UNA APLICACIÓN DE REDES NEURONALES EN LA APROXIMACIÓN DE ECUACIONES DIFERENCIALES PARCIALES EVOLUTIVAS

Darwin Ferney Tapie Taimal

UNIVERSIDAD DEL VALLE
FACULTAD DE CIENCIAS NATURALES Y EXACTAS
DEPARTAMENTO DE MATEMÁTICAS
SANTIAGO DE CALI
2025

Darwin Ferney Tapie Taimal

Proyecto de trabajo de investigación presentado como requisito parcial para optar al título de Magíster en Ciencias Matemáticas 7179

Director:
JUAN CARLOS MUÑOZ GRAJALES, PH.D

UNIVERSIDAD DEL VALLE
FACULTAD DE CIENCIAS
DEPARTAMENTO DE MATEMÁTICAS
SANTIAGO DE CALI
2025

Agradecimientos

A Dios, por la vida y por permitirme disfrutar y aprender del admirable mundo de las matemáticas. A mi esposa Andrea Paola y a mis padres, por su amor, paciencia, motivación constante y por ser el motor que me impulsa cada día. A mis amigos, en especial a Andrés, Alex, Jorge y John Ericson, por su compañía y por las experiencias compartidas que enriquecieron mi formación profesional.

Al profesor Juan Carlos Muñoz, director de este trabajo, por su constante dedicación, por la paciencia con la que me guió en reuniones y seminarios sobre problemas inversos y EDPs, y por la claridad de sus enseñanzas, que fueron esenciales para lograr finalizar esta investigación.

Resumen

En este trabajo nos enfocamos en la implementación y aplicación de redes neuronales informadas por la física (Physics Informed Neural Networks, PINNs[24]) para la resolución numérica de ecuaciones diferenciales parciales (EDPs) de tipo evolutivo. Las PINNs representan un enfoque innovador dentro del campo del aprendizaje automático científico [11], dado que integran principios físicos en el proceso de entrenamiento. Presentamos un estudio de la teoría relacionada con redes neuronales y su capacidad como aproximadores universales. Explicamos en detalle la aplicación del enfoque PINNs tanto para problemas directos como inversos, en modelos evolutivos como la ecuación de Burgers viscosa (VB) y la ecuación de Korteweg-de Vries (KdV) en una dimensión. En cuanto a las implementaciones numéricas, estas se realizan en la librería especializada PyTorch en Python.
Los resultados muestran que las PINNs aproximan adecuadamente soluciones, aunque presentan limitaciones en puntos de discontinuidad y su exactitud varía con la arquitectura elegida. Finalmente, se realiza un estudio comparativo con el método de elementos finitos (FEM), donde este destaca por su rapidez y escalabilidad, mientras que las PINNs ofrecen flexibilidad, bajo consumo de memoria, son libres de malla y presentan ventajas al ejecutarse en GPU.
Palabras clave: Redes neuronales, Redes neuronales informadas por la física (PINNs), Ecuaciones diferenciales parciales (EDPs), Análisis numérico.

Santiago de Cali, Colombia
Septiembre 2025

Darwin Ferney Tapie

Contenido

Capítulo 1
Introducción

Gracias a la creciente disponibilidad de datos y los recursos computacionales actuales, las redes neuronales artificiales han logrado grandes avances en tareas de aprendizaje automático, como la visión por computadora, el reconocimiento de imágenes, el procesamiento del lenguaje natural, entre otros [2, 3, 10]. Esto ha llevado a la comunidad científica a investigar su idoneidad también para tareas de computación de alto rendimiento. El resultado de esto es un nuevo campo de investigación denominado *aprendizaje automático científico*, donde estrategias como las redes neuronales profundas y el aprendizaje estadístico se aplican a problemas clásicos de aplicación en matemáticas aplicadas [11].

Tras décadas de investigación sobre la solución numérica de ecuaciones diferenciales parciales (EDPs), actualmente permanecen algunos desafíos significativos. Sin embargo, nuevas técnicas se presentan como posibles soluciones; en particular, las redes neuronales artificiales han emergido como una herramienta poderosa para aproximar funciones de alta dimensión no lineales [20]. En particular, las llamadas redes neuronales informadas por la física [24] (PINNs, por sus siglas en inglés) ofrecen un enfoque innovador para resolver EDPs al incorporar principios físicos fundamentales, expresados mediante ecuaciones diferenciales, directamente en la función de pérdida durante el entrenamiento de la red.

Nuestro objetivo central en este trabajo es implementar esta técnica en dos modelos evolutivos de la forma

$$\partial_t u(x,t) + \mathcal{N}[u;\lambda](x,t) = 0, \quad (x,t) \in [0,T] \times \Omega, \tag{1.1}$$

donde $\Omega \subset \mathbb{R}^n$ es un dominio acotado, $T > 0$ es el tiempo final, λ representa parámetros físicos y el operador $\mathcal{N}[\cdot;\lambda]$ modela fenómenos como la convección $u\,\partial_x u$, la difusión $\nu\,\partial_{xx} u$ o la dispersión $\lambda\,\partial_{xxx} u$. La solución deseada $u\colon [0,T]\times\Omega \to \mathbb{R}$ está sujeta a condiciones iniciales y de frontera de la forma

$$u(x,0) = g(x), \quad x \in \Omega, \tag{1.2}$$

$$\mathcal{B}[u] = 0, \quad t \in [0,T], \quad x \in \partial\Omega, \tag{1.3}$$

donde g es una función dada y $\mathcal{B}[\cdot]$ es un operador de frontera que representa condiciones tipo Dirichlet. En particular, consideramos la ecuación de Burgers viscosa (BV) [61] y la ecuación Korteweg-de Vries (KdV) en una dimensión [63].

La estrategia de las PINNs para resolver una EDP de la forma (1.1) consiste en construir una aproximación de la solución dada por una red neuronal $u_\theta(x,t)$ (donde θ denota los parámetros entrenables de la red), y $u_\theta : \Omega \times [0,T] \to \mathbb{R}$ se define como una composición de funciones lineales afines $W^\ell(\cdot) + b^\ell$ y funciones no lineales $\sigma^\ell(\cdot)$, conocidas como funciones de activación

$$u_\theta(z) := W^L \sigma^L(W^{L-1}\sigma^{L-1}(\cdots \sigma^1(W^0 z + b^0)\cdots) + b^{L-1}) + b^L. \tag{1.4}$$

Aquí W^ℓ y b^ℓ se conocen como matrices de pesos y vectores de sesgo, respectivamente, y $z = [x,t]^T$. La configuración de red neuronal considerada se conoce como *red neuronal alimentada hacia adelante de varias capas*, conocida también como *perceptrón multicapa* [57]. Esta red toma como entradas la posición x y el tiempo t, y devuelve un valor aproximado de la solución. Para asegurar que u_θ respete las ecuaciones y condiciones de contorno, la PINN se entrena usando una función de pérdida de la forma,

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{ic}} + \mathcal{L}_{\text{bc}} + \mathcal{L}_{\text{r}}, \tag{1.5}$$

donde $\mathcal{L}_{\text{ic}}$ mide el error de la red en los puntos de la condición inicial, $\mathcal{L}_{\text{bc}}$ penaliza el desajuste de las condiciones de frontera, y $\mathcal{L}_{\text{r}}$ evalúa el residuo de la ecuación (1.1) en un conjunto de puntos denominados de *colocación* del dominio. El residuo se calcula sustituyendo u_θ en la EDP y utilizando derivación automática para computar las derivadas parciales de u_θ respecto de x y t. El entrenamiento consiste en minimizar $\mathcal{L}(\theta)$ mediante algoritmos de optimización como Adam o L-BFGS-B, ajustando los parámetros de la red hasta que el residuo y los errores en las condiciones iniciales y de frontera se hagan pequeños o se logre una tolerancia deseada.

El segundo objetivo de este trabajo es analizar la capacidad de las PINNs en la resolución de problemas inversos para los dos modelos evolutivos en cuestión (BV y KdV); el primero se conoce como asimilación de datos donde el objetivo es reconstruir la condición inicial a partir de un conjunto de observaciones posiblemente ruidosas. El segundo es un problema denominado reconstrucción de parámetros, el cual consiste en determinar el parámetro desconocido λ en (1.1), a partir de un conjunto apropiado de datos observados. Estos problemas son fundamentales en diversas disciplinas, como ingeniería , física y ciencias ambientales. Sin embargo, su resolución suele ser compleja debido a la mala colocación de este tipo de problemas, lo que los hace sensibles a errores en los datos, o a la no unicidad de las soluciones [43, 42].

Por último, evaluamos el desempeño relativo de las PINNs frente al método de elementos finitos (FEM) en el modelo de Burgers viscoso (BV). La comparación se realizó en dos aspectos: (i) precisión de la solución, (ii) el coste computacional y capacidad de generalización paramétrica. En términos de precisión, las PINNs alcanzan errores del mismo orden que FEM para una arquitectura adecuada y una función de pérdida calibrada con términos físicos. Con respecto a costo computacional, FEM destaca por su rapidez en simulaciones directas, mientras que las PINNs aprovechan eficientemente las GPUs, lo que reduce de forma notable los

tiempos de entrenamiento y permite explorar arquitecturas más profundas sin penalizaciones excesivas.

Este trabajo está organizado de la siguiente manera: En el capítulo 2, se presentan los conceptos preliminares, se introduce de manera general la teoría relacionada con los problemas directos y de tipo inverso, la cual nos ayudará a comprender el problema de asimilación de datos y de reconstrucción de parámetros considerados en el presente trabajo. En el capítulo 3, se describen de manera general los modelos evolutivos estudiados, partimos de una forma simplificada de Navier–Stokes para motivar la ecuación de Burgers viscosa (BV) y la ecuación de Korteweg–de Vries (KdV), se discuten sus propiedades básicas y se especifican las condiciones iniciales y de frontera empleadas.
En el Capítulo 4, se introducen las redes neuronales artificiales y sus componente principales, se describe la neurona y la red multicapa, las funciones de activación más usadas y los algoritmos de entrenamiento por retropropagación y optimización. Adicionalmente, al final de este capítulo se establecen los teoremas de aproximación universal, los cuales garantizan la aproximación de una función continua por una red neuronal con una arquitectura específica.
En el capítulo 5 se presentan y desarrollan las redes neuronales informadas por la física (PINNs): la formulación de la aproximación, el uso de derivadas automáticas, la selección de puntos de colocación y el flujo de entrenamiento. En seguida, se presentan los resultados numéricos: se comparan soluciones exactas y aproximadas mediante gráficos y tablas de error, se analizan curvas de pérdida y tiempos de cómputo, y se contrasta el desempeño de las PINNs frente al método de elementos finitos (FEM) en el modelo (BV), considerando precisión, costo computacional y generalización paramétrica, así como el impacto del uso de GPU. Por último, en el Capítulo 6, se presentan las conclusiones generales de los resultados obtenidos en este trabajo y finalmente, en el apéndice se anexan un enlace de GitHub que contiene todos los programas utilizados para las simulaciones numéricas.

Capítulo 2
Preliminares

2.1 Conceptos generales

Definición 2.1. Sea X un espacio vectorial real o complejo. Una función $\|\cdot\| : X \to \mathbb{R}$ se llama *norma* si cumple las siguientes propiedades:

i) $\|x\| \geq 0$, para cualquier $x \in X$ y $\|x\| = 0$ si y solo si $x = 0$.

ii) $\|\alpha x\| = |\alpha|\|x\|$, para todos $x \in X$ y escalares α.

ii) $\|x + y\| \leq \|x\| + \|y\|$, para todos $x, y \in X$.

Si $\|\cdot\|$ es una norma de X, se dice que $(X, \|\cdot\|)$ es un *espacio vectorial normado* o simplemente un *espacio normado*, y se escribe X sin mencionar su norma cuando sea claro en el contexto cuál es su norma. Los ejemplos más usuales de espacio vectorial normado son $\mathbb{R}^d$ (o $\mathbb{C}^d$), para $d \in \mathbb{N}$ fijo, con cualquiera de las normas $\|\cdot\|_2, \|\cdot\|_\infty$ o $\|\cdot\|_1$, donde $x = (x_1, x_2, \ldots, x_d)^T$ y

$$\|x\|_2 := \left(\sum_{k=1}^{d} |x|_k^2\right)^{1/2}, \quad \|x\|_\infty := \max_{1\leq k\leq d} |x_k|, \quad \|x\|_1 := \sum_{k=1}^{d} |x_k|. \tag{2.1}$$

Una propiedad útil de definir una norma sobre un espacio vectorial V es que induce una métrica $d : V \times V \to \mathbb{R}_+$ sobre el espacio vectorial [1] como $d(u, v) = \|u - v\|$. Esto nos permite medir las distancias entre puntos en un espacio X, lo que será útil para definir la convergencia.

Definición 2.2.

(1) Dada una sucesión (x_n) en un espacio normado $(X, \|.\|)$, decimos que *converge* a un x si para todo $\epsilon > 0$ existe $N(\epsilon) \in \mathbb{N}$ tal que

$$\|x_n - x\| < \epsilon \quad \text{si} \quad n \geq N(\epsilon). \tag{2.2}$$

(2) Una sucesión (x_n) en un espacio normado $(X, \|.\|)$ se llama *sucesión de Cauchy* si para todo $\epsilon > 0$ existe un $N = N(\epsilon) \in \mathbb{N}$ tal que

$$\|x_m - x_n\| < \epsilon, \tag{2.3}$$

para todos $m, n \geq N$. El espacio normado $(X, \|.\|)$ se denomina *completo* si cualquier sucesión de Cauchy en X converge a un elemento de X.

El concepto de sucesión de Cauchy da lugar a la definición de un tipo fundamental de espacio vectorial.

Definición 2.3. (Espacio de Banach) Un espacio vectorial normado $(X, \|\cdot\|)$ se dice que es un *espacio de Banach* si X es completo con respecto a la norma $\|\cdot\|$.

Ejemplo 2.1. Sea m la medida de Lebesgue sobre $\mathbb{R}^n$. Para $p \in [1, \infty)$ y $n \in \mathbb{N}$ fijos, definimos la norma,

$$\|f\|_p = \left(\int_{\mathbb{R}^n} |f|^p dm \right)^{1/p},$$

para cualquier función $f : \mathbb{R}^n \to \mathbb{C}$ en el espacio vectorial

$$\mathscr{L}^p(\mathbb{R}^n) = \{f : \mathbb{R}^n \to \mathbb{C} : f \text{ medible y } \|f\|_p < \infty\}.$$

Aquí la integral $\int dm$ es con respecto a la medida de Lebesgue m. Si consideramos la relación de equivalencia $f \sim g$ si y solo si $f = g$ m-c.t.p., para $f, g \in \mathscr{L}^p(\mathbb{R}^n)$, entonces tenemos que el espacio cociente

$$L^p(\mathbb{R}^n) := \mathscr{L}^p(\mathbb{R}) / \sim, \tag{2.4}$$

resulta ser un espacio de Banach con la norma $\|\cdot\|_p$.

Definición 2.4. Un *espacio con producto interno* es un espacio vectorial sobre $\mathbb{R}$ (o $\mathbb{C}$) con una funión llamada *producto interno* $\langle \cdot, \cdot \rangle : X \times X \to \mathbb{R}$ (o $\mathbb{C}$) tal que para cualesquier $x, y, z \in X$ y $\alpha, \beta \in \mathbb{R}$ (o $\mathbb{C}$),

1. $\langle x, x \rangle \geq 0$ con la igualdad si y solo si $x = 0$.
2. $\langle x, y \rangle = \overline{\langle y, x \rangle}$; donde la barra representa el conjugado.
3. $\langle \alpha x + \beta y, z \rangle = \alpha \langle x, z \rangle + \beta \langle y, z \rangle$.

La fórmula $\|x\| = \sqrt{\langle x, x\rangle}$ para todo $x \in X$, define una norma en X, de manera que cualquier espacio con producto interno es también un espacio vectorial normado.

Definición 2.5. (Espacio de Hilbert) Un *espacio de Hilbert* es un espacio vectorial con producto interno $(\mathcal{H}, \langle\cdot,\cdot\rangle)$ el cual es completo con respecto a la norma $\|\cdot\|$ inducida por el producto interno.

Ejemplo 2.2. Para el caso $p = 2$, en el ejemplos 2.1, $L^2(\mathbb{R}^n)$ es un espacio de Hilbert con el producto interno definido por

$$\langle f, g\rangle = \int_{\mathbb{R}^n} f\overline{g}dx,$$

el cual induce la 2-norma

$$\|f\|_2 = \left(\int_{\mathbb{R}^n} |f|^2 dx\right)^{1/2}.$$

Definición 2.6. (Operador lineal) Sean X, Y dos espacios vectoriales sobre un mismo campo de escalares $\mathbb{K}$ ($\mathbb{R}$ o $\mathbb{C}$). Un *operador lineal* $T : D(T) \subset X \to Y$ es una función que satisface que

$$T(x + y) = T(x) + T(y),\ T(\lambda x) = \lambda T(x),$$

para todo $x, y \in X$, $\lambda \in \mathbb{K}$ ($\mathbb{R}$ o $\mathbb{C}$). Habitualmente se usan las notaciones $T(x)$ y Tx para indicar la acción del operador en un elemento $x \in D(T)$ del dominio.

El siguiente resultado nos da diferentes caracterizaciones para el concepto de continuidad de un operador lineal.

Teorema 2.1. *Sean X y Y dos espacios vectoriales normados y sea $T : X \to Y$ una transformación lineal. Los siguientes afirmaciones son equivalentes:*

1. *T es uniformemente continua;*
2. *T es continua;*
3. *T es continua en 0;*
4. *Existe $K > 0$ tal que $\|Tx\|_Y \leq K$ siempre que $x \in X$ y $\|x\| \leq 1$.*
5. *Existe $K > 0$ tal que $\|Tx\|_Y \leq K\|x\|$ para todo $x \in X$.*

2.2 Problemas inversos

Consideremos un fenómeno físico sobre el cual aplicamos un modelo matemático, tal que dados uno o más datos de entrada en el sistema, se obtiene uno o más datos de salida, donde el sistema de parámetros (conocida también como la regla) en general es una ecuación

diferencial ordinaria o parcial, una ecuación integral, una transformada lineal, etc. Este tipo de problemas a menudo tiene la siguiente estructura,

En ecuaciones diferenciales, los datos de entrada pueden ser condiciones iniciales, condiciones de frontera, dominios. Los parámetros, en este caso son los que afectan directamente a la ecuación y los datos de salida, la solución de la ecuación diferencial.

El estudio de este modelo induce a tres tipos de problemas:

1. **Problema directo:** Se cuenta con los datos de entrada, los parámetros y se desea encontrar la salida del modelo.

2. **Problema de reconstrucción:** Se dispone de los parámetros, los datos de salida y se desea encontrar la entrada del modelo.

3. **Problema de identificación:** Se tiene la entrada y salida del modelo, y se desea recuperar el sistema de parámetros.

Los problemas de reconstrucción e identificación son llamados *problemas inversos*, debido a que se conocen a priori las consecuencias, y a partir de esto, se deben conocer las causas. Formalmente, lo anterior se puede escribir como: Sean X,Y y $\mathscr{P}$ los espacios de las entradas, las salidas y los parámetros respectivamente. Sea $T_p : X \longrightarrow Y$ un operador, entonces los tipos de problemas anteriores, se pueden expresar matemáticamente de la siguiente manera:

1. Dado $p \in \mathscr{P}$ y $x \in X$, encontrar $y \in Y$, talque $y := T_p x$.

2. Dado $p \in \mathscr{P}$ y $y \in Y$, encontrar $x \in X$ tal que $T_p x = y$.

3. Dado $x \in X$ y $y \in Y$, encontrar $p \in \mathscr{P}$ tal que $T_p x = y$.

Una característica de los problemas inversos está vinculada con la noción de problema bien planteado introducida por Hadamard, quien estableció que el modelo matemático de un problema físico debe cumplir con las propiedades de existencia, unicidad y estabilidad de la solución. Una definición más precisa indica:

Definición 2.7. (Problema bien puesto) La ecuación $Tx = y$, con el operador $T : X \longrightarrow Y$ y X, Y espacios de Hilbert, es un problema bien puesto, si cumple las siguientes condiciones

A. El problema tiene solución (Existencia).

B. La solución del problema es única (Unicidad).

C. El problema depende continuamente de los datos (Estabilidad).

Si alguno de los ítems anteriores no se cumple, se dice que es un problema *mal puesto*. Es importante mencionar que un problema es bien puesto equivale a decir que el operador T es biyectivo y que el operador inverso T^{-1} es continuo.

Observación 2.1. El hecho de que $Tx = y$ sea un problema mal puesto, equivale a no poder encontrar x con $T^{-1}y$, debido a que T^{-1} es posible que no exista y si lo hace, el mayor obstáculo es la no continuidad de T^{-1}.

A continuación veamos un ejemplo de un problema inverso, mal puesto.

Ejemplo 2.3. (Integración-Derivación) El problema de la derivación se puede ver, gracias al teorema fundamental del cálculo, como un problema inverso, de la siguiente manera: Dado $g \in C^1[0,1]$ con $g(0) = 0$, encontrar $f \in C[0,1]$ tal que:

$$T[f(x)] := \int_0^x f(t)\mathrm{d}t + g(0) = g(x).$$

Este problema se puede clasificar como un problema de reconstrucción. Veamos que este problema $Tf = g$ es mal puesto. Sean las sucesiones $\{g_n(x)\}_n$ y $\{g_n'(x)\}_n$, definidas por,

$$g_n(x) = g(x) + \frac{1}{n}\sin(n^2x)$$

y

$$g_n'(x) = g'(x) + n\cos(n^2x).$$

De la anterior definición, y teniendo en cuenta la norma $\|\cdot\|_\infty$, se sigue que

$$\|g - g_n\|_\infty = \frac{1}{n} \to 0, \qquad \text{cuando } n \to \infty,$$

lo cual significa que g_n es una buena aproximación de g, mientras que g_n', no lo es, debido a que

$$\|g' - g_n'\|_\infty = n \to \infty, \qquad \text{cuando } n \to \infty.$$

Por lo tanto, T^{-1} no es continuo, lo que significa que $Tf = g$ es un problema mal puesto.

Capítulo 3
Modelos Evolutivos

En este trabajo consideramos la implementación de redes neuronales informadas por la física (PINNs) en la resolución de modelos evolutivos. Dichos modelos, descritos por ecuaciones diferenciales parciales, permiten representar fenómenos que cambian con el tiempo, como la propagación del calor, el movimiento de fluidos o la formación de ondas [48]. Estos procesos son fundamentales para comprender y anticipar el comportamiento de sistemas físicos, biológicos y de ingeniería, pues dependen tanto de las variaciones en el espacio como en el tiempo. Sin embargo, su complejidad hace que resolverlos de manera precisa y eficiente sea un reto, lo que motiva la búsqueda de métodos más flexibles e innovadores.

3.1 Modelo general

Matemáticamente, los modelos evolutivos de ecuaciones en derivadas parciales no lineales y parametrizadas, los cuales consideramos en este trabajo, suelen representarse como

$$\partial_t u(x,t) + \mathcal{N}[u;\lambda](x,t) = 0, \qquad (x,t) \in \Omega \times [0,T], \tag{3.1a}$$

$$u(0,x) = u_0(x), \qquad x \in \Omega, \tag{3.1b}$$

donde $\mathcal{N}[\cdot;\lambda]$ es un operador diferencial no lineal parametrizado por λ, que actúa sobre u, $\Omega \subset \mathbb{R}^n$ representa un dominio acotado, T es el tiempo final, y $u_0 : \Omega \to \mathbb{R}$ es la condición inicial dada. Adicionalmente, consideramos una condición de frontera Dirichlet homogéneas, es decir,

$$\mathcal{B}[u] = 0, \quad t \in [0,T], \quad x \in \partial\Omega. \tag{3.1c}$$

El problema formulado en (3.1a)-(3.1c) se clasifica como un *problema directo*; Esta configuración cubre una amplia variedad de problemas en la física matemática, como leyes de conservación, procesos de difusión, sistemas de advección-difusión y ecuaciones cinéticas [50].

Modelos de mecánica de fluidos

Ecuaciones de Navier-Stokes

Las ecuaciones de Navier-Stokes, fundamentales en matemáticas aplicadas y física, describen la dinámica de los fluidos y son esenciales para modelar fenómenos como el flujo de aire, agua u otros líquidos. Su derivación se sigue de principios fundamentales de la mecánica de fluidos, como se evidencia en diversas referencias clásicas [48, 50]. Estas ecuaciones suponen que el fluido es un medio continuo, y su estado se describe mediante las siguientes funciones,

$$\rho = \rho(x,t), \qquad u = u(x,t), \qquad p = p(x,t),$$

donde $\rho = \rho(x,t)$ representa la densidad, que puede ser constante o variar con la posición y el tiempo. $u = u(x,t)$ es la velocidad del flujo, que depende de una variable espacial $x \in U \subseteq \mathbb{R}^3$ y una variable temporal $t \in \mathbb{R}$. Y $p = p(x,t)$ denota la presión en un punto específico del espacio y tiempo. Cuando se considera que el fluido es incompresible, estas cantidades están relacionadas mediante las ecuaciones de Navier-Stokes incompresibles, comúnmente expresadas como

$$\frac{\partial u}{\partial t} - \nu\Delta u + (u\cdot\nabla)u + \frac{1}{\rho}\nabla p = f, \qquad (x,t) \in \Omega \times [0,T], \tag{3.2a}$$

$$\nabla \cdot u = 0, \qquad x \in \Omega, \tag{3.2b}$$

donde $f = f(x,t)$ representa una fuerza externa que actúa sobre el medio, y $\nu = \frac{\mu}{\rho}$ se conoce como el coeficiente de viscosidad cinemática y se define como la relación entre la viscosidad dinámica μ y la densidad ρ. En este contexto, el operador diferencial $\mathcal{N}[u;\lambda]$ que caracteriza el modelo general evolutivo se define como:

$$\mathcal{N}[u;\lambda] = -\nu\Delta u + (u\cdot\nabla)u, \qquad \lambda = (\nu),$$

donde λ contiene únicamente el parámetro de viscosidad cinemática ν.

3.2 Modelo de Burgers

A partir de las ecuaciones incompresibles de Navier-Stokes (3.2), podemos derivar lo que se conoce como ecuación de Burgers con viscosidad. En ausencia de presión ($\nabla p = 0$) y fuerzas externas ($f = 0$), este modelo describe un flujo que se organiza de manera natural por efecto de sus propias dinámicas en el que las propiedades del fluido evolucionan únicamente bajo la influencia de la viscosidad y las interacciones no lineales inherentes al flujo [61, 63]. Matemáticamente, dicha ecuación se describe como,

$$\frac{\partial u}{\partial t} + u\cdot\nabla u - \nu\Delta u = 0 \tag{3.3}$$

donde el segundo término la convierte en una ecuación no lineal; sin embargo, la ecuación puede transformarse en una ecuación lineal mediante una transformación de Hopf-Cole [47], esto lo convierte en una simplificación que sigue siendo útil para analizar propiedades fundamentales de los fluidos y analizar el desempeño de esquemas numéricos. En una dimensión un problema directo asociado la dicha ecuación se escribe como,

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0, \qquad (x,t) \in \Omega \times [0,T], \tag{3.4a}$$

$$u(0,x) = u_0(x), \qquad x \in \Omega, \tag{3.4b}$$

$$u(t,x) = u_b(t,x), \qquad (x,t) \in \partial\Omega \times (0,T], \tag{3.4c}$$

donde

$$\mathcal{N}[u;\lambda] = \lambda_1 u u_x - \lambda_2 u_{xx} \text{ y } \lambda = (\lambda_1, \lambda_2)\,; \quad (\cdot)_x = \frac{\partial(\cdot)}{\partial x} \quad (\cdot)_{xx} = \frac{\partial^2(\cdot)}{\partial x^2}.$$

Físicamente, este modelo es útil para estudiar la propagación de ondas y el comportamiento de sistemas no lineales en dinámicas simplificadas.

3.3 Modelo KdV

La ecuación Korteweg-de Vries (KdV) es una ecuación diferencial parcial no lineal que describe la propagación de ondas en medios dispersivos. Fue introducida en 1895 por Korteweg y de Vries para modelar ondas en canales poco profundos de agua, y desde su introducción se ha utilizado en una amplia variedad de campos, como óptica no lineal, acústica, física de plasmas y dinámica de fluidos [63, 50]. El modelo general asociado a la ecuación KdV tiene la forma,

$$\frac{\partial u}{\partial t} + \lambda_1 u\frac{\partial u}{\partial x} + \lambda_2\frac{\partial^3 u}{\partial x^3} = 0, \qquad (x,t) \in \Omega \times [0,T], \tag{3.5a}$$

$$u(0,x) = u_0(x), \qquad x \in \Omega, \tag{3.5b}$$

$$u(t,x) = u_b(t,x), \qquad (x,t) \in \partial\Omega \times (0,T], \tag{3.5c}$$

donde $u = u(x,t)$ describe la altura de la onda en un punto del espacio y tiempo. Y $\lambda > 0$ es un parámetro relacionado con la dispersión del medio. El modelo KdV es evolutivo, ya que describe cómo las ondas cambian su forma y posición en el tiempo, y es ampliamente utilizado en hidráulica, acústica y física de plasmas. En este caso, el operador $\mathcal{N}$ viene dado por,

$$\mathcal{N}[u;\lambda] = \lambda_1 u u_x + \lambda_2 u_{xxx}, \quad \lambda = (\lambda_1, \lambda_2)\,; \quad (\cdot)_x = \frac{\partial(\cdot)}{\partial x} \quad (\cdot)_{xxx} = \frac{\partial^3(\cdot)}{\partial x^3}.$$

La ecuación KdV es un modelo particularmente interesante porque, bajo ciertas condiciones, tiene soluciones clásicas conocidas como solitones.[1]. Estas tipo de ondas solitarias, que se mueven sin deformarse, convierten al modelo en uno ideal para probar cómo funcionan y qué tan efectivos son los métodos numéricos aplicados a ecuaciones no lineales.

[1] Un solitón es una onda solitaria que mantiene su forma y velocidad a lo largo del tiempo debido a un equilibrio exacto entre los efectos no lineales y la dispersión del medio.

Capítulo 4
Redes neuronales

En esta sección se presentan los conceptos, términos y definiciones fundamentales que se relacionan con el enfoque principal de este trabajo.

Introducción

La inteligencia artificial, conocida popularmente como (IA), se entiende como el conjunto de métodos y sistemas capaces de procesar información, aprender de la experiencia y tomar decisiones o ejecutar acciones que normalmente asociamos con comportamientos inteligentes en los seres humanos, tales como el reconocimiento de voz, la resolución de problemas o la toma de decisiones [4]. Dentro de este campo, se destaca el aprendizaje automático (AA) el cual constituye una de las áreas más activas, pues se centra en diseñar algoritmos que permiten a los sistemas mejorar su desempeño a partir de la experiencia. En términos prácticos, esto se logra analizando datos y reconociendo patrones, con el fin de predecir resultados o tomar decisiones de manera autónoma [5].

Entre las herramientas más destacadas del aprendizaje automático se encuentran el aprendizaje profundo (deep learning), basado en redes neuronales artificiales. Las cuales están inspiradas en la organización del cerebro humano, y han demostrado ser especialmente útiles para procesar información compleja, como imágenes, audio o texto. Dichas redes están formadas por capas de nodos interconectados; cada nodo aplica una operación matemática a sus entradas, y durante el entrenamiento los pesos de las conexiones se ajustan para reducir el error en las predicciones. Gracias a este proceso iterativo, el modelo va refinando su capacidad de aproximación y se vuelve cada vez más preciso.

