Universidad Autónoma de San Luis Potosí

instituto de física

Física Asistida por Redes Neuronales Artificiales

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

Maestro en Ciencias (Física)

PRESENTA:

Jose Plata Salas

DIRECTOR DE TESIS:
Dr. John Alexander Franco Villafañe

San Luis Potosí, México, 2023

1. Datos del alumno
Fís.
Jose
Plata
Salas
+52 55 37 66 55 54
Universidad Autónoma de San Luis Potosí
Instituto de Física
Física

2. Datos del tutor
Dr.
John Alexander
Franco
Villafañe

3. Datos del sinodal 1
Dr.
Juan Faustino
Aguilera
Granja

4. Datos del sinodal 2
Dr.
Enrique
González
Tovar

5. Datos del sinodal 3
Dr.
Pedro Ezequiel
Ramirez
Gonzalez

7. Datos del trabajo escrito
Física Asistida por Redes Neuronales Artificiales
65
2023

Declaración de autenticidad

Por la presente declaro que, salvo cuando se haga referencia específica al trabajo de otras personas, el contenido de esta tesis es original y no se ha presentado total o parcialmente para su consideración para cualquier otro título o grado en esta o cualquier otra Universidad. Esta tesis es resultado de mi propio trabajo y no incluye nada que sea el resultado de algún trabajo realizado en colaboración, salvo que se indique específicamente en el texto.

Jose Plata Salas. San Luis Potosí, México, 2023

"Los hombres olvidan siempre que la felicidad humana es una disposición de la mente y no condiciones de circunstancias"

Augusto Comte

Resumen

La presente tesis de maestría aborda tres aspectos fundamentales de la física moderna y explora cómo las redes neuronales artificiales pueden contribuir a su resolución. En primer lugar, se emplean técnicas de aprendizaje no supervisado para encontrar soluciones a ecuaciones diferenciales unidimensionales con condiciones de frontera preestablecidas. Específicamente, se investiga el comportamiento de flexión bajo compresión axial de una barra elástica. En segundo lugar, se aplican técnicas de aprendizaje supervisado para enseñar a una red neuronal a estimar los coeficientes de transporte de una partícula cuántica que se desplaza entre pozos y barreras. Esta aproximación supervisada permite obtener predicciones precisas sobre el transporte de la partícula. En tercer lugar, se intenta utilizar el aprendizaje supervisado para abordar el desafiante problema inverso de encontrar un Hamiltoniano correspondiente a un espectro de eigenvalores dado. Aunque se implementaron diversas estrategias supervisadas, los resultados obtenidos fueron limitados en este caso particular. En los dos primeros escenarios de estudio, se emplea el método de los elementos finitos como un mecanismo de validación para comparar los resultados obtenidos mediante el uso de redes neuronales artificiales. Sin embargo, en el último caso, debido a la complejidad del problema, se enfrentaron dificultades significativas para encontrar una solución satisfactoria utilizando el enfoque de aprendizaje supervisado. A través de la utilización de técnicas de aprendizaje supervisado y no supervisado, se logran avances significativos en la resolución de problemas de ecuaciones diferenciales, transporte de partículas cuánticas y el desafío del problema inverso. Estos resultados contribuyen al crecimiento y desarrollo de la física teórica y computacional.

Índice general

Índice de figuras

Capítulo 1

Introducción

Las redes neuronales artificiales (ANNs) surgieron en 1943 con el modelo pionero de neuronas simplificadas propuesto por McCulloh y Pitts, diseñado para emular la función de las neuronas biológicas [1]. Sin embargo, el entusiasmo inicial sobre estos modelos se vio mermado con la publicación del libro "Perceptrons"de Minsky y Papert en 1969, que destacaba las limitaciones inherentes a los perceptrones. Este análisis crítico llevó a un redireccionamiento significativo de los fondos destinados a la investigación en esta área, provocando que muchos científicos abandonaran el campo. Sin embargo, destacados investigadores como Teuvo Kohonen, Stephen Grossberg, James Anderson y Kunihiko Fukushima, perseveraron.

El resurgimiento en el interés por las ANNs llegó en la década de 1980, impulsado en gran medida por avances teóricos, siendo el algoritmo de retropropagación uno de los más significativos, y también por el aumento exponencial en la capacidad de procesamiento de computadoras. Esta renovación se evidenció en el incremento de proyectos financiados, la proliferación de conferencias, congresos y publicaciones en el ámbito del aprendizaje automático. Hoy en día, es común encontrar grupos de investigación dedicados a las redes neuronales en universidades alrededor del mundo, en diversos departamentos, desde psicología y física hasta ciencias computacionales y biología. En esencia, las redes neuronales artificiales se describen mejor como modelos computacionales caracterizados por su capacidad para aprender, generalizar, agrupar y organizar datos, y su funcionamiento se basa en el procesamiento en paralelo. Si bien estas capacidades no son exclusivas de las ANNs y también se pueden observar en modelos computacionales tradicionales, una cuestión clave es si las ANNs ofrecen ventajas sobre los enfoques convencionales para ciertas tareas. Hasta el momento, esta pregunta sigue siendo objeto de debate y estudio.

En las últimas décadas, con el auge de los datos masivos (big data) [2] y el aumento en la capacidad de procesamiento de las computadoras, especialmente con el desarrollo de unidades de procesamiento gráfico (GPUs) [3], las redes neuronales han emergido como una herramienta fundamental en una amplia gama de aplicaciones. Estas estructuras, que alguna vez fueron relegadas debido a sus limitaciones y desafíos en términos de entrenamiento, ahora alimentan algunos de los avances más impresionantes en campos como el reconocimiento de imágenes [4], procesamiento del lenguaje natural [5], medicina [6] y conducción autónoma [7], por nombrar solo algunos. Asimismo, la popularización y democratización de herramientas y plataformas de aprendizaje profundo, como TensorFlow [8] y PyTorch [9], han facilitado la implementación y experimentación, convirtiendo a las redes neuronales en una piedra angular de la inteligencia artificial moderna. A medida que nos adentramos en esta nueva era de la informática, las

redes neuronales continuarán desempeñando un papel crucial, abriendo caminos para soluciones innovadoras y redefiniendo lo que las máquinas pueden lograr.

Las redes neuronales han encontrado una aplicación revolucionaria en la física moderna, un campo que tradicionalmente dependía de ecuaciones determinísticas y modelos computacionales intensivos. Con la creciente complejidad de los problemas en física, desde la exploración de materiales cuánticos hasta la predicción de fenómenos complejos en la física de fluidos, las redes neuronales ofrecen herramientas que pueden aprender patrones y representar funciones no triviales sin requerir un modelo explícito. Un ejemplo notorio es el uso de redes neuronales para simular sistemas cuánticos y calcular funciones de onda en la mecánica cuántica [10]. Asimismo, en la física estadística, las redes neuronales han mostrado su potencial en la identificación de fases y transiciones de fase en sistemas magnéticos [11]. Otro ámbito fascinante es el uso de estas herramientas en la detección y clasificación de eventos en experimentos de física de partículas, como en los detectores del Gran Colisionador de Hadrones [12]. La capacidad de las redes neuronales para modelar relaciones complejas ha abierto nuevos paradigmas en la simulación y predicción en física. Estas técnicas, impulsadas por la disponibilidad de grandes conjuntos de datos y avances en potencia computacional, están llevando a la comunidad física a reexaminar métodos tradicionales y a adoptar enfoques basados en aprendizaje automático. Si bien todavía estamos en las primeras etapas de esta integración, la promesa de las redes neuronales en la física moderna es evidente y es probable que su influencia crezca en las próximas décadas.

En esta tesis de maestria estudiamos tres problemas de física moderna a través de redes neuronales. El primero corresponde al campo de la elasticidad, resolviendo ecuaciónes diferenciales para las flexiones de una viga. El segundo, corresponde al campo de la conducción de particular cuanticas a traves de barreras y el ultimo, corresponde a un problema muy general de intentar obtener un posible Hamiltoniano a primeros vecinos a partir de su espectro, también llamado problema inverso.

En el capitulo 2 de esta tesis, damos una introducción general al aprendizaje automatico de las redes neuronales. Revisaremos conceptos como Aprendizaje Supervisado y No Supervisado, Algoritmo de Backpropagation (Retropropagación) y Diferenciación Automática. En el capítulo 3, introducimos un tipo de red neuronal llamada: red neuronal asistida por la física (Physics-Informed Neural Networks, PINNs). Que son un tipo de red neuronal diseñada para resolver problemas que involucran ecuaciones diferenciales basadas en principios físicos. Estas redes incorporan conocimiento a priori sobre el problema a través de las leyes físicas subyacentes, y su diseño las distingue de las redes neuronales tradicionales [13]. Las PINNs han demostrado recientemente ser un enfoque innovador que combina el aprendizaje automático con la física teórica para resolver problemas útiles en campos tan diversos como la mecánica de fluidos [14], la termodinámica [15] y otros dominios donde las ecuaciones diferenciales juegan un papel central [16].

En el capítulo 4, se abordan tres aspectos fundamentales de la física moderna y se explora cómo las redes neuronales pueden contribuir a su resolución: (1) se emplean técnicas de aprendizaje no supervisado para encontrar soluciones a ecuaciones diferenciales unidimensionales con condiciones de frontera prestablecidas. En particular, se investiga el comportamiento de flexión bajo comprensión axial de una barra elástica. (2) Se aplican técnicas de aprendizaje supervisado para enseñar a una red neuronal a estimar los coeficientes de transporte de una partícula cuántica que se desplaza entre pozos y barreras y (3) se realiza un estudio, a través del aprendizaje supervisado, para abordar el problema inverso de encontrar una Hamiltoniano a partir de un espectro de

eigenvalores. Por último, en el capítulo 5, se discuten los resultados y conclusiones de este trabajo.

