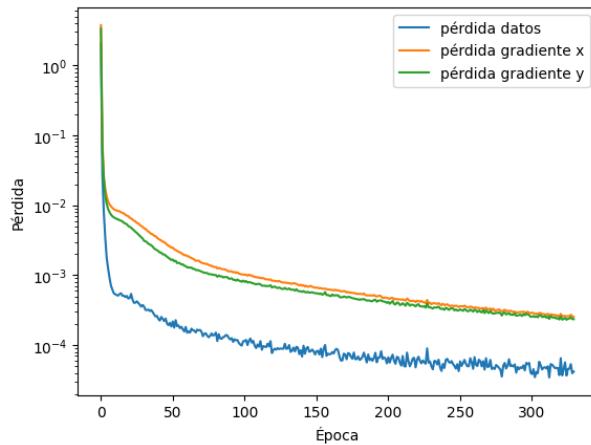


Figura 8.25: Evolución de los términos de la función de pérdida durante el entrenamiento para la curva $z = \sin(x)e^y$.

8.3 Experimento 2: El problema de Burgers

donde ν es un parámetro que representa la viscosidad cinemática y $u(t, x)$ representa la velocidad. El término de advección $u \frac{\partial u}{\partial x}$ describe el transporte de una sustancia o propiedad (como el calor o la humedad) a través de un fluido en movimiento e introduce no-linealidad en este problema. Representa cómo la velocidad del fluido en un punto afecta al cambio de velocidad en el espacio. Reproduciendo el experimento descrito en [Rai18], vamos a trabajar con una versión unidimensional del problema

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \left(\frac{0.01}{\pi}\right) \frac{\partial^2 u}{\partial x^2}, \quad t \in [0, 1], \quad x \in [-1, 1]$$

sujeta a las condiciones iniciales y de contorno

$$u(0, x) = -\sin(\pi x), \quad u(t, 1) = 0, \quad u(t, -1) = 0.$$

8.3.2. Generación de datos sintéticos

Siguiendo la estructura de entrada que *SciANN* requiere, es necesario construir un mallado del espacio de entrada $[0, 1] \times [-1, 1]$. Como los objetivos del entrenamiento no son ajustarnos a unos valores de salida, si no cumplir con una serie de restricciones, en este experimento no generaremos datos explícitos de salida para $u(t, x)$.

8.3.3. Construcción del modelo

Nuestro objetivo es, por tanto, construir un aproximador $\hat{u}(t, x)$ de $u(t, x)$ definido en el dominio $[0, 1] \times [-1, 1]$.

```

1 #Definición de entradas.
2 x = sn.Variable('x')
3 t = sn.Variable('t')
4
5 #Definición de la función a aprender.
6 u = sn.Functional('u', [t,x], 8*[20], 'tanh')
7
8 #Definición de las restricciones del modelo.
9 L1 = diff(u, t) + u*diff(u,x) - (0.01/pi)*diff(u, x, order=2)
10 TOL = 0.001
11 C1 = (1-sign(t - TOL)) * (u + sin(pi*x))
12 C2 = (1-sign(x - (-1+TOL))) * (u)
13 C3 = (1+sign(x - ( 1-TOL))) * (u)
14
15 #Definición del modelo.
16 burgers_model = sn.SciModel([x, t], [L1, C1, C2, C3])

```

Listing 8.4: Modelo en *SciANN* para el problema de Burgers.

Como podemos observar en el Listing 8.4, los elementos principales utilizados para la construcción de $\hat{u}(t, x)$ son:

- Variables de entrada: Definimos un objeto ‘t’ de tipo *Variable* para la variable temporal y un único objeto ‘x’ de tipo *Variable* para la variable espacial unidimensional.
- Funciones: Nuestro objetivo es aprender una única función, por lo que definimos un único objeto *Functional*. Como esta tarea de aprendizaje es considerablemente más compleja que las anteriores, contamos con un mayor número de capas. Además, hemos seleccionado la tangente hiperbólica como función de activación ya que tiene una media más cercana a 0 que otras funciones de activación, lo que permite preservar la varianza entre las entradas y salidas de cada capa y, por tanto, ayuda a la tarea de aprendizaje haciendo que los gradientes no se aproximen demasiado a 0 en las capas internas.
- Restricciones: Definimos tanto el PDE asociado (L_1) como las condiciones iniciales y de contorno especificadas (C_1, C_2, C_3). En vez de imponer de forma estricta estas condiciones, damos un margen de tolerancia de 0.001. Esta decisión ha sido tomada con el propósito de facilitar la tarea de aprendizaje y acelerar así la convergencia del modelo.

Observemos como, en contraposición con el experimento anterior, el objetivo del modelo es únicamente aproximar, es decir, no es necesario proporcionar de forma explícita evaluaciones discretas de $u(t, x)$: el cumplimiento de las restricciones definidas para cada entrada (t, x) nos permitirá predecir la función $u(t, x)$. Esta configuración induce una estructura no secuencial en el modelo, como muestra la [Figura 8.26](#).

8.3.4. Resultados

Los resultados obtenidos tras la experimentación (la [Figura 8.27](#) muestra los valores de pérdida y la [Figura 8.28](#) muestra los tiempos de entrenamiento) son bastante uniformes y no arrojan mucha luz. Además, se puede observar que el ajuste no es demasiado bueno, habiéndose encontrado otras implementaciones en [\[HJ21\]](#) que han dado resultados con pérdidas del orden de 10^{-6} . Tras una experimentación manual y visualizar la función de pérdida para el modelo descrito en el [Listing 8.4](#), en la [Figura 8.31](#) podemos observar cómo el aprendizaje es bastante errático, lo que hace que el método de parada anticipada no haya funcionado bien en nuestra experimentación.

Procedemos, por tanto, a repetir la experimentación para un número de épocas fijo lo suficientemente grande. Los resultados mostrados en la [Figura 8.29](#) y la [Figura 8.30](#) corresponden al entrenamiento con un mallado de 1000 datos y 3000 épocas completas. Como podemos ver, una vez más, el hiperparámetro más importante en términos de eficiencia vuelve a ser el número de capas ocultas. Además, en la [Figura 8.30](#) se aprecia que el ajuste es mejor para modelos más simples.

A la vista de los resultados obtenidos, sigue siendo difícil justificar hasta qué punto estos tiempos de ejecución se deben a la complejidad del modelo o a la naturaleza del problema. Por ello, se ha realizado un estudio comparativo de los modelos seleccionados en el [Listing 8.1](#) y [Listing 8.4](#), por ofrecer valores de pérdida equivalentes en sus respectivas tareas de aprendizaje. Lo primero que nos llama la atención tras una comparación directa entre el tiempo de ejecución por época señala que cada época del [Listing 8.4](#) triplica los tiempos del [Listing 8.1](#). Como vimos en la [Sección 8.2](#), el entrenamiento