4.1 Redes neuronales artificiales

Para entender la estructura compleja de una red neuronal de múltiples capas, es esencial conocer primero la estructura de una sola neurona. Las neuronas y las redes neuronales se han inspirado históricamente en sus equivalentes biológicos. Así, una neurona artificial puede explicarse de la misma manera que una biológica. Conceptualmente, una neurona, ya sea biológica o artificial, actúa como una unidad que recibe varias entradas y decide si debe activarse. Si se activa, transmite una señal a una o más neuronas. Juntas, estas neuronas pueden procesar información demasiado compleja para que una sola lo haga. En las neuronas biológicas, las entradas se reciben a través de las dendritas en forma de sustancias transmisoras. El procesamiento de estas entradas lo realizan los mecanismos biológicos internos de la célula, y la señal se transmite a través del axón. El axón libera nuevas sustancias transmisoras que pueden ser enviadas a otras neuronas [15].
Matemáticamente, una neurona artificial η puede representarse como una función que toma un vector de entrada n dimensional y un término de sesgo $b \in \mathbb{R}$, y produce un valor de salida y. Aquí $\eta : \mathbb{R}^{n+1} \to \mathbb{R}$. Para calcular este valor de salida, se realiza una combinación lineal de las entradas x_i y los pesos asociados w_i, incluyendo el término de sesgo b. Posteriormente, a esta combinación lineal se le aplica una función de activación f_h para obtener el resultado final. En consecuencia, la salida y se calcula como:

$$y = f_h \left(\sum_{i=1}^{n} w_i x_i + b \right) \tag{4.1}$$

La arquitectura de una sola neurona, conocida como perceptrón simple, puede visualizarse como en la siguiente figura 4.1. Si bien las redes neuronales artificiales se inspiran en las

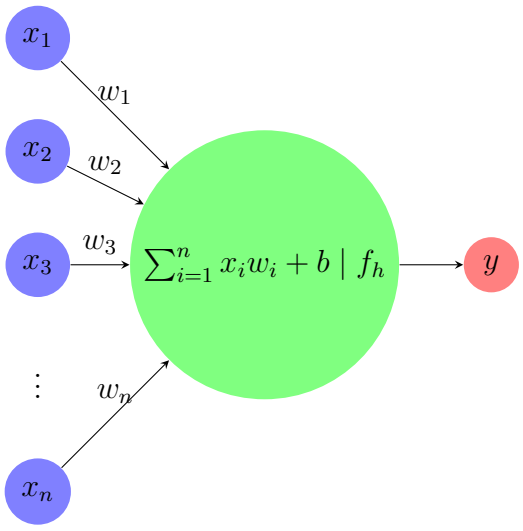


Figura 4.1. Perceptron simple con n entradas, pesos w_i, sesgo b, una neurona y una salida y.

neuronas biológicas y sus redes asociadas, existen diferencias fundamentales entre ambas [15].

En primera instancia, el procesamiento de información en las redes neuronales biológicas es considerablemente más lento en comparación con las redes artificiales [15]. Además, la conectividad en las redes neuronales artificiales es mucho menor que en las biológicas, con diferencias de varios órdenes de magnitud [15].

Otra diferencia importante radica en el modo en que estas redes se entrenan. Mientras que las conexiones entre las neuronas biológicas se fortalecen o debilitan de manera dinámica en función de su actividad, las redes neuronales artificiales suelen entrenarse utilizando un algoritmo conocido como retropropagación [6]. Este método requiere un agente externo para calcular los ajustes necesarios en las conexiones de la red. La retropropagación se explorará en mayor detalle en la sección 4.2.4

4.1.1 Funciones de activación

Antes de enfatizar en las estructuras más complejas de las redes neuronales (RN), es importante analizar la función f_h mencionada en (4.1), conocida como función de activación en el ámbito del aprendizaje automático. Esta función, que suele ser continua y en muchos casos no lineal, se define como $f_h : \mathbb{R} \to \mathbb{R}$. Su propósito es determinar el valor de salida de una neurona y, en consecuencia, decidir si esta se activa o no, produciendo un resultado distinto de cero.

En seguida, se presentan algunos ejemplos comunes de funciones de activación, junto con sus derivadas. Estas derivadas son fundamentales tanto para resolver ecuaciones diferenciales parciales (EDPs) [24] como para entrenar redes neuronales mediante el algoritmo de retropropagación [6].

Ejemplo 4.1. (Sigmoide) La función de activación sigmoide, también conocida como función logística, transforma cualquier entrada en un valor comprendido en el rango $[0, 1]$. Formalmente, esta función se define como $f_{\text{sigmoid}} : \mathbb{R} \to [0, 1]$, su expresión algebraica y su derivada vienen dadas por,

$$f_{\text{sigmoide}}(x) = \frac{1}{1 + e^{-x}}, \quad \frac{df_{\text{sigmoid}}}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2}. \tag{4.2}$$

La función sigmoide se usó mucho en las primeras redes neuronales, pero hoy en día se emplea menos porque puede causar problemas al entrenar, como gradientes muy pequeños y neuronas que dejan de aprender[22].

Ejemplo 4.2. (Tangente hiperbólica tanh**)** La función tangente hiperbólica $f_{\text{hyperbolic}} : \mathbb{R} \to [-1, 1]$ también se usa como una función de activación para redes neuronales artificiales. La cual viene dada por,

$$f_{\text{hyperbolic}}(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.3}$$

Es similar a la sigmoide, pero su rango es simétrico en torno al origen, esto la hace

adecuada para ciertas aplicaciones. Su derivada con respecto a x es

$$\frac{df_{\text{hyperbolic}}}{dx} = \frac{4e^{2x}}{\left(e^{2x}+1\right)^2} = 1 - \tanh(x)^2 \tag{4.4}$$

Observemos que la tangente hiperbólica está presente en la derivada de la función, lo que significa que calcular las derivadas de segundo orden y superiores puede hacerse de manera similar. Esto será especialmente útil al representar las soluciones de ecuaciones diferenciales parciales con redes neuronales y diferenciar las redes neuronales para representar las derivadas de orden superior [24].

Ejemplo 4.3. (ReLU) La función rectificadora lineal (ReLU) es una de las funciones de activación más populares en el aprendizaje profundo. $f_{\text{ReLU}} : \mathbb{R} \to \{x \in \mathbb{R} : x \geq 0\}$, la cual se define como:

$$f_{ReLU}(x) = \begin{cases} x, & \text{si } x \geq 0 \\ 0, & \text{en otro caso} \end{cases} \tag{4.5}$$

Observe que una de las mayores ventajas de usar esta función de activación es que requiere un esfuerzo computacional mínimo para evaluarla. Comparado con evaluar las expresiones de la ecuación (4.2) o (4.3), es menos costoso, especialmente cuando se tienen que evaluar varias funciones de activación en muchas capas en una red neuronal profunda. La derivada de la función ReLU también es simple de evaluar, lo que puede ayudar con el proceso de entrenamiento de la RN, nótese que,

$$\frac{df_{ReLU}}{dx} = \begin{cases} 1, & \text{si } x > 0 \\ 0, & \text{si } x < 0 \\ \text{indefinido} & \text{cuando } x = 0 \end{cases} \tag{4.6}$$

Sin embargo, por simplicidad, a veces se establece que la derivada de la función ReLU es cero para $x = 0$ para evitar que la función sea indefinida en ese punto. El problema con esta derivada es que su derivada con respecto a x es cero para todos los valores de x, por lo que no se puede usar cuando se requiere calcular derivadas de orden superior. Esto será un problema al considerar PINNs [24, 25].

Observación 4.1. (Sobre funciones de activación en la resolución de EDPs) Al resolver ecuaciones diferenciales parciales (EDPs) con redes neuronales, es fundamental que las funciones de activación sean diferenciables de acuerdo al orden de las derivadas requeridas. Por ejemplo, la función ReLU solo es diferenciable una vez (excepto en cero), lo que limita su uso en EDPs que requieren derivadas de orden superior. Para mejorar esta limitación, se han utilizado modificaciones de ReLU, por ejemplo la función polynomial-ReLU:

$$f_{\text{polynomial-ReLU}}(x) = \begin{cases} x^{n+1}, & \text{si } x \geq 0 \\ 0, & \text{en otro caso.} \end{cases} \tag{4.7}$$

Aquí, $n \in \mathbb{N}$ se elige según el orden más alto de derivadas requeridas. Aunque esta función mejora la representación de soluciones, incrementa el esfuerzo computacional y puede causar problemas de gradientes durante el entrenamiento [2, 6], los cuales se especifican más adelante en 4.2.4.

4.1.2 Perceptrones multicapa

Partiendo del concepto de una sola neurona o perceptrón simple, estas se pueden combinar para formar lo que se llama una *capa*. En este caso, los datos iniciales, denominados *entrada*, se proporcionan a la red neuronal. Cada uno de estos datos se multiplica por un valor específico llamado peso, que define la conexión con otras unidades dentro de la red. Luego, se agrega un ajuste llamado término de sesgo antes de enviar el resultado a una capa intermedia, conocida como *capa oculta* (ver figura 4.2). En cada neurona, las entradas y los términos de sesgo se suman, y se aplica una función de activación al resultado para producir la entrada para la siguiente capa. Si la siguiente capa es la llamada capa de salida, las entradas a cada neurona en dicha capa se suman y, dependiendo del tipo de salida deseada, podría ser necesario aplicar una función, como la función softmax[1], que se utiliza frecuentemente para tareas de clasificación [2, 4].

De manera más formal, una red neuronal con una sola capa oculta puede describirse como una función $\Phi : \mathbb{R}^m \longrightarrow \mathbb{R}^n$ que envía una entrada $\mathbf{x} \in \mathbb{R}^m$ hacia una salida $\mathbf{y} \in \mathbb{R}^n$. Este mapeo se define mediante las matrices de pesos $\mathcal{W}_1 \in \mathbb{R}^{d\times m}$ y $\mathcal{W}_2 \in \mathbb{R}^{n\times d}$, junto con los vectores de sesgos correspondientes $b_1 \in \mathbb{R}^d$ y $b_2 \in \mathbb{R}^n$. Aquí, $d \in \mathbb{N}$ representa el número de neuronas en la capa oculta, mientras que f_h denota la función de activación, la cual se aplica de manera componente a componente[2] a su entrada vectorial. La salida $\mathbf{y}$ se calcula mediante las siguientes tres etapas: primero, se toma la transformación afín inicial $\mathbf{z} = \mathcal{W}_1\mathbf{x} + b_1$. En seguida se aplica la función de activación, $\mathbf{h} = f_h(\mathbf{z})$, donde $\mathbf{h} \in \mathbb{R}^d$ representa las activaciones en la capa oculta. Finalmente, se aplica la transformación afín final, $\mathbf{y} = \mathcal{W}_2\mathbf{h} + b_2$. En consecuencia, la expresión que describe la salida de la red está dada por:

$$\mathbf{y} = \mathcal{W}_2 f_h\left(\mathcal{W}_1\mathbf{x} + b_1\right) + b_2, \tag{4.8}$$

[1]La **función softmax** es una transformación utilizada en redes neuronales para convertir los valores de la salida de la última capa en una distribución de probabilidad sobre las clases. Se define como:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \quad \forall i = 1, \ldots, n$$

donde z_i son las salidas de los nodos de la última capa. En el contexto de clasificación multiclase, la función softmax permite asignar a cada clase una probabilidad, facilitando la interpretación del modelo y la selección de la clase con mayor probabilidad como la predicción final.

[2]Con esto nos referimos a que f_h actúa de forma independiente en cada componente de un vector de entrada. Es decir, dado un vector $\mathbf{x} \in \mathbb{R}^k$, definido como $\mathbf{x} = (x_1, \ldots, x_k)$, la función f_h produce:

$$f_h(\mathbf{x}) = \left(f_h(x_1), \ldots, f_h(x_k)\right).$$

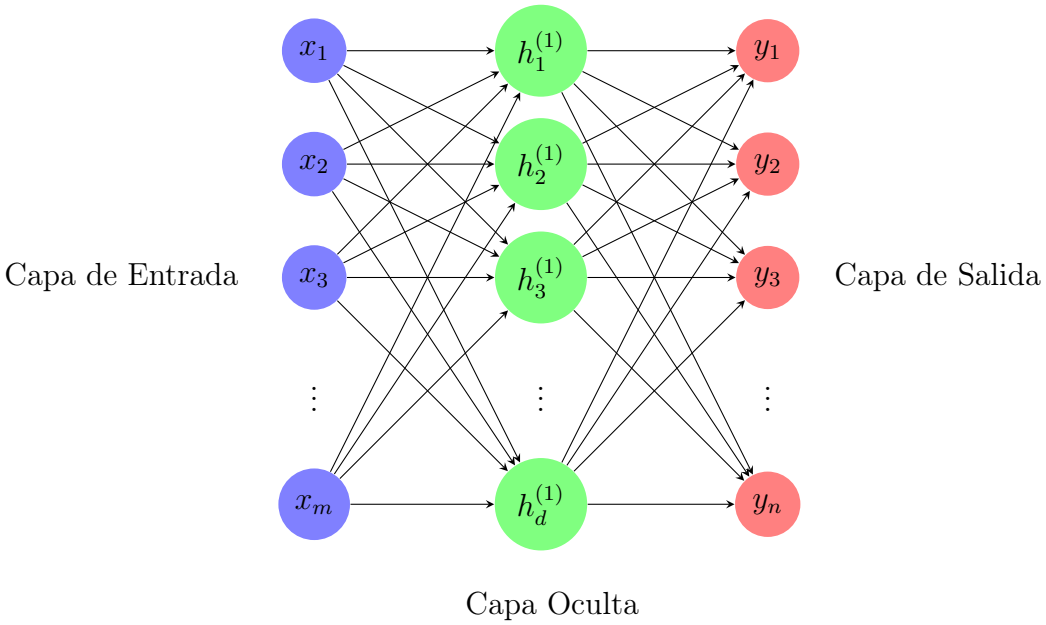


Figura 4.2. Tipo de red neuronal densa, con m entradas, una capa oculta y n salidas.

donde el encadenamiento representa la secuencia de transformaciones que ocurre a medida que la información fluye a través de las capas de la red. En este contexto, la red neuronal Φ pertenece al conjunto $\mathcal{R}$ de redes parametrizadas y queda completamente caracterizada por los pares matriz-vector asociados a cada capa, $(\mathcal{W}_1, b_1)$ para la capa oculta y $(\mathcal{W}_2, b_2)$ para la capa de salida. Observemos que, estas definiciones suponen que las dimensiones de las matrices y vectores están correctamente especificadas, lo cual verifica la validez de la ecuación (4.8).

Observación 4.2. (Sobre la notación) Una red neuronal $\Phi^1 : \mathbb{R}^m \longrightarrow \mathbb{R}^n$, con una sola capa oculta, puede expresarse como

$$\Phi^1 := \left[\left(\mathcal{W}_1^1, b_1^1\right), \left(\mathcal{W}_2^1, b_2^1\right) \right],$$

donde el primer par $(\mathcal{W}_1^1, b_1^1)$ corresponde a la capa oculta $\mathbf{z} = \mathcal{W}_1^1\mathbf{x} + b_1^1$ y el segundo par $(\mathcal{W}_2^1, b_2^1)$ corresponde a la capa de salida $\mathbf{y} = \mathcal{W}_2^1\mathbf{h} + b_2^1$. Esta notación es una lista ordenada que refleja el flujo secuencial de los datos; cada par $(\mathcal{W}_k, b_k)$ representa una transformación afín, y el orden de los pares corresponde a las capas: entrada $\rightarrow$ oculta $\rightarrow$ salida.

De acuerdo con lo anterior, se puede observar que la extensión de redes neuronales con una sola capa oculta a redes neuronales con dos capas ocultas se puede realizar de manera directa mediante un procedimiento conocido como concatenación[3]. En términos intuitivos, si

[3]Matemáticamente, si consideramos dos redes neuronales f_1 y f_2, donde $f_1 : \mathbb{R}^n \to \mathbb{R}^m$ y $f_2 : \mathbb{R}^m \to \mathbb{R}^p$, la concatenación de estas redes se puede expresar como una nueva red f tal que $f(x) = f_2(f_1(x))$ para $x \in \mathbb{R}^n$. Aquí, f_1 toma una entrada x y produce una salida $f_1(x)$, que luego se utiliza como entrada para f_2, resultando en la salida final $f(x)$.

la salida de una red neuronal se utiliza como entrada para otra red neuronal, esto se puede modelar como una red neuronal con dos capas ocultas, siempre y cuando las dimensiones de sus entradas y salidas sean compatibles.

Definición 4.1. Sean

$$\Phi^1 := \left[\left(\mathcal{W}_1^1, b_1^1\right), \left(\mathcal{W}_2^1, b_2^1\right)\right] \quad \text{y} \quad \Phi^2 := \left[\left(\mathcal{W}_1^2, b_1^2\right), \left(\mathcal{W}_2^2, b_2^2\right)\right],$$

dos redes neuronales de una sola capa oculta, entonces su concatenación se define como

$$\Phi^2 \circ \Phi^1 = \left[\left(\mathcal{W}_1^1, b_1^1\right), \left(\mathcal{W}_1^2\mathcal{W}_2^1, \mathcal{W}_1^2 b_2^1 + b_1^2\right), \left(\mathcal{W}_2^2, b_2^2\right)\right]. \tag{4.9}$$

Note que, de acuerdo con la Observación 4.2, la red neuronal $\Phi^2 \circ \Phi^1$ tendrá dos capas ocultas; la primera es la misma que la de Φ^1, representada por $(\mathcal{W}_1^1, b_1^1)$. La segunda capa oculta es una combinación de las capas ocultas de Φ^1 y Φ^2, representada por $(\mathcal{W}_1^2\mathcal{W}_2^1, \mathcal{W}_1^2 b_2^1 + b_1^2)$. Siguiendo esta misma metodología, se construyen las redes neuronales con muchas capas ocultas, las cuales pueden definirse de manera similar.

Definición 4.2. (Red neuronal con L capas ocultas) Una red neuronal $\Phi \in \mathcal{R}$ con L capas ocultas puede definirse como una aplicación $\Phi : \mathbb{R}^m \to \mathbb{R}^n$, dada por las tuplas matriz-vector

$$\Phi := \left[\left(\mathcal{W}_1, b_1\right), \ldots, \left(\mathcal{W}_{L+1}, b_{L+1}\right)\right], \tag{4.10}$$

que satisfacen la ecuación

$$\mathbf{y} := \mathcal{W}_{L+1} f_h\left(\mathcal{W}_L f_h\left(\ldots f_h\left(\mathcal{W}_1 \mathbf{x} + b_1\right)\ldots\right) + b_L\right) + b_{L+1}, \tag{4.11}$$

para una función de activación no especificada f_h. El número de neuronas en cada capa oculta se denota como d_i para $i \in \{1, \ldots, L\}$. Las matrices de pesos están dadas por $\mathcal{W}_1 \in \mathbb{R}^{d_1 \times m}$, $\mathcal{W}_j \in \mathbb{R}^{d_j \times d_{j-1}}$ para $j \in \{2, \ldots, L\}$, y $\mathcal{W}_{L+1} \in \mathbb{R}^{n \times d_L}$, mientras que los términos de sesgo son $b_k \in \mathbb{R}^{d_k}$ para $k \in \{1, \ldots, L\}$ y $b_{L+1} \in \mathbb{R}^n$.

Usando la definición anterior 4.2 es posible diseñar lo que se conoce como redes neuronales de avance (feedforward[4]) para cualquier número de capas $L \in \mathbb{N}$. En este contexto, la red neuronal multicapa puede visualizarse como un grafo, donde los círculos representan las neuronas individuales, las flechas denotan conexiones entre las neuronas en términos de los pesos para la siguiente capa, y las neuronas se disponen en columnas que representan lo que se conoce como capas. En la figura 4.3 puede visualizarse una red neuronal con dos capas ocultas, pero puede extenderse fácilmente a más capas.

Definición 4.3. La profundidad $\mathcal{D}$ de una red neuronal $\Phi \in \mathcal{R}$ con L capas ocultas y una capa de salida se define como la suma del número de capas ocultas y la capa de salida, $\mathcal{D}(\Phi) = L + 1$.

[4]Redes donde la información fluye en una sola dirección: desde la capa de entrada, a través de capas ocultas (si existen), hasta la capa de salida, sin ciclos o retroalimentación.

Proposición 4.1. *El número de parámetros $\mathcal{P}(\cdot)$ de una red neuronal $\Phi \in \mathcal{R}$ se define como el número total de entradas en la representación matriz-vector de la red según la definición 4.2, puede calcularse a partir del número de neuronas d_i en cada capa oculta i, considerando además que la capa de entrada tiene $d_0 = m$ neuronas y la capa de salida tiene $d_{L+1} = n$ neuronas, para $i \in \{0, 1, \ldots, L, L+1\}$. El número total de parámetros se puede calcular como*

$$\mathcal{P}(\Phi) = \sum_{i=1}^{L+1} (d_i \cdot d_{i-1} + d_i) \tag{4.12}$$

Demostración. Observemos que capa $i \in \mathbb{N}$ tiene d_i neuronas y la capa anterior tiene d_{i-1} neuronas; dado que cada nodo de la capa anterior está conectado a cada neurona de la siguiente capa, tendrá d_i conexiones, según el principio de multiplicación. Esto se repite d_{i-1} veces para cada neurona de la capa, lo que da el número de entradas de la matriz correspondiente $\mathcal{W}_{i-1}$. Además, para cada capa existen d_i sesgos, los cuales también deben sumarse, ya que están todos conectados a la siguiente capa. Repetir este procedimiento para todas las capas da el número total de parámetros.

La operación de concatenación, que se definió anteriormente para combinar la salida de una red neuronal con una sola capa oculta con la entrada de otra red neuronal con una sola capa oculta para crear una red neuronal con dos capas ocultas, ahora puede definirse para casos más generales.

□

Definición 4.4. Dadas dos redes neuronales $\Phi_1, \Phi_2 \in \mathcal{R}$ con profundidades L_1 y L_2 respectivamente

$$\begin{aligned}\Phi^1 &= \left[\left(\mathcal{W}_1^1, b_1^1\right), \ldots, \left(\mathcal{W}_{L_1}^1, b_{L_1}^1\right), \left(\mathcal{W}_{L_1+1}^1, b_{L_1+1}^1\right)\right] \\ \Phi^2 &= \left[\left(\mathcal{W}_1^2, b_1^2\right), \ldots, \left(\mathcal{W}_{L_2}^2, b_{L_2}^2\right), \left(\mathcal{W}_{L_2+1}^2, b_{L_2+1}^2\right)\right]\end{aligned}$$

su concatenación $\Phi_2 \circ \Phi_1$ se define como

$$\begin{aligned}\Phi^2 \circ \Phi^1 = \Big[&\left(\mathcal{W}_1^1, b_1^1\right), \ldots, \left(\mathcal{W}_{L_1}^1, b_{L_1}^1\right), \\ &\left(\mathcal{W}_1^2 \mathcal{W}_{L_1+1}^1, \mathcal{W}_1^2 b_{L_1+1}^1 + b_1^2\right), \left(\mathcal{W}_2^2, b_2^2\right), \ldots, \left(\mathcal{W}_{L_2+1}^2, b_{L_2+1}^2\right)\Big]\end{aligned}$$

asumiendo que las dimensiones son compatibles para las multiplicaciones de matrices.

Si denotamos una transformación afín como $\Psi_i^\Phi \in C\left(\mathbb{R}^{d-1}, \mathbb{R}^d\right), x \mapsto \mathcal{W}_i x + b_i$, donde los pares $(\mathcal{W}_i, b_i)$ corresponden a la representación matriz-vector de la red neuronal $\Phi \in \mathcal{N}$ según la definición 4.2, se puede definir lo que comúnmente se denomina (modelo feedforward o realización) de una red neuronal.

Definición 4.5. La realización de redes neuronales con más de una capa oculta también se conoce como perceptrón multicapa [6].

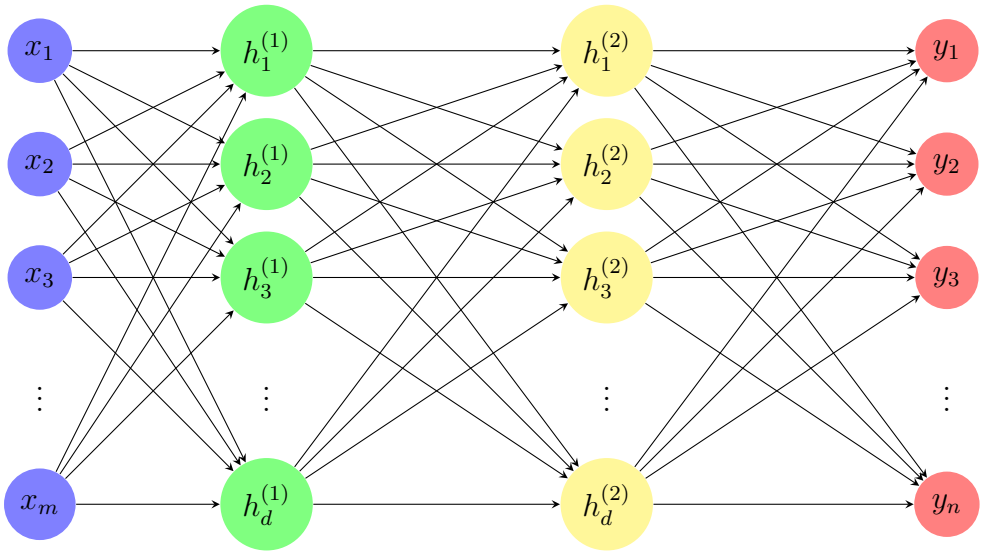


Figura 4.3. Red neuronal, con m entradas, dos capas ocultas y n salidas.

Observación 4.3. Finalmente notemos si una función de activación $f_h : \mathbb{R} \to \mathbb{R}$ es lineal en una red neuronal multicapa, entonces ésta es equivalente a una red con solo capa de entrada. En efecto, dada una aplicación lineal $f : V \to W$ satisface $f(x+y) = f(x) + f(y)$ y $f(cx) = cf(x)$. Para una red $\phi = [(A_1, b_1), (A_2, b_2)]$ con f lineal, se tiene

$$\begin{aligned} & y = A_2 f\left(A_1 x + b_1\right) + b_2 \\ \Leftrightarrow\ & y = A_2(f A_1 x + f b_1) + b_2 \\ \Leftrightarrow\ & y = (A_2 f A_1)x + (A_2 f b_1 + b_2) \end{aligned}$$

Esto equivale a una transformación afín $\hat{A}x + \hat{b}$, correspondiente a una red monocapa.

4.2 Entrenamiento de redes neuronales

Hasta este punto sabemos que una red neuronal simplemente es una función que satisface la definición 4.2, para aplicarla a un problema específico, es necesario determinar los pesos y sesgos adecuados que resuelvan dicho problema. Este proceso se denomina *entrenamiento de la red neuronal* y generalmente se realiza mediante dos métodos clave, el *descenso de gradiente* y la *retropropagación*, los cuales se explicarán en esta sección.

Antes de explicar qué significa entrenar una red neuronal, es preciso definir qué es un problema de aprendizaje.

Definición 4.6. Un problema de aprendizaje supervisado consta de los siguientes tres componentes [14],

1. **Datos de entrenamiento:** Se dispone de un conjunto de vectores de entrada $\mathbf{x}$, obtenidos (por ejemplo) de experimentos o simulaciones, junto con sus correspondientes salidas etiquetadas. Cada vector $\mathbf{x}$ forma parte del conjunto de entrenamiento.

2. **Supervisor:** Existe una función (o procedimiento) σ que, dada una entrada $\mathbf{x}$, proporciona la salida deseada $\mathbf{y} = \sigma(\mathbf{x})$. En este enfoque, consideramos que dicha función representa el conocimiento subyacente que queremos aproximar.

3. **Algoritmo de aprendizaje:** Un mecanismo que, a partir de la información contenida en los pares $(\mathbf{x}, \mathbf{y})$, es capaz de implementar un conjunto de funciones $f(\mathbf{x}, \alpha)$, donde $\alpha \in A$ representa el espacio de parámetros ajustables (pesos y sesgos).

Observación 4.4. En este contexto, un problema de aprendizaje no supervisado se caracteriza por la ausencia de salidas etiquetadas. En este caso, se dispone únicamente de un conjunto de vectores de entrada $\mathbf{x}$, y el objetivo es descubrir patrones o estructuras subyacentes en los datos, como agrupamientos o reducciones de dimensionalidad, sin la guía de un supervisor que indique la salida deseada.

Definición 4.7. (Funciones de pérdida) Para evaluar el desempeño de una función $f(\mathbf{x}, \alpha)$, según la definición anterior, es necesario cuantificar qué tan bien dicha función aproxima la salida proporcionada por el supervisor. Para ello se define la **función de pérdida**.

Definición 4.8. Una función de pérdida $\mathcal{L}(\cdot, \cdot)$ es un funcional que retorna un número real dado un valor de salida $\mathbf{y}$ (proporcionado por el supervisor) y la correspondiente salida predicha $f(\mathbf{x}, \alpha)$, con $\alpha \in A$. La función de pérdida debe elegirse de manera que su minimización corresponda a minimizar la diferencia entre la salida del supervisor y la salida predicha. Algunos ejemplos comunes son:

- **(Pérdida caracteristica).** La función de pérdida 0-1, $\mathcal{L}_{0-1}$, se define como:

$$\mathcal{L}_{0-1}(\mathbf{y}, f(\mathbf{x}, \alpha)) = \begin{cases} 0, & \text{si } \mathbf{y} = f(\mathbf{x}, \alpha) \\ 1, & \text{si } \mathbf{y} \neq f(\mathbf{x}, \alpha) \end{cases}$$

  Esta función se utiliza comúnmente en problemas de clasificación, donde tanto la salida del supervisor como la predicha son variables categóricas [6].

- **(Perdida cuadrática media MSE).** La función de pérdida de error cuadrático medio es una de las más empleadas en tareas de regresión. Consiste en calcular el promedio de los cuadrados de las diferencias entre los valores esperados y las predicciones de la red:

$$\mathcal{L}_{MSE}(\mathbf{y}, f(\mathbf{x}, \alpha)) = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{y_i} - f(\mathbf{x}_i, \alpha))^2$$

  donde $\mathbf{y} \in \mathbb{R}$ y $f(\mathbf{x}, \alpha) \in \mathbb{R}$. N es el número de ejemplos de entrenamiento o número de datos etiquetados disponibles.

Observación 4.5. En el contexto de resolver ecuaciones diferenciales parciales (EDP), este planteamiento se traduce en que, dada una EDP y un conjunto de condiciones iniciales y de contorno (las entradas) junto con las soluciones correspondientes (las salidas), se busca entrenar una red neuronal que aproxime la función solución de la EDP. De esta manera, el proceso de entrenamiento consiste en ajustar los parámetros de la red para minimizar la discrepancia entre la solución predicha y la solución deseada.