Si bien las redes neuronales artificiales no son la solución a todos los problemas, puesto que existen soluciones convencionales (fuera de las ANNs) que son muy poderosas; realizar una mezcla híbrida entre los diferentes métodos provee un paradigma completamente nuevo y más fuerte en la solución de problemas complejos y computacionalmente costosos [17]. Las ANNs se han utilizado exitosamente para resolver problemas tales como visión artificial, procesamiento de lenguaje natural y la teoría de juegos, por mencionar algunos. Su uso se ha incrementado para resolver problemas como las ecuaciones diferenciales ordinarias o parciales, por su capacidad para atender no linealidades, tales como lo son la transferencia de calor por convección. Es así como las PINNs han emergido recientemente como todo un nuevo paradigma de cómputo científico. Las ANNs en conjunto con los métodos tradicionales sugieren ser un método prometedor para la construcción de modelos con predicciones muy rápidas de sistemas dinámicos [16].

Capítulo 2

Redes Neuronales Artificiales

Las redes neuronales artificiales (ANNs, por sus siglas en inglés) son estructuras compuestas de densas capas de elementos adaptativos de procesamiento simples, llamadas neuronas o nodos, con las cuales es posible ejecutar tareas computacionales masivas paralelamente. Aunque las ANNs son drásticas abstracciones de las neuronas biológicas, la idea general de éstas no es reproducir el funcionamiento del cerebro, sino utilizar lo que sabemos del funcionamiento del cerebro para resolver problemas complejos. Las características principales que vuelven a las ANNs tan atractivas para la resolución de problemas incluyen: **no linealidad, insensibilidad al ruido, paralelización de procesamiento, aprendizaje, adaptación y generalización** [17, 18].

Aunque las ANNs son empíricas por naturaleza, tienen la capacidad de brindar solución para problemas definidos de forma precisa, así como para problemas ambiguos que tienen múltiples soluciones. Las distintas aplicaciones de las ANNs van desde el diagnóstico y detección de fallas de celdas fotovoltáicas [19], ciencias forenses [20, 21], detección de fraude en transacciones crediticias [22], medicina [23], entre otras. Una buena introducción al estado del arte en aplicaciones de redes neuronales artificiales se encuentra en [24].

Existen modernos programas que pueden resolver problemas numéricos y simbólicos a una increíble velocidad, pero se encuentran muy lejos del desempeño del cerebro humano para realizar tareas perceptivas tales como el reconocimiento de imágenes y lenguaje. Mientras que las computadoras requieren información precisa de entrada y seguir un conjunto de instrucciones secuenciales, el cerebro humano resuelve las tareas de forma distribuida y paralela, a lo largo de la corteza cerebral. Aunque se estima que los amplificadores operacionales y las compuertas lógicas pueden realizar operaciones varios órdenes de magnitud más rápido que las neuronas, se ha observado que las neuronas requieren un número considerablemente menor de operaciones al resolver un problema [18, 25]. Si la misma técnica de procesamiento de las neuronas pudiera ser implementado en amplificadores operaciones o compuertas lógicas, se podrían construir máquinas relativamente baratas y pequeñas para procesar información, al menos como la que procesa el cerebro.

Por lo tanto, existen fuertes razones para creer en la viabilidad de crear sistemas que procesen la información utilizando los mismos principios que aquellos de las redes neuronales biológicas [26]. A tales sistemas se les conoce como *redes neuronales artificiales, modelos conexionistas o conexionismo.*

Las redes neuronales artificiales (ANNs) son algoritmos basados en la función cerebral y que

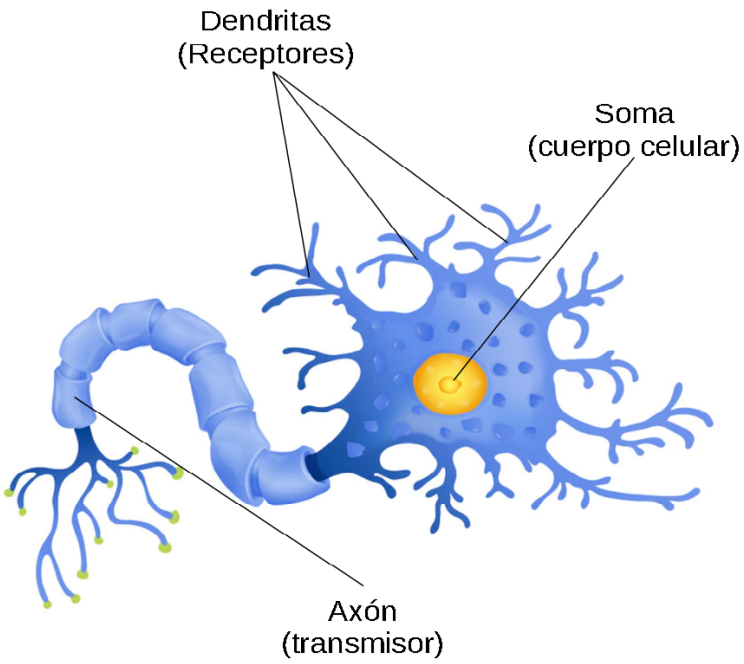

Fig. 2.1: Esquema de una neurona biológica. Se muestra el soma, de color amarillo, el axón que funge como transmisor y las dendritas que son los receptores de información de la neuorona.

son utilizados para modelar problemas complejos y para realizar predicciones. El término '*redes neuronales*' proviene entonces de la intención de simular el sistema neuronal biológico de forma artificial [27].

Las ANNs están inspiradas en la arquitectura de las neuronas en la corteza cerebral. El cerebro humano está compuesto por un gran número de neuronas interconectadas. Cada neurona es una celda que realiza una simple tarea, la cual consiste en dar respuesta a una señal de entrada, sin embargo, cuando consideramos una red de neuronas interconectadas, pueden realizar tareas complejas, tales como reconocimiento de voz, imágenes y texto con una increíble velocidad y precisión. Para el reconocimiento de un rostro, el cerebro tarda alrededor de unos cuantos cientos de milisegundos, mientras que una neurona tarda alrededor de unos milisegundos en procesar información. Esto indica que al cerebro humano le toma solo unos cientos de pasos realizar una tarea de esta naturaleza, en contraste con las miles de iteraciones que una computadora necesita para la misma tarea. El procesamiento tan corto en el cerebro implica que la información transmitida entre las neuronas debe ser realmente pequeña. A diferencia de los algoritmos de procesamiento tradicionales, no toda la información es transmitida entre una neurona y otra, sino sólo una pequeña parte.

2.1. Neurona biológica

Una neurona está compuesta de un gran cuerpo celular, llamado soma, y de fibras nerviosas: una prolongación alargada denominada axón (transmisor) para enviar impulsos y habitualmente

muchas ramificaciones denominadas dendritas (receptores) para recibirlos. La Figura 2.1 muestra el esquema de una neurona biológica. La conexión entre el final de una neurona (axón) y las dendritas de las neuronas de alrededor es llamada sinapsis, la cual es la unidad de comunicación entre dos neuronas. Señales electroquímicas se transmiten a través de la sinapsis. Cuando la señal total que recibe una señal supera el umbral de ésta, hace que la neurona 'se encienda', es decir, que envíe señales a las neuronas de su alrededor.

2.2. Neurona artificial

Inspirado en lo que sabemos de la neurona biológica consideremos, a continuación, el modelo más simple para una neurona artificial. Consideremos la función base como:

$$u = T + \sum_{j=1}^{n_i} w_j x_j \tag{2.1}$$

donde T es el umbral, los w_j son los pesos (sinápticos), u es la función base, x_j son las entradas y n_i la dimensión de la entrada.

La salida de una neurona artificial está dada por:

$$y = f(u) = f(T + \sum_{j=1}^{n_i} w_j x_j). \tag{2.2}$$

donde $f(x)$ es una función (de activación)[1] no lineal. Las funciones escalón:

$$f(x) = \begin{cases} 0, & \text{si } x < 0 \\ 1, & \text{si } x \geq 0, \end{cases} \tag{2.3}$$

signo:

$$f(x) = \begin{cases} -1, & \text{si } x < 0 \\ 1, & \text{si } x \geq 0, \end{cases} \tag{2.4}$$

y la función exponencial:

$$f(x) = \frac{1}{1 + e^{-\beta x}}, \ \beta > 0 \tag{2.5}$$

[1] Una función de activación debe satisfacer ser acotada y diferenciable.

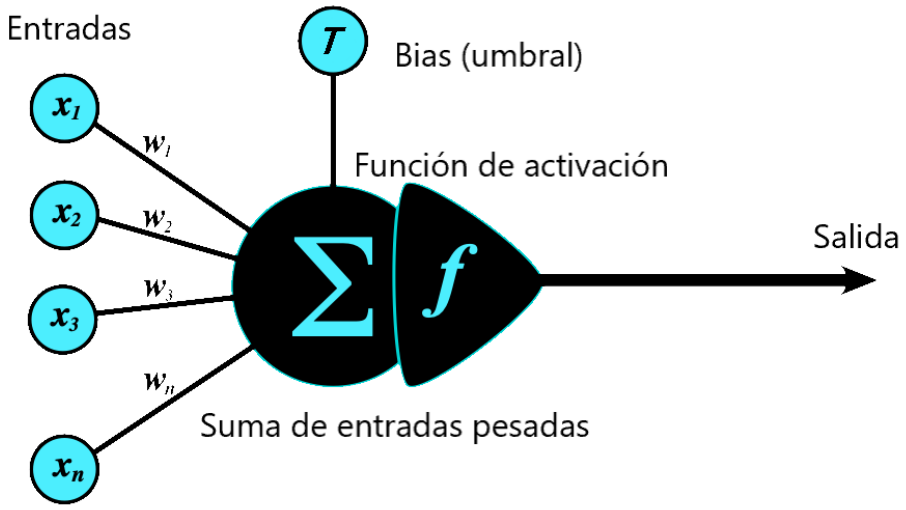

Fig. 2.2: Esquema de una neurona artificial.

son algunos ejemplos de funciones de activación.