8.3 Experimento 2: El problema de Burgers

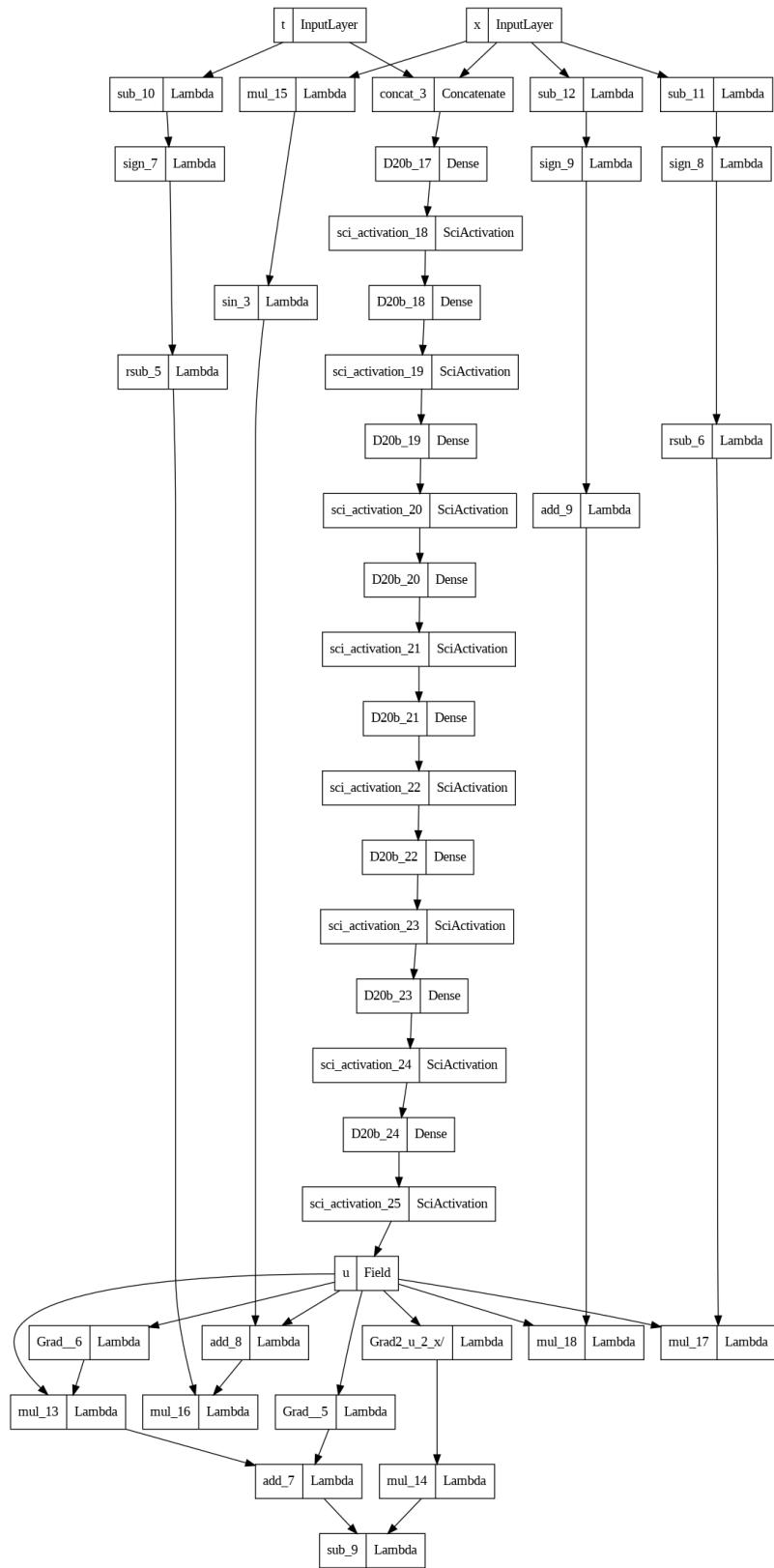


Figura 8.26: Representación interna en *SciANN* para el problema de Burgers.

8 Experimentación

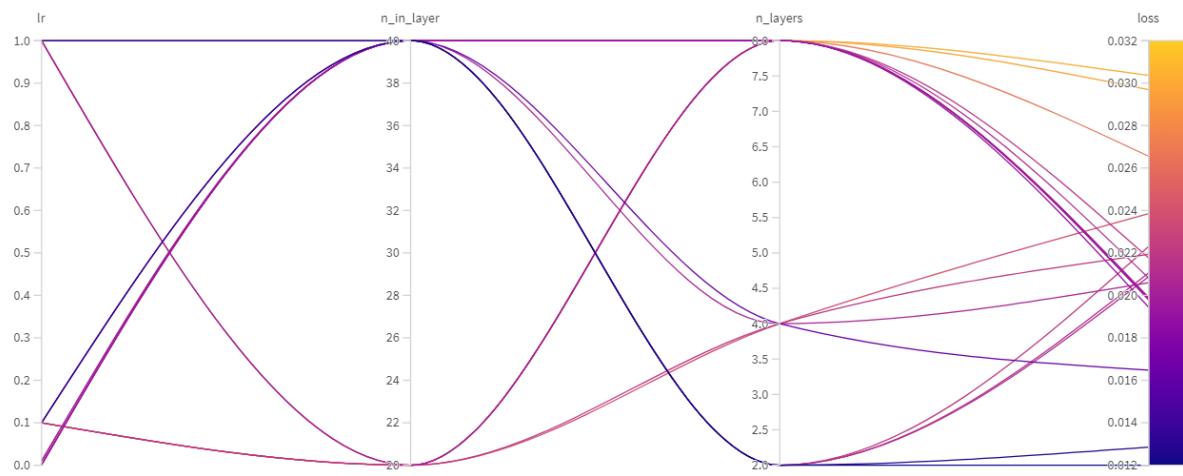


Figura 8.27: Desglose del ajuste de hiperparámetros para el problema de Burgers con parada anticipada bajo la métrica $loss$.

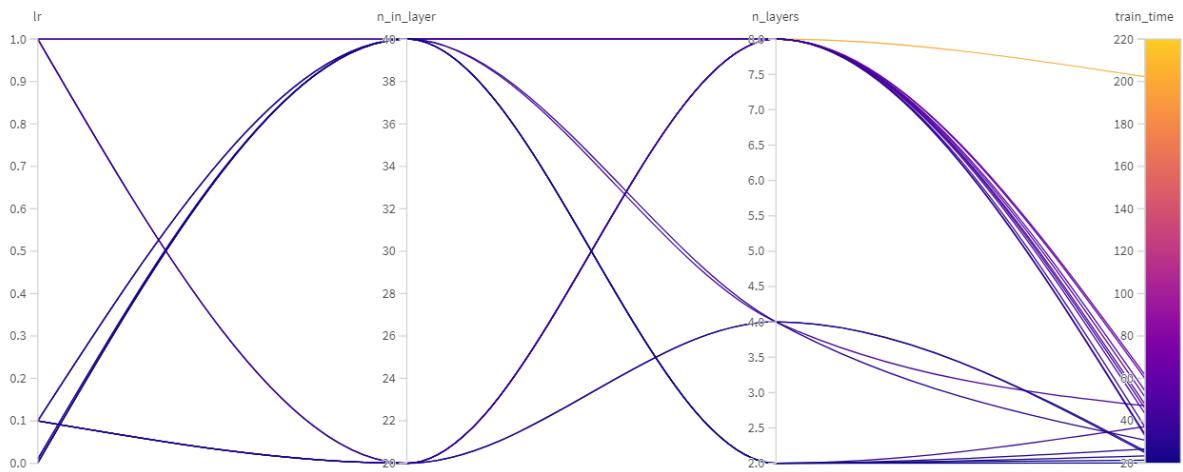


Figura 8.28: Desglose del ajuste de hiperparámetros para el problema de Burgers con parada anticipada bajo la métrica $train_time$.

8.3 Experimento 2: El problema de Burgers

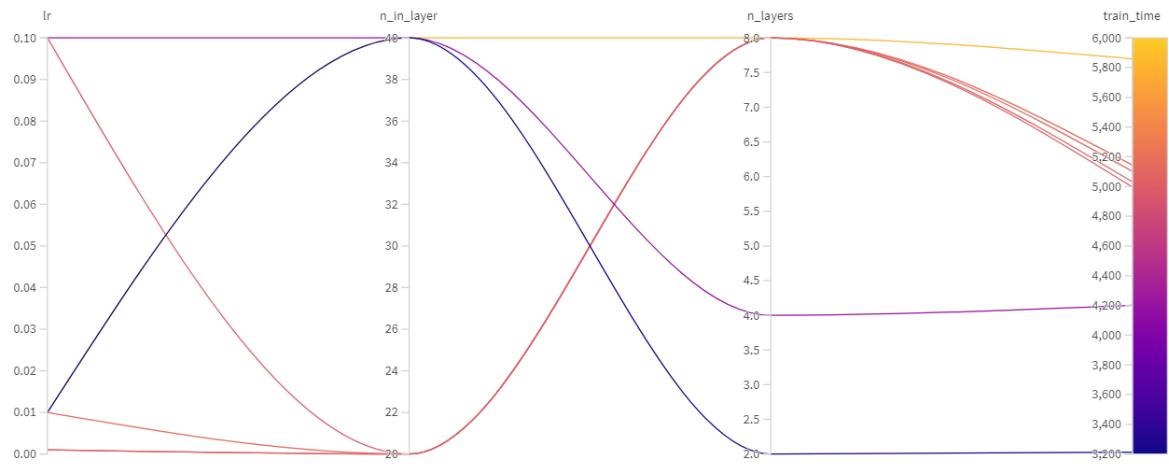


Figura 8.29: Desglose del ajuste de hiperparámetros para el problema de Burgers sin parada anticipada bajo la métrica *train_time*.