4.2.1 Algoritmos de optimización

Recordemos que el propósito del entrenamiento de una red neuronal consiste en ajustar sus parámetros, pesos y sesgos de modo que se minimice una función de pérdida que cuantifica el error entre las predicciones de la red y los "valores reales". Matemáticamente, dado un conjunto de datos compuesto por N pares entrada-salida (x_i, y_i), con $i = 1, \ldots, N$, y una red neuronal denotada por $\Phi_a(\cdot, \theta)$, donde $\theta \in \Theta$ representa el conjunto de parámetros del modelo, el objetivo del entrenamiento es resolver el siguiente problema de minimización:

$$\theta^* = \arg\min_{\theta} \mathcal{L}\left(y_i, \Phi_a(x_i, \theta)\right), \tag{4.13}$$

donde $\mathcal{L}(\cdot, \cdot)$ representa la función de pérdida. En este contexto, a continuación exploramos algunos algoritmos de optimización, los cuales desempeñan un papel muy importante ya que permiten aproximar la solución θ^* del problema anterior (4.13) mediante métodos iterativos que actualizan los parámetros en la dirección que reduce la pérdida.

4.2.2 Descenso de Gradiente estocástico

El descenso de gradiente se basa en la propiedad de que una función disminuye más rápido en la dirección de su gradiente negativo [6]. Usando esta información, dado un conjunto de parámetros Θ podemos formular una regla de actualización para los parámetros. La cual se realiza como:

$$\theta_{\text{new}} := \theta_{\text{old}} - \eta \nabla \mathcal{L}\left(\mathbf{y}, \Phi\left(x; \theta_{\text{old}}\right)\right) \tag{4.14}$$

donde $\theta_{\text{old}}, \theta_{\text{new}} \in \Theta$ son los parámetros de la red neuronal, los cuales se presentan como tuplas matriz-vector $\theta_i := (\mathcal{W}_i, b_i)$ para cada capa $i = 1, \ldots, L+1$. La constante $\eta \in \mathbb{R}$ se conoce como la tasa de aprendizaje y ajusta cuánto se actualizan los parámetros θ_{old} y la función de pérdida $\mathcal{L}$, junto con su par de entrada-salida especificado del conjunto de entrenamiento $(\mathbf{x}, \mathbf{y})$.

El descenso de gradiente se actualiza una vez que se han evaluado todos los datos de entrenamiento, lo cual puede ser muy lento ya que se pueden necesitar muchas actualizaciones si el conjunto de entrenamiento es grande, por lo que a menudo se utiliza una variación de este método, llamada *descenso de gradiente estocástico*[6]. En el descenso de gradiente estocástico, todo el conjunto de datos de entrenamiento se divide en conjuntos más pequeños, llamados

lotes, y el descenso de gradiente se actualiza para cada lote de datos de entrenamiento, antes de continuar con el siguiente lote. Cada recorrido por todos los lotes se llama una época.

El descenso de gradiente estocástico (SGD) sigue siendo una técnica ampliamente utilizada para la optimización de los parámetros en redes neuronales, destacándose entre sus variantes el optimizador Adam, por su eficiencia y robustez en diversos escenarios de entrenamiento [6]. No obstante, existen otros enfoques igualmente relevantes, como el algoritmo BFGS de memoria limitada (L-BFGS), el cual resulta útil en problemas donde se requiere una mayor precisión en la aproximación del mínimo [6, 10].

4.2.3 Diferenciación automática

Antes de describir la retropropagación, el procedimiento mediante el cual se ajustan los parámetros de una red neuronal minimizando una función de pérdida, conviene entender cómo se obtienen las derivadas de manera precisa y eficiente; este proceso se denomina diferenciación automática. A diferencia de la diferenciación numérica (por ejemplo, diferencias finitas) o la simbólica (derivación manual), la diferenciación automática autodiff se basa en descomponer cualquier función compleja como una red neuronal compuesta por múltiples capas y operaciones en una secuencia de operaciones elementales, cuyas derivadas son conocidas [2, 6]. Sea $y = f(x)$ un algoritmo general, que puede verse como una composición de varias funciones más simples $f = f_n \circ f_{n-1} \circ \cdots \circ f_2 \circ f_1$, y consideremos que estas composiciones pueden escribirse en términos de variables intermedias, es decir,

$$\begin{aligned} w_0 &= x, \\ w_1 &= f_1(w_0), \\ w_2 &= f_2(w_1), \\ &\vdots \\ w_n &= f_n(w_{n-1}) = y. \end{aligned}$$

El proceso para calcular $\frac{\partial y}{\partial x}$ mediante diferenciación automática en *modo directo* (forward mode), en términos de las variables intermedias w_i y las correspondientes w_i', consiste en propagar una semiderivada desde la entrada:

$$\begin{aligned} w_0' &= \frac{\partial x}{\partial x} = 1, \\ w_1' &= \frac{\partial w_1}{\partial w_0}\, w_0', \\ w_2' &= \frac{\partial w_2}{\partial w_1}\, w_1', \\ &\vdots \\ w_n' &= \frac{\partial w_n}{\partial w_{n-1}}\, w_{n-1}', \end{aligned}$$

de modo que $w'_n = \frac{\partial y}{\partial x}$. En cambio, en *modo inverso* (reverse mode) el cual es la base de la retropropagación se inicia la derivación en la salida y se retropropaga hacia la entrada:

$$\begin{aligned}
\bar{w}_n &= \frac{\partial y}{\partial w_n} = 1, \\
\bar{w}_{n-1} &= \bar{w}_n \frac{\partial w_n}{\partial w_{n-1}}, \\
\bar{w}_{n-2} &= \bar{w}_{n-1} \frac{\partial w_{n-1}}{\partial w_{n-2}}, \\
&\vdots \\
\bar{w}_0 &= \bar{w}_1 \frac{\partial w_1}{\partial w_0} = \frac{\partial y}{\partial x}.
\end{aligned}$$

Aquí, $\bar{w}_i = \frac{\partial y}{\partial w_i}$. Este esquema es especialmente eficiente cuando la función de interés tiene pocas salidas (por ejemplo, la pérdida escalar) y muchos parámetros o entradas, pues computa el gradiente completo en un solo paso de retropropagación.

En resumen, la diferenciación automática, ya sea en modo directo o inverso, aprovecha la descomposición en bloques elementales y la regla de la cadena para calcular derivadas exactas sin recurrir a trucos numéricos ni a derivaciones simbólicas completas.

Una pregunta natural es: ¿Cómo se pueden obtener o implementar en la práctica este tipo de técnicas como diferenciación automática? La respuesta rápida es mediante las bibliotecas de aprendizaje automático como TensorFlow o PyTorch [28, 29]. Ambas bibliotecas sobre Python, las cuales roporcionan mecanismos eficientes y optimizados para realizar diferenciación automática en modo inverso, permitiendo calcular gradientes de funciones complejas definidas por grafos computacionales. En PyTorch, por ejemplo, esto se logra mediante un sistema dinámico de grafos conocido como *define-by-run*, que construye el grafo computacional en tiempo de ejecución y facilita el uso interactivo.

4.2.4 Retropropagación (Backpropagation)

Como mencionamos anteriormente, para aplicar un algoritmo de optimización como el descenso del gradiente, necesitamos calcular las derivadas parciales de la función de pérdida $\mathcal{L}(\mathbf{y}, f(\mathbf{x}, \theta))$ con respecto a cada uno de los parámetros $\theta_i := (\mathcal{W}_i, b_i)$, para $i = 1, \ldots, L+1$, correspondientes a cada una de las L capas ocultas más la capa de salida.

Consideremos entonces una red neuronal del tipo perceptrón multicapa, o una realización-a como la descrita en la definición 4.1. Esta se puede expresar como una composición de funciones de la siguiente forma:

$$\mathcal{F}_a^\Phi := \Psi_{L+1}^\Phi \circ a \circ \Psi_L^\Phi \circ a \circ \cdots \circ a \circ \Psi_1^\Phi, \tag{4.15}$$

donde cada Ψ_i^Φ representa una transformación afín (una combinación lineal más un sesgo) seguida de una función de activación a.

Siguiendo lo explicado en la sección 4.2.3 sobre diferenciación automática, la derivada parcial de la función de pérdida respecto a los parámetros de cada capa se puede obtener aplicando la regla de la cadena. En particular, la derivada parcial de $\mathcal{L}(\mathbf{y}, f(\mathbf{x}, \theta))$ con respecto a los parámetros θ_i en la capa i-ésima se puede descomponer como:

$$\frac{\partial \mathcal{L}}{\partial \Psi_i^\Phi} = \frac{\partial \mathcal{L}}{\partial y} \cdot \frac{\partial y}{\partial \Psi_{L+1}^\Phi} \cdot \frac{\partial \Psi_{L+1}^\Phi}{\partial a} \cdot \frac{\partial a}{\partial \Psi_L^\Phi} \cdot \frac{\partial \Psi_L^\Phi}{\partial a} \cdot \frac{\partial a}{\partial \Psi_{L-1}^\Phi} \cdots \frac{\partial \Psi_{i+1}^\Phi}{\partial a} \cdot \frac{\partial a}{\partial \Psi_i^\Phi}. \tag{4.16}$$

En esta expresión, cada $\Psi_i^\Phi = \mathcal{W}_i \mathbf{x} + b_i$ representa una transformación afín en la i-ésima capa, donde $\mathbf{x}$ es la entrada a esa capa. Las derivadas parciales se entienden con respecto a los parámetros $\mathcal{W}_i$ y b_i. Además, dado que la red incluye funciones de activación a, también es necesario calcular sus derivadas.

El procedimiento de retropropagación comienza calculando la derivada de la función de pérdida con respecto a la salida final de la red, y luego se propagan estas derivadas hacia atrás, capa por capa. Para evitar calcular varias veces los mismos valores, se introducen variables auxiliares llamadas δ^i, que se actualizan de forma recursiva según el siguiente esquema:

$$\begin{aligned}
\delta^{L+1} &:= \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \Psi_{L+1}^\Phi}, \\
\delta^{L} &:= \frac{\partial \Psi_{L+1}^\Phi}{\partial a} \cdot \frac{\partial a}{\partial \Psi_L^\Phi} \cdot \delta^{L+1}, \\
&\vdots \\
\delta^{i} &:= \frac{\partial \Psi_{i+1}^\Phi}{\partial a} \cdot \frac{\partial a}{\partial \Psi_i^\Phi} \cdot \delta^{i+1}, \\
&\vdots \\
\delta^{1} &:= \frac{\partial \Psi_{2}^\Phi}{\partial a} \cdot \frac{\partial a}{\partial \Psi_1^\Phi} \cdot \delta^{2}.
\end{aligned}$$

Este algoritmo, como se mencionó antes, se llama *retropropagación* porque inicia desde la parte final de la red (la salida) y va *propagando hacia atrás* la información de los gradientes, capa por capa. Este mecanismo es fundamental para entrenar redes neuronales mediante métodos de optimización basados en gradientes, ya que permite actualizar de manera eficiente todos los parámetros del modelo.

En resumen, el proceso completo de entrenamiento consiste en aplicar repetidamente este esquema: calcular la pérdida, propagar los gradientes hacia atrás usando retropropagación y actualizar los parámetros mediante un algoritmo de optimización (como descenso de gradiente o alguno de sus variantes). Este ciclo se repite durante un número determinado de *épocas* o hasta que la función de pérdida alcance un valor suficientemente bajo, indicando que el modelo ha aprendido adecuadamente la tarea deseada.

Problemas comunes en el entrenamiento de redes neuronales

El proceso de entrenamiento de una red neuronal profunda se ve usualmente afectado por diversos problemas que comprometen su convergencia y generalización [5, 7]. El problema posiblemente más común es el de subajuste (*underfitting*) y sobreajuste (*overfitting*) de los datos de entrenamiento. El subajuste ocurre cuando la red neuronal no tiene suficiente complejidad para captar patrones complejos, por lo que su desempeño es deficiente en cualquier conjunto de datos. El sobreajuste es el problema inverso; la red es demasiado compleja y "memoriza" los datos de entrenamiento en lugar de aprender patrones generales, por lo que no puede generalizar a datos nuevos. Para mitigar ambos efectos existen varias soluciones conocidas colectivamente como métodos de regularización. Tales como, detener el entrenamiento temprano (early stopping) para evitar que la red aprenda en exceso. Aplicar *dropout*, que apaga aleatoriamente neuronas durante el entrenamiento. Añadir un término de penalización en la pérdida, por ejemplo la técnica LASSO[6, 7].

Por otro lado, el descenso por gradiente también puede quedarse atrapado en mínimos locales, impidiendo encontrar parámetros mejores [5]. Para evitarlo, a veces se emplea una tasa de aprendizaje adaptativa(p. ej. Adam, RMSProp) o se añade un término de momento, que suaviza las oscilaciones y facilita escapar de minimimos locales.
Adicionalmente, otras dificultades comunes son los problemas de gradientes que se anulan (*vanishing gradients*) y gradientes que explotan (*exploding gradients*) [6, 5]. Como se observa de la ecuación (4.16), el cálculo del gradiente mediante la regla de la cadena implica productos sucesivos de derivadas parciales. Si muchas de estas derivadas son menores que uno, el gradiente total tiende a anularse, dificultando el aprendizaje en capas profundas. Por el contrario, si varias derivadas son mayores que uno, el gradiente puede crecer exponencialmente, lo que lleva a inestabilidad numérica. Ambos fenómenos se agravan conforme aumenta la profundidad de la red.

Para hacer frente a estos problemas, se han desarrollado diversas modificaciones en la arquitectura de las redes. Una de las más destacadas es la incorporación de *conexiones residuales*, como las empleadas en las redes ResNet [5], las cuales facilitan el flujo de información y de gradientes entre capas al introducir atajos que conectan directamente capas no adyacentes. Esta estructura ayuda a preservar la magnitud de los gradientes durante la retropropagación, evitando tanto su atenuación como su crecimiento descontrolado. Por otro lado, funciones de activación como la ReLU (4.5), cuya derivada es constante (0 o 1), también desempeñan un papel clave al limitar la aparición de productos extremos en el cálculo del gradiente, contribuyendo así a una propagación más estable y eficiente.

Además de modificaciones estructurales, técnicas de entrenamiento más sofisticadas también pueden contribuir a contrarrestar estos problemas. Una de ellas es el *muestreo por importancia* (*importance sampling*), que ajusta la frecuencia con la que se seleccionan los datos durante el entrenamiento en función de su influencia esperada en el gradiente. Al priorizar ejemplos más informativos, se puede acelerar la convergencia y reducir la varianza del estimador del gradiente, mejorando la estabilidad del entrenamiento en redes profundas [8, 9].

4.3 Teoremas de aproximación de las redes neuronales

Dado que el objetivo central de este trabajo es aproximar la solución de una EDP (1.1) mediante el uso de una red neuronal, y considerando que ya se han introducido los conceptos básicos de este tipo de modelos, en lo que sigue se presentarán algunos resultados teóricos de gran relevancia en su estudio.

En particular, se abordará el *teorema de aproximación universal*, el cual establece que, bajo ciertas condiciones, las redes neuronales poseen la capacidad de aproximar funciones con un alto grado de precisión. Este tipo de resultados expone el potencial de las redes neuronales como aproximadores funcionales. Para un análisis más detallado sobre sus capacidades de aproximación, pueden consultarse referencias especializadas como [12, 13].

4.3.1 Aproximación Universal de Funciones para Redes Neuronales

Para entender por qué las redes neuronales son herramientas tan útiles, es necesario analizar tanto sus limitaciones como los motivos por los cuales pueden aproximar una gran variedad de funciones. Desde un enfoque teórico, esta capacidad ha sido respaldada por varios teoremas conocidos como *teoremas de aproximación universal*. Los primeros resultados en esta área se centraron en redes neuronales con una única capa oculta y funciones de activación sigmoidal [19, 20]. Posteriormente, se extendieron estos resultados con nuevos teoremas, como el de aproximación universal para redes neuronales, que es el enfoque que se abordará en esta sección.

Teorema 4.2. (Teorema de aproximación universal para redes neuronales (versión reformulada de [18])) *Sea $f \in C(K, \mathbb{R}^n)$ una función continua definida sobre un subconjunto compacto $K \subseteq \mathbb{R}^m$ y con valores en $\mathbb{R}^n$. Entonces, dado $\epsilon > 0$, existe una realización-N_a de una red neuronal con una sola capa oculta y función de activación $a \in C(\mathbb{R}, \mathbb{R})$, tal que,*

$$\sup_{x \in K} \|N_a(x) - f(x)\| < \epsilon \tag{4.17}$$

si y solo si a no es un polinomio.

Observación 4.6. En este contexto, una versión para $\mathbb{R}$ es: la *realización-N_a* de una red neuronal con una sola capa oculta corresponde a funciones de la forma

$$N_a(x) = \sum_{j=1}^{N} \alpha_j \, a(w_j \cdot x + b_j),$$

donde $a = f_h$ es la función de activación (infinitamente diferenciable y no polinómica), y los parámetros $\alpha_j \in \mathbb{R}$, $w_j \in \mathbb{R}^m$, $b_j \in \mathbb{R}$ son libres.

La desigualdad (4.17) indica que, para toda función continua f definida sobre un compacto K, es posible encontrar una red de este tipo que la aproxime uniformemente con precisión

arbitraria, siempre que a no sea un polinomio. En particular, la calidad de la aproximación (es decir, cuán pequeño es ϵ) depende de la elección adecuada de los parámetros (α_j, w_j, b_j) y del número de nodos ocultos N.

La demostración de este teorema es bastante elaborada y necesita que se establezcan varios lemas y proposiciones previamente, los cuales se tratarán en el resto de esta subsección. La estructura de la demostración se basa en el enfoque descrito en [18]. Comenzamos con la definición de las funciones tipo *ridge*.

Definición 4.9. Una función tipo *ridge* $g : \mathbb{R}^d \to \mathbb{R}$ es una función que puede expresarse en la forma,

$$g(x) = f(b \cdot x - \theta) \qquad \text{para todo } x \in \mathbb{R}^d, \tag{4.18}$$

donde el vector $b \in \mathbb{R}^d$ es un vector direccional, $\theta \in \mathbb{R}$ es un un escalar llamado sesgo. Y $f : \mathbb{R} \to \mathbb{R}$ es una función no lineal. (por ejemplo, ReLU(4.5) o Sigmiode(4.2))

Definimos el espacio generado por todas las funciones tipo *ridge* definidas sobre un conjunto $A \subseteq \mathbb{R}^d$ como:

$$R(A) = \text{span}\{f(b \cdot x) : f \in C(\mathbb{R}), b \in A\} \tag{4.19}$$

Teorema 4.3. (Teorema de Vostrecov-Kreines) *El conjunto $R(A)$ de funciones tipo* ridge *definidas sobre un conjunto compacto A es denso en $C\left(\mathbb{R}^d\right)$ (con la topología de convergencia uniforme sobre A) si y solo si no existe ningún polinomio homogéneo no trivial que se anule en A.*
(Este teorema se cita sin demostración; para detalles de la prueba, consulte [18])

Observación 4.7. Es importante notar que, al examinar el conjunto de todas las funciones tipo *ridge* $f \in C(\mathbb{R})$, donde se permite un único vector direccional $b \in \mathbb{R}^d$, se pueden considerar cualquier múltiplo escalar de dicho vector. Esto se debe a que estos vectores pueden ser normalizados para situarse en la hiperesfera unitaria S^{d-1}. Por lo tanto, para analizar la densidad del conjunto de funciones tipo *ridge*, no es necesario incluir todos los vectores $b \in \mathbb{R}^d$ de manera arbitraria. Es suficiente trabajar con los vectores unitarios en S^{d-1}, ya que cualquier dirección en $\mathbb{R}^d$ puede ser representada como uno de ellos mediante un múltiplo escalar.

Proposición 4.4. *Supongamos que el conjunto de todas las funciones $\mathcal{M}$, donde los pesos $\mathcal{A}$ y los sesgos $\mathcal{B}$ son subconjuntos de $\mathbb{R}$, está dado por*

$$\mathcal{M}(f_h; \mathcal{A}, \mathcal{B}) = \text{span}\{f_h(ax - b) : a \in \Lambda \subset \mathbb{R},\, b \in \mathcal{B}\} \tag{4.20}$$

el cual, en la topología de convergencia uniforme sobre conjuntos compactos, es denso en $C(\mathbb{R})$. Además, supongamos que $\mathcal{A} \subseteq S^{d-1}$, para lo cual el conjunto de funciones tipo ridge *$R(\mathcal{A})$ es denso en $C(\mathbb{R}^d)$ en la topología de convergencia uniforme sobre conjuntos compactos.*

Entonces, el conjunto de todas las redes neuronales posibles con una sola capa oculta, con pesos $w \in \mathcal{A}$ y sesgo $\theta \in \Theta$, dado por

$$\mathcal{N}(f_h; \Lambda \times \mathcal{A}, \Theta) = \text{span}\{f_h(w \cdot x - \theta) : w \in \mathcal{A},\, \theta \in \Theta\} \tag{4.21}$$

es denso en $C(\mathbb{R}^d)$, en la topología de convergencia uniforme sobre conjuntos compactos.

Demostración. Consideremos $f \in C(K)$ la función a aproximar, donde K es un subconjunto compacto de $\mathbb{R}^d$. Por el Teorema 4.3, el conjunto de funciones tipo *ridge* $R(A)$ es denso en $C(K)$. En consecuencia, dado un $\epsilon > 0$, existe una secuencia de funciones tipo *ridge* $g_i \in C(\mathbb{R})$ y $a^i \in A$, $i = 1, \ldots, r$, tal que

$$\left| f(x) - \sum_{i=1}^{r} g_i \left(a^i \cdot x\right) \right| < \frac{\epsilon}{2} \tag{4.22}$$

para todo $x \in K$. Como $a^i \cdot x \in \mathbb{R}$ y K es compacto, el producto escalar $a^i \cdot x$ pertenece a un intervalo acotado $[\alpha_i, \beta_i]$. Por la hipótesis de densidad de $\mathcal{M}\left(f_h; \mathcal{A}, b\right)$ en dicho intervalo, existen constantes $c_{ij} \in \mathbb{R}$, pesos $w_{ij} \in \mathcal{A}$ y sesgos $\theta_{ij} \in \Theta$ tales que

$$\left| g_i(t) - \sum_{j=1}^{m} c_{ij} f_h \left(w_{ij} t - \theta_{ij}\right) \right| < \frac{\epsilon}{2r} \tag{4.23}$$

para todo $t \in [\alpha_i, \beta_i]$ y $i = 1, \ldots, r$. Por tanto,

$$\left| f(x) - \sum_{i=1}^{r} \sum_{j=1}^{m} c_{ij} f_h \left(w_{ij} a^i \cdot x - \theta_{ij}\right) \right| < \epsilon \tag{4.24}$$

para todo $x \in K$.

□

Proposición 4.5. *Supongamos que la función $f_h \in C^\infty(\mathbb{R})$ está asociada a la función en el conjunto $\mathcal{M}(f_h; \mathcal{A}, \mathcal{B})$, tal como se definió previamente en la ecuación (4.20), y que f_h no es un polinomio. Entonces, el subespacio generado por $\mathcal{N}(f_h; \mathbb{R}, \mathbb{R})$ es denso en $C(\mathbb{R})$.*

Demostración. Dado que la función f_h es infinitamente diferenciable y que la derivada de una función en $\mathcal{N}(f_h; \mathbb{R}, \mathbb{R})$ también pertenece al subespacio lineal generado por $\mathcal{N}(f_h; \mathbb{R}, \mathbb{R})$, se sigue que,

$$\frac{1}{h}\big(f_h((a+h)t - b) - f_h(at - b)\big) \;\in\; \mathcal{N}(f_h; \mathbb{R}, \mathbb{R}) \quad (h \neq 0). \tag{4.25}$$

Al hacer tender $h \to 0$, esta diferencia converge a la derivada de $f_h(at - b)$ con respecto a a. En particular, evaluando en $a = 0$ obtenemos

$$\frac{d}{da}\, f_h(at - b)\bigg|_{a=0} \;=\; t\, f_h'(-b), \tag{4.26}$$

que por continuidad pertenece a la cerradura $\overline{\mathcal{N}(f_h;\mathbb{R},\mathbb{R})}$. De manera completamente análoga, si diferenciamos k veces respecto a a y luego evaluamos en $a=0$, resulta

$$\frac{d^k}{da^k}\, f_h(at-b)\bigg|_{a=0} \;=\; t^k\, f_h^{(k)}(-b), \tag{4.27}$$

que también pertenece a la misma cerradura, para todo $k \in \mathbb{N}$. Como por hipótesis $f_h^{(k)}(-b) \neq 0$, concluimos que el monomio t^k (y por tanto todo polinomio) puede aproximarse arbitrariamente bien por combinaciones de funciones en $\mathcal{N}(f_h;\mathbb{R},\mathbb{R})$. Finalmente, aplicando el Teorema de Stone–Weierstrass[5] , obtenemos que el espacio lineal generado por $\mathcal{N}(f_h;\mathbb{R},\mathbb{R})$ es denso en $C(K)$ para cualquier compacto $K \subset \mathbb{R}$, lo que completa la demostración.

□

Proposición 4.6. *Si $f_h \in C(\mathbb{R})$ y no es un polinomio, entonces el espacio lineal generado por $\mathcal{N}(f_h;\mathbb{R},\mathbb{R})$ es denso en $C(\mathbb{R})$. Es decir, toda función continua en $\mathbb{R}$ puede aproximarse uniformemente en compactos por combinaciones lineales finitas de funciones en $\mathcal{N}(f_h;\mathbb{R},\mathbb{R})$.*

Demostración. Sea $C_0^\infty(\mathbb{R})$ el conjunto de funciones suaves con soporte compacto. Para $\phi \in C_0^\infty(\mathbb{R})$, se define la convolución de f_h y ϕ como

$$f_h * \phi := \int_{-\infty}^{\infty} f_h(t-y)\phi(y)dy \tag{4.28}$$

la cual está bien definida y converge para todo $t \in \mathbb{R}$, debido a que ϕ tiene soporte compacto y f_h es continua. Además, al considerar una función compuesta del tipo $f_h(at-b)$, donde $a, b \in \mathbb{R}$, también se puede escribir su convolución con ϕ como,

$$f_h(at-b) * \phi(y) = \int_{-\infty}^{\infty} f_h(at-b-y)\phi(y)dy \tag{4.29}$$

Dado que esta operación no modifica la estructura de convolución más allá de una transformación afín en el argumento, se sigue que

$$\overline{\mathcal{N}(f_h * \phi;\mathbb{R},\mathbb{R})} \subseteq \overline{\mathcal{N}(f_h;\mathbb{R},\mathbb{R})} \quad \text{para todo } a \in \mathbb{R}.$$

Ahora, definamos $f_\phi := f_h * \phi$. Dado que la convolución de una función continua con una función suave y de soporte compacto es también infinitamente diferenciable, entonces $f_\phi \in C^\infty(\mathbb{R})$, por la proposición 4.5 se concluye que para cualquier $k \in \mathbb{N}$ y $b \in \mathbb{R}$, el producto $t^k f_\phi^{(k)}(-b)$ pertenece al conjunto $\mathcal{N}(f_\phi;\mathbb{R},\mathbb{R})$.

Si $\mathcal{N}(f;\mathbb{R},\mathbb{R})$ no es denso, entonces existiría algún k tal que $t^k \notin \overline{\mathcal{N}(f;\mathbb{R},\mathbb{R})}$. Por tanto, $f_\phi^{(k)}(-b) = 0$ para todo b y toda ϕ, implicando que f_ϕ es un polinomio de grado a lo sumo $k-1$. No obstante, se puede construir una sucesión de funciones $\{\phi_n\} \subset C_0^\infty(\mathbb{R})$ tal que

[5]Sea A una subálgebra de $C(K)$, el espacio de funciones continuas en un compacto $K \subset \mathbb{R}$, que separa puntos y no se anula en ningún punto de K. Entonces el cierre de A con respecto a la norma del supremo coincide con todo $C(K)$, es decir, $\overline{A} = C(K)$. En particular, los polinomios son densos en $C(K)$.

$f_{\phi_n} := f_h * \phi_n \to f$ uniformemente sobre conjuntos compactos. Esto significa que f sería el límite uniforme de funciones f_{ϕ_n}, cada una de las cuales es un polinomio de grado a lo sumo $k-1$. Como el conjunto de polinomios de grado a lo sumo $k-1$ es cerrado en la topología de convergencia uniforme sobre compactos, se concluiría que f también debe ser un polinomio de grado a lo sumo $k-1$. Sin embargo, esta conclusión contradice la hipótesis inicial de que f no es un polinomio de grado finito, por lo que se obtiene una contradicción. En consecuencia, el conjunto $\mathcal{N}(f;\mathbb{R},\mathbb{R})$ es denso. □

Teniendo en cuenta lo anterior, es posible demostrar el teorema de aproximación universal de funciones para redes neuronales.

Demostración. Teorema de Aproximación Universal de Funciones para Redes Neuronales 4.2
Usando la proposición 4.6 y observando que si f es un polinomio, entonces la propiedad de densidad no puede mantenerse, se prueba esencialmente el teorema. Fijar el grado de f en cualquier entero positivo implica que el espacio lineal generado por el conjunto de todas las redes neuronales con una sola capa oculta no puede abarcar todo $\mathbb{R}^d$. □

Ejemplo 4.4. (Aproximación de una función continua mediante una red neuronal)

Con el fin de ilustrar un ejemplo del Teorema 4.2. Consideremos la función continua $f(t) = \frac{1}{2}\sin(t)$ definida sobre el compacto $K = [-\pi, \pi]$. De acuerdo con el teorema, existe una red neuronal de una sola capa oculta y función activación no polinómica capaz de aproximar f uniformemente con precisión arbitraria. En este ejemplo empleamos la función de activación tangente hiperbólica (4.3). De modo que, la N_a-realización de una red neuronal o simplemente red totalmente conectada del tipo *feedforward*, tiene la forma

$$N_a(t) = \sum_{i=1}^{N} \alpha_i \tanh\left(w_i t + b_i\right) \tag{4.30}$$

Donde N denota el número de neuronas en la capa oculta (en este caso usamos $N = 10$) y los parámetros libres $\{\alpha_j, w_j, b_j\}$. Para la implementación computacional generamos un conjunto de entrenamiento etiquetado con 10 puntos uniformemente espaciados en $[-\pi, \pi]$, de la forma $\{t_i, f(t_i)\}$ los cuales se generan sintéticamente. El entrenamiento se formula con una función de pérdida basada en mínimos cuadrados (4.8). Así,

$$\mathcal{L}(\theta) = \frac{1}{10}\sum_{i=1}^{10}\big(N_a(t_i;\theta) - f(t_i)\big)^2,$$

donde θ agrupa todos los parámetros de la red. Para el proceso de optimización utilizamos un método de tipo L-BFGS. Aquí θ se inicializa de forma aleatoria utilizando los inicializadores por defecto de PyTorch, en el rango de $(-1, 1)$. El procedimiento de entrenamiento (minimización de la función de pérdida) se ejecuta durante 2500 épocas con tasa de aprendizaje o tolerancia 10^{-3}. Tras el entrenamiento, comparamos N_a con f tanto en los 10 puntos de entrenamiento como en una malla densa para la visualización.