2.3. Descripción general de una red neuronal artificial

Una ANN está modelada en semejanza a las redes neuronales biológicas. De igual forma que en la red neuronal biológica, una ANN es una interconexión de nodos, análogos a las neuronas. Cada ANN tiene tres componentes principales: la caracterización de nodos, la topología de la red y las reglas de aprendizaje. El primero, determina como serán procesadas las señales por los nodos, tales como definir el número de entradas y salidas en un nodo, los pesos asociados a cada entrada y salida, así como la función de activación. La topología de la red determina la forma en que los nodos están acomodados en la red. Finalmente, las reglas de aprendizaje determinan cómo se inician y ajustan los pesos. A continuación se describen brevemente los tres componentes.

2.3.1. Caracterización del nodo

El modelo más simple para un nodo (neurona artificial) se muestra en la Figura 2.2. Cada nodo recibe múltiples entradas de otros nodos que tienen pesos asociados, análogo a la sinapsis. Cuando la suma de las entradas bajo sus pesos supera el umbral del nodo, se activa y pasa una señal a los nodos siguientes. Este proceso se puede expresar matemáticamente como:

$$y = f\left(\sum_{i=0}^{n_i} w_j x_j + T\right) \tag{2.6}$$

donde w_i son los pesos correspondientes a cada entrada x_i, T es el valor del umbral de la neurona, f es la función de activación y y es la salida de la neurona hacia las neuronas vecinas.

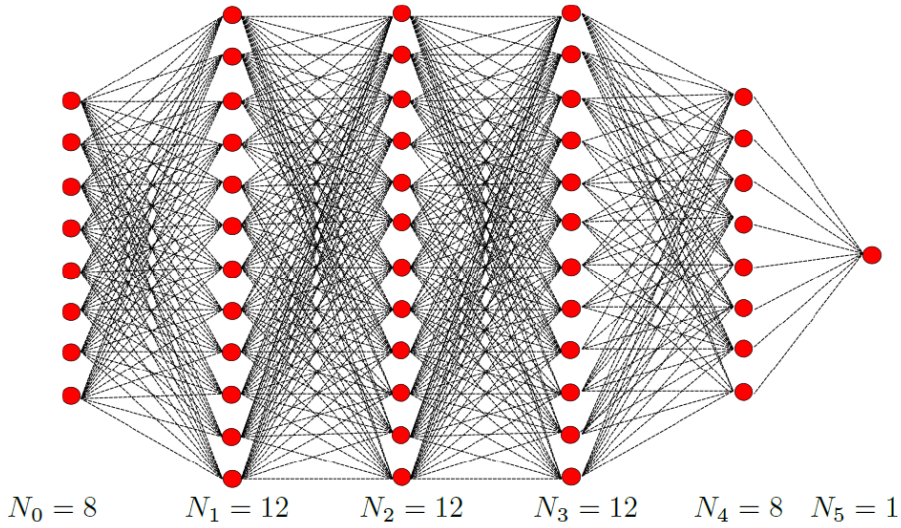

Fig. 2.3: Ilustración de la topología general de una red. En el ejemplo, se considera una red con 8 neuronas de entradas, 3 capas ocultas (de 12 neuronas cada una), una cuarta capa con 8 neuronas y finalmente una capa de salida con una única neurona. Los puntos rojos corresponden a las neuronas.

2.3.2. Topología de la red

En la arquitectura general de una ANN, los nodos están organizados en forma de arreglos lineales, llamados capas. Una ilustración de un ejemplo de red neuronal se muestra en la Figura 2.3 Regularmente, una ANN se estructura de una capa de entrada, una capa de salida y capas ocultas. No existe un número determinado para las capas ocultas, una ANN puede contener varias capas ocultas, aunque puede no tener ninguna. El diseño de una red neuronal (para un problema en específico) consiste en determinar el número de nodos en cada capa, el número de capas en la red, y la forma de las conexiones entre las redes. Usualmente, estos parámetros son elegidos por intuición y optimizados a lo largo de una sucesión de experimentos [28]. Existen también algunos métodos racionales para la elección de los parámetros, Concretamente en [29] hacen uso de un algoritmo genético en modelos QSAR [30].

2.3.3. Reglas de aprendizaje

La ANN considera reglas de aprendizaje para entrenar la red. Durante el entrenamiento, los pesos son ajustados a los valores que mejor modelan la tarea encomendada. El aprendizaje puede considerarse en dos grandes categorías: aprendizaje supervisado y aprendizaje no-supervisado. En el primero, un conjunto de entrenamiento está dado, este conjunto contiene pares de valores de entrada y salida. En el entrenamiento, los pesos de la red son ajustados de forma que se minimice el error de los valores de salida de la red y los valores de salida correctos. El conjunto de entrenamiento debe ser una muestra representativa del problema en cuestión. Si el conjunto no cumple esta regla, no se puede producir un modelo confiable que generalice a los datos. Para las redes que consisten en este tipo de aprendizaje, la red debe ser entrenada primero. Cuando la red produzca los valores deseadas de salida, ésta puede ponerse en funcionamiento. De forma opuesta, en el aprendizaje no-supervisado la ANN no utiliza los valores de salida de un conjunto

de aprendizaje, sino que ésta intenta descubrir la tendencia y/o patrón de los datos en cuestión. Diferentes tipos de redes, requiren distintas reglas de aprendizaje.

Históricamente, el aprendizaje *Hebbiano* es la regla de entrenamiento más vieja y una de las más utilizadas aún en nuestros días [18]. Durante el año 1961, Donald Hebb propuso un modelo de aprendizaje en el que el valor del umbral de la neurona crezca siempre y cuando, tanto la neurona anterior como la siguiente se activen simultanéamente. De esta manera, las conexiones más usadas son las que más peso tienen, de cierta forma, simulando la forma en que se comporta el cerebro humano, a través de la repetición.

Decimos que una red neuronal que utiliza el aprendizaje hebbiano incrementa los pesos de las neuronas en la red considerando los niveles de excitación de las neuronas inicial y final en la conexión. De este modo, el aprendizaje de la red se realiza iterando el proceso sucesivamente con los datos iniciales, sin considerar un valor objetivo o salida. Por esta razón, este tipo de aprendizaje se considera no supervisado. Aquellas redes en las que se utilizan los valores de salida u objetivos, son llamadas de aprendizaje supervisado. A continuación se describen brevemente cada uno de los tipos.

2.3.4. Aprendizaje supervisado

El aprendizaje supervisado, a diferencia de la regla hebbiana, considera un conjunto de pares de datos: entradas y salidas. El modelo realiza su aprendizaje iterando la red sobre las entradas y salidas, y midiendo el error entre la salida de la red y la salida de entrenamiento dada a la red. Luego, utilizando la red y un conjunto de pares de prueba (estos deben ser distintos a los de entrenamiento, es decir, deben ser datos no vistos por la red antes) se considera el mejor modelo a aquel que minimiza el error entre la salida de la red y la salida del conjunto de prueba. El objetivo principal es aprender una función de mapeo desde las entradas hasta las salidas. Una vez que el algoritmo ha sido entrenado adecuadamente, puede predecir la salida para nuevas entradas desconocidas.

Durante los años 60, Rosenblatt trabajó en el desarrollo de modelos de aprendizaje supervisado [31]. A este tipo de redes neuronales se le conoce como perceptrón. Rosenblatt demostró que un perceptrón de una sola red es capaz de aprender distintas funciones, de hecho cualquier función correspondiente a un problema linealmente separable.[2] Consideremos el modelo más simple para un perceptrón, *el modelo de neurona de McCulloch y Pitts*. En este modelo, consideramos un par de conjuntos (de entrenamiento) de datos, entradas y salidas, es decir, $U_N = \{(\vec{x_1}, y_1), (\vec{x_2}, y_2), ..., (\vec{x_N}, y_N)\}$. De este modo, en el par k, tenemos que en la entrada $\vec{x_k}$, la regla de actualización de los pesos $w = [w_0, w_1, ..., w_M]^T$ se define como:

$$w(k+1) = w(k) + \eta(y_k - o_k)\vec{x_k} \tag{2.7}$$

donde o_k es la salida de la red y el parámetro η es un valor entre 0 y 1, llamado *tasa de aprendizaje*, y se utiliza para ajustar la velocidad en la convergencia de la red [25].

[2] Un problema linealmente separable es aquel en cuales las clases se pueden separar por un hiperplano.

2.3.5. Aprendizaje no supervisado

La principal característica del aprendizaje no supervisado recae en que no utiliza pares de entradas y salidas, sino únicamente entradas. De forma concisa, decimos que una red neuronal encuentra los patrones en los datos sin la necesidad de la intervención humana.

En general, los algoritmos de aprendizaje no supervisado son utilizados para tres principales actividades: clusterización, asociación y reducción de dimensionalidad. La clusterización (*clustering*) es una técnica de minería de datos en la cual conjuntos de datos son clasificados según sus similitudes y/o diferencias. Las reglas de asociación son un método para descubrir relaciones entre las variables de un conjunto de datos. Por último, la reducción de dimensionalidad es una técnica (principalmente para manejar conjuntos de datos con alta dimensionalidad) en donde se reduce el número de entradas a un número manejable preservando la integridad del conjunto de datos. Ejemplos de algoritmos de reducción de dimensionalidad son: a) PCA (Principal component analysis), b) SVD (single value decomposition)[25, 32].

2.4. Tipos de redes neuronales

2.4.1. Neurona de McCulloh y Pits (Perceptrón)

Una neurona de *McCulloch* y *Pitts* (perceptrón) es una neurona con función de activación escalón (Heaviside) de la forma:

$$x \in \mathbb{R}^d \mapsto H\left(\sum_{i=1}^{d} w_i x_i + T\right), \tag{2.8}$$

donde $d \in \mathbb{N}$, $H(x)$ es la función de Heaviside, i.e, $H(x) = 0$ si $x \leq 0$, $H(x) = 1$ si $x > 0$. w_i es el peso asociado a la entrada x_i y T es el umbral. La neurona de McCulloch y Pitts, recibe d señales. Si estas señales multiplicadas por sus pesos son sumadas y superan el umbral, entonces la señal 'se enciende' y regresa un 1. En otro caso, la neurona permanece inactiva.

Una ANN se construye a partir de la conexión de múltiples neuronas juntas en el sentido de que la salida de una neurona forme una entrada para las siguientes. Un modelo simple para una red con dichas características es el *perceptrón multicapa*. Este modelo fue diseñado por Rossenblatt[31].