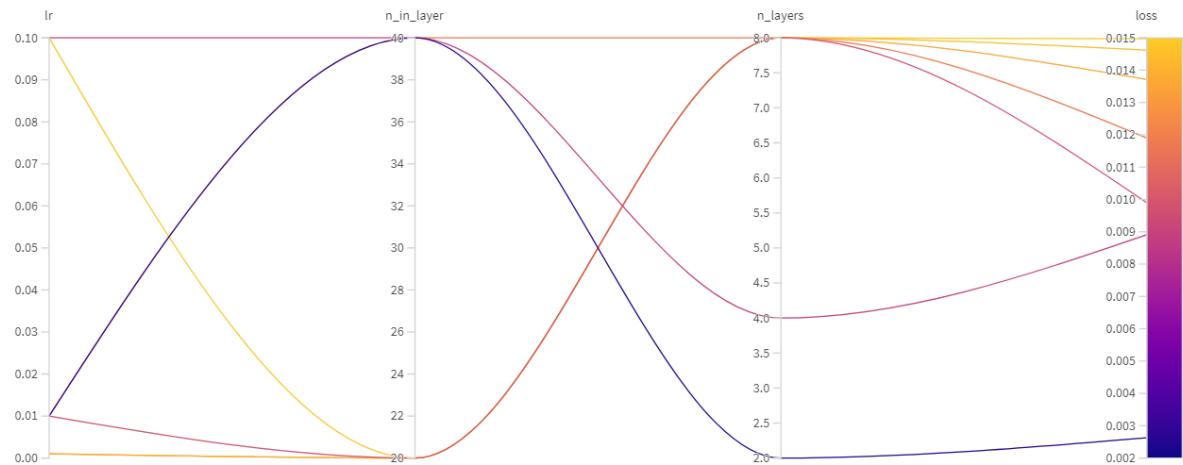


Figura 8.30: Desglose del ajuste de hiperparámetros para el problema de Burgers sin parada anticipada bajo la métrica *loss*.

8 Experimentación

en *SciANN* es muy sensible al tamaño de conjunto de datos. Sin embargo, este factor no es influyente en nuestro problema ya que el experimento asociado al Listing 8.1 iba asociado a un conjunto de entrenamiento mayor que el seleccionado para el problema de Burgers.

Por otro lado, se ha explorado el resumen de ambos modelos. Para las implementaciones propuestas, Listing 8.1 y Listing 8.4 tienen 233 y 3021 parámetros respectivamente. Por tanto, la variación de tiempos por época recae exclusivamente en la complejidad del modelo.

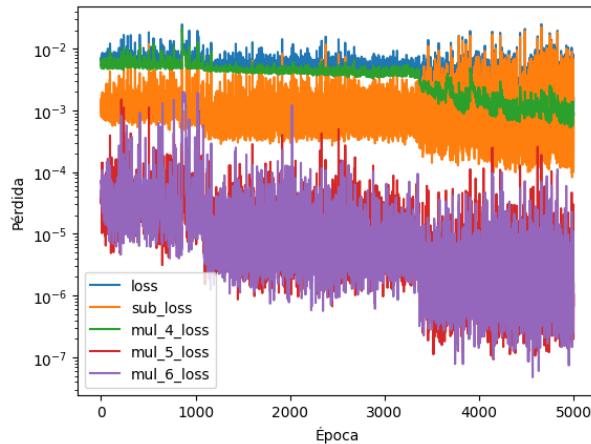


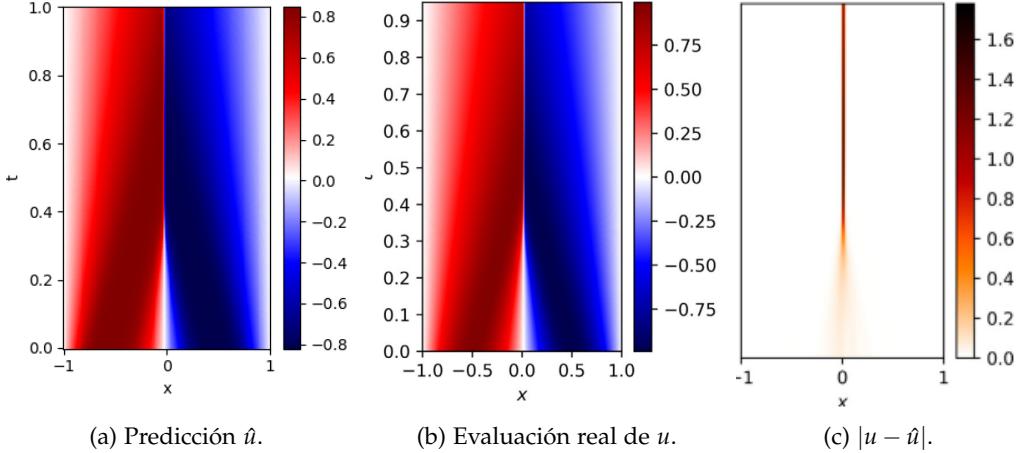
Figura 8.31: Evolución de la función de pérdida en el experimento 2.

Por último, en la Figura 8.32 podemos comparar la predicción obtenida para el modelo de Listing 8.4 con los valores reales de u . Podemos por tanto concluir que, aunque la predicción es buena, la tarea de aprendizaje es muy poco directa, como quedó reflejado en la Figura 8.31. Esto hace que necesitemos una tarea de aprendizaje muy larga, lo que lleva un coste computacional muy elevado. Sería necesario un estudio posterior para determinar hasta qué punto este coste computacional se debe a la implementación de *SciANN* o a la complejidad del modelo en si.

8.4. Experimento 3: Aproximación de operadores

8.4.1. Motivación

Hasta ahora se ha mostrado la capacidad de *SciANN* para aproximar funciones partiendo de información explícita o implícita sobre ellas. Con este experimento se pretende poner a prueba la capacidad de *SciANN* para aproximar operadores bajo la hipótesis de que, como vimos en el Capítulo 6, tomando un número lo suficientemente grande de neuronas por capa, es posible alcanzar una buena aproximación, sea cual sea nuestra elección de función de activación para el modelo. El experimento tiene otro objetivo que es la comparación de *SciANN* con *DeepONet* [LJK19], otra librería de Python para la resolución de PINNs que ya ha demostrado soportar la aproximación de operadores tanto lineales como no lineales.

Figura 8.32: Comparación de \hat{u} y u en el problema de Burgers extraída de [HJ21].

8.4.2. Descripción del problema

Debido al elevado coste computacional que suponen los modelos complejos en *SciANN*, hemos escogido un operador muy sencillo. Consideremos entonces un sistema dinámico sujeto a la ecuación diferencial ordinaria

$$\begin{cases} \frac{d}{dx}s(x) = u(x) \\ s(a) = s_0 \end{cases}$$

donde recordemos que u , que pertenece a un compacto V en $C[a, b]$, define la señal de entrada al sistema y $s : [a, b] \rightarrow \mathbb{R}$ describe la señal de salida.

Entonces podemos reformular este problema como la búsqueda de un operador $G : V \rightarrow C[a, b]$ que satisfaga:

$$(Gu)(x) = s_0 + \int_a^x u(t)dt.$$

Tomando como condición de contorno $s(0) = 0$ y como dominio de u el intervalo $[0, 1]$, esto equivale a:

$$G : u(x) \mapsto s(x) = \int_0^x u(t)dt.$$

Recordando el proceso de entrenamiento de *SciANN* descrito en la Subsección 7.2.3, podemos ver que la librería está orientada a aprender u a través de G , pero no G a través de u . Es cierto que en [HJ21] se muestra la capacidad de *SciANN* para aprender ecuaciones asociadas a PDE, que pueden ser vistas como operadores, pero únicamente en los casos en los que un operador describe un PDE parámetrico,

y el aprendizaje se restringe a dichos parámetros.