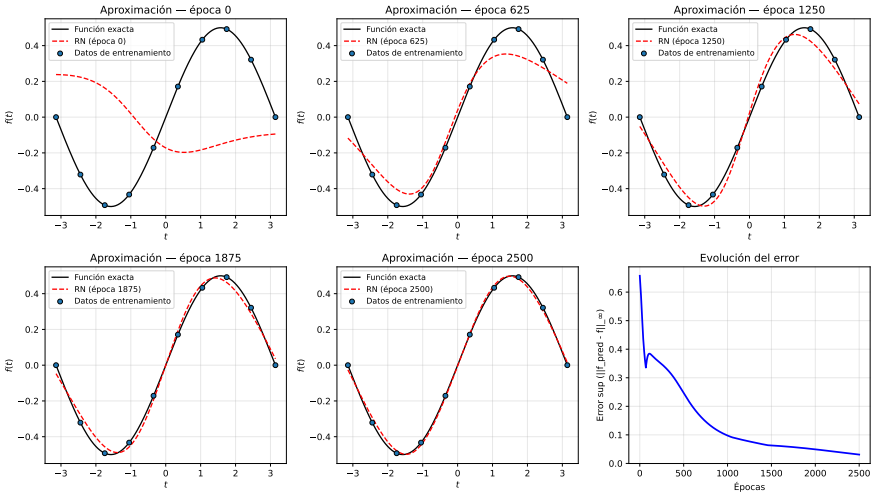


Figura 4.4. Aproximación de $f(t) = \frac{1}{2}\sin(t)$ por una red neuronal de una capa oculta con activación tanh, y tasa se aprendizaje 10^{-3}. Se muestran la función exacta, la aproximación para diferentes iteraciones, los 10 puntos de entrenamiento y el error $\|N_a(t_i) - f(t_i)\|_\infty$ como función de la iteración

En la Fig. 4.4 se muestra la curva de $f(t)$ junto con la aproximación de la red y los puntos de entrenamiento. Este experimento confirma empíricamente el enunciado del teorema. Incluso cuando los datos de entrenamiento son escasos, una red neuronal, con una sola capa oculta y activación no polinómica puede aproximar con alta precisión una función continua sobre un compacto.

Observación 4.8. Adicionalmente, en este punto conviene destacar algunos aspectos positivos de las redes neuronales. En particular, los resultados de Barron [17] muestran que, para una clase específica de funciones, las redes pueden mitigar la llamada *maldición de la dimensionalidad*. Este fenómeno se refiere al crecimiento exponencial del volumen del espacio de entrada conforme aumenta la dimensión, lo que dificulta la aproximación y generalización de funciones mediante métodos tradicionales. Por ejemplo, al resolver ecuaciones diferenciales parciales con Diferencias Finitas, es necesario construir una malla sobre el dominio: en bajas dimensiones esto es manejable, pero en altas, el número de puntos crece exponencialmente. Así, representar un dominio de 500 dimensiones con apenas 10 puntos por dimensión exigiría 10^{500} nodos, lo cual resulta completamente impracticable. En contraste, se ha demostrado que las redes neuronales pueden superar este obstáculo, pues no requieren una malla explícita para representar soluciones [18, 17]. Estas ideas se discutirán con mayor detalle más adelante.

Capítulo 5
Aprendizaje profundo basado en física

En este punto contamos con todos los preliminares necesarios para presentar el enfoque central de este trabajo, las redes neuronales informadas por la física (PINNs por su siglas en ingles). Este enfoque combina las capacidades de aproximación y aprendizaje de las redes neuronales profundas con los principios y restricciones impuestos por las leyes físicas que describen sistemas del mundo real, escritas como ecuaciones diferenciales ya sean ordinarias (EDP) o parciales (EDP). En el contexto de aprendizaje profundo, esta estrategia es conocida como *aprendizaje profundo basado en física* [11]. En las secciones siguientes, profundizaremos en la teoría y aplicaciones de este enfoque innovador que une el aprendizaje automático con la física asociada a un modelo en específico.

5.1 Principios generales de las PINNs

Como se ha mencionado anteriormente, las redes neuronales juegan un papel muy importante como aproximadores universales en diferentes contextos; en particular, estamos interesados en el objetivo de aproximar la solución de ecuaciones diferenciales de tipo evolutivo 3, mediante una red neuronal. Este propósito no es nuevo, los primeros trabajos relacionados con este enfoque se remontan a principios de la década de 1990 [52, 53, 54]; sin embargo, en ese entonces, la limitada capacidad de cómputo generó un desinterés en el uso de estas técnicas.
En consecuencia, en la actualidad, gracias al creciente poder de cómputo, las redes neuronales han vuelto a ser muy atractivas, generando un renovado interés en su aplicación. El término PINN y el aumento de la investigación en esta área se establecieron en dos artículos clave [24, 25], y posteriormente consolidados en [26]. Como resultado las redes neuronales informadas por la física, comúnmente abreviadas como PINNs (por sus siglas en inglés), fueron introducidas por primera vez en 2017 por Raissi, Perdikaris y Karniadakis [24, 25, 26] como un enfoque eficiente y directo para usar redes neuronales en la resolución de ecuaciones diferenciales parciales. Aunque se han presentado muchas modificaciones a esta estrategia,

el enfoque original ha demostrado ser altamente efectivo en la aproximación de soluciones a problemas complejos.

Las PINNs integran principios físicos directamente en la estructura de la red neuronal, permitiendo la incorporación de conocimiento previo sobre la dinámica del sistema en el proceso de aprendizaje. Esta estrategia no solo aproxima la solución, sino que también reduce la necesidad de grandes cantidades de datos de entrenamiento [24].

Para hacer un recorrido de cómo es en específico el funcionamiento y aplicación de las PINNs, consideremos ecuaciones diferenciales parciales de la forma

$$u_t + \mathcal{N}[u;\lambda] = 0 \tag{5.1}$$

donde $\mathcal{N}[u;\lambda]$ es un operador diferencial no lineal que puede depender de varios parámetros λ (por ejemplo (3.3), (3.5)), y $u(x,t)$ denota la función $u : \Omega \times I \subseteq \mathbb{R}^{n+1} \to \mathbb{R}$ que se desea determinar a partir de la ecuación diferencial parcial, para $x \in \Omega \subset \mathbb{R}^n$ y $t \in I \subset \mathbb{R}$.
Sujeta a las condiciones iniciales y de frontera

$$u(x,0) = g(x), \quad x \in \Omega, \tag{5.2}$$
$$\mathcal{B}[u] = 0, \quad t \in [0,T], \quad x \in \partial\Omega, \tag{5.3}$$

donde $\mathcal{B}[\cdot]$ es un operador de frontera que representa condiciones de Dirichlet, Neumann, Robin o periódicas.

El supuesto fundamental en el que se basan las PINNs es que la función solución desconocida u, que satisface una ecuación diferencial parcial, así como sus derivadas parciales, puede aproximarse mediante una red neuronal. Estas derivadas se calculan eficientemente utilizando técnicas de diferenciación automática 4.2.3. Formalmente, una EDP puede escribirse utilizando la notación de multiíndices[1] , como:

$$F\left(D^k u(\mathbf{x}), D^{(k-1)}u(\mathbf{x}), \ldots, Du(\mathbf{x}), u(\mathbf{x}), \mathbf{x}\right) = 0, \tag{5.4}$$

donde D^i representa el conjunto de todas las derivadas parciales de orden i de u con respecto a las variables espaciales y temporales, y $\mathbf{x} = (x,t) \in \Omega \times I \subseteq \mathbb{R}^n \times \mathbb{R}$ denota la variable independiente. En el contexto de las PINNs, se asume que la función u puede ser aproximada por una red neuronal $\Phi_a(\mathbf{x})$, y que sus derivadas parciales pueden ser calculadas mediante

[1] Sea $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ un multiíndice, donde cada $\alpha_i \in \mathbb{N}_0$. Entonces, la derivada parcial de orden $|\alpha| = \alpha_1 + \cdots + \alpha_n$ se define como:

$$D^\alpha u = \frac{\partial^{|\alpha|} u}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} \cdots \partial x_n^{\alpha_n}}.$$

De modo que una EDP puede definirse como una expresión de la forma:

$$F\left(D^k u(\mathbf{x}), D^{(k-1)}u(\mathbf{x}), \ldots, Du(\mathbf{x}), u(\mathbf{x}), \mathbf{x}\right) = 0,$$

donde D^k denota el conjunto de derivadas parciales de orden k, definidas en términos de multiíndices, y $u : U \subseteq \mathbb{R}^n \to \mathbb{R}$ es la función diferenciable desconocida. En este contexto, F es una función conocida que actúa como un funcional de los términos anteriores.

diferenciación automática. Estas derivadas se denotan como $\tilde{D}^i\Phi_a(\mathbf{x})$, donde el símbolo $\tilde{D}$ indica que las derivadas se obtienen automáticamente a partir de la arquitectura y los parámetros de la red.

En consecuencia, bajo el supuesto de que la solución $u(\mathbf{x})$ de una ecuación diferencial parcial puede ser aproximada por una red neuronal $\Phi_a(\mathbf{x})$ con función de activación a, se plantea que la red y sus derivadas, calculadas mediante diferenciación automática, deben aproximar suficientemente bien la función F definida por la EDP. Matemáticamente, esta hipótesis se puede expresar de la siguiente manera,

$$\begin{aligned}\sup_{\mathbf{x}\in\Omega\times I} \Big| & F\left(D^k u(\mathbf{x}), D^{(k-1)}u(\mathbf{x}), \ldots, Du(\mathbf{x}), u(\mathbf{x}), \mathbf{x}\right) \\ & -F\left(\tilde{D}^k\Phi_a(\mathbf{x}), \tilde{D}^{(k-1)}\Phi_a(\mathbf{x}), \ldots, \tilde{D}\Phi_a(\mathbf{x}), \Phi_a(\mathbf{x}), \mathbf{x}\right)\Big| < \epsilon,\end{aligned} \tag{5.5}$$

para todo $\epsilon > 0$. En vista de que la ecuación diferencial parcial se satisface exactamente por u, es decir,

$$F\left(D^k u(\mathbf{x}), \ldots, Du(\mathbf{x}), u(\mathbf{x}), \mathbf{x}\right) = 0,$$

la expresión anterior se puede simplificar como:

$$\sup_{\mathbf{x}\in\Omega\times I} \left|F\left(\tilde{D}^k\Phi_a(\mathbf{x}), \ldots, \tilde{D}\Phi_a(\mathbf{x}), \Phi_a(\mathbf{x}), \mathbf{x}\right)\right| < \epsilon. \tag{5.6}$$

Para que esta aproximación sea válida, es necesario que la red neuronal $\Phi_a(\mathbf{x})$ reproduzca con suficiente precisión tanto la función u como sus derivadas parciales hasta orden k dentro del dominio. En términos formales, este criterio se pueden formular como

$$\sup_{\mathbf{x}\in\Omega\times I} |u(\mathbf{x}) - \Phi_a(\mathbf{x})| < \epsilon_0, \tag{5.7}$$

para todo $\epsilon_0 > 0$, donde $\Phi_a(\mathbf{x}, t)$ denota la realización de una red neuronal Φ, con función de activación a y variable de entrada $\mathbf{x}$.

Adicionalmente, se requiere que cada derivada parcial de u involucrada en la ecuación sea también aproximada por la correspondiente derivada obtenida mediante diferenciación automática en la red. Esto implica que, para cada multiíndice j con $1 \leq j \leq k$, se satisfaga,

$$\sup_{\mathbf{x}\in\Omega\times I} \left|D^j u(\mathbf{x}) - \tilde{D}^j\Phi_a(\mathbf{x})\right| < \epsilon_j, \tag{5.8}$$

con $\epsilon_j > 0$ arbitrario, suponiendo que la red neuronal sea continuamente diferenciable respecto a todas las derivadas involucradas en la ecuación diferencial parcial, lo cual es una de las razones por las que es tan importante que las funciones de activación sean diferenciables. La validez de estas hipótesis está sustentada en el Teorema de Aproximación Universal 4.2, el cual garantiza la capacidad de aproximación de las redes neuronales, para una red de tamaño

y complejidad suficientes.

El aspecto fundamental que distingue a las PINNs de las redes neuronales convencionales es el uso de un funcional de pérdida basado en la física del problema, en lugar de depender exclusivamente de datos observacionales. Para ilustrar este enfoque, comenzamos representando la solución desconocida $u(x,t)$ de la EDP mediante una red neuronal profunda $u_\theta(x,t)$, de modo que $u_\theta(x,t) \approx u(x,t)$, donde θ denota el conjunto de parámetros entrenables de la red (por ejemplo, pesos y sesgos). Esta aproximación nos permite definir el residuo de la EDP, denotado por $\mathcal{R}_\theta(x,t)$, de la siguiente manera, como también se propone en [26]:

$$\mathcal{R}_\theta(x,t) := \frac{\partial u_\theta}{\partial t}(x_r,t_r) + \mathcal{N}[u_\theta;\lambda](x_r,t_r) \tag{5.9}$$

lo anterior, con el objetivo de acortar la notación en la definición del error cuadrático medio 4.8 asociado a los puntos de evaluación $\{(x_r^i,t_r^i)\}_{i=1}^{N_r}$ en la ecuación diferencial parcial, es decir:

$$\mathcal{L}_{MSE_r}(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r}\left|\mathcal{R}_\theta(x_r^i,t_r^i)\right|^2. \tag{5.10}$$

donde N_r denota el número total de puntos de evaluación. Como es bien sabido, además de estos puntos, un problema bien planteado de ecuaciones diferenciales parciales también debe incluir condiciones de frontera y posiblemente condiciones iniciales (5.3), (5.2). Los puntos sobre estas fronteras se incorporan en el modelo como los puntos de la forma $\{x_{bc}^i,t_{bc}^i\}_{i=1}^{N_{bc}}$, donde N_{bc} es el número total de puntos en este conjunto. Y los puntos de la condición inicial, de la forma $\{x_{ic}^i\}_{i=1}^{N_{ic}}$, aquí N_{ic} representa el número total de puntos en el conjunto. En estos casos, el error cuadrático medio para los puntos de frontera se expresa como:

$$\mathcal{L}_{MSE_{bc}}(\theta) = \frac{1}{N_{bc}}\sum_{i=1}^{N_{bc}}\left|\mathcal{B}[u_\theta](x_{bc}^i,t_{bc}^i)\right|^2 \tag{5.11}$$

$$\mathcal{L}_{MSE_{ic}}(\theta) = \frac{1}{N_{ic}}\sum_{i=1}^{N_{ic}}\left|u_\theta(\mathbf{x}_{ic}^i,0) - g(\mathbf{x}_{ic}^i)\right|^2, \tag{5.12}$$

De esta manera, juntando las ecuaciones (5.10) , (5.11) y (5.12), la función de pérdida total en el dominio de la ecuación diferencial parcial se expresa como,

$$\mathcal{L}_{MSE} = \mathcal{L}_{MSE_{ic}} + \mathcal{L}_{MSE_{bc}} + \mathcal{L}_{MSE_r} \tag{5.13}$$

Y utilizando esta ecuación como función de pérdida para la red neuronal $u_\theta(\cdot,\cdot)$, se puede resolver una ecuación diferencial parcial asociada a un problema bien puesto 2.7. Nótese que los dos términos en (5.13), denominados "pérdida inicial", $\mathcal{L}_{MSE_{ic}}$, y "pérdida de frontera", $\mathcal{L}_{MSE_{bc}}$, alientan la predicción de la red a coincidir con la solución verdadera (y/o sus derivadas) en un subconjunto de ubicaciones conocidas del dominio. Mientras que el tercer término, denominado "pérdida física", $\mathcal{L}_{MSE_r}$, busca asegurar que la solución PINN cumpla

con la ecuación diferencial subyacente del sistema minimizando el residual[2] de la ecuación diferencial en un conjunto de puntos, $\{(x_r^i, t_r^i)\}_{i=1}^{N_r}$, muestreados en todo el dominio. Intuitivamente, la pérdida física empuja a la red neuronal a aprender una solución que sea consistente con la ecuación diferencial subyacente, mientras que la pérdida de frontera asegura que la solución sea única al hacerla coincidir con la condición inicial y/o condiciones de frontera conocidas.

Un recorrido esquemático sobre el funcionamiento de las PINNs se puede ver en la figura 5.1. Una PINN es una red neuronal $u_\theta(x,t)$ con parámetros entrenables θ, que aproxima directamente la solución $u(x,t)$ de una ecuación diferencial, es decir, $u_\theta(x,t) \approx u(x,t)$. El proceso comienza suministrando los pares de coordenadas (x,t) como entrada a la red, la cual produce una aproximación $u_\theta(x,t)$ en cada punto del dominio. A partir de esta salida, se calculan las derivadas necesarias (como $\partial_t u_\theta$, $\partial_x u_\theta$, $\partial_{xx} u_\theta$, etc.) mediante diferenciación automática para construir el residual $\mathcal{R}_\theta(x,t)$. Luego, se evalúan tres contribuciones a la función de pérdida total (5.13). Si este último es menor que un umbral de tolerancia ϵ, el entrenamiento se detiene. En caso contrario, se retropropaga el gradiente de $\mathcal{L}_{MSE}$ para actualizar los parámetros θ, repitiendo el ciclo hasta alcanzar la convergencia.

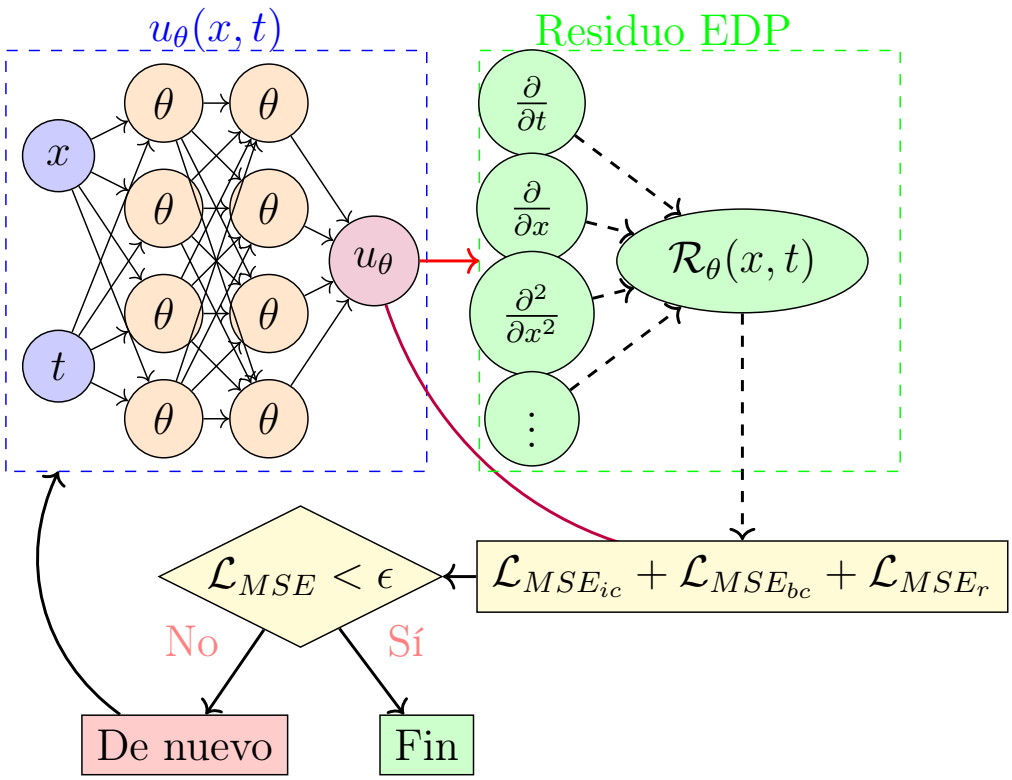


Figura 5.1. Esquema de una red neuronal informada por la física (PINN).

[2] $\mathcal{R}_\theta(\cdot,\cdot)$ se conoce como *residual* porque cuantifica el error o "residuo" que queda tras insertar la solución aproximada en la EDP. Si u_θ fuera la solución exacta, entonces $\mathcal{R}_\theta(x,t) = 0$ en todo el dominio. Por lo tanto, minimizar el residual es equivalente a hacer que la solución propuesta satisfaga la ecuación diferencial en el sentido más fiel posible.

Observación 5.1. Una característica distintiva de las (PINNs) es su capacidad para aproximar soluciones de ecuaciones diferenciales parciales (EDPs) sin requerir datos etiquetados en el sentido clásico del aprendizaje supervisado. Esto se debe a que, en lugar de ajustar la red neuronal a datos de salida conocidos, como se hace en los métodos de aprendizaje supervisado, las PINNs se entrenan directamente para satisfacer la ecuación diferencial y las condiciones de frontera o iniciales impuestas por el problema.

Aunque es posible incorporar datos etiquetados (por ejemplo, mediciones experimentales o soluciones parciales) para enriquecer el entrenamiento de la PINNs y mejorar su precisión (en un esquema que sería híbrido entre aprendizaje supervisado y no supervisado), el poder de las PINNs radica en su capacidad de funcionar en un régimen no supervisado puro para un problema bien planteado. Esto significa que no es necesario disponer de ejemplos de la solución exacta de la EDP antes de aplicar la red neuronal. Como resultado, las PINNs permiten abordar la resolución de ecuaciones que podrían no haber sido resueltas previamente o problemas para los cuales los métodos numéricos tradicionales, como los métodos de elementos finitos o volúmenes finitos, pueden ser costosos o inviables debido a la dimensionalidad, la geometría compleja o la escasez de datos [27].

5.2 PINNs desde el enfoque de optimización multiobjetivo

Como se ha observado al entrenar una PINN para resolver una EDO o EDP, se incorporan las distintas condiciones físicas (ecuación diferencial, condiciones iniciales y de frontera, datos, etc.) como términos de una función de pérdida (5.13). Cada término de la pérdida evalúa qué tan bien se cumple un objetivo específico, por ejemplo minimizar el error en el residual de la EDP o en las condiciones de frontera. En consecuencia, el entrenamiento de una PINN típicamente se plantea como un problema de optimización multiobjetivo donde varios criterios deben satisfacerse simultáneamente [55]. Por ejemplo, en la ecuación (5.13) los objetivos se entrenan conjuntamente, por tanto, pertenecen a la clase optimización multiobjetivo,

$$\mathcal{L}(\theta) = (\mathcal{L}_{MSE_{ic}}, \mathcal{L}_{MSE_{bc}}, \mathcal{L}_{MSE_r})^T \tag{5.14}$$

donde cada uno de los términos se interpreta como se ha mencionado anteriormente. Sin embargo, es usual encontrar más condiciones iniciales o de frontera, inclusive mediciones del problema directo. De modo que, en general podemos encontrar múltiples objetivos $\mathcal{L}_i(\theta)$ (cada uno asociado a un conjunto de puntos de entrenamiento y un tipo de condición). Así $\mathcal{L}(\theta)$ puede generalizarse como

$$\mathcal{L}(\theta) = (\mathcal{L}_1(\theta), \ldots, \mathcal{L}_k(\theta))^T \tag{5.15}$$

De este modo se combinan en una única funcion de costo escalar mediante una suma pondera,

$$\mathcal{L}(\theta) = \sum_{i=1}^{k} \lambda_i \mathcal{L}_i(\theta), \quad \text{con } \lambda_i \in \mathbb{R}^+ \tag{5.16}$$

donde $\mathcal{L}_i$ es el valor del ésimo objetivo (o término de pérdida i), y λ_i es el peso que se le otorga a dicho término en la combinación lineal. Esta estrategia es conocida como escalarización

lineal de objetivos múltiples y convierte el problema multiobjetivo en uno de optimización escalar ajustando las ponderaciones λ_i de cada término. En el contexto de PINNs, la función de perdida total (5.13) también se puede generalizar y escribirse como sigue

$$\mathcal{L}(\theta) = \lambda_{IC}\mathcal{L}_{IC} + \lambda_{BC}\mathcal{L}_{BC} + \lambda_{res}\mathcal{L}_{res} \tag{5.17}$$

Una pregunta fundamental es cómo elegir apropiadamente los pesos λ_i de cada término en la pérdida. La selección de estos coeficientes determina la prioridad relativa de cada objetivo durante el entrenamiento de la PINN y, por ende, influye fuertemente en la calidad de la solución obtenida. En la práctica, la elección de las escalas λ_i determina *qué* objetivos influyen con mayor intensidad en los gradientes durante el entrenamiento y, por ende, *hacia dónde* se desplaza la solución en el espacio de pérdidas [55].

Si bien las redes neuronales poseen la capacidad de aproximar cualquier función suave, la optimización basada en gradientes sobre una combinación lineal de términos (5.16) o (5.17) no garantiza alcanzar el óptimo global de la formulación multiobjetivo. Escoger manualmente los λ_i mediante búsqueda en rejilla se torna rápidamente intratable cuando el número de objetivos k crece, y además impide que dichos pesos evolucionen de forma coherente a lo largo del entrenamiento.

Dado lo anterior, ha surgido un cuerpo de trabajo dedicado a estrategias adaptativas para balancear pérdidas en PINNs, es decir, métodos que ajustan automáticamente los λ_i durante el entrenamiento, evitando la necesidad de determinarlos a mano. A continuación, describimos algunos enfoques relevantes:

- **Escalarización estática informada:** Consiste en una estrategia simple, la cual es elegir los pesos de modo que todas las componentes de la pérdida contribuyan de forma equivalente inicialmente. Por ejemplo, podríamos fijar λ_i inversamente proporcional a la magnitud inicial de $\mathcal{L}_i$, de forma que $\lambda_i\mathcal{L}_i$ tenga un orden de magnitud similar para todos i al inicio. Esto puede evitar arranques dominados por un solo término. Sin embargo, esta solución no garantiza que más adelante en el entrenamiento las contribuciones sigan equilibradas.

- **Adaptación mediante tasas de aprendizaje**: (Learning Rate Annealing (LRA))es un método propuesto específicamente en el contexto de PINNs por Wang et al.[56]. La idea es ajustar dinámicamente los pesos usando las estadísticas de los gradientes de cada término. Un ejemplo de implementación es reducir la tasa de aprendizaje efectiva para aquellos objetivos cuyos gradientes son grandes (o sus errores decrecen lentamente), y aumentarla para aquellos cuyos gradientes son pequeños (o errores decrecen rápidamente). En la práctica, LRA mantiene estimaciones del tamaño relativo de cada gradiente $\nabla_\theta\mathcal{L}_i$ y actualiza λ_i a lo largo del entrenamiento para igualar esas escalas. En esencia, LRA introduce una retroalimentación: si la pérdida i está descendiendo muy lento (relativamente), aumenta su peso para empujarla más, y viceversa.

- **GradNorm (Normalización de Gradientes)**: Introducido por Chen et al. [55]. También ha sido aplicado al contexto PINNs. En GradNorm, los coeficientes λ_i se

consideran parámetros entrenables adicionales. Su actualización se diseña para que la tasa de reducción de todas las pérdidas sea equilibrada. Concretamente, GradNorm calcula la velocidad a la que cada $\mathcal{L}_i$ está disminuyendo con respecto a su valor inicial, y ajusta λ_i para igualar esas velocidades [55] . Si una pérdida ha caído mucho más rápido que las otras (es decir, esa parte de la tarea se está aprendiendo fácilmente), GradNorm disminuye su peso para no "sobre-aprenderla" y reserva capacidad para las demás; por el contrario, si una pérdida apenas ha bajado, aumenta su peso para reforzarla. Todo esto se realiza mediante una regla de actualización diferenciable durante el entrenamiento, a menudo con un optimizador secundario que actualiza los λ_i.

- **SoftAdapt:** Propuesto por Heydari et al. [55]. Es otro método de ajuste automático que prescinde de calcular gradientes por separado, usando en cambio las propias pérdidas como indicador. SoftAdapt asume que la magnitud de los gradientes está relacionada con la magnitud de las pérdidas (en muchos casos, gradiente grande proviene de error grande). Por tanto, en lugar de balancear gradientes explícitamente, este enfoque pondera según la magnitud relativa de $\mathcal{L}_i$ en cada etapa. Se calcula, por ejemplo, una puntuación inversamente proporcional al orden de cada pérdida (dando más peso al término con error mayor, para empujarlo más) y luego se aplica una función softmax para normalizar las ponderaciones. De esta manera, cada iteración ajusta ligeramente los λ_i otorgando más atención a los objetivos que siguen altos en error. SoftAdapt resulta más sencillo computacionalmente que GradNorm, ya que no requiere derivadas individuales ni optimizadores adicionales; a cambio, asume que equiparar pérdidas conduce a equilibrar gradientes de forma indirecta [55].

En resumen, el uso de técnicas automáticas para el ajuste de los coeficientes λ_i busca eliminar la necesidad de seleccionarlos manualmente, permitiendo que el propio algoritmo encuentre durante el entrenamiento un balance óptimo entre los distintos términos de la función de pérdida. En ausencia de estos mecanismos, algunos autores, como en [55], recomiendan basarse en información física o numérica del problema directo para orientar la elección. Por ejemplo, se sugiere normalizar las ecuaciones (para evitar discrepancias por unidades) o realizar una búsqueda sistemática de los λ_i, utilizando herramientas como la curva-L[3], que permite visualizar el compromiso entre el ajuste a la EDP y a los datos, eligiendo un punto de equilibrio óptimo.

Adicionalmente, en problemas con pocos objetivos, también es viable explorar combinaciones extremas (dar peso predominante a un término y observar el comportamiento) para analizar la sensibilidad. No obstante, en aplicaciones más complejas, especialmente en PINNs con múltiples restricciones físicas, la tendencia actual es emplear algoritmos adaptativos (como los ilustrados anteriormente) que eviten decisiones arbitrarias y mejoren la eficiencia del entrenamiento.

[3]La curva-L es una técnica gráfica utilizada comúnmente en problemas inversos. Consiste en representar la norma del error de ajuste frente a la norma del término de regularización para distintos valores de λ, generando una curva con forma de "L". El punto de máxima curvatura se interpreta como un compromiso óptimo entre fidelidad a los datos y estabilidad de la solución.

5.3 Software para PINNs

Antes de seguir a las aplicaciones, es conveniente mencionar que existen diferentes tipos y formas de implementar las PINNs computacionalmente [27], de manera eficiente ha resultado ventajoso construir nuevos algoritmos basados en librerías de aprendizaje automático (ML), como TensorFlow [28], PyTorch [29], Keras [30] y JAX [32]. Se han desarrollado varias bibliotecas de software diseñadas específicamente para el aprendizaje automático informado por la física, las cuales están contribuyendo al rápido desarrollo del campo (véase la tabla 5.1). En la actualidad, algunos de los entornos de desarrollo destacables son DeepXDE[33], SimNet[34], PyDEns[35], NeuroDiffEq[36], NeuralPDE[37], SciANN[38] y ADCME[39]. Dado que Python es el lenguaje predominante para el aprendizaje automático, la mayoría de estas bibliotecas están escritas en Python, a excepción de NeuralPDE y ADCME, que están escritas en Julia. Estas bibliotecas usan mecanismos de diferenciación automática como los proporcionados por TensorFlow. Algunas, como DeepXDE y SimNet, pueden actuar como solucionadores completos; es decir, el usuario define el problema y el solucionador se encarga de los detalles. Otras, como SciANN y ADCME, funcionan como puente (interfaz) que utilizan funciones de bajo nivel de otras bibliotecas como TensorFlow.

En este trabajo nos enfocamos en el uso principalmente de *PyTorch* para las simulaciones computacionales, de modo que haremos énfasis en seguida.