2.4.2. Perceptrón multicapa (MLP)

Definición 2.4.1 *Sea $d, L \in \mathbb{N}, L \geq 2$ y $\rho : \mathbb{R} \to \mathbb{R}$. Entonces un perceptrón multicapa (MLP) con entrada $d-$dimensional, L capas (una de entrada, una de salida y $L-2$ capas ocultas) con funciones de activación ρ_i es una función F que se puede escribir como:*

$$x \mapsto F(x) := \rho_L(T_L(\rho_{L-1}(T_{L-1}(...\rho_1(T_1(x))...)))), \tag{2.9}$$

donde $T_{l(x)} = A_l x + b_l$, *y* $(A_l)_{l=1}^L \in \mathbb{R}^{N_l \times N_{l-1}}$, $b_l \in \mathbb{R}^{N_l}$ *con* $N_l \in \mathbb{N}$, $N_0 = d$ *y* $l = 1, 2, ..., L$. *Además* $\rho_i : \mathbb{R} \to \mathbb{R}$.

Las neuronas en el modelo MLP corresponden a una aplicación $\rho_i : \mathbb{R} \to \mathbb{R}$ aunque, en contraste, con la neurona de McCulloch y Pitts, ρ_i es una función de activación arbitraria. En la Figura 2.3 se muestra el esquema de una MLP. Se debe tener en cuenta que la MLP, no permite conexiones aleatorias entre las neuronas, sino entre aquellas que forman parte de capas adyacentes, y únicamente de las capas inferiores a las superiores (en el sentido de la capa de entrada hacia la de salida). Mientras que las MLP y sus variaciones son, probablemente, el tipo de redes neuronales más utilizado en la práctica. Su arquitectura y mecanismos de aprendizaje son muy diferentes a las redes neuronales biológicas.

2.5. Funcionamiento de un perceptrón

Consideremos el problema de encontrar los coeficientes de una relación conocida: la transformación de temperatura de grados celsius (C) a farenheit (F). Como sabemos esta relación viene dada como $F = 1.8 \times C + 32$.

Considerando una única neurona artificial, la salida de ésta viene dada como
$s_k = w_1^k \times C + w_2^k$, por lo que el aprendizaje de la neurona en este problema consiste en entrenar los parámetros w_1^k y w_2^k de forma que ajusten mejor la transformación de grados celsius a farenheit. s_k es el valor de salida de la red para la iteración k.

Utilizando el lenguaje de programación Python, se realiza la construcción de una neurona artificial desde cero. Primero, construimos los pares de datos para el ejercicio. Para la temperatura en grados celsius se crean 20 elementos (de números enteros) en el rango $[0, 30]$, para su correspondiente temperatura en farenheit, realizamos la transformación conocida $F = 1.8 \times C + 32$. Utilizando estos pares de datos, entrenaremos a la neurona para determinar dichos valores.

Dado que para este ejercicio, tenemos un procesamiento del tipo supervisado, tenemos las siguientes reglas de actualización de los pesos (considerando un α dado como tasa de aprendizaje) de la red:

$$w_1^k = w_1^{k-1} + \alpha \times (t_k - o_k) \times x_k \tag{2.10}$$
$$w_2^k = w_2^{k-1} + \alpha \times (t_k - o_k) \tag{2.11}$$

Además, se inicializan los valores $w_1^1 = 1$ y $w_2^1 = 1$. Se realizan hasta un máximo de 25k iteraciones (éstas se realizan aleatoriamente sobre los 20 pares de datos, celsius y farenheit).

Un fragmento del código utilizado se muestra a continuación:

Snippet 2.1: Transformación Celsius-Farenheit

```
#Loading libraries needed
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt

#Data creation
c = np.random.randint(0,30,size=20)
f = [1.8*element + 32 for element in c]

#Simple neuron code
w1 = 1
w2 = 1
alp = 5e-3

i = 0
W1,W2,E,Index = [],[],[],[]

while i < 2.5e4:
    j = random.randint(0,len(c)-1)
    s = w1*c[j] + w2

    w1 = w1 + alp*(f[j]-s)*c[j]
    w2 = w2 + alp*(f[j]-s)

    W1.append(w1)
    W2.append(w2)
    E.append(abs(f[j]-s))
    Index.append(i)

    i+=1
```

El aprendizaje de la neurona artificial ocurre apenas a los 0.2 s, considerando las 25k iteraciones. A continuación se muestran dos gráficas que consisten con el aprendizaje de los parámetros w_1 (Figura 2.4) y w_2 (Figura 2.5) en función del número de iteraciones. Los resultados del aprendizaje dan como valores $w_1 = 1.8000000000000522$ y $w_2 = 31.999999999999545$, que dan como resultado una transformación con una tolerancia de error menor a 10^{-13}.

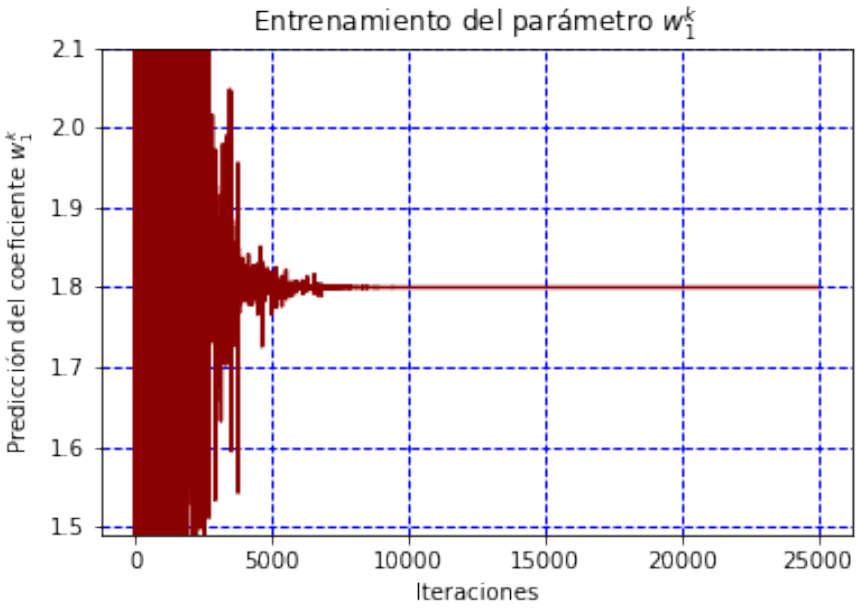

Fig. 2.4: Gráfica del coeficiente w_1 en función del número de iteraciones. En este primer caso, se muestra que la red converge a un valor 1.8000000000000522. En este caso, la figura se limita al intervalo en $[1.49, 2.1]$ del eje de las ordenadas.

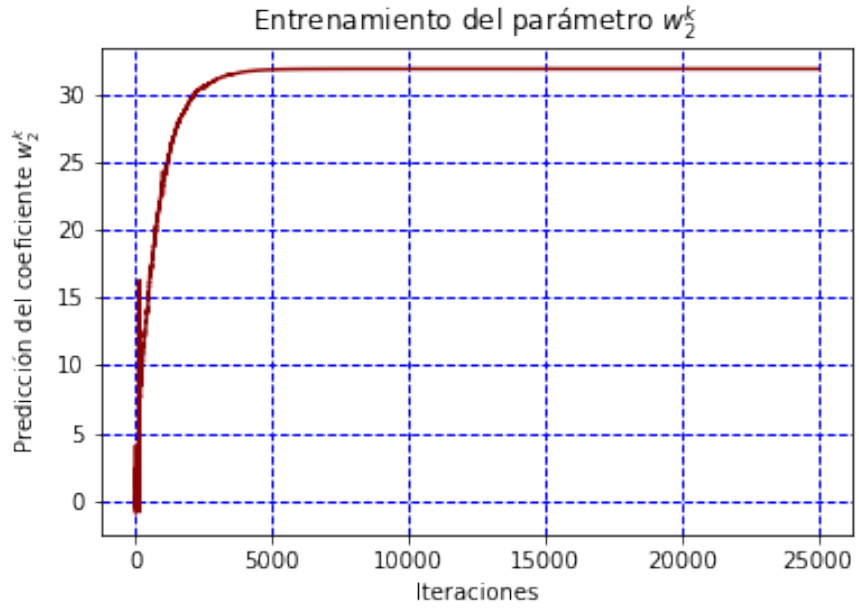

Fig. 2.5: Gráfica del coeficiente w_2 en función del número de iteraciones. En este primer caso, se muestra que la red converge a un valor 31.999999999999545.

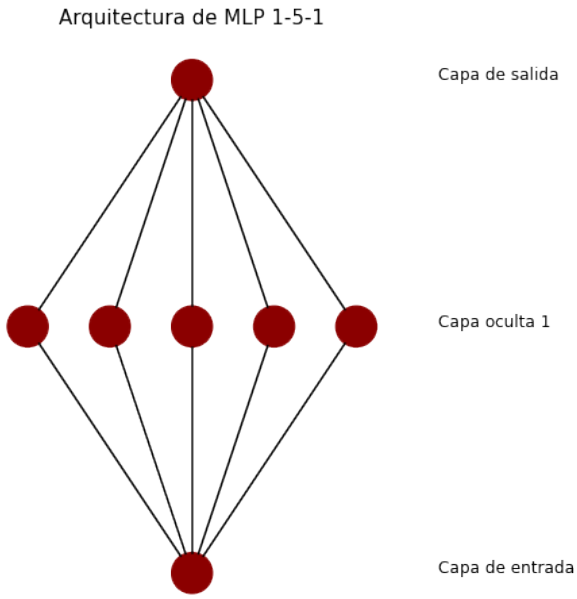

Fig. 2.6: Esquema de una red neuronal artificial (MLP) con una configuración 1-5-1, es decir, una capa de entrada, 5 neuronas en la capa oculta intermedia y finalmente una capa de salida.