Este impedimento hace que la función de pérdida asociada a nuestro modelo no vaya a poder beneficiarse de términos extra propios de PINNs que ayuden al aprendizaje. Dicho aprendizaje consistirá, por tanto, en una tarea de regresión en la que a través de la arquitectura propuesta en el Capítulo 6 aprenderemos el operador G .

8.4.3. Obtención de datos

Nuestro objetivo es replicar el experimento realizado en [LJK19], por lo que los datos han sido obtenidos directamente del proyecto, que se encuentra disponible en <https://github.com/lululxvi/deepenet>. Como acabamos de discutir en la Subsección 8.4.2, el objetivo es minimizar el error a partir de unos datos explícitos. Esto recuerda a la regresión realizada en la Sección 8.2 y nos da información sobre la estructura de datos explícitos para $G(u)(y)$ que necesitamos.

Como se vió en la figura Figura 6.1, vamos a contar con dos subredes independientes con sus respectivas entradas. Esta arquitectura facilita que cada una aprenda por su cuenta. En [LJK19] dan nombre a estas subredes, así que para facilitar la explicación de la estructura vamos a utilizar esa misma terminología. Teniendo en mente la Figura 6.1, vamos a llamar subred branch a la de la parte inferior del esquema, que toma evaluaciones de u en m sensores $u(x_1), \dots, u(x_m)$, y subred trunk a la de la parte superior, que únicamente toma como entrada valores de y . En la Figura 8.33 mostramos una representación simplificada de la arquitectura y sus entradas.

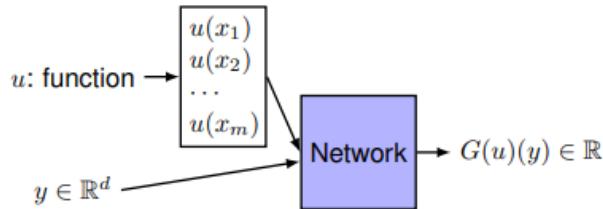


Figura 8.33: Arquitectura de las entradas necesarias para el experimento 3 propuesta en [LJK19].

Por tanto, un conjunto de entrenamiento de n elementos para la subred branch consistirá en una matriz:

$$\begin{bmatrix} u_1(x_1) & u_1(x_2) & \cdots & u_1(x_m) \\ \vdots & \vdots & \ddots & \vdots \\ u_n(x_1) & u_n(x_2) & \cdots & u_n(x_m) \end{bmatrix}$$

donde cada dato de entrada es un vector $[u_i(x_1), \dots, u_i(x_m)]^T$ que representa las muestras recogidas de una función u_i , con $i = 1, \dots, n$. Para este ejemplo se proporciona un conjunto de entrenamiento con $n = 150$ datos.

En la subred trunk, el conjunto de entrenamiento consiste en la partición que hemos realizado del intervalo $[0, 1]$. Para este experimento se ha tomado una partición con $m = 100$ elementos, lo que nos

8.4 Experimento 3: Aproximación de operadores

da un vector $y = [y_1, \dots, y_{100}]$, donde cada dato de entrada corresponde al valor y_j , con $j = 1, \dots, 100$.

Además, para estimar el error cuadrático de cada salida de la red se proporciona una matriz con evaluaciones explícitas del operador:

$$\begin{bmatrix} G(u_1)(x_1) & G(u_1)(x_2) & \cdots & G(u_1)(x_m) \\ G(u_2)(x_1) & G(u_2)(x_2) & \cdots & G(u_2)(x_m) \\ \vdots & \vdots & \ddots & \vdots \\ G(u_n)(x_1) & G(u_n)(x_2) & \cdots & G(u_n)(x_m) \end{bmatrix}.$$

SciANN no soporta que el número de datos entrenamiento de cada subred sea distinto, es decir, requiere que n sea igual a m . Para solucionar esto, se ha hecho una aumentación del conjunto de datos repitiendo valores y manteniendo la consistencia entre los datos de entrada y la matriz de comparaciones con la salida.

En concreto, el conjunto de datos para la subnet trunk es ahora un vector de 150 elementos $y = [y_1, \dots, y_{n=100}, y_1, \dots, y_{n-m=50}]$ y, de forma análoga, la matriz de evaluaciones de G se ha extendido a

$$\begin{bmatrix} G(u_1)(x_1) & G(u_1)(x_2) & \cdots & G(u_1)(x_{n=100}) & G(u_1)(x_1) & \cdots & G(u_1)(x_{n-m=100}) \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ G(u_n)(x_1) & G(u_n)(x_2) & \cdots & G(u_n)(x_{m=100}) & G(u_n)(x_1) & \cdots & G(u_n)(x_{n-m=50}) \end{bmatrix}.$$

Como cada fila de la matriz de salidas tiene ahora n elementos, mantener la integridad entre entradas y salidas requiere modificar también el conjunto de muestras de funciones u de la siguiente forma:

$$\begin{bmatrix} u_1(x_1) & u_1(x_2) & \cdots & u_1(x_{m=100}) & u_1(x_1) & u_1(x_2) & \cdots & u_1(x_{n-m=50}) \\ \vdots & \vdots & \ddots & \vdots & \vdots & & & \vdots \\ u_n(x_1) & u_n(x_2) & \cdots & u_n(x_{m=100}) & u_n(x_1) & u_n(x_2) & \cdots & u_n(x_{n-m=50}) \end{bmatrix}.$$

8.4.4. Construcción del modelo

Para comprender mejor la arquitectura a implementar, nos ayudaremos de la Figura 8.34 y la Figura 8.35, extraídas de [LJK19]. En ellas se puede ver cómo DeepONet implementa dos arquitecturas inspiradas en la arquitectura dada por el Teorema 6.5. Para este experimento nos ceñiremos a la la arquitectura de la Figura 8.34, que es un caso particular del teorema para $M = 1$ y $N = p$.

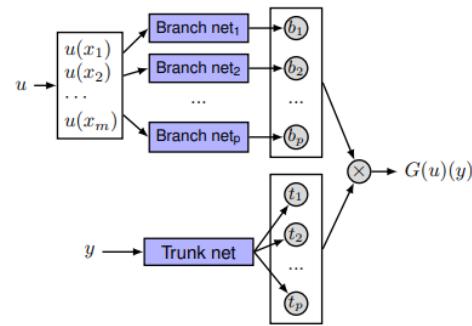


Figura 8.34: Representación compacta de la arquitectura propuesta en el Capítulo 6 para $M = 1$ y $N = p$.

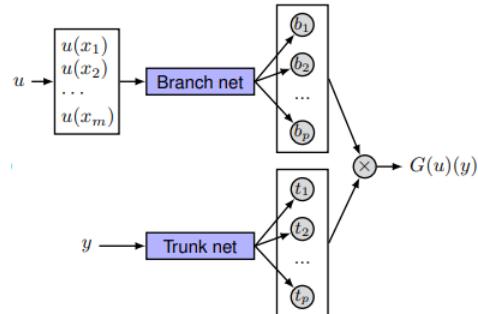


Figura 8.35: Representación compacta de la arquitectura modificada propuesta en DeepONet.

8.4.4.1. Primera aproximación

La estructura de clases de *SciANN* actualmente soporta funciones, pero no operadores. Como la arquitectura de redes neuronales en *SciANN* viene ligada a funciones, es necesario cambiar a un enfoque constructivo en el que definamos exactamente las entradas y subredes neuronales que necesitamos para nuestra arquitectura, así como las relaciones entre ellas. Es por esto que se propone el modelo siguiente ([Listing 8.5](#)):

```

1 #Número de sensores.
2 m = 100
3
4 #Número de neuronas por capa.
5 n_in_layer = 40
6
7 #Definición de entradas.
8 x = sn.Variable("x")
9 ux = [sn.Variable(f'ux{i}') for i in range(m)]
10
11
12 #Definición de subredes.
13 subnet_branch = sn.Functional('branch_out', ux, [n_in_layer], activation='sigmoid')
14 subnet_trunk = sn.Functional("trunk_out", x , [n_in_layer], 'sigmoid')
15
16 #Producto cartesiano de las salidas de ambas.
17 s = sn.utils.dot(subnet_branch, subnet_trunk)
18 d1 = sn.Data(s)
19
20 #Definición del modelo.
21 modelo_operador = sn.SciModel(inputs = ux+[x], targets=[d1], loss_func='MSE',optimizer="adam")
```

[Listing 8.5](#): Primera aproximación para el modelo de aprendizaje de operadores.