PyTorch como herramienta para implementación de PINNs

En el desarrollo de PINNs es fundamental calcular derivadas de las salidas de la red respecto a sus entradas. Esto puede lograrse mediante *diferenciación automática* utilizando bibliotecas como *PyTorch*, que ofrece la rutina `torch.autograd` para propagar gradientes hacia atrás y calcular derivadas con respecto a las coordenadas [29]. En particular, `torch.autograd.grad` permite obtener derivadas de primer y mayor orden estableciendo `create_graph=True` y marcando los tensores de entrada con `requires_grad=True`. Esta funcionalidad es clave para construir el residuo de la EDP (por ejemplo, $\mathcal{R}_\theta$ en (5.9)) a partir de combinaciones de derivadas temporales y espaciales. Asimismo, *PyTorch* proporciona operaciones *vector–Jacobian* y *Jacobian–vector* (VJP/JVP), útiles para el cálculo eficiente de gradientes y derivadas direccionales en problemas con términos de alto orden.

La construcción del funcional de pérdida requiere definir tres subconjuntos de puntos: el *interior* (o de colación), utilizado para evaluar el término físico $\mathcal{R}_\theta$; la *frontera*, donde se imponen las condiciones de contorno (5.3) (Dirichlet, Neumann o Robin); y los *iniciales*, empleados para imponer la condición inicial (5.2). En PyTorch, estos puntos se representan como tensores (x, t) sobre los que se evalúa la red $\hat{u}_\theta(x, t)$. El muestreo del interior se realiza de forma aleatoria con `torch.rand` (reescalando al dominio), mientras que para fronteras e iniciales es común emplear `torch.linspace` por su regularidad. Los tensores para los que se requieran derivadas (interior y, si aplica, puntos con condiciones Neumann/Robin) deben crearse con `requires_grad=True`. En condiciones de tipo Neumann, por ejemplo, $\partial_n \hat{u}_\theta$ se evalúa con `autograd` (como $\partial_x \hat{u}$ en 1D) y se penaliza su desviación respecto al valor prescrito.

Nombre del software	Uso	Lenguaje	Servidor/Backend
SimNet	Solucionador	Python	TensorFlow, PyTorch
PyDEns	Solucionador	Python	TensorFlow
NeuroDiffEq	Solucionador	Python	PyTorch
NeuralPDE	Solucionador	Julia	Julia
SciANN	Interfaz	Python	TensorFlow
ADCME	Interfaz	Julia	TensorFlow
GPyTorch	Interfaz	Python	PyTorch
Neural Tangents	Interfaz	Python	JAX

Tabla 5.1. Bibliotecas relacionadas con el aprendizaje automático informado por la física. En este trabajo se emplea exclusivamente *PyTorch* como librería base.

Una vez generados los puntos de entrenamiento, la implementación del funcional de pérdida total (5.13) en PyTorch se realiza sumando varios términos asociados a distintas partes del problema. Primero, se evalúa la red en los puntos del dominio y se calcula el residuo $\mathcal{R}_\theta$ mediante derivadas automáticas (u_t, u_x, u_{xx} en el caso de Burgers), usando `create_graph=True` para permitir derivadas de orden superior. Luego, se calcula el error cuadrático medio (MSE) sobre el residuo:

$$\texttt{torch.nn.functional.mse\_loss}(\mathcal{R}_\theta, \mathbf{0}),$$

donde $\mathbf{0}$ es un tensor[4] de ceros del mismo tamaño que $\mathcal{R}_\theta$. Para las condiciones de contorno e iniciales, se aplica el mismo criterio comparando la salida de la red con los valores impuestos. Finalmente, la pérdida total se obtiene como la suma de estos términos.

En cuanto a la optimización, PyTorch dispone de algoritmos estándar como Adam y L-BFGS a través de `torch.optim`. Una estrategia habitual en PINNs es iniciar el entrenamiento con Adam (descenso de gradiente estocástico adaptativo) para explorar el espacio de parámetros, y luego refinar con L–BFGS (método cuasi–Newton) por su buena capacidad de convergencia documentada en la literatura.

[4]En matemáticas, un tensor es una generalización de los escalares, vectores y matrices, que representa un objeto multilineal capaz de mapear varios vectores (y/o covectores) a un número real. Formalmente, un tensor de rango n puede considerarse como un arreglo multidimensional cuyos índices indican sus componentes en una base dada. En el contexto de las PINNs, un tensor en *PyTorch* es la estructura de datos fundamental para almacenar y manipular valores numéricos (por ejemplo, coordenadas espaciales, valores de la red o residuos de la EDP) y admite operaciones con gradientes automáticos, lo que permite calcular derivadas necesarias para construir la función de pérdida.

5.4 Aplicación de redes neuronales basadas en la física

A continuación, se presentan diferentes ejemplos de aplicación de las PINNs a diferentes problemas, inicialmente el primero para ilustrar la importancia de agregar información de la física del modelo al momento de entrenar una red neuronal, en seguida diferentes problemas directos e inversos en EDPs dispersivas que ilustran lo robustas que pueden ser estas nuevas técnicas.

5.4.1 Simulando el oscilador Armónico

Las redes neuronales pueden fallar al momento de predecir comportamientos futuros de un sistema físico debido a la falta de datos suficientes. Un ejemplo ilustrativo se presenta en [31], donde se modela el comportamiento de un oscilador armónico subamortiguado. El objetivo es encontrar la función de desplazamiento $u(t)$, que satisface la siguiente ecuación diferencial ordinaria (EDO):

$$m\frac{d^2u}{dt^2} + \mu\frac{du}{dt} + ku = 0, \tag{5.18}$$

donde m representa la masa, μ el coeficiente de fricción, y k la constante del resorte. Particularmente en el caso subamortiguado, el cual describe oscilaciones con amortiguamiento leve. Matemáticamente, esta condición se expresa como:

$$\delta < \omega_0, \quad \text{donde} \quad \delta = \frac{\mu}{2m}, \quad \omega_0 = \sqrt{\frac{k}{m}}. \tag{5.19}$$

Además, se imponen las siguientes condiciones iniciales:

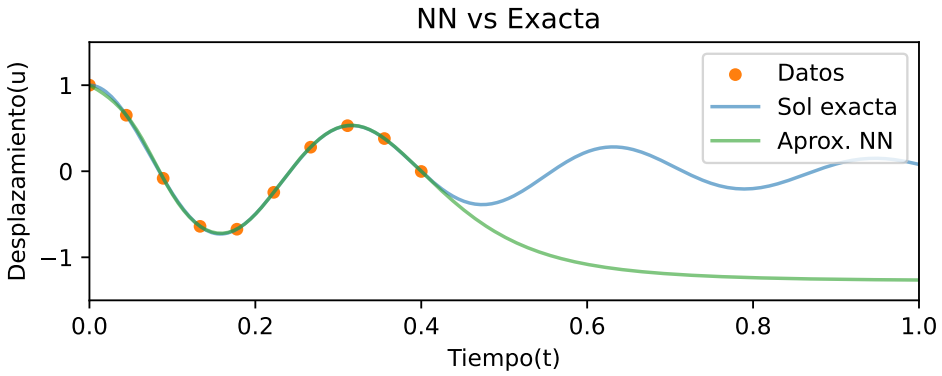
$$u(0) = 1, \quad \frac{du}{dt}(0) = 0. \tag{5.20}$$

Para este problema, se conoce una solución analítica dada por:

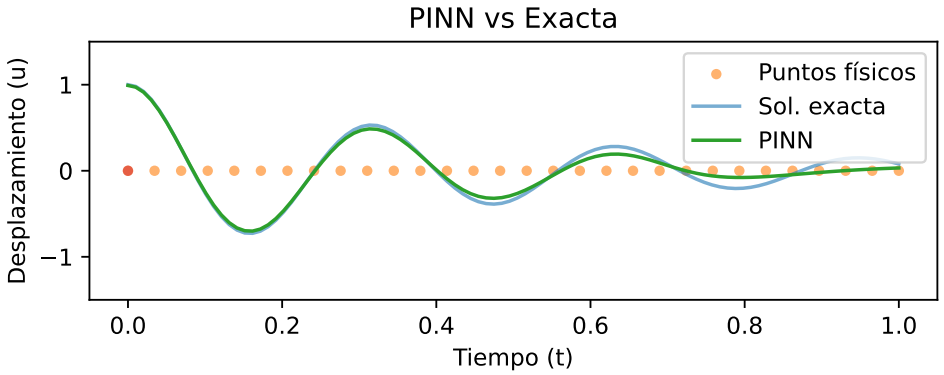
$$u(t) = 2Ae^{-\delta t}\cos(\phi + \omega t), \quad \text{donde} \quad \omega = \sqrt{\omega_0^2 - \delta^2}. \tag{5.21}$$

Los objetivos de considerar este problema, conociendo a priori la solución exacta, son primero evaluar el rendimiento de un enfoque de aprendizaje supervisado y segundo comparar el desempeño de una red neuronal clásica 4.1 con el enfoque de redes neuronales informadas por la física (PINNs), para tiempos futuros en el modelo oscilador armónico subamortiguado.

Para esto, inicialmente se aplica un enfoque de aprendizaje supervisado donde se entrena una red neuronal alimentada con datos sintéticos generados a partir de la solución exacta (5.21), evaluada en un subconjunto de tiempos $t_i \in [0, T_{\text{med}}]$. Estos datos (representados como puntos naranjas en la Figura 5.2a) se utilizan para ajustar los parámetros de una red neuronal clásica o simple, que se entrena minimizando el funcional de pérdida cuadrática media (MSE):



(a) Red neuronal NN vs Solución exacta.



(b) PINN vs Solución exacta.

Figura 5.2. Comportamiento en tiempos futuros de redes neuronales y PINNs.

$$\frac{1}{N}\sum_{i=1}^{N}\left|u_{\mathrm{NN}}(t_i;\theta)-u_{\mathrm{true}}(t_i)\right|^2, \tag{5.22}$$

donde $u_{\mathrm{NN}}(t_i;\theta)$ representa la salida de la red neuronal con parámetros θ evaluada en el tiempo t_i, y $u_{\mathrm{true}}(t_i)$ corresponde a los valores verdaderos (exactos) usados como referencia. La red utilizada es una arquitectura del tipo *feedforward* completamente conectada 4.1, compuesta por tres capas ocultas, cada una con 50 neuronas y funciones de activación tanh (4.3), y una capa de salida lineal. El modelo se implementa en PyTorch y se entrena durante 10001 épocas con el optimizador Adam y una tasa de aprendizaje (tolerancia) de 10^{-3}. Se configuró el entorno de ejecución para producir resultados y se usaron únicamente 10 puntos de entrenamiento uniformemente distribuidos en el intervalo $[0, 0{,}4]$.

En cuanto al desempeño, se observa que la red neuronal ajusta correctamente los datos dentro del intervalo de entrenamiento, (curva verde gráfica 5.2a) pero su predicción para tiempos futuros (más allá de $T_{med} = 0{,}4$) tiende a alejarse de la solución exacta (curva azul gráfica 5.2a). Este comportamiento es esperado en redes supervisadas cuando el dominio de

inferencia extrapola los datos vistos en el entrenamiento, pero no es capaz de generalizar. En este caso, la pérdida durante el entrenamiento fue baja (del orden de 10^{-5} a 10^{-6}), lo cual indica buen ajuste local, pero no garantiza generalización más allá del intervalo entrenado (ver figura 5.3a). Aunque existen diferentes técnicas de aprendizaje automático para mejorar la predicción de la solución (aumentar el número de puntos de entrenamiento, incluir técnicas de regularización, utilizar validación cruzada, entre otras). Estamos interesados en mirar la incidencia de incorporar la física del modelo a la red neuronal como estrategia de aprendizaje. Los detalles completos de la implementación y los parámetros utilizados en el experimento se encuentran disponibles en el repositorio github.

Incidencia al agregar la física durante el entrenamiento

En el segundo enfoque incorporamos de manera explícita la dinámica del oscilador armónico subamortiguado al proceso de entrenamiento. Como se ha mencionado anteriormente, la idea fundamental consiste en enriquecer el funcional de pérdida con los residuos de la ecuación diferencial, de modo que el modelo no dependa exclusivamente de datos supervisados. Sea $\hat{u}(t;\theta)$ la salida de la red neuronal parametrizada por θ, la cual se define con la misma arquitectura del experimento anterior. Definimos la pérdida total (5.13) de $\hat{u}$, como

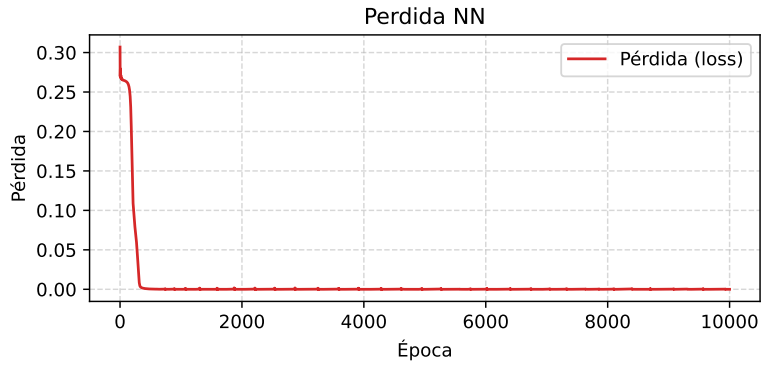
$$\mathcal{L}(\theta) = \mathcal{L}_{\mathrm{BC}_1}(\theta) + \lambda_1 \mathcal{L}_{\mathrm{BC}_2}(\theta) + \lambda_2 \mathcal{L}_{\mathrm{phys}}(\theta),$$

donde

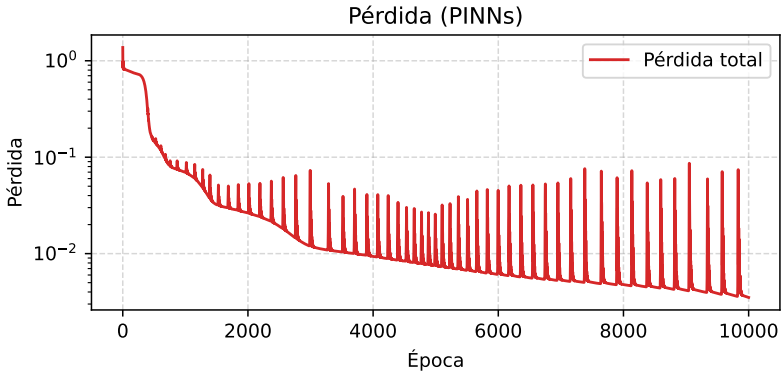
$$\mathcal{L}_{\mathrm{BC}_1} = \left|\hat{u}(0;\theta) - 1\right|^2, \quad \mathcal{L}_{\mathrm{BC}_2} = \left|\tfrac{d}{dt}\hat{u}(0;\theta)\right|^2,$$

$$\mathcal{L}_{\mathrm{phys}} = \frac{1}{N_r}\sum_{i=1}^{N_r} \left| \underbrace{\tfrac{d^2}{dt^2}\hat{u}(t_i;\theta) + \mu\, \tfrac{d}{dt}\hat{u}(t_i;\theta) + k\, \hat{u}(t_i;\theta)}_{\mathcal{R}_\theta(t_i)} \right|^2 .$$

Aquí los términos $\mathcal{L}_{\mathrm{BC}_1}$ y $\mathcal{L}_{\mathrm{BC}_2}$ indican la pérdida de las condiciones iniciales (5.20), y los puntos $\{t_i^{(r)}\}$ son muestras aleatorias uniformes en $[0,1]$. λ_1 y λ_2 son parámetros que se utilizan para balancear los términos en la función de pérdida y asegurar la estabilidad durante el entrenamiento (los valores seleccionados fueron similares a los elegidos por B. Moseley en [31]). Una característica esencial del enfoque PINN es que la red neuronal no se entrena únicamente con datos observados o sintéticos, sino que también se le exige cumplir la dinámica interna del sistema físico, evaluando directamente el residuo $\mathcal{R}_\theta$ de la ecuación diferencial en dicho conjunto de puntos seleccionados dentro del dominio de interés. En este experimento, dichos puntos corresponden a instantes de tiempo $t \in [0,1]$, donde se evalúa la validez de la solución aproximada $\hat{u}(t;\theta)$ que genera la red. La muestra equiespaciada de $N_r = 30$ puntos permite evaluar la consistencia de la red con la ecuación diferencial, aun cuando no se disponga de valores exactos de $u(t)$ en tales puntos. Este enfoque constituye una forma de regularización inductiva: al imponer que la red respete la física del sistema en todo el dominio, se logra una solución coherente incluso fuera del intervalo cubierto por las condiciones iniciales.



(a) Pérdida NN



(b) Pérdida PINN

Figura 5.3. Comportamiento de los funcionales de pérdida (x- época vs y-pérdida, escala normal).

Desde el punto de vista computacional, este proceso se apoya en la capacidad de PyTorch de realizar autodiferenciación sobre tensores, lo cual permite calcular derivadas de cualquier orden respecto al tiempo t activando la opción `requires_grad_()`. Esta característica es crucial para evaluar el residuo $\mathcal{R}_\theta(t)$ con precisión en los puntos de colocación. Para más detalles de la implementación computacional, puede consultar el repositorio github donde se encuentran los dos enfoques.

Al comparar las predicciones obtenidas con el modelo PINN frente al enfoque supervisado, se evidencia una diferencia fundamental en su comportamiento para tiempos $t > 0{,}4$, como se muestra en la figura 5.2b. Mientras el modelo clásico tiende a colapsar, la PINN preserva la estructura amortiguada de la solución exacta, confirmando que la información física actúa como guía estructural y mejora la convergencia del entrenamiento.

Este fenómeno puede interpretarse como una capacidad de generalización inducida por la física, ya que la PINN, al incorporar explícitamente la ecuación diferencial y las condiciones iniciales como términos del funcional de pérdida, logra extender el comportamiento correcto de la solución más allá de los datos supervisados. En otras palabras, la red no memoriza únicamente los datos vistos, sino que aprende a respetar la estructura dinámica del sistema.

Para afianzar lo anterior, podemos estimar la L^2-norma [5] para comparar tanto las aproximaciones dadas por el enfoque supervisado como el enfoque dado por las PINNs con la solución exacta de nuestro primer modelo de prueba. Los resultados para diferentes épocas del entrenamiento y $t = 1{,}0$ se pueden ilustrar en la siguiente tabla 5.2,

Época	$\|u_{NN} - u_{exacta}\|_{L^2}$	$\|u_{PINN} - u_{exacta}\|_{L^2}$
2000	0.827496	0.128075
4000	0.822963	0.087577
6000	0.817514	0.094435
8000	0.816946	0.110728
10000	0.813500	0.110932

Tabla 5.2. Comparación del error en norma L^2 entre los enfoques supervisado (NN) y las redes neuronales informadas por la (PINN) para el oscilador armónico subamortiguado en distintas épocas, estos valores pueden verificarse en el repositorio github

En la Figura 5.3(b) se aprecia que el funcional de pérdida PINN disminuye de forma uniforme, reflejando cómo el modelo asimila tanto las condiciones iniciales como la propia ecuación diferencial.

En resumen, este enfoque híbrido resulta especialmente poderoso en problemas donde los datos supervisados son escasos: al incorporar la ecuación física y las condiciones iniciales o de frontera, la red generaliza más allá del intervalo entrenado y alcanza precisiones que superan en órdenes de magnitud al método puramente supervisado, sin incrementar la cantidad de datos de entrenamiento. Por otro lado, el muestreo aleatorio de los puntos de colocación es suficiente para cubrir el dominio temporal, lo que reduce la necesidad de diseñar rejillas fijas. Además, la autodiferenciación agiliza la implementación y asegura la precisión numérica de las derivadas.

5.5 Problema directo asociado a la ecuación de Burgers

En este trabajo nos enfocamos en aplicar redes neuronales informadas por la física (PINNs) a modelos evolutivos (3.1) gobernados por ecuaciones diferenciales parciales. Como punto de partida, consideramos la ecuación de Burgers en una dimensión, dada por (3.4). Esta ecuación constituye un modelo fundamental en el estudio de fenómenos no lineales que involucran difusión y convección; adicionalmente, captura características esenciales de las ecuaciones de Navier-Stokes (3.2) en un contexto simplificado.

[5] El error de aproximación puede cuantificarse mediante la L^2-norma discreta definida como:

$$\|u - \hat{u}\|_{L^2} := \left(\frac{1}{N} \sum_{i=1}^{N} [u(t_i) - \hat{u}(t_i)]^2 \right)^{1/2},$$

donde $\{t_i\}_{i=1}^N$ es un conjunto de puntos de evaluación en el dominio temporal, y N es su cardinalidad.

Entre sus aplicaciones más relevantes se encuentra la modelación de ondas de choque, la dinámica de gases compresibles y el flujo vehicular en una sola dirección [61, 62]. Gracias a su estructura matemática relativamente accesible y su capacidad para representar dinámicas no lineales complejas, la ecuación de Burgers es ampliamente utilizada como modelo de prueba para verificar métodos numéricos. Además, bajo ciertas condiciones como viscosidad positiva y dominios adecuados, admite una solución exacta mediante la transformada de Cole-Hopf [47], lo que permite validar aproximaciones numéricas y computacionales con un referente analítico confiable.

Utilizando condiciones de frontera tipo Dirichlet, el *problema directo* asociado a la ecuación de Burgers en una dimensión consiste en determinar la evolución temporal del campo escalar $u(x,t)$ en un dominio espacio temporal dado, a partir de una condición inicial y valores de frontera especificados. La formulación matemática es la siguiente:

$$\begin{cases} \dfrac{\partial u}{\partial t} + u\dfrac{\partial u}{\partial x} = \nu \dfrac{\partial^2 u}{\partial x^2}, & \text{en } (x,t) \in (-1,1) \times (0,1], \\ u(x,0) = u_0(x), & x \in [-1,1], \\ u(t,-1) = u(t,1) = 0, & t \in [0,T], \end{cases} \tag{5.23}$$

Aquí, $\nu > 0$ representa la viscosidad cinemática del medio y la condición $u(x,0) = u_0(\cdot)$ define un perfil el instante inicial. Este problema es considerado bien planteado en el sentido de Hadamard. En particular, para $\nu > 0$, la ecuación de Burgers se comporta como una ecuación parabólica no lineal, lo que garantiza la suavidad y estabilidad de la solución bajo condiciones adecuadas. Por ejemplo, se ha estudiado en [50, 51], la existencia y unicidad de la solución, para $u(\cdot,\cdot)$ y u_0 en espacios adecuados.

Para el enfoque PINNs, aproximamos $u(\cdot,\cdot)$ mediante una red neuronal $\hat{u}_\theta(\cdot,\cdot)$ parametrizada por un vector de pesos θ. Comenzamos con un ejemplo en particular cuando $\nu = \frac{1}{100\pi}$ y la condición inicial viene dada por,

$$u_0(x) = \frac{2\nu\pi \sin(\pi x)}{\sigma + \cos(\pi x)}, \qquad \sigma > 1 \tag{5.24}$$

Elegimos esta función dado que conocemos a priori la solución exacta del problema directo (5.23) mediante la transformada de Cole-Hopf [42]. En consecuencia, podemos analizar el desempeño de las PINNs en este modelo en particular. Para este caso, la red neuronal recibe como entrada puntos en el dominio espaciotemporal (x,t) y produce como salida una aproximación de la solución $\hat{u}(x,t)$. La función $\hat{u}$ es una red neuronal completamente conectada con dos capas ocultas de 50 neuronas cada una y función de activación tangente hiperbólica. Siguiendo la metodología PINNs, para incorporar la estructura física del problema se define la función de pérdida total

$$\mathcal{L}(\theta) = \lambda_{ic}\mathcal{L}_{\text{ic}}(\theta) + \lambda_{bc}\mathcal{L}_{\text{bc}}(\theta) + \lambda_r\mathcal{L}_r(\theta) \tag{5.25}$$

Los componentes de esta ecuación vienen dados por,

$$\mathcal{L}_{\mathrm{r}} = \frac{1}{N_r}\sum_{i=1}^{N_r}\left|\underbrace{\frac{\partial \hat{u}}{\partial t} + \hat{u}\frac{\partial \hat{u}}{\partial x} - \nu\frac{\partial^2 \hat{u}}{\partial x^2}}_{\mathcal{R}_\theta}\right|^2 .$$

$$\mathcal{L}_{\mathrm{IC}} = \frac{1}{N_0}\sum_{i=1}^{N_0}|\hat{u}(x_i,0) - u_0(x_i)|^2 \quad \text{y} \quad \mathcal{L}_{\mathrm{BC}} = \frac{1}{N_b}\sum_{i=1}^{N_b}\left(|\hat{u}(-1,t_i)|^2 + |\hat{u}(1,t_i)|^2\right).$$

Aquí la pérdida residual de la EDP $\mathcal{L}_r$ está evaluada en puntos de la forma (x_i, t_i) con $1 \leq i \leq N_r$, los cuales son muestreados aleatoriamente en el dominio interior $(x,t) \in (0,1)\times(0,T]$. El término $\mathcal{L}_{IC}$ mide la pérdida en la condición inicial, garantiza que $\hat{u}$ coincida con la condición inicial $u_0(x)$, en el instante inicial $t = 0$, en este caso $\mathcal{L}_{IC}$ se mide en puntos $\{x_i\}_{i=1}^{N_0}$ sobre el eje espacial. Finalmente $\mathcal{L}_{BC}$ mide la pérdida en la frontera e impone que $\hat{u}$ respete las condiciones Dirichlet homogéneas en $x = -1$ y $x = 1$, se evalúa en puntos $\{t_i\}_{i=1}^{N_b}$ del intervalo temporal $[0,1]$. Adicionalmente, los coeficientes $\lambda_{ic}, \lambda_{bc}, \lambda_r$ en (5.25) permiten ponderar la importancia relativa de cada término durante el entrenamiento, para este experimento consideramos únicamente el funcional de pérdida como en (5.13), es decir los dichos es calares iguales a 1.

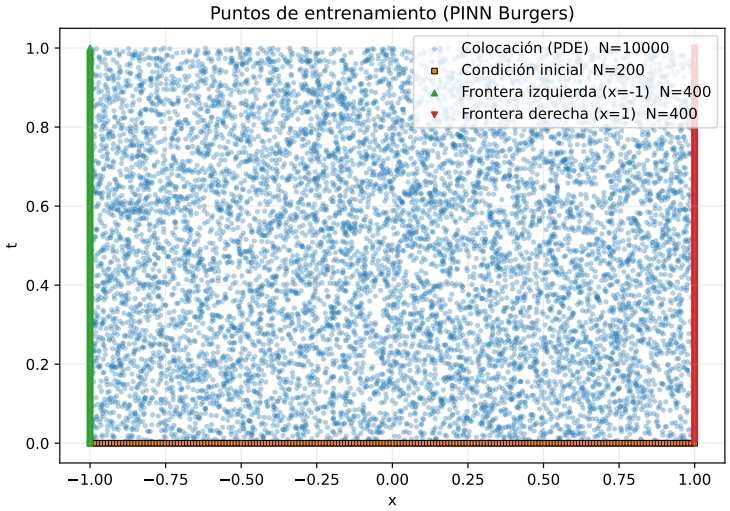
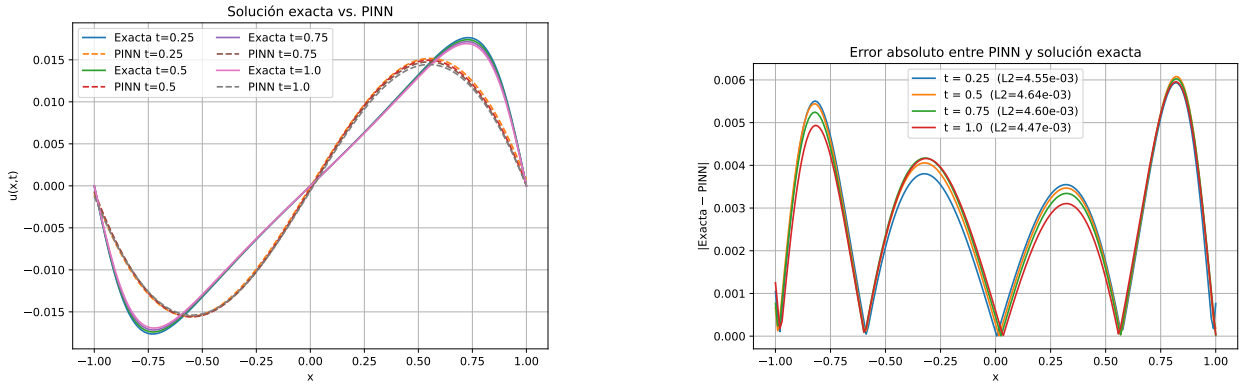


Figura 5.4. Puntos de entrenamiento para la simulación de PINNs

Para el entrenamiento se generan $N_r = 10,000$ puntos interiores de colocalización $(x_\mathrm{i}, t_\mathrm{i})$ de manera aleatoria en el dominio $[-1,1]\times[0,1]$. Dentro de la implementación computacional estas coordenadas se marcan con `requires_grad=True` para habilitar el cálculo de derivadas automáticas respecto a x y t. Además, se utilizan $N_0 = 200$ puntos uniformemente distribuidos en el espacio para $t = 0$, a fin de imponer la condición inicial, y N_b puntos en la frontera (400 en $x = -1$ y 400 en $x = 1$) para imponer las condiciones de frontera para todo t en $[0,1]$.



(a) Solución exacta vs aproximación PINN **(b)** Curvas de error absoluto

Figura 5.5. Resultados experimentación para diferentes t_i del problema directo

La cantidad significativa de puntos interiores asegura una cobertura adecuada del dominio y contribuye a una mejor aproximación de la solución.

El entrenamiento de la PINN se realiza utilizando el optimizador Adam, con una tasa de aprendizaje de $\epsilon = 10^{-3}$ para un total de 10000 épocas. En cada iteración, se calcula la función de pérdida total, se computan los gradientes mediante retropropagación y se actualizan los parámetros de la red, siguiendo el esquema presentado en la figura 5.1.

Una vez entrenado el modelo se obtuvo una disminución total del funcional 5.25 del orden de 10^{-5} y, terminada la simulación, se evalúa la precisión de la aproximación PINN frente a la solución exacta. Para ello, se consideran cuatro instantes de tiempo relevantes: $t = 0{,}25$, $t = 0{,}5$, $t = 0{,}75$ y $t = 1{,}0$. Para cada uno de estos tiempos, se calcula la solución exacta y la aproximación PINN sobre una malla espacial uniforme de 200 nodos en $x \in [-1, 1]$. Esta comparación permite obtener tanto las curvas de la solución como las curvas de error absoluto en función de x (ver figura 5.5). En 5.5b, se observa que el error absoluto puntual presenta picos alrededor de las regiones de mayor curvatura de onda senoidal.

Los resultados muestran una buena correspondencia visual entre la solución exacta y la aproximación de la PINN. El error absoluto y relativo presenta valores del orden de 10^{-3} y 10^{-1}, respectivamente, como se muestra en la siguiente tabla,

t	$\|u_{\text{exact}} - u_{\text{PINN}}\|_{L^2}$	$\dfrac{\|u_{\text{exact}} - u_{\text{PINN}}\|_{L^2}}{\|u_{\text{exact}}\|_{L^2}}$
0.25	4.628e-03	2.834e-01
0.50	4.695e-03	2.910e-01
0.75	4.612e-03	2.894e-01
1.00	4.393e-03	2.789e-01

Tabla 5.3. Errores absoluto y relativo en norma L^2 para distintos tiempos.

Estos valores muestran que la PINN logra una aproximación notablemente precisa de la solución exacta a lo largo del tiempo.