2.6. Funcionamiento de un MLP

Consideremos de nuevo el problema anterior pero en este caso, abordaremos el problema por medio de una MLP. El diseño de esta red neuronal artificial está constituido por una neurona como capa de entrada, una capa oculta con 5 neuronas artificiales, las cuales servirán como procesamiento en la transformación y finalmente una capa de salida. Un esquema del MLP con la configuración descrita se puede ver en la Figura 2.6.

Un fragmento del código utilizado para generar la simulación se muestra a continuación:

Snippet 2.2: Funcionamiento de un MLP

```
#Loading libraries needed
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt

#Data creation
c = np.random.randint(0,30,size=20)
f = [1.8*element + 32 for element in c]

### Creating the model ###
# Multiple neuron model (MLP)
inputLayer = keras.layers.Dense(units = 1, input_shape = [1],\
activation = 'relu')
hiddenLayer = keras.layers.Dense(units = 5, activation = 'relu')
outputLayer = keras.layers.Dense(units = 1)
model = keras.Sequential([inputLayer,hiddenLayer,outputLayer])
```

```
#Model compilation. Defining learning rate and loss method.
model.compile(optimizer = keras.optimizers.Adam(3e-1),\
loss = 'mean_squared_error')

# Defining epochs. *epoch: iteration over training dataset.
historic = model.fit(data['c'],data['f'],epochs = 1000, verbose = 0)

# Testing
model.predict([100]) #Result:array([[211.99986]], dtype=float32)

#Getting weights of the model.
model.get_weights()
```

Con la definición de este modelo, tenemos un modelo secuencial de 18 parámetros totales, pues tenemos 2 valores para la capa de entrada (pues sólo tiene una neurona), 10 valores para la capa oculta (con 5 neuronas) y 6 valores para la capa de salida (5 para la lectura de los valores de salida de la capa oculta además del umbral para la neurona en esta capa). El aprendizaje para este tipo de red resultó ser mucho más rápido, pues observando la Figura 2.7, podemos concluir que antes de la epoch[3] 200 se obtenía una convergencia muy aceptable. Sin embargo, en este modelo multicapa no es posible darle una interpretación física a los parámetros obtenidos. Éstos se muestran a continuación.

El entrenamiento de la red dio como resultado el conjunto de los 18 valores para la red, obteniéndose la lista de valores (de forma aleatoria) para los 18 parámetros:

```
#Model weights
model.get_weights()

#Output
[[0.3936084]], #Input layer
10.010584], #Input layer

[ 2.2583277 , -1.7494329 , -1.6211609 , \
-0.6625211 , -0.48154092]], #Hidden layer
[-0.40898386, -1.8016506 , -2.5881705 ,  \
0.        ,  0.        ], #Hidden layer

[ 2.0249813 ], #Output layer
[ 1.3222066 ], #Output layer
[ 1.5077455 ], #Output layer
[ 0.2340045 ], #Output layer
[-0.80062604]], #Output layer
[-12.950986]] #Output layer
```

[3]Iteración sobre todo el conjunto de entrenamiento

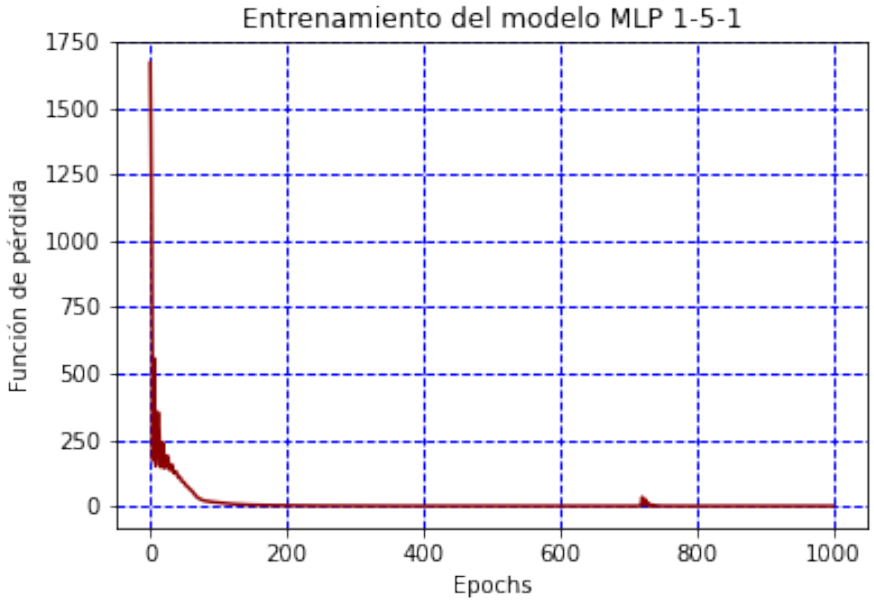

Fig. 2.7: Gráfica de la función de pérdida vs Epochs.

Capítulo 3

Física asistida por redes neuronales artificiales (PINNs)

La física asistida por redes neuronales artificiales es una técnica recientemente desarrollada que pretende solucionar ecuaciones diferenciales ordinarias y parciales. Las PINNs son técnicas de resolución de ecuaciones diferenciales parciales. Las PINNS aproximan soluciones de PDEs a partir del entrenamiento de una ANN por medio de la minimización de una función de pérdida que incluye términos referentes a las condiciones iniciales y de borde a través del dominio (espacio-tiempo), es decir, incorpora términos de la ecuación diferencial dentro de la función de costo. Éstas son redes que, dada una entrada produce una solución estimada para cada punto, después del entrenamiento. El concepto básico detrás del entrenamiento de una PINN es que no requiere resultados previos de simulaciones o experimentos. Aún más, es un técnica a la cual no se le necesita definir un mallado o partición del dominio, si no que convierte el problema en uno de optimización de una función de pérdida. Su mecanismo integra el modelo matemático a través de un término de penalización, que actúa para restringir el espacio de soluciones físicas aceptables. El entrenamiento de una PINN es un tipo de aprendizaje no supervisado porque no se requieren resultados previos, como simulaciones o experimentos.

El algoritmo de PINN se basa en una técnica que no requiere una partición o mallado del dominio, sino que transforma el problema de resolver ecuaciones diferenciales en un problema de optimización que minimiza la función de costo. El proceso de esta técnica se apoya en la integración del modelo matemático en la red, junto con la inclusión de un término de penalización proveniente de la ecuación original, el cual actúa para restringir el conjunto de soluciones posibles. En otras palabras, las PINNs consideran la ecuación diferencial del problema y la física subyacente, en lugar de solo tratar de deducir una solución a través de datos de entrada y salida.

En 1994, Dissanayake propuso la idea de utilizar técnicas de redes neuronales para resolver ecuaciones diferenciales [33]. En su artículo, empleó una red neuronal simple para aproximar una ecuación diferencial parcial (PDE), utilizando los resultados de aproximación universal de finales de la década de 1980. La red neuronal consistía en dos capas ocultas con 3, 5 y 10 nodos en cada una, y la función de pérdida se utilizó para aproximar el error L^2 [1] en el interior y la frontera del punto de colocación. Para evaluar los gradientes, se utilizaron diferencias finitas, y la pérdida se

[1] La norma L^2 o *distancia euclidiana* está definida como $||x||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$.

evaluó entre cada iteración utilizando una aproximación quasi-newtoniana.

La inclusión sistemática de la física en una red neuronal tiene sus orígenes en la investigación de Owhadi en 2015, quien demostró el poder de estas técnicas [34]. En 2017, Raissi y sus colaboradores [35] utilizaron un proceso de regresión gaussiana para construir representaciones de operadores funcionales y calcular la incertidumbre en las soluciones de diversos problemas. Este trabajo se expandió en una colaboración posterior con Karniadakis [36]. En 2017, Raissi y sus colaboradores introdujeron las PINNs como una nueva familia de técnicas para resolver problemas de ecuaciones diferenciales [37]. En 2019, en la segunda parte de esta serie de artículos, ilustraron la solución de ecuaciones diferenciales como las ecuaciones de Schrödinger, Burgers y Allen-Cahn [13]. En éstos últimos, se crearon dos tipos de PINNs que pueden resolver tanto la solución que gobierna el modelo matemático como ajustar los parámetros basándose en los datos observables.

3.1. Los bloques fundamentales de una PINN

Las PINNs pueden resolver ecuaciones diferenciales que se expresen, de forma general, como:

$$\begin{aligned} \mathbf{F}(u(z);\gamma) &= \mathbf{f}(z) \qquad z \in \Omega \\ \mathbf{B}(u(z)) &= \mathbf{g}(z) \qquad z \in \partial\Omega, \end{aligned} \tag{3.1}$$

definida en el dominio $\Omega \subset \mathbb{R}^d$, $d \in \mathrm{N}$ y frontera $\partial\Omega$. Donde $z := [x_1, x_2, ..., x_{d-1}; t]$ indica el vector coordinado de espacio tiempo, u representa la función desconocida, γ son los parámetros físicos del problema, $\mathbf{f}$ determina las condiciones iniciales del problema, $\mathbf{F}$ es un operador diferencial, no necesariamente lineal. Finalmente, en el operador $\mathbf{B}$ se establecen las condiciones de borde y $\mathbf{g}$ es la función en el borde. Con esta metolodología pueden resolverse problemas Dirichlet, Neumann, Robin o condiciones de borde periódicos.

La Ecuación 3.1 puede describir numerosos sistemas físicos incluyendo problemas directos e inversos. El objetivo de los problemas directos es encontrar la función u para todo z, mientras que γ, los parámetros, están dados. En el caso de los problemas inversos, γ puede ser determinado de datos experimentales.

En la metodología de PINN, $u(z)$ es aproximada computacionalmente por una ANN, parametrizada por un conjunto de parámetros θ, que determinan una aproximación de la forma:

$$\hat{u}_\theta(z) \approx u(z);$$

donde $\hat{(.)}_\theta$ expresa una aproximación por ANN realizada con los parámetros θ.