Sin embargo, mediante el resumen del modelo y el esquema extraído por Tensorflow, que se muestra en la [Figura 8.36](#), se puede observar que el producto escalar de la librería no está bien implementado.

8.4.4.2. Segunda aproximación

Debido a las dificultades encontradas, optamos por implementar el producto escalar manualmente. Recordemos que, aunque estemos definiendo las subredes como objetos de la clase *Functional*, esto no va a tener ninguna repercusión en la tarea de entrenamiento, pues no hemos definido ninguna restricción que añada términos extra a la función de pérdida.

```

1 #Definición de las entradas.
2 x = sn.Variable("x")
3 ux = [sn.Variable(f'ux{i}') for i in range(m)]
4
5
6 #Definición de subredes.
7
8 subnet_branches = [sn.Functional(f'branch_out_{i}', ux, [1], activation='tanh') for i in range
(N)]
```

8 Experimentación

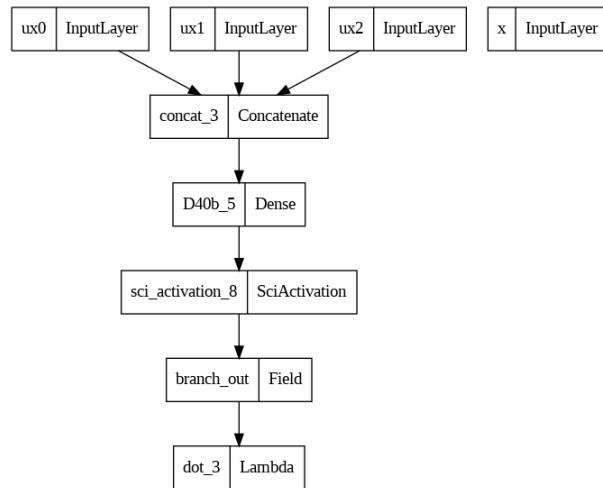


Figura 8.36: Representación interna simplificada del experimento 3: primera aproximación.

```

9 subnet_trunk = sn.Functional("trunk_out",x ,[N], 'sigmoid')
10 #Construcción manual del producto cartesiano de las subredes.
11 productos = []
12
13 for i in range(N):
14     productos.append(sn.utils.mul(subnet_branches[i],subnet_trunk))
15
16 s = sn.utils.add(productos[0],productos[1])
17
18 for i in range(2,N):
19     s = sn.utils.add(s,productos[i])
20
21 #Restricciones del modelo.
22 d1 = sn.Data(s)
23
24 #Definición del modelo.
25 modelo_operador= sn.SciModel(inputs = ux+[x], targets=[d1], loss_func='MSE',optimizer="adam")
  
```

Listing 8.6: Segunda aproximación para el modelo de aprendizaje de operadores.

En la [Figura 8.37](#) se puede observar cómo en este caso el producto entre ambas subredes se ha implementado correctamente mediante capas *Lambda* de Keras.

8.4.5. Interpretación de los resultados

En esta sección vamos a comentar y comparar los resultados obtenidos para la experimentación realizada a través de la segunda aproximación al modelo. Además de realizar un ajuste de los hiperparámetros más importantes, vamos a experimentar con el tamaño de conjunto de entrenamiento. Como quedó

8.4 Experimento 3: Aproximación de operadores

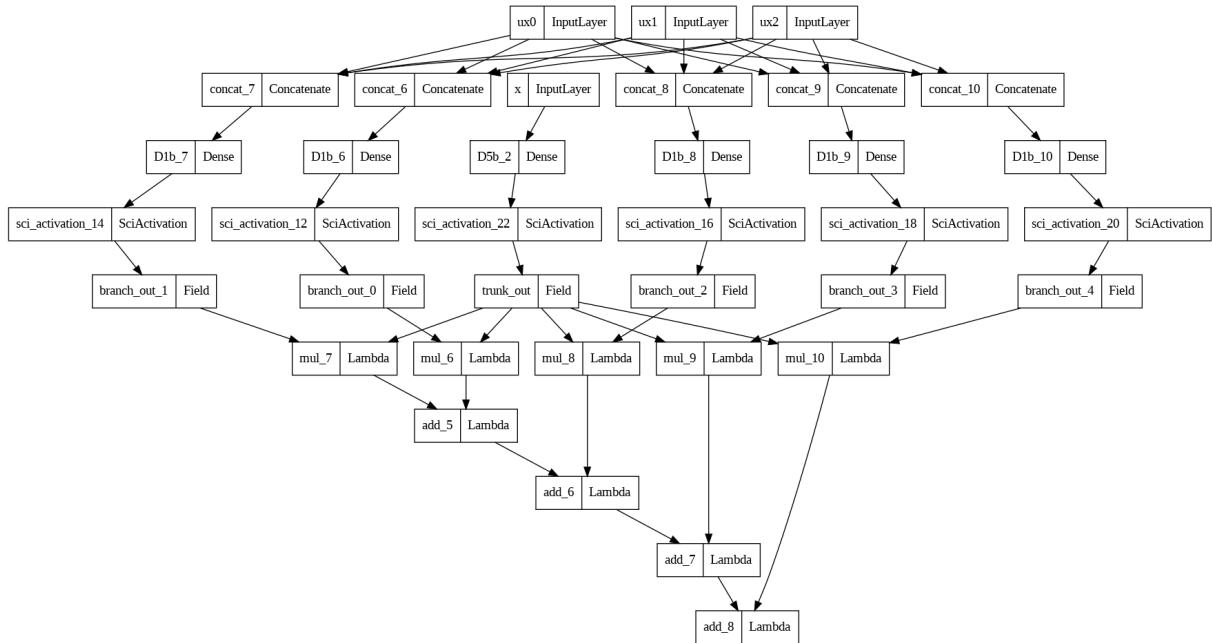


Figura 8.37: Representación interna del experimento 3: segunda aproximación.

plasmado en la Subsección 8.4.3 y en la Subsección 8.4.4, ampliar el conjunto de entrenamiento requiere ampliar el número de entradas y por tanto modificar la arquitectura del modelo. Por ello, la experimentación en *W&B* se ha realizado mediante dos proyectos distintos correspondientes a los dos conjuntos de entrenamiento seleccionados. Estos proyectos pueden consultarse [aquí](#).

8.4.5.1. Resultados para el entrenamiento con conjunto de datos reducido

En la Figura 8.38 podemos ver cómo la experimentación no arroja mucha luz a cerca de qué hiperparámetros funcionan mejor. Al margen de alguna configuración especialmente mala, hay configuraciones muy dispares que dan resultados relativamente parecidos. Por otro lado, comparando la Figura 8.38 con la Figura 8.39, se puede ver que la configuración que minimiza la pérdida en validación también minimiza la pérdida en entrenamiento. Esto es positivo, ya que indica que durante el entrenamiento no hemos sobre-ajustado el modelo al conjunto de validación. Tras ver estos resultados, se contemplan dos hipótesis.

La primera, que no tengamos lo suficientes datos para realizar una buena tarea predictiva. La segunda hipótesis tiene que ver con que la arquitectura propuesta no sea buena para esta tarea de predicción. Ambas hipótesis pueden estar relacionadas, como explicábamos en la Subsubsección 8.4.4.2, con que no se hace uso de ningún tipo de restricción propia de PINNs que ayuden a la tarea de entrenamiento. En presencia de pocos datos, dichas restricciones pueden aportar información extra que sea clave para completar satisfactoriamente la tarea de entrenamiento.