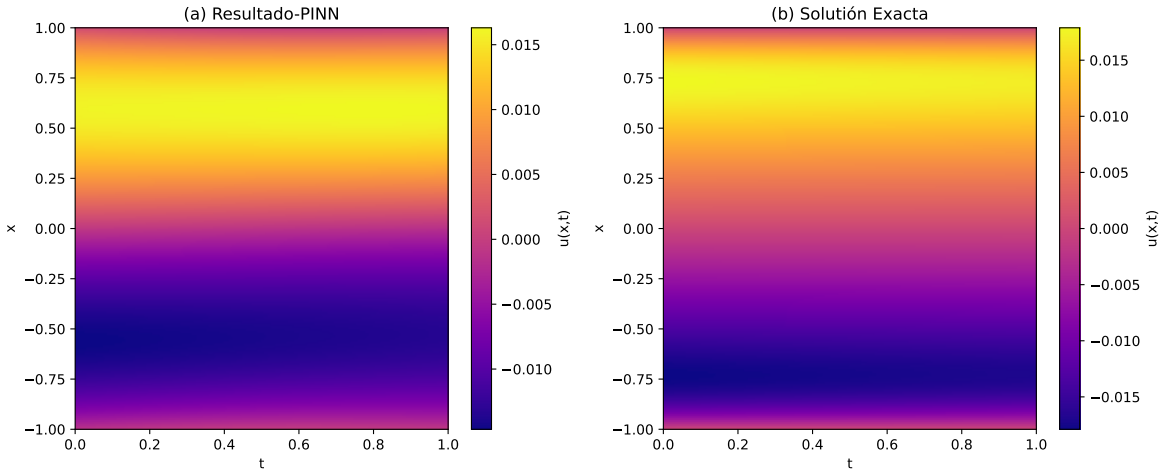


Figura 5.6. Resultados 2D del experimento

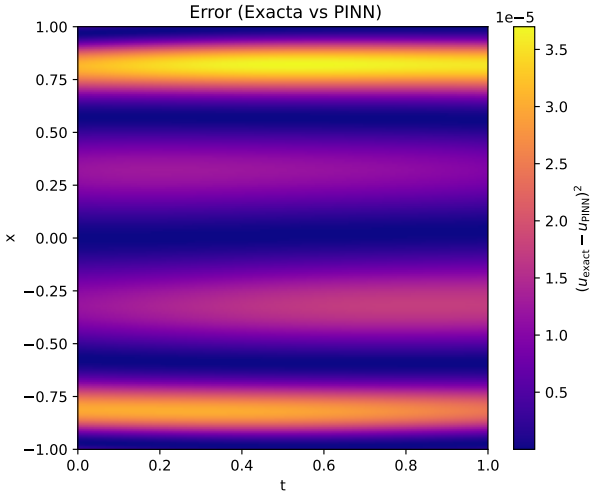


Figura 5.7. Error medido como mapa de calor

Para una visualización global tanto de la solución como del comportamiento del error, se genera un mapa de calor bidimensional sobre una malla densa de 200×200 puntos en el dominio $x \in [-1, 1]$, $t \in [0, 1]$. Para el caso de la solución se plotean la aproximación PINN y la solución exacta (ver figura 5.6) aquí el mapa de calor muestra la distribución espacial y temporal de los valores de la función $u(x, t)$ en todo el dominio, donde el subgráfico (a) corresponde a la aproximación obtenida mediante la red neuronal informada por la física y el subgráfico (b) a la solución exacta, evidenciando la excelente concordancia entre ambas representaciones. En el caso de la figura 5.7 l el mapa evidencia que el error permanece uniformemente bajo en todo el dominio, sin presentar concentraciones significativas en ninguna región particular, lo que indica que la red neuronal ha capturado adecuadamente tanto la dinámica temporal como la espacial de la ecuación de Burgers bajo las condiciones dadas.

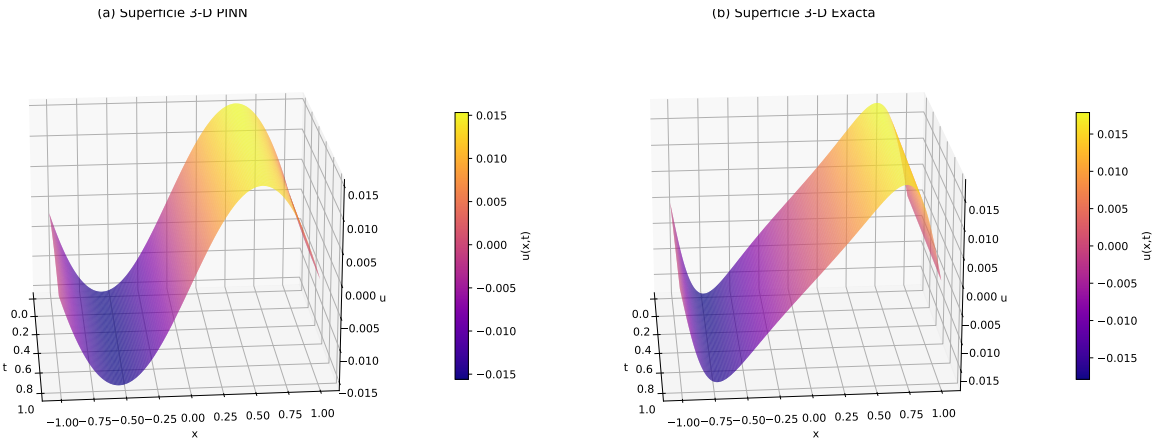


Figura 5.8. Resultados 3D del experimento PINN vs exacta

Finalmente, como actividad pedagógica, se representan en 3-D las superficies de $u_{\text{PINN}}(x,t)$ y $u_{\text{exact}}(x,t)$, interpoladas sobre la malla uniforme de 200×200 nodos en el dominio $x \in [-1,1]$, $t \in [0,1]$. Cada gráfica muestra la posición x en el eje horizontal, el tiempo t en profundidad y la magnitud de $u(x,t)$ en el eje vertical, coloreada con la paleta *plasma* para realzar los gradientes. La comparación evidencia la alta fidelidad de la PINN al reproducir la disipación viscosa y las ondulaciones de la solución exacta, con discrepancias prácticamente imperceptibles en todo el intervalo espacio-temporal.

La implementación completa de este experimento, junto con un *notebook* interactivo que permite rotar, hacer *zoom* y explorar punto a punto la simulación, se encuentra disponible en el repositorio de código abierto: `github.com/DarwinFer/Burgers1D_PD` donde se incluyen las instrucciones detalladas para reproducir el experimento y personalizar los parámetros de entrenamiento.

En conclusión, la arquitectura y entrenamiento descritos permiten que la PINN alcance una alta precisión en la resolución de la ecuación de Burgers, con errores sistemáticamente pequeños tanto en sentido puntual como en norma L^2, demostrando así la efectividad de este enfoque basado en aprendizaje profundo informado por la física.

5.5.1 PINNs vs FEM

En este experimento deseamos comparar el desempeño de las PINNs con el conocido método de elementos finitos (FEM), consideramos un problema directo similar al considerado en (5.23), pero con la condición inicial,

$$u_0(x) = \sin(\pi x), \quad x \in (-1,1). \tag{5.26}$$

Cuya solución exacta no es conocida, en consecuencia el método FEM se considera entonces como referencia numérica fiable para evaluar el desempeño de las PINNs. El método

de elementos finitos (FEM) discretiza el dominio espacial $[-1, 1]$ mediante una malla uniforme de 400 elementos lineales, donde la solución $u(x, t)$ se aproxima mediante funciones base "P_1" denotadas por $\{\phi_j\}$, de modo que $u_h(x, t) = \sum_{j=1}^{N} u_j(t)\phi_j(x)$. Aplicando la formulación débil de Galerkin a la ecuación de Burgers, se obtiene un sistema semi-discreto de ecuaciones diferenciales ordinarias (EDOs):

$$\mathbf{M}\frac{d\mathbf{u}}{dt} + \mathbf{u} \circ (\mathbf{Du}) + \nu\mathbf{Ku} = \mathbf{0}, \tag{5.27}$$

donde $\mathbf{M}, \mathbf{D}, \mathbf{K}$ se conocen como matrices características, las cuales se ensamblan mediante productos internos: $\mathbf{M}$ (denominada matriz de masa masa) integra $\phi_i\phi_j$, $\mathbf{K}$ (rigidez) sus derivadas $\nabla\phi_i\nabla\phi_j$, y $\mathbf{D}$ (convección) $\nabla\phi_i\phi_j$. La no-linealidad convectiva $u\partial_x u$ se discretiza eficientemente mediante el producto de Hadamard $(\mathbf{u} \circ (\mathbf{Du}))$, preservando la estructura física de la ecuación. Las condiciones de frontera de Dirichlet $(u(\pm 1, t) = 0)$ se imponen fuertemente anulando du_i/dt en los nodos correspondientes. Finalmente, el sistema de EDOs (5.27) se resuelve temporalmente con el método BDF implícito adaptativo (implementado en `solve_ivp` de SciPy[71]), que maneja automáticamente la rigidez mediante pasos adaptativos con tolerancias 10^{-9} (absoluta) y 10^{-6} (relativa). Esta combinación garantiza estabilidad numérica incluso para el término convectivo dominante, siendo el esquema temporal implícito particularmente adecuado para problemas difusivos-convectivos [70].

La PINN implementada se mantiene idéntica arquitectura al caso anterior: 2 capas ocultas de 50 neuronas con activación tanh, optimizada con Adam ($\epsilon = 10^{-3}$, 10000 épocas). Los puntos de entrenamiento ($N_r = 10,000$ interiores, $N_0 = 200$ iniciales, $N_b = 400$ frontera) y ponderaciones ($\lambda_{ic} = \lambda_{bc} = \lambda_r = 1$) se conservan. La única modificación considerable es la condición inicial, implementada en $\mathcal{L}_{\text{IC}}$.

$$\mathcal{L}_{\text{IC}}(\theta) = \frac{1}{N_0}\sum_{i=1}^{N_0} |\hat{u}(x_i, 0) - u_0(x_i)|^2$$

de esta forma se actualiza la función de pérdida total (5.13). Finalizamos el entrenamiento de la PINN y se observa el siguiente comportamiento del funcional de pérdida,

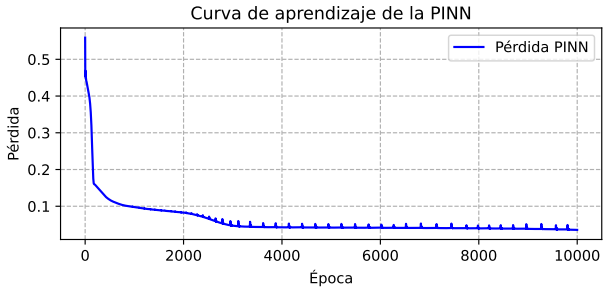


Figura 5.9. Curva de aprendizaje de la PINN.

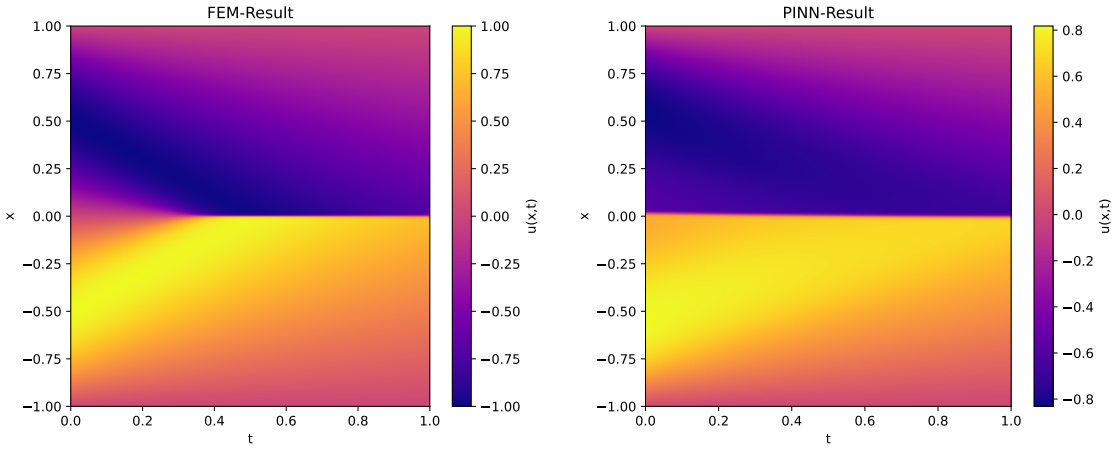
Para cuantificar la diferencia entre la solución FEM u_h y la aproximación PINN $\hat{u}$, eva-

luamos la norma L^2 y la norma infinito [6] para $e = u_h - \hat{u}$ y $\{x_j\}$ es la malla común de 200 nodos espaciales. En la siguiente tabla se resumen los resultados para este experimento.

t	$\|e(t)\|_{L^2}$	$\|e(t)\|_{L^\infty}$
0.25	$1{,}3 \times 10^{-1}$	$4{,}5 \times 10^{-1}$
0.50	$1{,}2 \times 10^{-1}$	$5{,}8 \times 10^{-1}$
0.75	$5{,}2 \times 10^{-2}$	$3{,}0 \times 10^{-1}$
1.00	$1{,}9 \times 10^{-2}$	$2{,}3 \times 10^{-1}$

Tabla 5.4. Errores globales FEM–PINN a distintos tiempos.

Se observa que los errores se mantienen en el rango 10^{-2}–10^{-1}, confirmando que la PINN aproxima con buena fidelidad la solución de referencia aún sin disponer de la forma analítica exacta. Los resultados graficos de las simulaciónes son los siguientes



(a) Aproximación FEM **(b)** Aproximación PINN

Figura 5.10. Resultados aproximación de la solución

Las aproximaciones mediante el Método de Elementos Finitos (FEM) y la Red Neuronal Informada por la Física (PINN) (figuras 5.10a 5.10b) reproducen con fidelidad la disipación de la onda inicial de Burgers sobre el dominio espacial $x \in [-1, 1]$ y temporal $t \in [0, 1]$. En particular, ambas soluciones muestran el aplanamiento progresivo de la curva senoidal y mantienen la condición de frontera homogénea en $x = \pm 1$.

No obstante, al comparar las barras de color se observa que la PINN tiende a subestimar ligeramente la amplitud máxima (aproximándose a $\pm 0{,}8$) frente al rango completo $\pm 1{,}0$

6

$$\|e(t)\|_{L^2} = \left(\frac{1}{|\Omega_x|}\sum_j |e(x_j,t)|^2\right)^{1/2} \qquad \text{y} \qquad \|e(t)\|_{L^\infty} = \max_j |e(x_j,t)|,$$

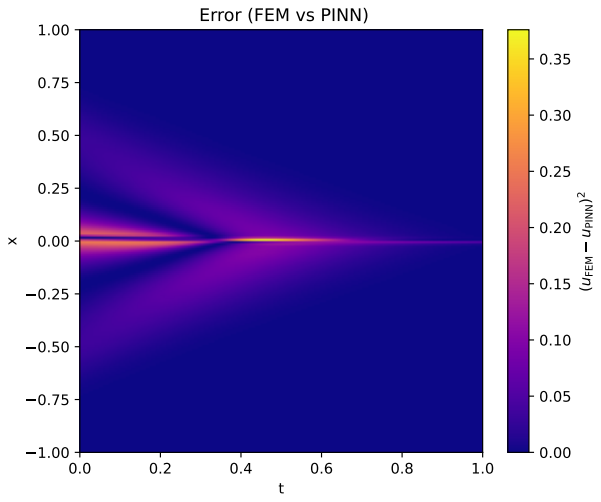


Figura 5.11. Error medido como mapa de calor FEM-PINN

registrado por la FEM, lo cual sugiere una pequeña discrepancia en las crestas iniciales de la onda.

El mapa de error cuadrático $(u_{\text{FEM}} - u_{\text{PINN}})^2$, evaluado sobre una malla densa de 200×200 puntos (figura 5.11), revela que la máxima diferencia se localiza aproximadamente en torno a $t \in (0{,}2, 0{,}4)$ y muy cerca de $x = 0$, donde los gradientes espaciales son más pronunciados y la no linealidad convectiva $u\,u_x$ impone un reto de aproximación mayor lo cual es de esperarse. Hacia tiempos posteriores, el error decrece de forma notable, lo que indica que la PINN alcanza una mejor concordancia con la disipación viscosa una vez que el perfil se suaviza.

En cuanto al coste computacional, la implementación FEM en una dimensión implicó el ensamblaje y la resolución de sistemas lineales dispersos en cada paso de tiempo, lo cual, para una malla de tamaño 200×200, suele ejecutarse en pocos segundos en CPU de propósito general. Por el contrario, la PINN demanda una fase de entrenamiento significativa: con Adam a tasa de aprendizaje 10^{-3} durante 10 000 épocas, muestreando aleatoriamente 10^4 puntos interiores en cada iteración y cientos de puntos de contorno e iniciales, cada ciclo de forward–backward que incluye el cálculo de derivadas por *autograd* puede costar milisegundos en GPU o varias décimas de segundo en CPU. En consecuencia, el entrenamiento completo puede extenderse a varios minutos o decenas de minutos, dependiendo del hardware disponible. No obstante, una vez finalizado, la PINN ofrece evaluaciones prácticamente instantáneas en cualquier punto del dominio, sin necesidad de reensamblar ni resolver nuevos sistemas lineales, lo que resulta ventajoso en escenarios de consulta rápida o de análisis paramétrico, lo anterior valida las ideas presentadas en [72]. La experimentación completa de este experimento se encuentra disponible `github.com/DarwinFer/Burgers1DPINNvsFEME3`.

5.6 Análisis de costo computacional

Para validar las últimas ideas presentadas, a continuación presentamos un estudio el cual tiene como propósito comparar el coste computacional de dos enfoques numéricos distintos aplicados a la resolución de la ecuación de Burgers. Para ello tomamos ambos métodos implementados sobre un mismo dominio: $x \in [-1, 1]$ y $t \in [0, 1]$, con condición inicial $u(x, 0) = -\sin(\pi x)$ y condiciones de frontera homogéneas $u(-1, t) = u(1, t) = 0$. Se utilizó una viscosidad de referencia dada por $\nu = 0{,}01/\pi$, en el problema directo (5.23).

En cuanto a las configuraciones evaluadas, FEM se aplicó sobre mallas con 401 y 801 nodos, correspondientes a 400 y 800 elementos, respectivamente. Por otro lado, las PINNs fueron entrenadas utilizando arquitecturas con 50 y 100 neuronas por capa, considerando profundidades de 3 y 4 capas. Esto permitió analizar cómo el tamaño y complejidad de la red afecta su rendimiento computacional.

Durante cada ejecución se midió el tiempo total de cómputo, el tiempo normalizado (es decir, el tiempo por nodo en FEM o por parámetro en PINN), así como la memoria pico[7] utilizada por cada enfoque. Todos los experimentos se realizaron sobre CPU `Intel(R) Xeon(R)` y GPU habilitada (`Tesla T4`) de google colab, para garantizar consistencia en la comparación respecto a los métodos.

Tabla 5.5. Coste computacional conjunto (Hardware **HW** CPU/GPU)

Método	**Configuración**	**Tiempo (s)**	**Tiempo Norm.**	**Memoria (MB)**	**HW**
FEM	Nodos: 401	0.249	$6{,}20\times10^{-4}$	16.04	CPU
FEM	Nodos: 801	0.731	$9{,}13\times10^{-4}$	32.04	CPU
PINN	Neuronas: 50, Capas: 3	2.944	$5{,}55\times10^{-4}$	0.021	CPU
PINN	Neuronas: 50, Capas: 4	2.972	$3{,}79\times10^{-4}$	0.031	CPU
PINN	Neuronas: 100, Capas: 3	4.639	$2{,}25\times10^{-4}$	0.082	CPU
PINN	Neuronas: 100, Capas: 4	5.916	$1{,}93\times10^{-4}$	0.123	CPU
FEM	Nodos: 401	0.574	$1{,}43\times10^{-3}$	11.15	CUDA
FEM	Nodos: 801	0.086	$1{,}07\times10^{-4}$	18.84	CUDA
PINN	Neuronas: 50, Capas: 3	2.254	$4{,}25\times10^{-4}$	18.55	CUDA
PINN	Neuronas: 50, Capas: 4	2.805	$3{,}57\times10^{-4}$	18.69	CUDA
PINN	Neuronas: 100, Capas: 3	2.451	$1{,}19\times10^{-4}$	19.06	CUDA
PINN	Neuronas: 100, Capas: 4	2.617	$8{,}50\times10^{-5}$	19.40	CUDA

Los resultados obtenidos se resumen en la Tabla 5.5. Se observa que en CPU, FEM logra resolver el problema en menos de un segundo, incluso en configuraciones con alta resolución espacial. En cambio, PINN requiere entre 3 y 6 segundos, dependiendo de la arquitectura utilizada. Mientras que con GPU, el panorama cambia notablemente las PINNs tardan entre

[7]En CPU la memoria reportada para PINN corresponde a una estimación de *parámetros*×4 bytes; en GPU se usó el máximo de memoria reservada, que incluye activaciones y *buffers* del optimizador. Por ello, los valores CPU/GPU de memoria no son estrictamente comparables en términos absolutos.

2,25 y 2,61 segundos en entrenar y evaluar por cada configuración, es decir, casi a la mitad del tiempo observado en CPU para las mismas redes (ver filas CUDA de la Tabla 5.5). Se observa que la configuración más grande (100 neuronas, 4 capas) es la que mejor aprovecha la tarjeta, con 2,617s y el menor tiempo normalizado $\left(8{,}5 \times 10^{-5}\right)$.

A partir de lo anterior y los valores de la Tabla 5.5 podemos extraer algunas conclusiones. En CPU, FEM es el método más rápido por ejecución y consigue tiempos por debajo del segundo, por lo que resulta adecuado cuando se necesita una respuesta puntual con rapidez. Al utilizar GPU, las PINNs se aceleran de forma notable y, además, las arquitecturas más grandes son las que mejor aprovechan la tarjeta al registrar un menor tiempo normalizado. En el caso de FEM, la GPU sólo ofrece una ventaja clara cuando la malla es más fina (801 nodos); con mallas pequeñas, la CPU sigue siendo competitiva. En cuanto a memoria, en CPU las PINNs son significativamente más ligeras que FEM, mientras que en GPU ambos enfoques quedan en el mismo orden de magnitud. Por último, el comportamiento al escalar es sencillo de interpretar: en PINN, aumentar neuronas o capas reduce el tiempo por parámetro, y en FEM sobre CPU el tiempo por nodo crece con el refinamiento; en FEM sobre GPU la ganancia aparece principalmente con mallas grandes.

Adicionalmente, la Figura 5.12 ilustra gráficamente la comparación entre los tiempos de ejecución y el uso de memoria para todas las configuraciones estudiadas.

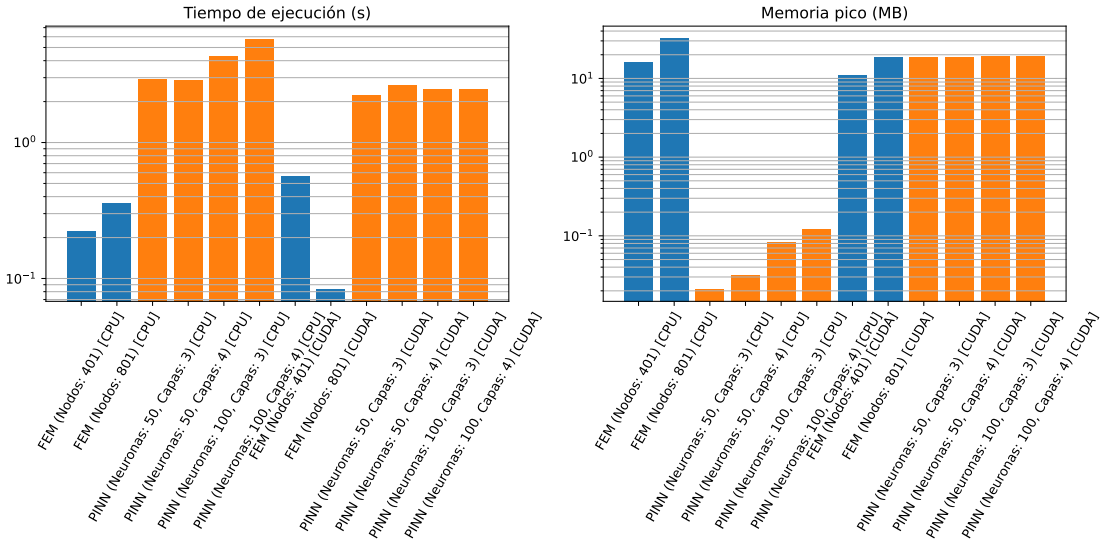


Figura 5.12. Comparación de tiempos de ejecución y uso de memoria

Respecto al uso de memoria en CPU, las PINNs ocupan poca memoria (entre 0,021 y 0,123 MB), muy por debajo de FEM (16–32 MB). En GPU, los consumos se mueven en el orden de decenas de MB para ambos enfoques (11–19 MB), con poca variación entre configuraciones de PINN. Lo cual indica que, si se piensa trabajar sólo en CPU, las PINNs son especialmente atractivas cuando la memoria es limitada; en GPU, la distancia en memoria entre métodos se acorta.

Al observar el tiempo normalizado (columna cuatro Tabla 5.5), surgen tendencias interesantes. Mientras que en FEM el tiempo por nodo crece con la resolución, en PINN este valor disminuye al incrementar la complejidad de la red, sugiriendo una mejora en la eficiencia relativa por parámetro.

Los resultados muestran que FEM sobresale por su rapidez y predictibilidad, siendo particularmente útil en aplicaciones donde se requiere una única solución precisa y rápida. Además, su madurez como método lo hace ampliamente validado y documentado [72].

Sin embargo, PINN ofrece ventajas que no deben subestimarse. Su eficiencia en memoria, combinada con su capacidad de generalización una vez entrenada, permite utilizar la red como un evaluador instantáneo para condiciones nuevas. Esto es especialmente útil en contextos donde se desea obtener soluciones rápidas para múltiples escenarios sin necesidad de reentrenar desde cero.

A pesar de que su proceso de entrenamiento inicial es más costoso, la red entrenada permite evaluaciones posteriores con tiempos despreciables. En este sentido, PINNs actúan como una inversión inicial que rinde frutos en aplicaciones repetitivas o paramétricas.

La elección entre FEM y PINN dependerá, entonces, del contexto: si se dispone de recursos suficientes y se busca velocidad inmediata, FEM es la opción preferida. En contraste, si el sistema impone restricciones de memoria o demanda evaluaciones múltiples y rápidas, PINN puede resultar más conveniente [72].

5.6.1 Análisis de Precisión en Función de la Viscosidad para la Ecuación de Burgers

Siguiendo con el estudio de precisión de los dos enfoques numéricos (FEM y PINNs) en el problema de Burgers (5.23) donde el parámetro de viscosidad ν regula el balance entre efectos convectivos y difusivos. Buscamos evaluar cómo varía la precisión de ambos métodos al modificar ν, considerando distintas configuraciones representativas.

Los experimentos se realizaron, en condiciones iguales al experimento anterior, el dominio $x \in [-1, 1]$, $t \in [0, 1]$, con condiciones de frontera homogéneas y una condición inicial sinusoidal: $u(x, 0) = -\sin(\pi x)$. Se analizaron tres valores de viscosidad

$$\nu = \{0{,}0003, 0{,}0032, 0{,}0318\} \tag{5.28}$$

equivalentes a $10^{-3}/\pi$, $10^{-2}/\pi$ y $10^{-1}/\pi$, respectivamente. Para FEM se consideraron mallas con 401 y 801 nodos, mientras que para PINNs se utilizaron redes con 50 y 100 neuronas, y arquitecturas de 3 y 4 capas, utilizados en el análisis anterior. Se empleo la métrica del error L^2 para cuantificar la precisión, comparando la solución de ambos métodos. Se realizósobre `Intel(R) Xeon(R)` y GPU habilitada (`CUDA`) de Google Colab.

En cuanto a los resultados obtenidos, la Tabla 5.6 resume valores representativos (CPU) por viscosidad y configuración; al replicar el experimento en GPU se obtuvieron resultados del mismo orden y con pocas variaciones, lo que indica que el hardware no altera de forma apreciable la calidad de la aproximación. Se observa que los errores son levemente invariantes frente a cambios en ν, lo que sugiere que tanto FEM como PINNs manejan adecuadamente

las distintas proporciones de términos convectivos y difusivos. Para visualizar de manera compacta el comportamiento conjunto CPU/GPU, en la Tabla 5.7 se reporta el promedio del error L^2 en los tres valores de ν por configuración y tipo de dispositivo.

Tabla 5.6. Errores L^2 (CPU) para diferentes valores de viscosidad ν.

Método	**Configuración**	$\nu = 0{,}0003$	$\nu = 0{,}0032$	$\nu = 0{,}0318$
FEM	401 nodos	0.0407	0.0414	0.0408
FEM	801 nodos	0.0204	0.0205	0.0206
PINN	50N, 3 capas	0.0502	0.0503	0.0497
PINN	50N, 4 capas	0.0374	0.0374	0.0373
PINN	100N, 3 capas	0.0251	0.0250	0.0251
PINN	100N, 4 capas	0.0186	0.0189	0.0188

Tabla 5.7. Error medio $\overline{E}_{L^2}$ por configuración y *hardware* (promedio sobre ν).

Método	**Configuración**	$\overline{E}_{L^2}$ (CPU)	$\overline{E}_{L^2}$ (GPU)
FEM	401 nodos	0.040824	0.040671
FEM	801 nodos	0.020688	0.020387
PINN	50 neuronas, 3 capas	0.049928	0.050059
PINN	50 neuronas, 4 capas	0.037462	0.037460
PINN	100 neuronas, 3 capas	0.025031	0.024966
PINN	100 neuronas, 4 capas	0.018803	0.018814

A nivel comparativo, FEM con 801 nodos y PINN con 100 neuronas y 4 capas alcanzan errores similares cercanos a 2×10^{-2}, tanto en CPU como en GPU, lo que confirma que la precisión final depende esencialmente de la *resolución* (en FEM) o de la *capacidad de la red* (en PINN), y no del dispositivo de cómputo. En configuraciones más simples, FEM con 401 nodos supera a las PINNs pequeñas (50 neuronas y 3 capas), donde el error puede rondar 5×10^{-2} en ambos dispositivos.

Adicionalmente, La Figura 5.13 proporciona un resumen visual de los errores en forma de mapa de calor, facilitando la comparación entre configuraciones.