Con esto dicho, la ANN debe aprender a aproximar la ecuación diferencial a través de buscar los parámetros θ que definan a la ANN por medio de la minimización de la función de costo que depende en la ecuación diferencial $\mathcal{L}_\mathcal{F}$, las condiciones de borde $\mathcal{L}_\mathcal{B}$ y eventualmente los datos

generados en el dominio $\mathcal{L}_{data}$:

$$\theta^* = \arg\min_\theta(\omega_\mathcal{F}\mathcal{L}_\mathcal{F}(\theta) + \omega_d\mathcal{L}_{data}(\theta) + \omega_\mathcal{B}\mathcal{L}_\mathcal{B}(\theta)) \tag{3.2}$$

donde $\omega_\mathcal{F}$, $\omega_{\mathcal{L}_{data}}$ y $\omega_\mathcal{B}$ son los pesos de cada tipo de pérdida[2].

En pocas para palabas, podemos decir que las PINNs pueden ser vistar como aprendizaje no supervisado cuando la red neuronal se entrena utilizando las ecuaciones físicas y las condiciones de borde; por otro lado, en el caso de problemas inversos o cuando algunas propiedades son extraídas de datos con ruido, PINN se puede considerar como una metodología de aprendizaje supervisado.

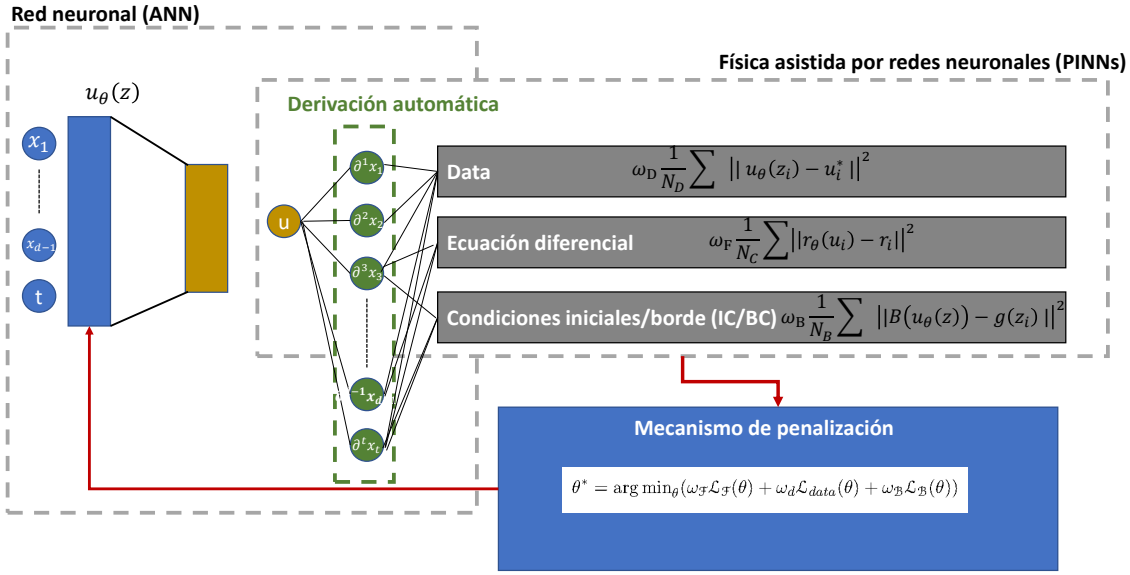

Fig. 3.1: Esquema de una PINN. Una PINN se componen de tres elementos fundamentales: una red neuronal, un componente funcional que contiene la diferenciación automática y un mecanismo de retroalimentación (la minimización de la función de costo). La primera produce una salida para un vector en el dominio, la segunda determina por medio de la diferenciación automática las derivadas parciales respecto a los parámetros θ y finalmente el tercero determina el siguiente conjunto de parámetros θ para la siguiente iteración. El proceso en general, consiste en minimizar θ^* utilizando un optimizador, de acuerdo a una tasa de aprendizaje previamente elegida.

La Figura 3.1 resume los bloques fundamentales de las PINNs que se han discutido a lo largo de esta sección. La PINN tiene tres componentes: una red neuronal, un componente funcional que contiene la diferenciación automática y un mecanismo de penalización. En primer lugar, la ANN, $\hat{u}_\theta$, acepta un vector del dominio de espacio-tiempo y calcula un valor de salida. En segundo lugar, el componente funcional, encargado de calcular la derivada utilizando la diferenciación automática respecto a los parámetros de la ecuación diferencial, las condiciones iniales y de borde; así como la de los datos creados por la red neuronal previa. De esta forma, se le inyectan las

[2]En cada iteración, se actualizan los pesos con la regla $\omega_{x_{i+1}} = \omega_{x_i} + \alpha\frac{\partial\mathcal{L}(\theta)}{\partial\omega_{x_i}}$, donde x representa a la variable y el subíndice i el número de iteración. α es la tasa de aprendizaje de la red.

ecuaciones físicas al modelo computacional. Por último, el mecanismo de penalización minimiza la función de pérdida para encontrar los nuevos parámetros θ^* de la ANN del primer paso. Este mecanismo se itera hasta encontrar el conjunto de parámetros θ^* que determinan la solución a la ecuación diferencial.

3.2. Feed-forward neural networks (FFNN)

Una red neuronal de propagación hacia adelante (feed-forward neural network), también conocida como perceptrón multicapa (MLP), es una colección de neuronas organizadas en capas que realizan cálculos secuenciales entre ellas. Éste tipo de redes están completamente conectadas, dado que las neuronas en capas adyacentes están conectadas. En cada neurona se realiza una operación matemática que aplica una función de activación a la suma pesada de sus propias entradas más un factor constante. Dicho esto, dada una entrada $x \in \Omega$, una MLP transforma esta entrada a una salida, a través del conjunto de capas de neuronas que componen un mapeo afín-lineal [3] entre las neuronas (en las capas sucesivas) con funciones de activación no lineales en las unidades, resultando así en una composición de funciones. De esta forma, para una MLP de L capas, tenemos, retomando la Ecuación 2.9:

$$\rho_i(z_i; A_i, b_i) = f_i(A_i \cdot z_i + b_i) \tag{3.3}$$

con lo que la función buscada $u(x)$ se aproxima mediante la transformación:

$$\hat{u}_\theta(z) = f_L \circ f_{L-1} \cdots \circ f_1(z) \tag{3.4}$$

donde cada ρ_i se define como el producto interior (euclidiano, puesto $E = H = \mathbb{R}$) de dos espacios E_i y H_i, siendo $\rho_i \in E_i \times H_i$. Además α_i es una función de activación escalar (no lineal). De esta forma se define:

$$\hat{u}_\theta(z) = T_L(z) \circ T_{L-1} \cdots \circ T_1(z) \tag{3.5}$$

para cualquier $1 \leq k \leq L$, donde $T_l(x) = A_l x + b_l$. De este modo, una FFNN consiste de una capa de entrada, una capa de salida y $K - 2$ capas ocultas. Con eston, retomamos la definición dada en la Subsección 2.4.2.

[3] Una **transformación afín (lineal)** es una transformación geométrica que preserva líneas y paralelas. Más en general, podemos decir que una transformación afín es un automorfismo de un espacio afín (en este caso, los espacios euclidianos son espacios afínes), es decir, es una función (biyectiva) que mapea un espacio afín a otro preservando la dimensión de los subespacios afines (puntos mandan a puntos, líneas a líneas, planos en planos y así sucesivamente).

3.2.1. Múltiples FFNN

En general, aunque la mayoría de las publicaciones emplean una única FFNN en el proceso, un número creciente de publicaciones referentes a PINNs con múltiples MLP han sido estudiados. Una arquitecura compuesta de cinco FFNN fue propuesta en [38]. A continuación se mencionan algunos ejemplos. En un problema de Stefan de dos fases, una ANN es utilizada para modelar la interfase desconocida entre las dos diferentes fases, mientras que otra describe las dos distribuciones de temperatura de las fases [39]. Por otro lado, en [40], Moseley et al. en lugar de utilizar una ANN sobre todo el dominio, utilizaron un conjunto de ANNs, una para cada subdominio.

3.3. Inyección de Física al modelo

Con finalidad de poder insertar las leyes físicas de la ecuación diferencial a la PINN, son necesarias las derivadas de la red neuronal de salida, es decir $u_\theta(z)$ respecto a los parámetros de la ANN. Dado que la u se aproxima a través de una ANN que elige funciones de activación suaves, es posible derivar $u_\theta(z)$. Existen cuatro métodos para calcular derivadas: manual, simbólica, numérica y automáticamente. Calcular las derivadas manualmente puede ser correcto, pero dado que no es automático es impráctico en este caso.

Cuando se trata de funciones en geometrías complicadas, en diferentes escalas de tiempo y espacio o con interfaces discontinuas, los métodos de diferenciación numéricos y simbólicos, como las derivadas por diferencias finitas, no son buenas aproximaciones. Por otro lado, la diferenciación automática (DA) supera numerosas restricciones sobre los métodos tradicionales como errores por la precisión de punto flotante, diferenciación numérica o uso intensivo de memoria, en el caso simbólico [41, 42]. La DA es también conocida como diferenciación algorítmica, que es más razonable, dado que en DA no se deriva automáticamente, si no que se realizan evaluaciones analíticas de estas. Una explicación clara y concisa del algoritmo puede encontrarse en [43]. Consideremos una función $f : \mathrm{R}^n \to \mathrm{R}^m$, a la cual queremos calcular el jacobiano J después de obtener el grafo de todas las operaciones que componen la expresión matemática. Además, la DA puede calcular tanto la derivada hacia adelante como la hacia atrás. En esencia, la DA incorpora una PDE dentro de la función de pérdida de la red neuronal, donde la ecuación residual[4] está dada por:

$$r_{\mathbf{F}}[\hat{\mathbf{u}}_\theta](z) = r_\theta(z) := \mathbf{F}(\hat{\mathbf{u}}_\theta(z); \gamma) - \mathbf{f} \tag{3.6}$$

y de forma similar los correspondientes residuos para las condiciones iniciales y/o de borde se obtienen al sustituir $\hat{\mathbf{u}}_\theta$ en la segunda ecuación de la Ecuación 3.1, es decir:

$$r_{\mathbf{B}}[\hat{\mathbf{u}}_\theta](z) := \mathbf{B}(\hat{\mathbf{u}}_\theta(z)) - \mathbf{g}(\mathbf{z}) \tag{3.7}$$

[4]Una ecuación residual representa la diferencia entre la solución analítica a un problema y una aproximación a dicha solución. Una ecuación de este tipo determina que tan buena es la aproximación a la solución. Idealmente, se desea que la ecuación residual sea lo más cercana a cero, sin embargo, en la práctica, raramente la ecuación residual es exactamente cero debido a errores de cálculo numérico (precisión y redondeo).