8 Experimentación

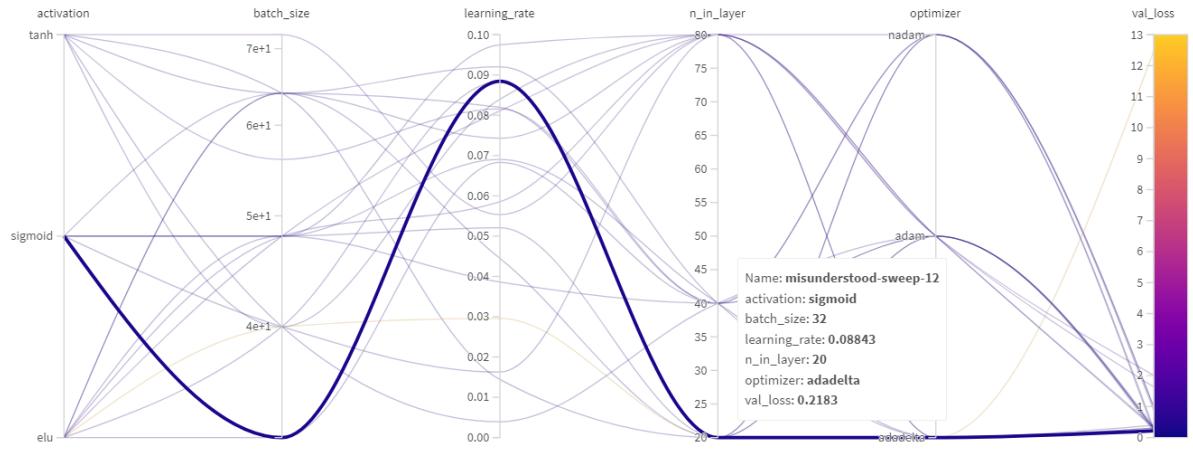


Figura 8.38: Desglose del ajuste de hiperparámetros para el experimento 8.3. con conjunto de datos reducido bajo la métrica *val_loss*.

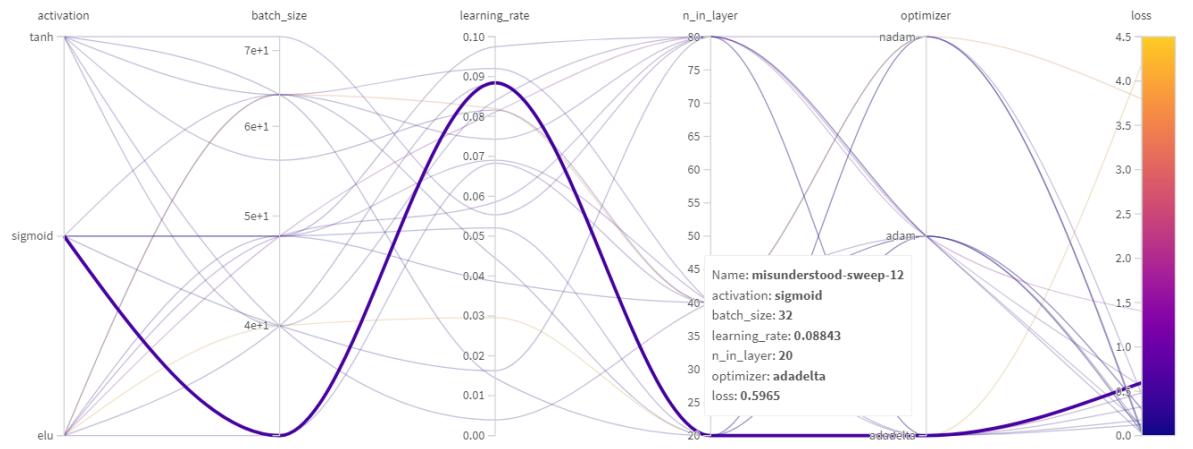


Figura 8.39: Desglose del ajuste de hiperparámetros para el experimento 8.3. con conjunto de datos reducido bajo la métrica *loss*.

8.4 Experimento 3: Aproximación de operadores

W&B ofrece también un ranking con los parámetros que han sido más relevantes durante el proceso de entrenamiento y los que están más correlacionados con la métrica seleccionada para el entrenamiento. En la [Figura 8.40](#) podemos ver que valores altos en error están correlacionados con utilizar la función de activación ELU y el optimizador *adadelta*. Sin embargo, este optimizador también está presente en la mejor configuración de parámetros encontrada. De nuevo, estos resultados apuntan a que con tan pocos datos no es posible entrenar bien el modelo.

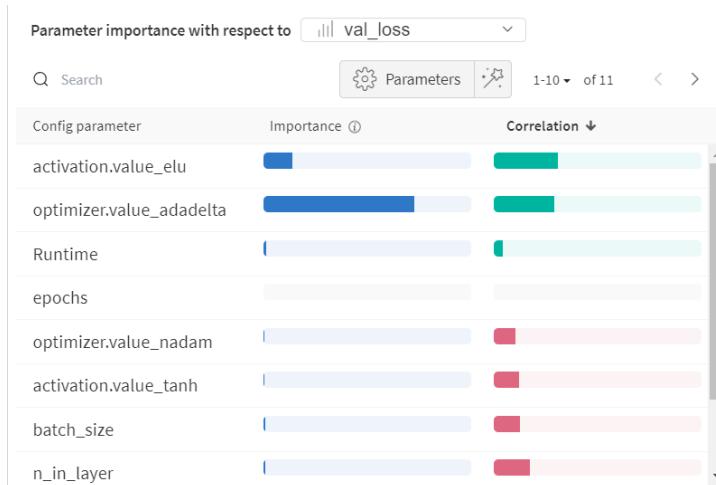


Figura 8.40: Ranking de parámetros correlacionados con la métrica *val_loss* para el experimento 8.3. con conjunto de datos reducido.

Tras realizar el ajuste de hiperparámetros, se ha seleccionado el modelo que minimiza la pérdida en validación y realizado un estudio comparativo entre su evolución en entrenamiento y la evolución del modelo con la configuración de parámetros propuesta en *DeepONet*, esto es, el que se corresponde con el [Listing 8.6](#). El modelo que miniza la pérdida cuenta con activación sigmoidal, 20 neuronas en la capa oculta, optimizador adadelta y un tamaño de lote de 32 ejemplos. Aunque el entrenamiento propuesto en [\[LJK19\]](#) se realiza con conjunto de validación, hemos realizado también una comparativa con la evolución de entrenamiento sin utilizar conjunto de validación. Al igual que en la batería de experimentación con diferentes hiperparámetros, para el entrenamiento se ha utilizado el criterio de parada anticipada bajo las métricas de *val_loss* y *loss* respectivamente.

Como podemos ver en la [Figura 8.41](#), el entrenamiento gobernado por *val_loss* termina muy pronto y alcanza una precisión del orden de 10^{-1} para *val_loss* y de 10^{-2} para la métrica *loss*. Por otro lado, el entrenamiento sin conjunto de validación alcanza valores de pérdida del orden de 10^{-4} cercanos a las precisiones alcanzadas por la implementación en *DeepONet*. Sin embargo, en ambos casos, al realizar la predicción sobre el conjunto de test la precisión no baja de 10^{-1} .

8 Experimentación

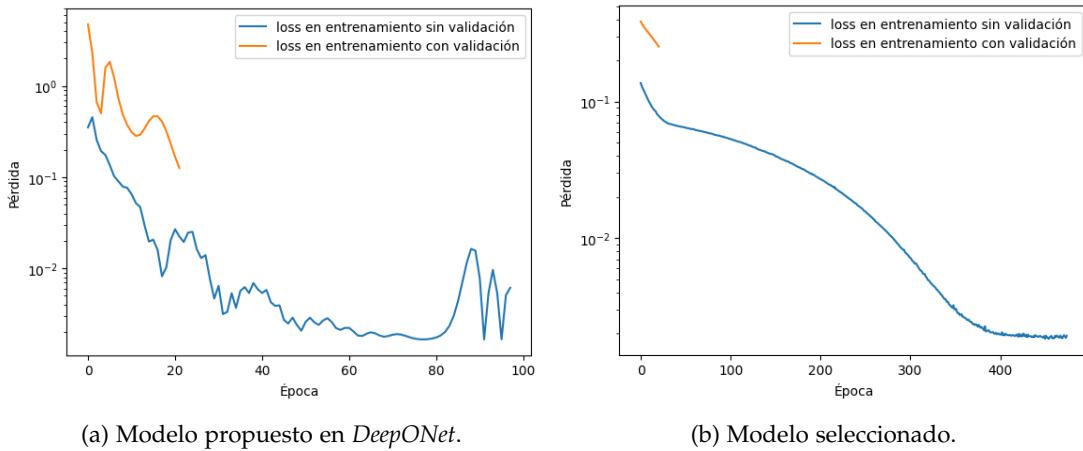


Figura 8.41: Comparación entre los modelos seleccionados para el conjunto de entrenamiento reducido.