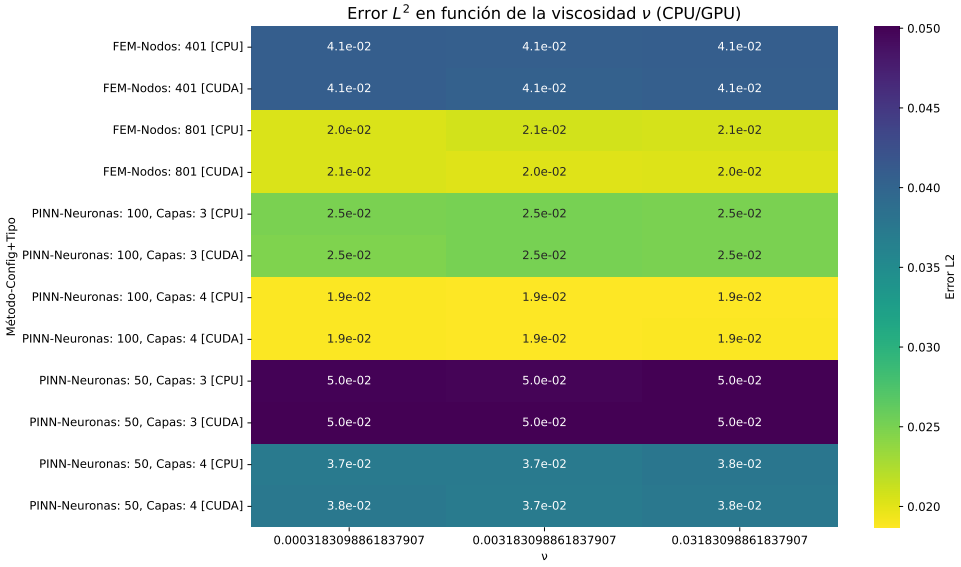


Figura 5.13. Mapa de calor del error L^2 según configuración y viscosidad

La tendencia es coherente con lo descrito anteriormente, al aumentar de 401 a 801 nodos en FEM el error disminuye aproximadamente a la mitad; en PINN, aumentar neuronas o capas también reduce el error de forma sistemática. Además, la variación del error frente a ν es leve, lo cual indica que ambos enfoques capturan adecuadamente el equilibrio entre difusión y convección en el rango analizado.

En conclusión, la precisión alcanzada por FEM y PINN es prácticamente equivalente. Sin embargo, las formas en que ambos métodos escalan son distintas, FEM mejora con refinamiento espacial, mientras que PINN lo hace ajustando su arquitectura. Esta diferencia puede influir en la selección del método dependiendo del contexto y los recursos disponibles. En entornos con restricciones de malla o geometrías complejas, PINN ofrece una alternativa flexible y prometedora [72].

5.6.2 Generalización Paramétrica para la Ecuación de Burgers

Finalmente estamos interesados en la capacidad de generalización de las Redes Neuronales informadas por la física (PINNs) frente al método de elementos finitos (FEM) en el problema directo asociado la ecuación de Burgers (5.23). El propósito principal radica en evaluar si una PINN entrenada en un rango de viscosidades es capaz de predecir con precisión soluciones para nuevos valores de ν. Además, en comparar el coste computacional frente a FEM para dichas evaluaciones.

En el experimento, la PINN (100 neuronas y 4 capas) se entrenó con valores de ν entre 0,0003 y 0,0318, es decir, $\nu \in (10^{-3}/\pi, 10^{-1}/\pi)$, evaluándose posteriormente en tres valores de prueba: uno fuera del rango (0,0002) y dos dentro (0,0016 y 0,0159). El análisis se realizó en $t = 1{,}0$, comparando las soluciones obtenidas por PINN con referencias generadas mediante FEM.

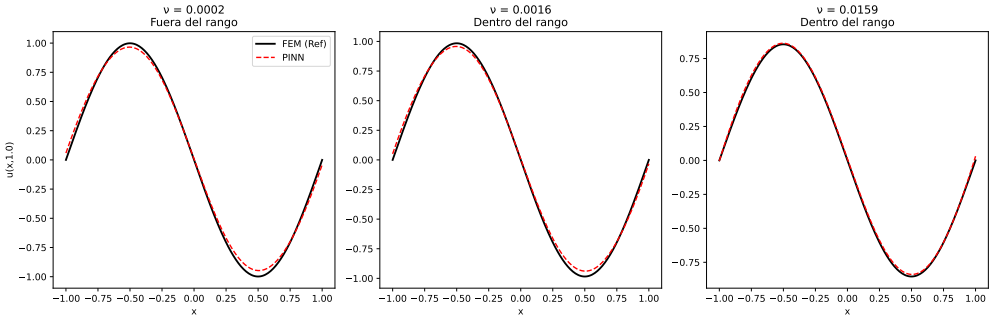


Figura 5.14. Soluciones FEM y PINN en $t = 1{,}0$ para distintos valores de ν

Las simulaciones muestran que, dentro del rango de entrenamiento, la PINN reproduce con alta fidelidad la solución de FEM, con errores medios cuadráticos entre 0,37 y 0,48. En el caso fuera de rango, la red mantiene la forma general de la solución aunque con desviaciones más notorias cerca de $x = 0$. Lo cual nos indica que la PINN aprendió la dependencia física esencial de ν, pero su extrapolación es más limitada.

En cuanto al coste computacional, la ejecución se realizó sobre GPU (CUDA) de google colab. El entrenamiento inicial de la PINN requirió 2.8249 s y un pico de memoria de 20.28 MB. Tras el entrenamiento, la evaluación para nuevos valores de viscosidad es prácticamente instantánea dentro del rango de entrenamiento, con tiempos de 0.55 ms para $\nu = 0{,}001592$ y 0.70 ms para $\nu = 0{,}015915$. Fuera del rango, como esperamos, el tiempo de inferencia aumenta, pero se mantiene muy bajo, 23.38 ms para $\nu = 0{,}000159$. En contraste, FEM presenta tiempos por simulación entre 2.90 s y 3.90 s (disminuyendo levemente al crecer ν).

La Figura 5.15 compara estos tiempos. Se observa una aceleración significativa de PINN frente a FEM.

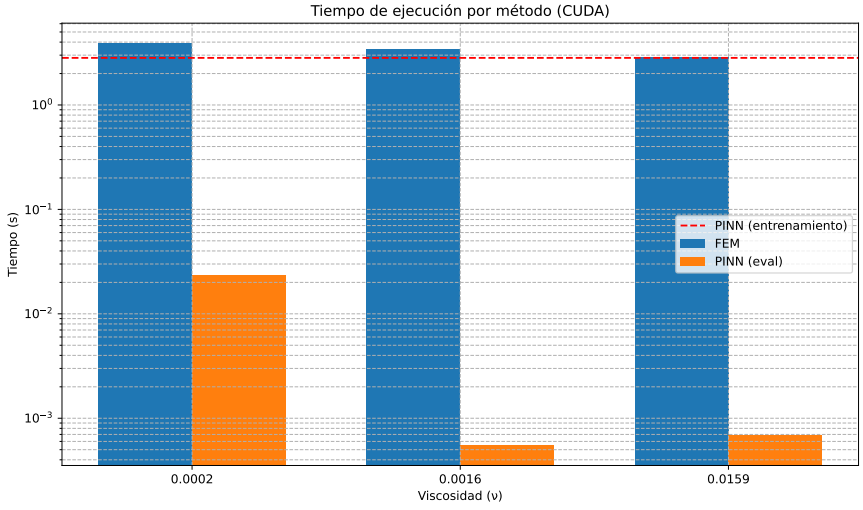


Figura 5.15. Comparación de tiempos de cómputo en FEM y PINN (GPU, CUDA). La línea discontinua roja indica el tiempo de entrenamiento de la PINN; las barras muestran el tiempo por evaluación.

Adicionalmente, se realizó un análisis del balance temporal, utilizando la regla de equilibrio $N^\star = \frac{t_{\text{train}}}{t_{\text{FEM}} - t_{\text{eval}}}$, en este caso el experimento indico que $N^\star$ se encuentra aproximadamente entre 0.7 y 1.0. Esto significa que con una sola evaluación la PINN ya recupera el costo de entrenamiento frente a FEM en GPU (CUDA). De este modo, FEM resulta adecuado para verificaciones precisas caso a caso, mientras que la PINN es claramente ventajosa para exploraciones paramétricas (p. ej., análisis de sensibilidad o barridos de viscosidad ν) con múltiples evaluaciones.

En resumen, de acuerdo con los resultados de la Tabla 5.15 las PINNs demuestran una notable eficiencia computacional en GPU. Una vez entrenadas, permiten evaluaciones de milisegundos con aceleraciones de dos a cuatro órdenes de magnitud frente a FEM, y con errores menores dentro del dominio de entrenamiento.

Finalmente, se pone a disposición el repositorio `EstudioCostoComputacionalE4`, donde se encuentran las simulaciones correspondientes a cada uno de los análisis realizados anteriormente.

5.7 Problema inverso asociado a la ecuación de Burgers 1D

En seguida estamos interesados en considerar un problema inverso asociado a la ecuación de Burgers y resolverlo mediante PINNs. Supongamos que tenemos información asociada al problema directo (5.23), es decir, disponemos de un conjunto de datos (posiblemente ruidosos),

$$\mathcal{D} \;=\; \big\{(x_i, t_i, u_i)\big\}_{i=1}^{N_d} \subset (-1,1)\times(0,1]\times\mathbb{R}, \qquad u_i \approx u(x_i,t_i),$$

y deseamos conocer la condición inicial subyacente, esto es; buscamos *reconstruir* la condición inicial $u_0 \in H_0^1(-1,1)$ tal que la solución de (5.23) reproduzca las mediciones dadas, este problema fue estudiado previamente desde un enfoque conocido como *asimilación de datos* [42]. En los experimentos con datos sintéticos, consideraremos como verdad la condición inicial

$$u_0^*(x) \;=\; -\sin(\pi x), \qquad x \in [-1,1],$$

la cual se utiliza únicamente para *generar* $\mathcal{D}$ y para evaluar el error de reconstrucción; no se emplea durante el entrenamiento inverso. La aproximación mediante PINNs se fundamenta en el aprendizaje simultáneo de la solución $u(x,t)$ y la condición inicial $u_0(x)$ mediante una red neuronal $\hat{u}_\theta(x,t)$, donde θ denota los parámetros de la red.

La estimación de los parámetros se plantea como un problema de optimización en el que buscamos minimizar una función de pérdida compuesta por varios términos: ajuste a los datos, cumplimiento de la física, imposición de condiciones de frontera y una regularización especial sobre la condición inicial. De forma general, esta función de pérdida se escribe como

$$\mathcal{L}(\theta) = \lambda_d\,\mathcal{L}_{\text{data}}(\theta) + \lambda_f\,\mathcal{L}_{\text{phys}}(\theta) + \lambda_b\,\mathcal{L}_{\text{bc}}(\theta) + \lambda_{\text{reg}}\,\mathcal{L}_{\text{reg}}(\theta), \tag{5.29}$$

donde cada coeficiente positivo $\lambda_\bullet$ controla la importancia relativa de cada término.

El primer término, $\mathcal{L}_{\text{data}}$, mide el error cuadrático medio entre la predicción de la red $\hat{u}_\theta$ y las mediciones disponibles:

$$\mathcal{L}_{\text{data}}(\theta) = \frac{1}{N_d}\sum_{i=1}^{N_d} \big|\,\hat{u}(x_i,t_i;\theta) - u_i\,\big|^2 .$$

El segundo, $\mathcal{L}_{\text{phys}}$, evalúa el residuo de la ecuación de Burgers en un conjunto de puntos de colación (x_j^f, t_j^f):

$$\mathcal{L}_{\text{phys}}(\theta) = \frac{1}{N_f}\sum_{j=1}^{N_f} \Big|\,\mathcal{R}_\theta(x_j^f,t_j^f)\,\Big|^2 .$$

El término de frontera, $\mathcal{L}_{\text{bc}}$, impone que la solución sea nula en los extremos $x=-1$ y $x=1$ para todo instante de tiempo:

$$\mathcal{L}_{\text{bc}}(\theta) = \frac{1}{N_b}\sum_{j=1}^{N_b} \big(\,|\hat{u}(-1,t_j^b;\theta)|^2 + |\hat{u}(1,t_j^b;\theta)|^2\,\big) .$$

Dado que el problema inverso es del tipo hacia atrás en el tiempo, es intrínsecamente inestable (la difusión atenúa la información de las condiciones iniciales), se añade un término de regularización sobre el perfil inicial en $t = 0$. Este término no penaliza directamente la amplitud, sino que controla la variación espacial mediante el seminorma de H^1:

$$\mathcal{L}_{\text{reg}}(\theta) = \frac{1}{N_0} \sum_{j=1}^{N_0} \left| \partial_x \hat{u}(x_j^0, 0; \theta) \right|^2 . \tag{5.30}$$

De esta forma se evitan oscilaciones espurias de alta frecuencia sin alterar de manera artificial el nivel de energía de la condición inicial.

La selección de los pesos $\lambda_\bullet$ puede realizarse manualmente, buscando que todos los términos tengan magnitudes comparables. Una vez entrenada la red, es decir, al obtener

$$\theta^\star \in \arg\min_\theta \mathcal{L}(\theta),$$

definimos la condición inicial reconstruida como

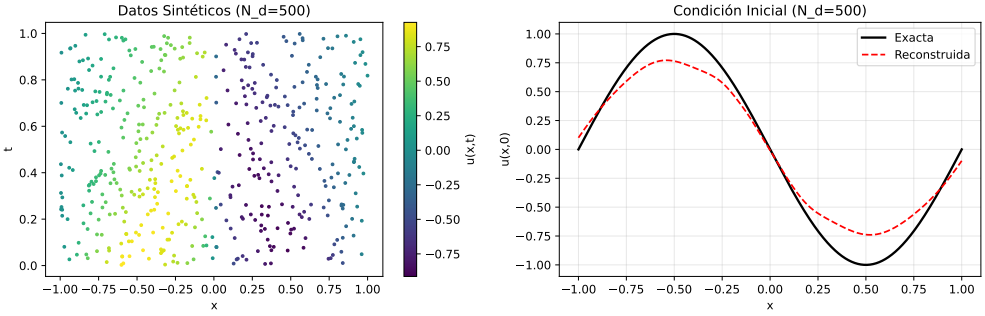
$$u_0^{\text{PINN}}(x) = \hat{u}(x, 0; \theta^\star), \qquad x \in [-1, 1].$$

Debido a que la condición de frontera se aplica para todo $t \in [0, 1]$, en particular se cumple que $\hat{u}(\pm 1, 0; \theta^\star) = 0$, por lo que u_0^{PINN} pertenece a $H_0^1(-1, 1)$ en un sentido aproximado.

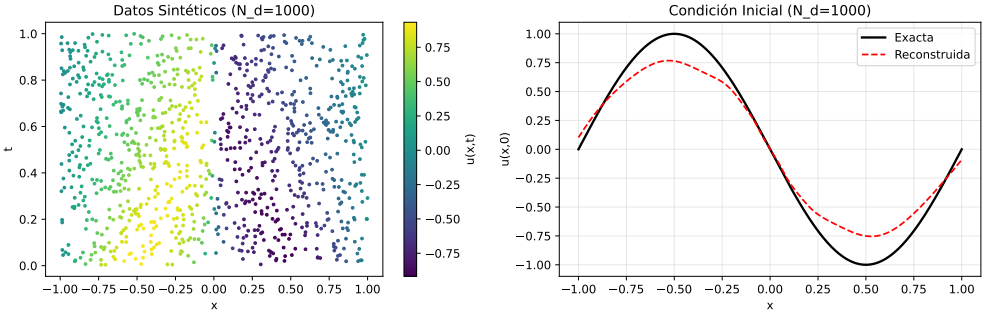
En los experimentos primero empezamos generando datos sintéticos, para diferentes tamaños de *dataset*, a partir de una PINN entrenada para el problema directo de la ecuación de Burgers unidimensional con $\nu = 0{,}01$, dominio $x \in [-1, 1]$, $t \in [0, 1]$ y condición inicial $u(x, 0) = -\sin(\pi x)$. El modelo directo tiene arquitectura totalmente conectada de tres capas con 50 neuronas y activación tanh en cada capa oculta, y se entrena mediante Adam con tasa 10^{-3} durante 5,000 épocas, hasta este punto se define de igual manera como se realizó en 5.5.1. La función de pérdida total (5.25), con pesos $(\lambda_{\text{r}}, \lambda_{\text{bc}}, \lambda_{\text{ic}}) = (0{,}4, 0{,}3, 0{,}3)$.

Con el modelo directo entrenado, se muestrean de forma uniforme pares (x_d, t_d) en el dominio y se evalúa $u_d = u_{\theta_{\text{fwd}}}(x_d, t_d)$ para construir tres conjuntos sintéticos $\mathcal{D}$ con $N_d \in \{500, 1000, 1500\}$. En las figuras 5.16a, 5.16b, 5.16c correspondientes, el panel izquierdo de cada fila muestra la nube de puntos coloreada por $u(x, t)$, lo que define una cobertura espacial y temporal razonablemente uniforme dispuesta para resolver el problema inverso.

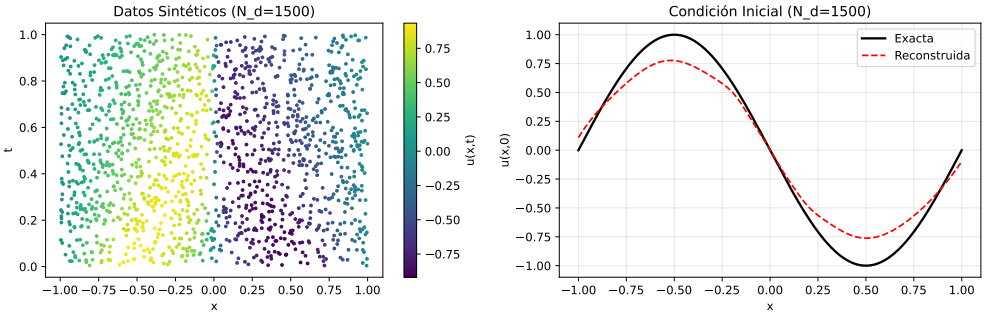
En cuanto al problema inverso, se entrena de nuevo una PINN $\hat{u}(\cdot, \cdot; \theta)$ con la misma arquitectura, pero ahora ajustando simultáneamente a los datos (x_d, t_d, u_d) y a términos físicos y de regularización (5.29). Se utilizan $N_f = 15{,}000$ puntos de colocación interiores para el residual de la EDP, $N_b = 400$ puntos de contorno para imponer $u(\pm 1, t) = 0$, y un término sobre el perfil inicial en $t = 0$ que penaliza el gradiente espacial de la condición inicial (5.30). La pérdida total para el problema inverso queda en la forma (5.29) con coeficiente o pesos $(\lambda_d, \lambda_f, \lambda_b, \lambda_{\text{reg}}) = (1, 1, 1, 10^{-2})$. El optimizador utilizado es Adam con tasa 10^{-3} y 6,000 épocas por cada tamaño N_d. La reconstrucción de la condición inicial se evalúa y plotea en una malla de 1,000 puntos sobre $[-1, 1]$, los resultados para cada lote de puntos se ilustran en la parte derecha de las figuras 5.16a, 5.16b, 5.16c, justo con el resultado deseado. Así mismo, el error relativo para cada tamaño se reporta en la tabla de errores correspondiente.



(a) Datos sintéticos $N_d = 500$ y resultado problema inverso



(b) Datos sintéticos $N_d = 1000$ y resultado problema inverso



(c) Datos sintéticos $N_d = 1500$ y resultado problema inverso

Figura 5.16. Resultados del problema inverso con diferentes cantidades de datos sintéticos.

Tamaño data set N_d	Norma L^2
500	$2{,}2790 \times 10^{-1}$
1000	$2{,}2399 \times 10^{-1}$
1500	$2{,}1646 \times 10^{-1}$

Tabla 5.8. Error relativo para diferentes tamaños de datos

En todos los casos, el valor final es muy similar, con una leve mejora al aumentar N_d. Esta tendencia también se refleja en el error relativo de la condición inicial, que decrece de $2{,}2790 \times 10^{-1}$ para $N_d = 500$ a $2{,}2399 \times 10^{-1}$ para $N_d = 1000$ y a $2{,}1646 \times 10^{-1}$ para

$N_d = 1500$. Lo anterior indica que, aunque las mejoras son reales, cada vez rinden menos al sumar más datos. En consecuencia, lo que incide en la aproximación es la propia forma del modelo y las reglas de suavizado que le pusimos, más no la falta de datos.

Las figuras de comparación de la condición inicial $u(x, 0)$ muestran de forma cualitativa la misma pauta: la curva reconstruida (línea roja discontinua) aproxima la forma senoidal exacta (línea negra continua), pero tiende a subestimar la amplitud en las cercanías de los máximos y mínimos y a suavizar los cambios de pendiente. Esta "*suavización*" es coherente con el término de regularización $\lambda_{\text{reg}}\mathcal{L}_{\text{reg}}(\theta)$ que penaliza gradientes grandes en $t = 0$ y, por construcción, favorece perfiles menos pronunciados. Al incrementar N_d la curva roja se acerca a la negra y reduce el error L^2, aunque persiste una ligera pérdida de amplitud que sugiere explorar un reajuste de pesos en (5.29) o una arquitectura con mayor capacidad de alta frecuencia, con el fin de lograr una coincidencia más fina en las crestas.

En síntesis, entrenar un modelo directo para crear datos sintéticos y luego resolver el inverso combinando datos, física y un leve suavizado, entrega reconstrucciones estables y razonables de la condición inicial. Al aumentar N_d la métrica L^2 mejora de forma sostenida, aunque el avance es moderado porque hay que equilibrar cuánto seguimos a los datos y cuánta suavidad imponemos al modelo. Lo anterior nos indica que en presencia de datos reales utilizar las PINNs para mejorar la aproximación en problemas inversos de este tipo, si bien depende de una cantidad considerable de datos, las PINNs junto con una configuración adecuada logran reconstruir perfiles iniciales aún sin la presencia de grandes cantidades de datos. La implementación completa de este problema tipo inverso puede consultarse en el siguiente repositorio `ProblemaInversoBurger1-1D_E4`.

5.8 Problema de identificación de parámetros asociado a la ecuación KdV

En el marco de este trabajo, ampliamos el análisis hacia la ecuación de Korteweg–de Vries (KdV), un modelo matemático que describe la propagación de ondas solitarias en medios dispersivos y que ha jugado un papel fundamental en el desarrollo de la teoría de ondas no lineales. Esta ecuación surge en diversos contextos físicos, como en la dinámica de ondas superficiales poco profundas, en la física de plasmas y en la transmisión de pulsos en fibras ópticas, donde el balance entre no linealidad y dispersión permite la aparición de soluciones de tipo solitón [74].

Consideramos un problema unidimensional clásico asociado a la ecuación de KdV, el cual se expresa como

$$\begin{cases} \partial_t u + \lambda_1\, u\, \partial_x u + \lambda_2\, \partial_x^3 u = 0, & (x,t) \in [-10,10] \times (0,1], \\ u(x,0) = u_0(x), & x \in [-10,10], \\ u(\pm 10, t) = 0, & t \in [0,1]. \end{cases} \tag{5.31}$$

donde λ_1 se denomina coeficiente de no linealidad, λ_2 es el coeficiente de dispersión, $u_0(x)$ es la condición inicial y $u(t,x)$ representa el campo escalar de interés. Se asume regularidad suficiente para garantizar la existencia y unicidad de la solución, tal como se establece en [74]. Bajo condiciones apropiadas, la KdV admite soluciones especiales denominadas *solitones*[8], las cuales han sido objeto de estudio intensivo desde su descubrimiento.

En este caso, la solución tipo solitón al problema directo (5.31) viene dada por la forma equivalente

$$u(x,t) \;=\; \frac{\frac{c}{2}}{\cosh^2\!\left(\frac{\sqrt{c}}{2}\,(x-ct)\right)} = \frac{c}{2}\,\operatorname{sech}^2\!\left(\frac{\sqrt{c}}{2}\,(x-ct)\right), \qquad c>0, \tag{5.32}$$

donde c representa la velocidad constante del solitón, que se desplaza sin deformarse. Al imponer condiciones de contorno homogéneas $u(\pm 10, t) = 0$, el problema está bien planteado en el sentido de Hadamard, dado que la ecuación es evolutiva y dispersiva, y su solución es suave y decae rápidamente hacia los extremos del dominio espacial [73, 74].

5.8.1 Aproximación mediante PINNs de la solución a la KdV

Para asegurarnos de la fiabilidad de las PINNs en la reproducción del comportamiento evolutivo de la ecuación KdV, y antes de resolver el problema inverso ilustramos las simulaciones de la solución del problema directo. Destacaremos principalmente un estudio en la arquitectura de la red neuronal y como esta incide en la calidad de aproximación de la

[8] Un solitón es una onda solitaria que mantiene su forma y velocidad de propagación debido a un equilibrio exacto entre los efectos no lineales y la dispersión del medio.

solución, usaremos un enfoque no supervisado. Es decir, no usaremos datos experimentales para entrenar la PINN, sino únicamente para comparar los resultados de aproximación en la solución. Los experimentos se realizaron sobre GPU (CUDA) de google colab. Como ya es habitual en este trabajo, para el enfoque PINNs aproximamos la solución al problema directo (5.31) mediante una red neuronal $\hat{u}_\theta(x,t;\theta)$ con parámetros θ, impuesta sobre la condición $u_0(x)$, la cual se obtiene evaluando solución exacta (5.32). Es decir,

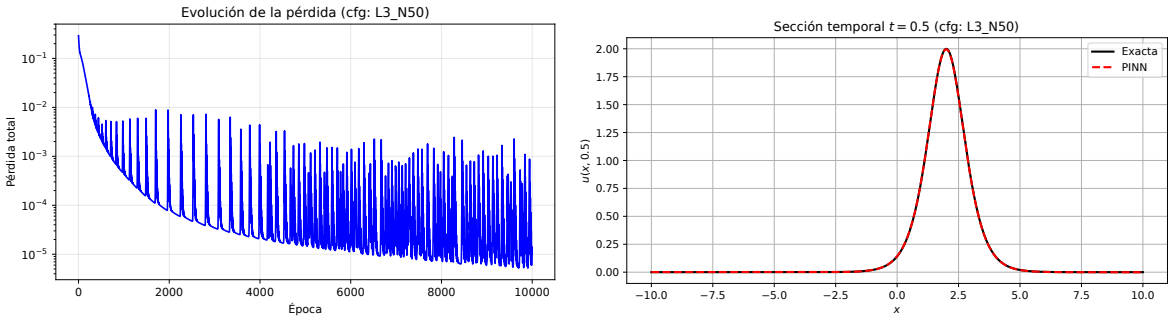
$$u(x,0) = u_0(x) = \frac{c}{2}\text{sech}^2\left(\frac{\sqrt{c}}{2}x\right), \tag{5.33}$$

para este experimento consideramos $c = 4{,}0$ como velocidad del solitón. Adicionalmente, el dominio computacional empleado es espacialmente acotado $x \in [-10, 10]$ y el horizonte temporal $t \in [0, 1]$, con parámetros físicos $\lambda_1 = 6{,}0$ y $\lambda_2 = 1{,}0$ para la no linealidad y la dispersión, respectivamente.

El entrenamiento de la PINN se planteó con una configuración de tres capas ocultas de 50 neuronas cada una y función de activación tanh. El optimizador Adam con tasa de aprendizaje 10^{-3} se utilizó durante 10,000 épocas. Análogamente a ejemplos anteriores, la función de pérdida total dada por

$$\mathcal{L}(\theta) = \lambda_f\,\mathcal{L}_{\text{phys}}(\theta) + \lambda_{\text{ic}}\,\mathcal{L}_{\text{ic}}(\theta) + \lambda_b\,\mathcal{L}_{\text{bc}}(\theta) \tag{5.34}$$

el cual combina términos residuales de la EDP ($\mathcal{L}_f$), condición inicial ($\mathcal{L}_i$) y condiciones de frontera Dirichlet homogéneas ($\mathcal{L}_{bc}$), con pesos unitarios para todos los términos. El muestreo incluyó 20,000 puntos interiores, 2,000 puntos iniciales y 1,000 puntos frontera, todos distribuidos uniformemente.



(a) Evolución de la perdida **(b)** Aproximación PINN para KdV en $t = 0{,}5$

Figura 5.17. Perdida total y aproximación PINN vs Solución exacta $t = 0{,}5$

Tras el entrenamiento, los resultados obtenidos muestran que la red neuronal propuesta logra una excelente aproximación de la solución exacta del problema directo de KdV. En la Figura 5.17(a) se observa la evolución del funcional de pérdida total durante las 10,000 épocas. Notamos una rápida disminución de la pérdida en las primeras iteraciones, seguida de una estabilización oscilatoria con valores mínimos cercanos a 10^{-5}. Este comportamiento es característico cuando se utilizan redes neuronales profundas con activaciones suaves como

tanh y optimizadores como Adam. Las oscilaciones observadas reflejan los desafíos de optimizar un problema con múltiples términos en la función de pérdida. No obstante, el descenso global sostenido indica una buena convergencia.

Por otro lado, en la Figura 5.17(b), se compara la solución exacta con la predicción de la PINN para un tiempo fijo $t = 0{,}5$. Se puede observar que ambas curvas coinciden casi perfectamente, lo que indica que la red neuronal ha aprendido no solo la forma del solitón, sino también su desplazamiento en el tiempo. Esto valida la capacidad del modelo PINN para capturar tanto la no linealidad como los efectos dispersivos que caracterizan a la ecuación de KdV.

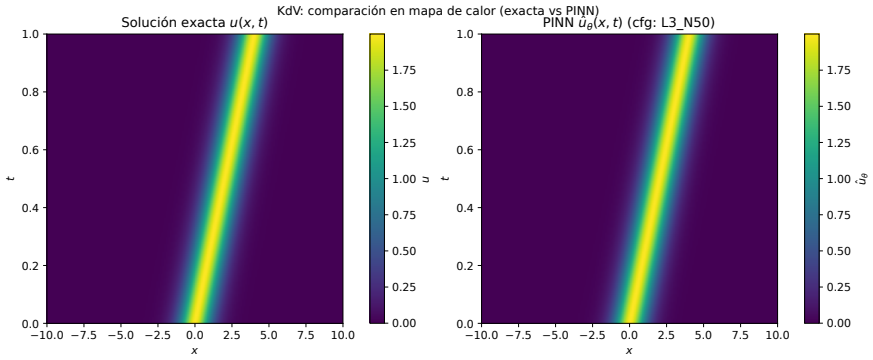


Figura 5.18. Comparación de las solución exacta vs la aproximación PINN para todo el dominio

Para una visualización más global, la Figura 5.18 muestra mapas de calor para la solución exacta y la solución aproximada por la PINN en todo el dominio espaciotemporal $[-10, 10] \times [0, 1]$. La semejanza visual entre ambos mapas confirma nuevamente la fidelidad de la aproximación.

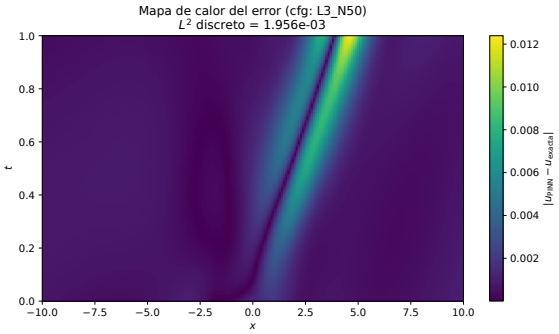


Figura 5.19. Mapa de calor del error absoluto

Adicionalmente, presentamos el error absoluto entre ambas soluciones en la Figura 5.19 mediante un mapa de calor. Este muestra que los errores se concentran en una franja estrecha a lo largo de la trayectoria del solitón, especialmente en lo que se conoce como frente de propagación. Esta acumulación de error puede deberse a la naturaleza altamente localizada del solitón, donde ligeras discrepancias en la forma o velocidad de desplazamiento generan

errores notables. Sin embargo, los valores máximos del error son del orden de 10^{-2}, mientras que el error medio cuadrático discreto calculado en norma L^2 es $\|u_{\text{PINN}} - u_{\text{exacta}}\|_{L^2} = 1{,}956 \times 10^{-3}$, lo cual es un excelente resultado para una aproximación no supervisada. Para una visualización del código de implementación y los resultados de este experimento, consulte `ProblemadirectpKdV1D` .