Haciendo uso de los dos residuos anteriores, es posible determinar que tan buena o mala aproximación es u_θ para satisfacer la solución al problema de la Ecuación 3.1. Para determinar la solución exacta, se busca que los residuos anteriores sean cero. Cuando la red está compuesta de muchas capas la DA incorpora la regla de la cadena, dado que se deriva jerárquicamente desde la capa de salida a la capa de entrada. Aún así, existen algunos problemas donde esta metodología no puede aplicarse. En particular, Pang et al. [44] en 2019, sustituyeron operadores diferenciales fraccionarios por sus versiones discretas, las cuales fueron incorporadas dentro de la función de pérdida.

3.4. Estimación de paramétros en el aprendizaje

La metodología de PINN para la determinación de los parámetros θ de la red neuronal $\hat{u}_\theta$, consiste en la minimización de la función de pérdida, es decir:

$$\theta = \arg\min_\theta \mathcal{L}(\theta) \tag{3.8}$$

donde

$$\mathcal{L}(\theta) = \omega_{\mathbf{F}}\mathcal{L}_{\mathbf{F}}(\theta) + \omega_{\mathbf{B}}\mathcal{L}_{\mathbf{B}}(\theta) + \omega_{\mathbf{d}}\mathcal{L}_{\mathbf{data}}(\theta). \tag{3.9}$$

Los tres términos de la ecuación Ecuación 3.9 se refieren a los errores en la descripción de las condiciones iniciales y de borde, ambas incluídas en el factor $\mathcal{L}_{\mathbf{B}}$, a la pérdida respecto a la ecuación diferencial $\mathcal{L}_{\mathbf{F}}$ y a la validación de la solución $\mathcal{L}_{\mathbf{data}}$.

La formulación para $\mathcal{L}_{\mathbf{F}}$ viene dada como:

$$\mathcal{L}_{\mathbf{F}} = \int_\Omega \left[\mathbf{F}(\hat{u}_\theta) - \mathbf{f}(z_i)\right]^2 dz \tag{3.10}$$

La definición dada en la Ecuación 3.10 no sólo es útil para el estudio teórico, si no que también se encuentra implementado en una paquete de PINN (NVIDIA Modulus[5]), permitiendo estrategias de integración más complejas tales como el muestreo con frecuencias más altas en áreas específicas del dominio para encontrar mejores resultados. Debemos tener en cuenta, que si se utiliza una PINN como metodología supervisada, los parámetros de la red neuronal se escogen a través de minimizar la diferencia entres los valores de salida y las predicciones del modelo; en el caso contrario, únicamente las diferencias residuales de la PDE son tomadas en cuenta.

El término, $\mathcal{L}_{\mathbf{F}}$, representa la pérdida producida por la discordancia con la ecuación diferencial del problema $\mathbf{F}$. Ésta forza la ecuación diferencial a los puntos de colocación, los cuales pueden ser elegidos uniformemente o distribuidos aleatoriamente sobre el dominio Ω. Las dos pérdidas

[5]Modulo de código abierto para la construcción, entrenamiento y ajuste de parámetros físicos en el modelo con una interfaz en Python.

restantes se utilizan para poder interpolar la ANN sobre los datos conocidos. La pérdida causada por la discordancia con la data (es decir, el valor de u) es denotado como $\mathcal{L}_{\mathbf{data}}$. Éste último fuerza $\hat{u}_\theta$ para empatar con las mediciones obtenidas de u sobre los puntos (z, u^*), los cuales pueden estar dados como datos sintéticos o mediciones. El término de peso ω_d toma en cuenta la calidad de las medidas. El último término es la pérdida debido al desempate con las condiciones iniciales y de frontera, $\mathbf{B}(\hat{u}_\theta) = \mathbf{g}$ de la Ecuación 3.1.

El problema resultante es entonces un problema de optimización. Una implementación genérica de la función de pérdida considera la formulación del error cuadrático medio[6]:

$$\mathcal{L}(\theta) = \omega_{\mathbf{F}} \frac{1}{N_c} \sum_{i=1}^{N_c} ||r_\theta(u_i) - r_i||^2 + \omega_{\mathbf{B}} \frac{1}{N_B} \sum_{i=1}^{N_B} ||\mathbf{B}(\hat{u}_\theta(z)) - \mathbf{g}(z_i)||^2 + \omega_d \frac{1}{N_d} \sum_{i=1}^{N_d} ||\hat{u}_\theta(z_i) - u_i^*||^2. \tag{3.11}$$

para calcular el error en la aproximación de $u(t, \mathbf{x})$. La mayoría de las paqueterías de *machine learning*, incluyendo TensorFlow y PyTorch, tienen implementada la diferenciación automática. La DA permite a la PINN implementar cualquier requerimiento de PDE así como de condición de borde, sin tener que realizar ninguna discretización numérica. En el contexto de las redes neuronales y el aprendizaje automático, la diferenciación automática es una técnica fundamental para calcular derivadas de funciones. A diferencia de otros métodos que aproximadamente calculan derivadas, la diferenciación automática obtiene derivadas exactas al descomponer funciones en operaciones elementales y aplicar reglas de derivación. Es esta técnica la que permite que el algoritmo de retropropagación calcule gradientes de manera eficiente y precisa. La diferenciación automática es crucial para optimizar las redes neuronales, ya que proporciona la dirección y magnitud exactas para actualizar los pesos y minimizar el error durante el entrenamiento. Gracias a herramientas modernas de aprendizaje profundo, esta diferenciación se realiza de forma transparente para los usuarios, permitiéndoles centrarse en el diseño y la arquitectura de la red.

3.5. Constricciones suaves y duras

Las condiciones de borde e iniciales (BC/IC) forman parte de la Ecuación 3.11, en el segundo término. Como se ha mencionado, se busca minimizar $\mathcal{L}(\theta)$ de forma, que este término (y los otros términos) concurrir a cero. En general, estas condiciones pueden ser tratadas como constricciones suaves, es decir, como un término de penalización en la ecuación de pérdida (Ecuación 3.11) [45]. Las desventajas de esta técnica radican en dos puntos principalmente: no existe garantía de satisfacer la condición correctamente (es decir, igualar exactamente a cero debido a errores numéricos o de redondeo). Alternativamente, se puede tratar este tipo de condición por medio de una *constricción dura*, donde la metodología es distinta a la anterior, pues en ésta se elige una solución particular que únicamente satisface la ecuación en los puntos de borde $\partial\Omega$. El *ansatz* particular es de la forma [46]:

[6] El error cuadrático medio (MSE) se define como el promedio del cuadrado de las diferencias entre el valor aproximado y el real. Para un conjunto de datos de tamaño N, definimos $MSE = \frac{1}{N}\sum_{i=1}^{N}[y_i - f(x_i)]^2$; donde y_i es la variable objetivo y $f(x_i)$ es el valor predicho por el modelo en tal punto.

$$\hat{u}(z;\theta) = u^{\text{par}}(z;\theta) + D(z;\theta)\tilde{u}(z;\theta), \tag{3.12}$$

donde u^{par} es la solución particular en el borde, $\tilde{u}$ es la red que determina u^* en los puntos interiores del dominio y $D(z;\theta) = 0$ cuando $z \in [(\partial\Omega \times [0,T]) \cup (\Omega \times \{0\}))]$. De esta forma las condiciones de borde/iniciales se satisfacen completamente. En particular para este trabajo, se utiliza la segunda metodología.

3.6. Aspectos de convergencia

Finalmente, la validación para la teoría de PINNs recae en investigar la convergencia de la red neuronal $\hat{u}_\theta$ a la solución $u(z)$ del problema de la Ecuación 3.1. Para esto, consideremos una configuración de la ANN donde el total de los coeficientes del vector de parámetros θ es la cardinalidad de la red. Así, podemos considerar una clase que se compone de todas las redes neuronales que satisfacen tener n parámetros en su arquitectura:

$$\mathcal{H}_n = \{u_\theta : |\theta| = n\} \tag{3.13}$$

De esta forma, la capacidad de aprendizaje de una PINN, recae en la cardinalidad de los parámetros de la red. En este sentido, debemos medir cómo la secuencia de medidas de $\hat{u}_\theta$, converge a la solución del problema de la Ecuación 3.1. Un resultado reciente en esta dirección fue obtenido por De Ryck et al. [47], donde se prueba que la diferencia $\hat{u}_\theta(n) - u$ converge a cero cuando el tamaño de la red, con función de activación tanh, tiende a infinito, es decir:

$$\hat{u}_\theta(n) \to u, \qquad n \to \infty. \tag{3.14}$$

Con lo descrito anteriormente, en la práctica, la PINN únicamente requiere la elección de una clase y una función de activación dada una colección de puntos de colocación. Dado que la cantidad y calidad de los puntos de entrenamiento afectan $\mathcal{H}_n$, el objetivo es minimizar la pérdida encontrando $\hat{u}^*_\theta \in \mathcal{H}_n$, por medio de un proceso de optimización. Aún si la solución exacta $u \in \mathcal{H}_n$ y se defina un mínimo global, no hay garantía que la solución encontrada coincida con la solución analítica u. Aún así, un primer trabajo de este tema en PINN, Shin et al. en [48] muestran que la secuencia de mínimos $\hat{u}^*_\theta$ converge fuertemente a la solución de ecuaciones diferenciales parciales elípticas y parabólicas.