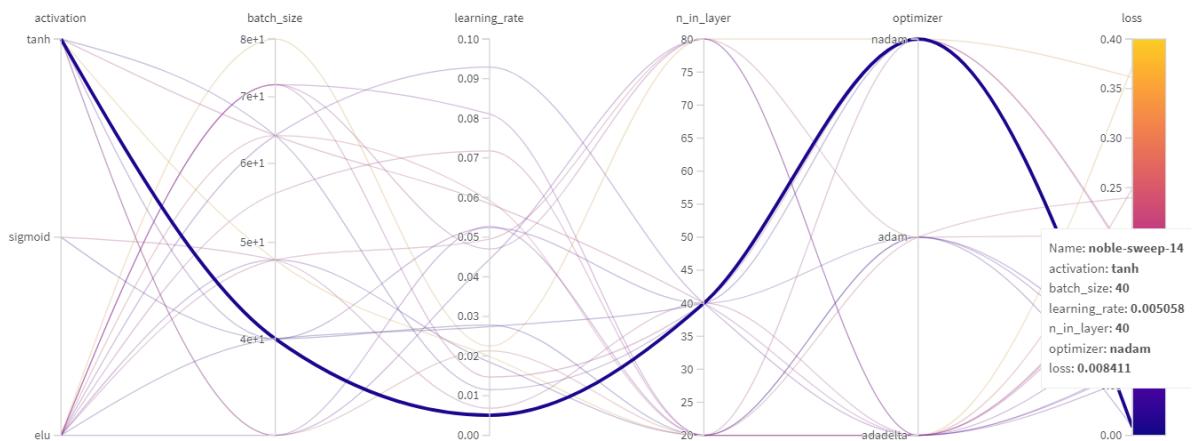
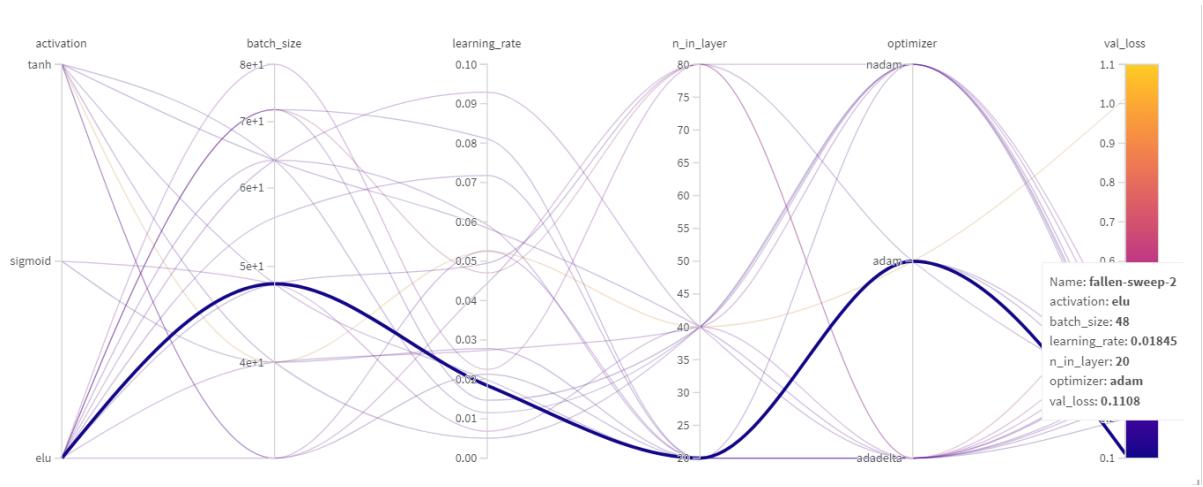
8.4.5.2. Resultados para el entrenamiento con conjunto de datos ampliado

En el diagrama de la Figura 8.42 se muestran los detalles de la experimentación realizada con todos los datos de entrenamiento disponibles, adaptando la arquitectura del modelo para ello. Como se puede ver en una comparación con la Figura 8.43, aunque la pérdida en validación se mantenga en el orden de 10^{-1} , la pérdida en entrenamiento sí disminuye. Una vez más, el modelo se sobre-ajusta a los datos de entrenamiento, cuya precisión difiere significativamente de la obtenida en validación. De estos resultados deducimos que las limitaciones para la predicción se deben a la arquitectura y proceso de entrenamiento del modelo, ya que para los mismos conjuntos de datos el modelo de [LJK19] realiza predicciones en validación con precisiones del orden de 10^{-4} .

Aunque no hayamos conseguido mejoras significativas en la tarea de aprendizaje, el ranking mostrado en la Figura 8.44 parece dar información más coherente sobre el modelo que el ranking del apartado anterior. El parámetro de configuración más correlacionado con valores altos en *val_loss* es la anchura de la capa profunda, *n_in_layer*. Volviendo a la Figura 8.42, podemos corroborar que la mayoría de combinaciones que partían de un valor elevado para este parámetro han resultado en errores de validación grandes. Para justificar esto, es necesario recordar que la predicción no es buena en validación pero sí en entrenamiento. Esto indica un sobre-ajuste a los datos de entrenamiento. Este escenario puede deberse a trabajar con modelos complejos, con demasiados grados de libertad, como es el caso de nuestro modelo cuando *n_in_layer* toma valores grandes. La aproximación del operador es mejor para modelos más simples y, por tanto, no es que nuestra tarea sea demasiado compleja para el modelo, si no que la arquitectura y proceso de entrenamiento no se adecuan bien al aprendizaje.

Por último, volvemos a realizar una comparación entre los entrenamientos del mejor modelo encontrado durante la experimentación y el modelo propuesto en el Listing 8.6. Como podemos ver en la Figura 8.45, la evolución es muy similar a la que obteníamos en el apartado anterior. Aunque para ambos modelos se ha conseguido más precisión en el conjunto de entrenamiento, recordemos que la precisión en los conjuntos de validación y test se mantiene en el orden de 10^{-1} , por lo que consideraremos

8.4 Experimento 3: Aproximación de operadores



8 Experimentación

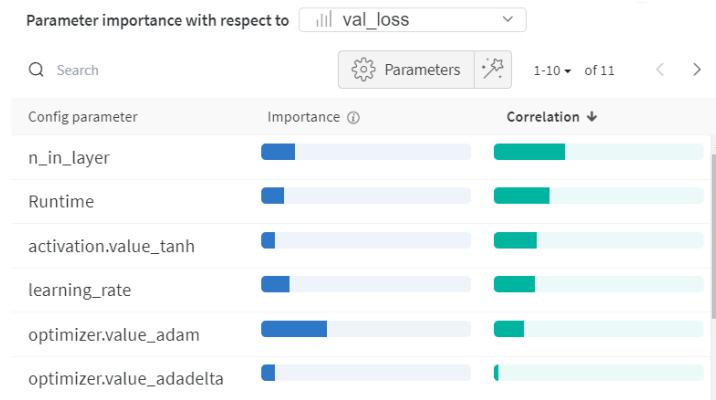


Figura 8.44: Ranking de parámetros correlacionados con la métrica val_loss para el experimento 8.3, con conjunto de datos ampliado.

que ampliar el conjunto de entrenamiento no ha mejorado la tarea de aprendizaje.