Un análisis rápido de la buena capacidad de aproximación de las PINNs indica que la configuración (tres capas ocultas con 50 neuronas y activación tanh) resulta adecuada para capturar la dinámica del solitón sin sobreajuste, dado que el modelo tiene suficiente capacidad expresiva para aproximar funciones suaves con transiciones rápidas. No obstante, el uso de un número considerable de puntos de entrenamiento en el interior del dominio (20,000) y en las condiciones iniciales y de frontera (3,000 en total) favoreció una buena cobertura del dominio y una adecuada penalización del residuo de la EDP, lo cual es clave para el éxito de las PINNs. Además, se emplearon pesos unitarios para cada término de la pérdida (5.34), lo que balancea la influencia de los datos iniciales, condiciones de frontera y cumplimiento de la física, evitando dominancia de algún término.

En resumen, el enfoque PINN aplicado al problema directo de KdV con esta arquitectura logra una excelente aproximación, demostrando la capacidad del método para resolver ecuaciones no lineales y dispersivas sin requerir datos experimentales. Estos resultados no solo validan el modelo propuesto, sino que también sientan las bases para abordar el problema inverso en secciones posteriores, bajo el supuesto de que la red es capaz de aprender la dinámica del sistema con una alta precisión.

Tratamiento de un problema inverso asociado a la ecuación KdV

Volviendo al problema de interés de esta sección, nuestro objetivo se centra en un problema inverso asociado a la ecuación de KdV, estudiado en [58, 59]. Este consiste en estimar los coeficientes de no linealidad λ_1 y dispersión λ_2 a partir de datos observados de la solución $u(x,t)$ (posiblemente ruidosos). Este problema es de gran relevancia práctica, ya que en muchos contextos experimentales los parámetros físicos subyacentes son desconocidos y deben inferirse a partir de mediciones.

Formalmente, supongamos que contamos con un conjunto de datos $\{(x_i, t_i, u_i)\}_{i=1}^{N_d}$, donde u_i representa una medición de la solución en el punto (x_i, t_i). El problema inverso se plantea como

$$\text{Dados } \{(x_i, t_i, u_i)\}_{i=1}^{N_d}, \text{ determinar } \lambda_1 \text{ y } \lambda_2 \text{ tales que se satisfaga (5.31).} \tag{5.35}$$

Para resolver este problema mediante el enfoque PINNs, representamos la solución mediante una red neuronal $\hat{u}_\theta$, parametrizada por θ, y optimizamos simultáneamente tanto los parámetros de la red como los coeficientes físicos λ_1 y λ_2. De esta manera, el enfoque se implementa incorporando directamente la ecuación de KdV (5.31) en la función de pérdida total. Para ello, se definen los siguientes términos: en primer lugar, la pérdida asociada a los datos conocidos; en segundo lugar, la pérdida física que penaliza el residuo de la EDP; además, las pérdidas que imponen condiciones iniciales y de frontera; y, opcionalmente, un

término de regularización sobre los parámetros del modelo. Matemáticamente, estos términos se expresan como

$$\mathcal{L}_{\text{datos}}(\theta) = \frac{1}{N_d}\sum_{i=1}^{N_d} |\,\hat{u}_\theta(x_i,t_i) - u_i\,|^2\,, \tag{5.36}$$

$$\mathcal{L}_{\text{física}}(\theta,\lambda_1,\lambda_2) = \frac{1}{N_f}\sum_{j=1}^{N_f} \left|\partial_t\hat{u}_\theta + \lambda_1\hat{u}_\theta\,\partial_x\hat{u}_\theta + \lambda_2\,\partial_x^3\hat{u}_\theta\right|^2\Big|_{(x_j,t_j)}\,, \tag{5.37}$$

$$\mathcal{L}_{\text{CI}}(\theta) = \frac{1}{N_0}\sum_{k=1}^{N_0} |\hat{u}_\theta(x_k,0) - u_0(x_k)|^2\,, \tag{5.38}$$

$$\mathcal{L}_{\text{CF}}(\theta) = \frac{1}{N_b}\sum_{m=1}^{N_b} \left(|\hat{u}_\theta(-10,t_m)|^2 + |\hat{u}_\theta(10,t_m)|^2\right). \tag{5.39}$$

Finalmente, un término opcional de regularización puede escribirse como

$$\mathcal{L}_{\text{reg}} = \|\lambda\|_2^2, \qquad \lambda = (\lambda_1,\lambda_2), \tag{5.40}$$

De esta forma, el funcional total queda definido por

$$\mathcal{L}(\theta,\lambda_1,\lambda_2) = \omega_d\,\mathcal{L}_{\text{datos}} + \omega_f\,\mathcal{L}_{\text{física}} + \omega_{ci}\,\mathcal{L}_{\text{CI}} + \omega_{cf}\,\mathcal{L}_{\text{CF}} + \omega_r\,\mathcal{L}_{\text{reg}}, \tag{5.41}$$

donde cada $\omega_\bullet \geq 0$ es un coeficiente que puede ajustarse de manera estática o adaptativa para mejorar la aproximación al problema.

Pasando a la implementación numérica, primero suponemos que la condición inicial u_0 es conocida y, en primera instancia, no empleamos regularización ($\omega_r = 0$). Generamos dos conjuntos de datos de entrenamiento $\mathcal{D}$ sobre el dominio $x \in [-10,10]$, $t \in [0,1]$, a partir de la solución exacta (5.32). Un primer conjunto, *datos limpios (0 % ruido)*, con $N_d = 1500$ pares (x_i,t_i) muestreados uniformemente con etiquetas $u(x_i,t_i)$. Y un segundo conjunto, *datos ruidosos (5 %)*, que consiste de los mismos (x_i,t_i) pero con $u_i = u(x_i,t_i) + \varepsilon_i$, donde $\varepsilon_i \sim \mathcal{N}(0,\sigma_\varepsilon^2)$, con varianza fijada como $\sigma_\varepsilon = 0{,}05\,\sigma_u$. Aquí, σ_u es la desviación estándar de los valores limpios $u(x_i,t_i)$ en el conjunto de entrenamiento, esto es,

$$\sigma_u \;=\; \left(\frac{1}{N_d-1}\sum_{i=1}^{N_d}\big(u(x_i,t_i)-\bar{u}\big)^2\right)^{1/2}, \qquad \bar{u} \;=\; \frac{1}{N_d}\sum_{i=1}^{N_d} u(x_i,t_i).$$

Adicionalmente, muestreamos $N_0 = 2000$ puntos para la CI en $t = 0$, $N_b = 500$ instantes para las CF en $x = \pm 10$, y $N_f = 10^4$ puntos de colocación para evaluar el término físico. Los parámetros verdaderos del modelo son $\lambda_1^\star = 6{,}0$ y $\lambda_2^\star = 1{,}0$; la velocidad del solitón elegida fue $c = 4{,}0$.

La Figura 5.20 resume la distribución de puntos y perfiles de referencia (exactos) empleados para el entrenamiento (paneles: datos limpios, datos con ruido, condición inicial y perfil en $t = 1$).

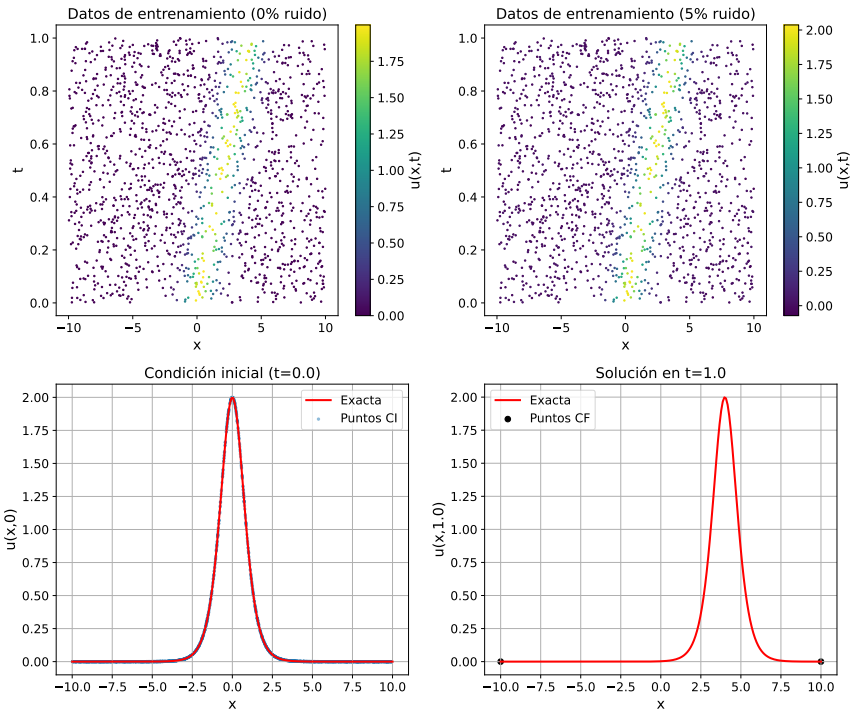


Figura 5.20. Datos de entrenamiento y perfiles de referencia para KdV: (a) 0 % ruido, (b) 5 % ruido, (c) condición inicial $t = 0$, (d) solución exacta en $t = 1$.

La red utilizada para tratar el problema inverso es un perceptrón multicapa (MLP) totalmente conectado, con capa de entrada de dimensión 2 (las variables (x, t)), tres capas ocultas con 50 neuronas cada una y una capa de salida escalar que aproxima $u(x, t)$. Las capas ocultas emplean la función de activación $\tanh(\cdot)$ y la salida es lineal. Los parámetros físicos se tratan como parámetros libres del modelo: $\lambda_1, \lambda_2 \in \mathbb{R}$. El entrenamiento utiliza `Adam` con tasa de aprendizaje 10^{-3} durante 20000 épocas y pesos unitarios en (5.41). Las derivadas espaciales y temporales requeridas por KdV se calculan mediante la rutina de derivación automática `autograd` de PyTorch.

El experimento se ejecutó en Google Colab con GPU habilitada (`Tesla T4`) y PyTorch `2.6.0+cu124`. La Figura 5.21 muestra (i) la evolución de la pérdida total, (ii)–(iii) la identificación de los parámetros λ_1 y λ_2 para datos limpios y ruidosos, y (iv) la comparación entre la solución exacta y las aproximaciones PINN en $t = 0{,}5$.

Al concluir el entrenamiento, los parámetros estimados se mantuvieron muy cercanos a los valores reales: para *datos limpios* se obtuvo $\hat{\lambda}_1 = 5{,}9699$ y $\hat{\lambda}_2 = 0{,}9883$; mientras que con *datos ruidosos* (5 % de ruido) alcanzamos $\hat{\lambda}_1 = 5{,}9503$ y $\hat{\lambda}_2 = 0{,}9826$. En ambos escenarios, el error relativo se mantiene por debajo de 1 %, lo que evidencia una identificación precisa incluso en presencia de mediciones ruidosas. Para visualizar la cercanía entre el modelo verdadero y las ecuaciones aprendidas, la Tabla 5.9 resume las expresiones resultantes (redondeadas a tres decimales).

En resumen, a lo largo de la optimización observamos una convergencia estable y prácti-

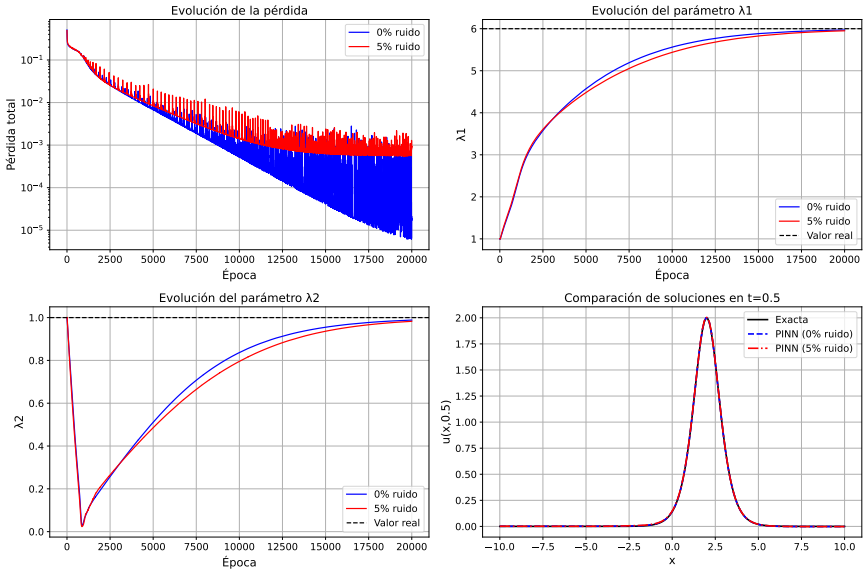


Figura 5.21. Evolución de la pérdida y de los parámetros identificados, y comparación de soluciones en $t = 0{,}5$.

Tabla 5.9. Resultados de identificación de parámetros.

Modelo	**Ecuación identificada**
PDE correcta	$u_t + 6{,}000\, u\, u_x + 1{,}000\, u_{xxx} = 0$
Identificado (0 % ruido)	$u_t + 5{,}970\, u\, u_x + 0{,}988\, u_{xxx} = 0$
Identificado (5 % ruido)	$u_t + 5{,}950\, u\, u_x + 0{,}983\, u_{xxx} = 0$

camente monótona de λ_1 y λ_2; la contribución física tiende a dominar el costo computacional por las derivadas espaciales de tercer orden que exige KdV, pero aun así la PINN preserva la estructura del solitón en tiempos intermedios (por ejemplo, $t = 0{,}5$), con buena aproximación tanto en amplitud como en soporte espacial. En conjunto, estos resultados refuerzan la idea de que, con una arquitectura densa basada en activaciones tanh, el optimizador Adam y un balance simple de pérdidas, es posible alcanzar estimaciones subporcentuales de los parámetros de no linealidad y dispersión a partir de datos escasos y ruidosos. Finalmente, es importante destacar que la implementación completa de este problema se puede consultar en el siguiente repositorio `ProblemaInversoKdV1D`.

Capítulo 6
Conclusiones

Este trabajo tuvo como objetivo principal el estudio, la implementación y evaluación de las llamadas redes neuronales informadas por la física (PINNs) en la aproximación de soluciones de ecuaciones diferenciales parciales (EDPs) evolutivas, particularmente en la ecuación de Burgers viscosa (BV) y ecuación de Korteweg-de Vries (KdV), tanto en su versión directa como en dos problemas inversos representativos: asimilación de datos (reconstrucción de la condición inicial) e identificación de parámetros físicos. A lo largo del desarrollo se construyó un marco teórico que abarca los fundamentos matemáticos del aprendizaje profundo y los teoremas de aproximación universal para redes neuronales. También, se contrastó el desempeño de las PINNs frente al método de elementos finitos (FEM), se analizó el costo computacional utilizando CPU y GPU, y se exploró la capacidad de generalización paramétrica en el caso de BV. Además, se desarrollaron implementaciones reproducibles en PyTorch, disponibles en un repositorio abierto de GitHub creado por el autor.

En cuanto a los problemas directos, los resultados confirman la validez y el potencial del enfoque dado por las PINNs, al incorporar el conocimiento físico en el entrenamiento mediante la minimización del residual de la EDP. Se encuentra que éstas superan a las redes supervisadas convencionales, como se apreció en la comparación mostrada en la sección 5.4.1, la cual evidencia cómo una red tradicional falla en la extrapolación para tiempos futuros, mientras que la PINN preserva la estructura física de la solución del oscilador armónico. La precisión, medida con la norma L^2 (Tabla 5.2), demuestra que la física actúa como un regularizador natural que guía la optimización hacia soluciones generalizables y consistentes.

La implementación de las PINNs en PyTorch permitió la posibilidad de calcular derivadas de alto orden de manera adecuada mediante diferenciación automática, evitando errores de discretización de los métodos clásicos y conduciendo a soluciones continuas y diferenciables en todo el dominio, disponibles en cualquier punto sin necesidad de interpolación. Para la ecuación BV en 1D se obtuvieron errores en norma L^2 del orden de 10^{-3} a lo largo del intervalo temporal, capturando adecuadamente el comportamiento difusivo–convectivo. En el caso de la KdV, la red reprodujo con precisión la dinámica de una solución de tipo solitón;

la discrepancia se concentró cerca del frente de propagación, con un error medio L^2 igualmente del orden de 10^{-3}. Esta evidencia respalda la capacidad de las PINNs para resolver con alta precisión EDP's evolutivas, con una arquitectura moderada y un muestreo considerable de puntos de colocación.

La comparación con FEM en el problema (BV) mostró que, si se evalúa una única ejecución en CPU, FEM resulta más veloz, mientras que la PINN, tras su entrenamiento, permite consultas prácticamente instantáneas en cualquier punto del dominio. En términos cuantitativos, la diferencia FEM–PINN se mantuvo un error de órdenes 10^{-2} y 10^{-1}, respectivamente según el tiempo, lo cual indica que la PINN alcanza una precisión aceptable con la ventaja adicional de ofrecer soluciones continuas y evaluables en tiempo real. En cuanto al coste computacional, FEM mostró una notable rapidez en la resolución, alcanzando la aproximación en tiempos muy pequeños en configuraciones de malla medias y finas cuando se ejecutó en CPU. Por su parte, las PINNS requirieron un esfuerzo levemente mayor en este mismo entorno. Sin embargo, al cambiar el cálculo a GPU, se observó una aceleración significativa que las situó en un rango de competitividad frente al enfoque tradicional, especialmente cuando se consideró el escenario de múltiples evaluaciones.

En cuanto al uso de memoria, la aproximación mediante PINNs se destacó por su velocidad en CPU, mientras que en GPU ambos métodos se mantuvieron dentro de márgenes razonables, sin que esto representara una limitación práctica en los experimentos realizados.

Respecto a la generalización paramétrica, una PINN entrenada en un rango de viscosidades reprodujo con precisión la solución para nuevos valores de ν dentro del intervalo, con tiempos de inferencia de milisegundos; fuera del rango, conservó la forma cualitativa con desviaciones esperables. Este resultado muestra que, una vez finalizado el entrenamiento, las PINNs pueden utilizarse como modelos aproximados para explorar diferentes configuraciones de parámetros de manera eficiente.

En los problemas inversos considerados, la aproximación mediante PINNs de la asimilación de datos en el problema (BV) (reconstrucción de la condición inicial) alcanzó errores relativos L^2 cercanos a 0,22, con una leve mejora al aumentar el tamaño del conjunto de datos. La regularización tipo seminorma H^1 estabilizó la inversión hacia atrás en el tiempo y evitó perturbaciones no deseadas, a costa de un ligero suavizado de las crestas; este comportamiento es coherente con la naturaleza mal planteada del problema. Por otro lado, en el problema de identificación de parámetros para KdV, la PINN estimó las constantes λ_1 y λ_2 con error relativo porcentualmente pequeño, tanto con datos exactos como con 5 % de ruido (por ejemplo, $\hat{\lambda}_1 \approx 5{,}97$ frente a 6,00 y $\hat{\lambda}_2 \approx 0{,}99$ frente a 1,00), evidenciando robustez frente a mediciones ruidosas.

Sin embargo, el estudio presenta algunas limitaciones que orientan un trabajo futuro. Por ejemplo, la extensión a 2D/3D y a dominios más complejos, el manejo de condiciones de frontera no homogéneas, la incorporación de muestreos más estructurados o la validación con problemas que cuenten con datos reales. Por otro lado, el balance de pérdidas se trabajó con ponderaciones fijas y activación tanh; una elección poco adecuada puede afectar la convergencia o sesgar la solución, en especial en problemas inversos.

A partir de lo anterior, se abren varias líneas prometedoras: explorar balances de pérdidas más flexibles para estabilizar el entrenamiento, usar estrategias de optimización progresiva

y remuestreo adaptativo, considerar formulaciones que reduzcan la sensibilidad a derivadas de orden alto, incorporar métodos que manejen la incertidumbre y, finalmente, apoyarse en arquitecturas más avanzadas como redes de operadores para mejorar la generalización y acelerar simulaciones con múltiples consultas. En el ámbito de los problemas inversos, resulta especialmente atractivo el enfoque híbrido que combina FEM con PINNs, el cual ha sido señalado por diversos autores como una estrategia novedosa para aprovechar la robustez numérica de los métodos clásicos y la capacidad de aprendizaje y extrapolación de las redes neuronales.

Capítulo A
Apéndice

Todas las implementaciones computacionales de este trabajo pueden encontrarse en el siguiente perfil de GitHub Darwin.Tapie.

Bibliografía

[1] Kreyszig, E. Introductory Functional Analysis with Applications isbn: 9780471504597. https://books.google.se/books?id=AQtMEAAAQBAJ (Wiley, 1991).

[2] Y. LeCun, Y. Bengio, G. Hinton, "Deep learning,"Nature, vol. 521, no. 7553, 2015, pp. 436-444.

[3] A. Krizhevsky, I. Sutskever, G. E. Hinton, Ïmagenet classification with deep convolutional neural networks,.ªdvances in Neural Information Processing Systems, vol. 25, 2012, pp. 1097-1105.

[4] Russell, S. J., Norvig, P. (2016). *Artificial Intelligence: A Modern Approach*. Pearson Education Limited.

[5] Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning*. MIT Press.

[6] Aggarwal, C. C. Neural Networks and Deep Learning. A Textbook 497 isbn: 978-3-319-94462-3 (Springer, Cham, 2018).

[7] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Journal of Machine Learning Research, vol.15, pp.1929–1958, 2014.

[8] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, Çurriculum learning", in Proceedings of the 26th Annual International Conference on Machine Learning (ICML), 2009, pp. 41–48.

[9] G. Alain, A. Lamb, C. Sankar, A. Courville, and Y. Bengio, "Variance reduction in SGD by distributed importance sampling", arXiv preprint arXiv:1511.06481, 2015.

[10] I. Goodfellow et al., Deep Learning, MIT Press, Cambridge, 2016

[11] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, L. Zdeborová, "Machine learning and the physical sciences,Reviews of Modern Physics, vol. 91, no. 4, 2019, Art. no. 045002.

[12] , H. N. Approximation properties of a multilayered feedforward artificial neural network. Advances in Computational Mathematics 1, 61-80 (1993).

[13] , H. N. y Poggio, T. A. Deep vs. shallow networks : An ap- proximation theory perspective. (2016).

[14] Vapnik, V. An overview of statistical learning theory. IEEE Transactions on Neural Networks 10, 988–999 (1999).

[15] Akomolafe, D. Scholars Research Library Comparative study of biological and artificial neural networks. European Journal of Applied Engineering and Scientific Research 2, 36–46 (Jan. 2013)

[16] Kreines, B. A. V. M. A. Approximation of continuous functions by superpositions of plane waves. *Dokl. Akad. Nauk SSSR*, **140**, 1326–1329 (1961).

[17] A. R. Barron, Universal approximation bounds for superpositions of a sigmoidal function, *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, 1993.

[18] A. Pinkus, Approximation theory of the MLP model in neural networks," *Acta Numerica*, vol. 8, pp. 143–195, 1999.

[19] G. Cybenko, Approximation by superpositions of a sigmoidal function, *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[20] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (1989) 359-366.

[21] R. Bellman, Dynamic Programming, Princeton University Press, Princeton, N. J., 1957

[22] Glorot,X.y Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. En Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249–256).

[23] W. E, "The mathematics of emerging applications: An application of high-dimensional partial differential equations to finance,"Journal of Computational Finance, vol. 8, no. 1, 2004, pp. 7-24.

[24] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics informed deep learning (Part I): Data-driven solutions of nonlinear partial differential equations, arXiv:1711.10561 (2017).

[25] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics informed deep learning (Part II): Data-driven discovery of nonlinear partial differential equations, arXiv:1711.10566 (2017).

[26] M. Raissi, P. Perdikaris, and 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 (2019), 686–707.

[27] Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S. & Yang, L. *Physics-informed machine learning.* Nature Reviews Physics **3**, 422–440 (2021)

[28] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A System for Large-Scale Machine Learning,ïn 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265-283.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, (..), "PyTorch: An Imperative Style, High-Performance Deep Learning Library,.ªdvances in Neural Information Processing Systems 32 (NeurIPS 2019), pp. 8024-8035.

[30] Chollet, F. et al. Keras — Deep learning library. *Keras* https://keras.io (2015).

[31] Moseley, B., Markham, A., & Nissen-Meyer, T. (2023). *Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. Advances in Computational Mathematics.*

[32] Frostig, R., Johnson, M. J. & Leary, C. Compiling machine learning programs via high-level tracing. *Syst. Mach. Learn.* (2018).

[33] Lu, L., Meng, X., Mao, Z. & Karniadakis, G. E. DeepXDE: a deep learning library for solving differential equations. *SIAM Rev.* **63**, 208–228 (2021).

[34] Hennigh, O. et al. NVIDIA SimNet: An AI-accelerated multi-physics simulation framework. Preprint at https://arxiv.org/abs/2012.07938 (2020).

[35] Koryagin, A., Khudozkov, R. & Tsimfer, S. PyDEns: a Python framework for solving differential equations with neural networks. Preprint at https://arxiv.org/abs/1909.11544 (2019).

[36] Chen, F. et al. NeuroDiffEq: A python package for solving differential equations with neural networks. *J. Open Source Softw.* **5**, 1931 (2020).

[37] Rackauckas, C. & Nie, Q. DifferentialEquations.jl — a performant and feature-rich ecosystem for solving differential equations in Julia. *J. Open Res. Softw.* **5**, 15 (2017).

[38] Haghighat, E. & Juanes, R. SciANN: a Keras/TensorFlow wrapper for scientific computations and physics-informed deep learning using artificial neural networks. *Comput. Meth. Appl. Mech. Eng.* **373**, 113552 (2020).

[39] Xu, K. & Darve, E. ADCME: Learning spatially-varying physical fields using deep neural networks. Preprint at https://arxiv.org/abs/2011.11955 (2020).

[40] Jan Blechschmidt, "PDEs by Neural Networks,"*GitHub Repository*, https://github.com/janblechschmidt/PDEsByNNs, acceso el 20 de agosto de 2024.

[41] Maziar Raissi, Applied Deep Learning, *GitHub Repository*, https://github.com/maziarraissi/Applied-Deep-Learning, acceso el 20 de agosto de 2024.

[42] Tapie D. .Estudio Numérico de un Problema de Asimilación de Datos para la Ecuación de Burgers con Viscosidad en 1D", Trabajo de Grado Matemáticas, Universidad del Valle, 2023.

[43] Mishra, S. & Molinaro, R. Estimates on the generalization error of Physics-Informed Neural Networks (PINNs) for approximating a class of inverse problems for PDEs. *Research Report No. 2020-46, Seminar für Angewandte Mathematik, ETH Zürich* (2020).

[44] Gianatti J. (2017). Métodos numéricos para resolver problemas de control óptimo. Tesis Doctoral CONICET - UNR.

[45] Jørgen S. Dokken, Sebastian K. Mitusch y Simon W. Funke (2020). Derivadas de forma automáticas para PDE transitorias en FEniCS y Firedrake , arXiv: 2001.10058 [math.OC].

[46] Sebastian K. Mitusch, Simon W. Funke y Jørgen S. Dokken (2019). dolfin-adjoint 2018.1: adjuntos automatizados para FEniCSand Firedrake , Journal of Open Source Software, 4 (38),1292.

[47] E. Hopf. The Partial Differential Equation, Comm. Pure Appl. Math. 3, pp.201-230-1950

[48] Taylor, M. (2010). Partial Differential Equations III: Nonlinear Equations (Vol. 117). Springer Science y Business Media.

[49] R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*, Society for Industrial and Applied Mathematics (SIAM), 2007.

[50] LeVeque, R. J. (2002). Finite Volume Methods for Hyperbolic Problems. Cambridge University Press.

[51] Lundvall J, Kozlov V, and Weinerfelt P, Iterative methods for data assimilation for Burgers's equation, Journal of Inverse and Ill-Posed Problems 14 , 505–535.2006

[52] H. Lee and I. S. Kang, Neural algorithm for solving differential equations, J. Comput. Phys. 91 (1990), no. 1, 110–131

[53] I. E. Lagaris, A. Likas, and D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (1998), no. 5, 987–1000

[54] I. E. Lagaris, A. Likas, and D. G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw. 11 (2000), no. 5, 1041–1049

[55] Bischof, J., Oevermann, M., & Meinlschmidt, H. (2022). Multi-objective optimization for PINNs. *arXiv preprint arXiv:2212.07900*.

[56] Wang, S., Teng, Y., & Perdikaris, P. (2021). Understanding and mitigating gradient pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5), A3055–A3081.

[57] J. Blechschmidt, O. G. Ernst (2021), Three Ways to Solve Linear and Nonlinear Partial Differential Equations Using Neural Networks linear and nonlinear partial differential equations using neural networks - A Review, GAMM Mitteilungen 2021, TODO.

[58] M. Raissi and G. E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, Journal of Computational Physics 357 (2018), 125–141.

[59] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, Journal of Computational Physics 348 (2017), 683–693.

[60] S. H. Rudy et al., Data-driven discovery of partial differential equations, Science Advances 3 (2017), no. 4.

[61] C. Basdevant et al., Spectral and finite difference solutions of the Burgers equation, Computers & Fluids 14 (1986), no. 1, 23–41.

[62] A. Sharma y D. J. K. C., "Burgers' Equation and Traffic Flow", Journal of Institute of Engineering (JIE), vol.17, no.1, pp.1–7, abril 2023.

[63] T. Dauxois, Fermi, Pasta, Ulam and a mysterious lady, 2008, arXiv:0801.1590.

[64] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, 2015, arXiv:1502.05767.

[65] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, 249–256.

[66] L. Bottou, F. E. Curtis, and J. Nocedal, Optimization methods for large-scale machine learning, SIAM Review 60 (2018), no. 2, 223–311.

[67] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980 (2014).

[68] C. Rao, H. Sun, and Y. Liu, Physics-informed deep learning for incompressible laminar flows, Theoretical and Applied Mechanics Letters 10 (2020), no. 3, 207–212.

[69] E. Hairer, S. P. Nørsett, and G. Wanner, Solving Ordinary Differential Equations I: Nonstiff Problems, Springer Series in Computational Mathematics, vol. 8, Springer-Verlag, Berlin Heidelberg, 1993,

[70] T. Gustafsson et al., "scikit-fem: A Python package for finite element assembly," Journal of Open Source Software5 (2020), no.52, 2369.

[71] P. Virtanen et al., "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," Nature Methods 17 (2020), 261–272, y la documentación de `solve_ivp` consultada en julio de 2025.

[72] T. G. Grossmann, U. J. Komorowska, J. Latz, and C.-B. Schönlieb, *Can Physics-Informed Neural Networks beat the Finite Element Method?*, arXiv preprint arXiv:2302.04107 (2023).

[73] Lax, P. D. & Levermore, C. D. The small dispersion limit of the Korteweg-de Vries equation. *Communications on Pure and Applied Mathematics*, 28(3), 321–377 (1975).

[74] Bona, J. L. & Smith, R. The initial-value problem for the Korteweg-de Vries equation. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 278(1287), 555–601 (1975).