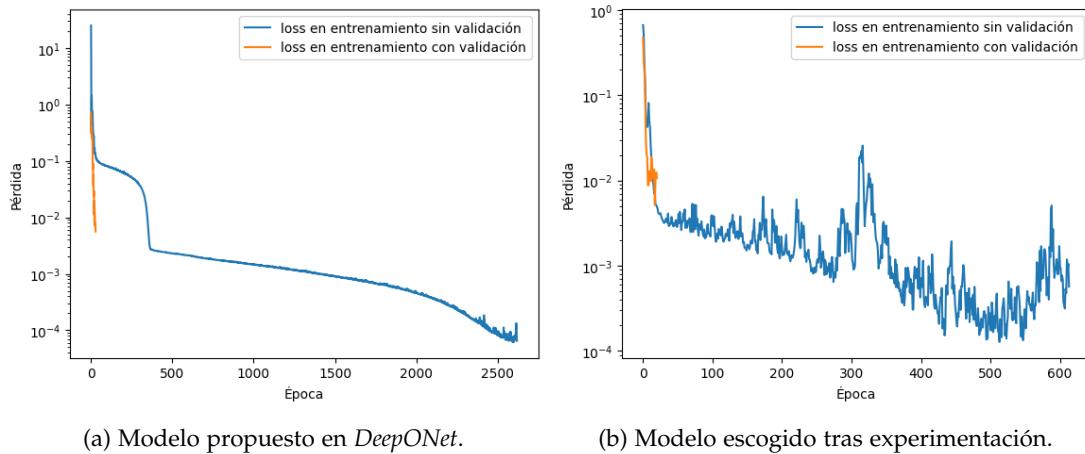


Figura 8.45: Comparación entre los modelos seleccionados para el conjunto de entrenamiento ampliado.

8.4.5.3. Interpretación

Tras las experimentaciones realizadas, hemos visto que los modelos pueden llegar a ajustarse a los datos de entrenamiento proporcionados pero, sin embargo, no generalizan bien. La poca capacidad de generalización se mantiene incluso para los modelos más simples, lo que descarta un sobre-ajuste a los datos de entrenamiento debido a la complejidad del modelo. Este hecho, junto con los buenos resultados obtenidos en [LJK19] para el mismo conjunto de datos, demuestra que la incapacidad para aproximar operadores no recae sobre el conjunto de datos seleccionado, si no sobre las abstracciones, arquitectura y proceso de entrenamiento que a día de hoy SciANN soporta para esta tarea. Con todo

8.4 Experimento 3: Aproximación de operadores

esto, se concluye que ninguno de los modelos ha sido capaz de realizar la tarea de aprender operadores globalmente. Esto se atribuye principalmente a la incapacidad de *SciANN* para integrar el aprendizaje de operadores desde el enfoque de las PINNs.

9 Conclusiones y futuros trabajos

En este Trabajo Fin de Grado se han estudiado los aspectos más importantes de la Teoría de Distribuciones con el objetivo de comprender, desarrollar e implementar los resultados propuestos en el artículo [CC95] para la aproximación de operadores de Lipschitz continuos entre espacios de Banach. Para su desarrollo se ha seguido la planificación propuesta en el [Capítulo 1](#).

Para hacer frente a los resultados propuestos en [CC95], en el [Capítulo 3](#) se ha realizado un estudio de los fundamentos de los Espacios Vectoriales Topológicos: su construcción, convergencia y tipos, haciendo énfasis en el espacio de funciones test para poder introducir la Teoría de Distribuciones. En el [Capítulo 4](#) se han trabajado los resultados más importantes de esta teoría y se han presentado las funciones de decrecimiento rápido con el objetivo de introducir las distribuciones temperadas, que son el ambiente de trabajo para algunos de los resultados principales de [CC95]. Para ello, además, ha sido necesario recordar algunos resultados estudiados durante el grado, que quedan recogidos en el [Capítulo 2](#).

Por otro lado, en el [Capítulo 5](#) se han presentado los aspectos fundamentales de los modelos de aprendizaje profundo actuales, incluyendo su contextualización dentro del aprendizaje automático, la definición del modelo de red profunda prealimentada y las particularidades del proceso de entrenamiento de este tipo de redes.

Posteriormente, en el [Capítulo 6](#), se han expuesto y demostrado los resultados del artículo [CC95] publicado por Tianping Chen y Robert Chen en 1995, que responde a las restricciones impuestas a las funciones de activación sigmoidales en los modelos de aprendizaje profundo vigentes en la época, así como a la incapacidad de las de redes neuronales diseñadas hasta el momento para hacer frente a problemas regidos por funcionales u operadores no lineales.

Para la implementación de estos resultados se ha escogido un acercamiento mediante Physics-Informed Neural Networks (PINNs). En el [Capítulo 7](#) se recogen los aspectos más importantes del estudio bibliográfico realizado en la materia. Se ha hecho especial énfasis en la librería *SciANN* por haber sido la librería escogida para desarrollar este proyecto.

Una vez introducido el concepdo de PINN, en el [Capítulo 8](#) se ha realizado una serie de experimentaciones para evaluar la capacidad de *SciANN* para realizar distintas tareas de predicción. En estas experimentaciones hemos medido la robustez de los modelos de *SciANN* frente a ruido y su escalabilidad frente a grandes tamaños de datos y frente a modelos complejos. También se ha realizado una estimación del coste computacional que se atribuye al proceso de entrenamiento propio de las PINNs.

La experimentación culmina integrando la arquitectura propuesta en el [Capítulo 6](#) para la aproximación de operadores con *SciANN* y comparando los resultados con los obtenidos para la librería

9 Conclusiones y futuros trabajos

DeepONet [LJK19], la única librería orientada a PINNs que a día de hoy soporta el aprendizaje de operadores. Tras la experimentación, se ha concluido que *SciANN* no cuenta con las abstracciones necesarias para realizar correctamente esta tarea. Sin embargo, se considera que integrar el aprendizaje de operadores en la librería podría dar resultados muy positivos, pues la arquitectura propuesta en el [Capítulo 6](#) cuenta con pocas capas ocultas, lo que aseguraría un buen comportamiento de la librería frente al problema.

Con todo esto, se consideran cumplidos los objetivos iniciales del trabajo. No obstante, las limitaciones que *SciANN* ha mostrado durante la implementación del mismo dejan varias vías de trabajo futuro abiertas:

- Corregir la implementación que *SciANN* propone para el producto cartesiano.
- Añadir la abstracción de operador mediante una clase de *SciANN*.
- Trabajar en una integración más sólida de *SciANN* con *Weights&Biases*.
- Estudiar en detalle la implementación de la librería *DeepONet* para identificar los elementos en los que difiere de *SciANN*.
- Ampliar la experimentación incluyendo más ejemplos de operadores no lineales.

Bibliografía

- [BIKP20] Matija Burić, Marina Ivašić-Kos, y Goran Paulin. Object detection using synthesized data. En *Proceedings of ICT Innovations*, Ohrid, April 2020. University of Rijeka. Presented at ICT Innovations, April 2020.
- [BPRS18] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, y J.M. Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, 2018.
- [CC95] T. Chen y H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its applications to dynamic systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.
- [Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.
- [GBC16] Ian Goodfellow, Yoshua Bengio, y Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [HG16] Dan Hendrycks y Kevin Gimpel. Gaussian Error Linear Units (GELUs). En *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, páginas 10–18, 2016.
- [HJ21] E. Haghighat y R. Juanes. SciANN: A Keras/TensorFlow wrapper for scientific computations and physics-informed deep learning using artificial neural networks. *Computer Methods in Applied Mechanics and Engineering*, 373, 2021.
- [HSW89] K. Hornik, M. Stinchcombe, y H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
- [K79] G. Köthe. *Topological Vector Spaces*, volumen 159 de *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag, 1979.
- [KUMH17] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, y Sepp Hochreiter. Self-normalizing neural networks. En *Advances in Neural Information Processing Systems*, páginas 971–980, 2017.
- [Lin70] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. Master's thesis, University of Helsinki, 1970. Master's Thesis (in Finnish).
- [LJK19] Lu Lu, Pengzhan Jin, y George Em Karniadakis. DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of

Bibliografía

- operators. *Journal of Applied Mathematics*, 42(3):123—145, 2019.
- [Mar18] Toru Maruyama. *Fourier Transforms of Measures*, páginas 121–163. Springer Singapore, Singapore, 2018.
- [MP43] Warren S. McCulloch y Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- [MS11] Miguel Martín Suárez. *Análisis Funcional*. Universidad de Granada, 2.0 edición, December 2011.
- [Rai18] M. Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *Journal of Machine Learning Research*, 19:1–24, 2018.
- [RPK19] M. Raissi, P. Perdikaris, y G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
- [Rud91] Walter Rudin. *Functional Analysis*. McGraw-Hill, Inc., second edición, 1991.
- [SSBD14] Shai Shalev-Shwartz y Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.