3.7. Problemas resueltos con metodología PINN

3.7.1. Ecuaciones diferenciales ordinarias (EDOs)

Las EDOs se pueden utilizar para simular complejos sistemas no lineales, que son difíciles de modelar utilizando modelos físicos [49]. Un sistema de EDOs puede escribirse como:

$$\frac{d\mathbf{u}(\mathbf{x},t)}{dt} = f(\mathbf{u}(\mathbf{x},t),t) \tag{3.15}$$

con condiciones iniciales $\mathbf{B}(\mathbf{u}(t)) = \mathbf{g}(t)$. Lai et al.[49], en 2021, utilizaron la metodología PINN para identificación estructural, es decir, determinar las propiedades físicas de un sistema. En este artículo, se verifica el funcionamiento de PINN con un sistema de 4 grados de libertad de bloques de masa y resortes, que además incluye una no-linealidad cúbica.

3.7.2. Ecuaciones diferenciales parciales (EDPs)

Las EDPs forman parte de una gran cantidad de modelos matemáticos que describen fenómenos físicos. Dichas ecuaciones usualmente se resuelven con diferentes métodos numéricos. Entre dichas estrategias destacan los métodos de diferencias finitas, elementos finitos, volumen finito, entre otros [50-52]. La estabilidad y convergencia de estas metodologías se han estudiado a profundidad, por lo que se tiene un marco de trabajo sólido para aproximar y resolver EDPs [53, 54].

3.7.2.1. EDPs estacionarias

En general, un problema estacionario[7] es de la forma:

$$\begin{aligned} \mathbf{F_s}(u(z)) &= f(z) \qquad z \in \Omega \\ \mathbf{B}(u(z)) &= 0 \qquad z \in \partial\Omega \end{aligned}$$

donde $\Omega \subset \mathbb{R}^d$ con d dimensiones y frontera $\partial\Omega$ [55].

En este sentido, Dwivedi y Srinivasan [56] en 2020, resolvieron distintos problemas estacionarios en una y dos dimensiones. En [57], Ramabathiran y Ramachandran, resolvieron PDEs elípticas, tales como la ecuación de Poisson, tanto con bordes regulares como irregulares. La ecuación de Laplace-Beltrami se solucionó en superficies tridimensionales (3D) de geometrías complejas. Este artículo además provee una discusión acerca del tamaño de la muestra, la estructura de la PINN y la convergencia de la solución (Fang, Zhan [58]).

[7] Las ecuaciones diferenciales parciales estacionarias son aquellas en las que las soluciones no dependen del tiempo.

La metodología PINN también ha sido aplicada a ecuaciones Eikonales, es decir, EDPs hipérbolicas, que se pueden escribir en la forma:

$$||\nabla u(z)||^2 = \frac{1}{v^2(z)} \qquad z \in \Omega, \tag{3.16}$$

donde v es la velocidad y u es un tiempo de activación desconocido. Las ecuaciones Eikonales describen propagación de ondas, que puede tratar, en otros problemas, algunos de ondas sismícas [59] y de activación cardiaca [60].

A través de la implementación de EikoNet [61], Smith et al. [62], incorporaron una PINN para resolver una ecuación Eikonal 3D con la finalidad de encontrar el tiempo de vuelo en estructuras heterogéneas tridimensionales. Dado que el modelo propuesto únicamente es válido para una velocidad fija, realizar un cambio mínimo en la velocidad requiere un reentrenamiento. EikoNet esencialmente predice el tiempo que se requiere para ir de punto (fuente) a otro (receptor), por lo que tiene un amplio rango de aplicaciones.

3.7.2.2. EDPs no estacionarias

Las ecuaciones diferenciales no estacionarias describen la evolución en el tiempo. Las PINNs han probado ser también fiables también para este tipo de problemas.

3.7.2.2.1. Difusión Para un material compuesto, Amini Niaki et al. [63] estudiaron la ecuación que modela la transferencia de calor está dada por:

$$\frac{\partial T}{\partial t} = a\frac{\partial^2 T}{\partial x^2} + b\frac{d\alpha}{dt}, \tag{3.17}$$

donde a, b son parámetros y el segundo término $\alpha \in (0, 1)$ es debido al grado de curado[8] de la reacción.

En [63], proponen el uso de PINN con dos subredes desconectadas y el uso de un algoritmo de entrenamiento secuencial que automáticamente adapta los parámetros iniciales, incrementando la convergencia del modelo. En éste demuestran que la PINN predice correctamente la temperatura máxima, es decir, la exoterma que ocurre en el material compuesto debido al calor interno.

[8] El grado de curado es referido al nivel de reacción química que ha ocurrido durante el proceso de curado. El curado es un proceso utilizado para endurecer o solidificar un material a través de una reacción química. En general, este tipo de reacciones son exotérmicas, es decir, liberan calor.

3.7.2.3. Otros problemas

Otros problemas han sido resueltos en diferentes temas, tales como física cuántica [64-67] o medicina [68-70]. Respecto a los trabajos en física cuántica, se centran en el entrenamiento de redes con datos generados a través de una simulación con la finalidad de predecir los estados cuánticos futuros. Se ha demostrado que es posible poderlos obtener con alta precisión [66, 67]. En medicina, en [68], Raissi et al. utilizaron PINNS para la predicción de la dinámica del COVID-19. Demostraron que la arquitectura es capaz de aprender las características fundamentales de la enfermedad y de predecir la evolución del padecimiento en distintos pacientes. En [70], Ham et al. lograron determinar satisfactoriamente la relación entre imágenes de tejidos y la rígidez de éstos. Los autores demostraron que su modelo puede inferir con precisión la rígidez de los tejidos en tiempo real.

Capítulo 4

Redes neuronales artificiales aplicadas a sistemas físicos

4.1. Solución numérica de ecuaciones diferenciales: Runge-Kutta vs PINNs

4.1.1. Mecanismos biestables

4.1.1.1. Introducción

Los dispositivos mecánicos biestables son útiles para crear relevadores, válvulas, circuitos, celdas de memorias; por mencionar algunas de sus aplicaciones. Una de las mayores ventajas de dichos mecanismos es que pueden aplicar fuerza de contacto sin la necesidad de fuente de poder continua. Una aplicación reciente de estos mecanismos biestables se encuentra en los sistemas microelectromecánicos (MEMs), donde se emplean para ayudar a cuantificar la aceleración gravitacional. En esta sección se realizan dos tipos de resultados. Primero, se reproducen las soluciones de un mecanismo biestable descrito en [71], aunque a diferencia del artículo, donde realizan el cálculo a través de análisis de elementos finitos, en este trabajo se utilizan PINNs.

Fig. 4.1: Esquema de un dispositivo mecánico biestable. En color negro, la primer posición de equilibrio, que coincide con el primer eigenvalor de la ecuación diferencial. En color verde, la segunda posicion de equilibro y en color rojo, la tercera.

4.1.1.2. Modelo matemático de un dispositivo mecánico biestable

El diseño original de un dispositivo mecánico biestable tiene su inspiración en la viga fija de los extremos, donde la viga está axialmente fija de ambos extremos. La ecuación que describe a una viga fija sujeto a una carga axial p es:

$$EI\frac{d^4w}{dx^4} + p\frac{d^2w}{dx^2} = 0 \tag{4.1}$$

donde w es el desplazamiento lateral de la viga, E es el módulo de Young del material de la viga y el momento de inercia es I. Este problema está bien estudiado por [71], de donde se tienen los siguientes resultados.

Modelo de mecanismo biestable con momento de inercia constante I

$$EI\frac{d^4w}{dx^4} + p\frac{d^2w}{dx^2} = 0, \qquad \text{(Haz sujeto a fuerza axial)} \tag{4.2a}$$

$$w(0) = w(L) = 0, \qquad \text{(Condición de borde I)} \tag{4.2b}$$

$$w'(0) = w'(L) = 0, \qquad \text{(Condición de borde II)} \tag{4.2c}$$

$$N^2 = \frac{pl^2}{EI}, \qquad \text{(Normalización de fuerza axial)} \tag{4.2d}$$

$$\sin\left(\frac{N}{2}\right)\left[\tan\left(\frac{N}{2}\right) - \frac{N}{2}\right] = 0, \qquad \text{(Condición de solución no trivial)} \tag{4.2e}$$

$$w_j(x) = C[1-\cos\left(N_j\frac{x}{l}\right)], \qquad \text{(j = 1,3,5,...)} \tag{4.2f}$$

$$N_j = (j+1)\pi, \qquad \text{(j = 1,3,5,...)} \tag{4.2g}$$

$$w_j(x) = C[1 - 2\frac{x}{l} - \cos\left(N_j\frac{x}{l}\right) + \frac{2\sin\left(N_j\frac{x}{l}\right)}{N_j}], \qquad \text{(j = 2,4,6,...)} \tag{4.2h}$$

$$N_j = 2.86\pi, 4.92\pi, ... \qquad \text{(j = 2,4,6,...)} \tag{4.2i}$$

4.1.1.3. Comparación de soluciones numéricas: Mathematica vs DeepXDE

Utilizando los resultados teóricos del modelo de mecanismo biestable, tenemos la forma de realizar soluciones numéricas, dado que conocemos los valores N_j para los cuales la ecuación tiene solución

no trivial.

Para la comparación entre resultados numéricos y las gráficas teóricas con ayuda de la Ecuación 4.2, se han realizado dos soluciones por métodos distintos:

- Método de diferencias finitas, en Mathematica [72], en particular utilizando el método de Runge-Kutta.
- DeepXDE [73], una paquetería computacional desarrollada en Python, para la solución de ecuaciones diferenciales parciales.

4.1.1.3.1. Comparaciones numéricas: Mathematica vs PINN Para la comparación, entre las soluciones tradicionales (diferencias finitas) y los métodos con redes neuronales, se comparará las soluciones normalizadas por cada método. Primero, utilizando el software Mathematica, un lenguaje de programación multipropósito utilizado por científicos, ingenieros y estudiantes de la comunidad científica. La primer versión fue lanzada de 1988, y su última versión (13.0.1.0) lanzada en febrero de 2022, por lo que cuenta con más de 30 años de desarrollo como respaldo. Posteriormente, se utilizó DeepXDE, una librería desarrollada en Python para la solución de ecuaciones diferenciales parciales a través de redes neuronales.

Primero, para el desarrollo de las simulaciones en Mathematica, se utiliza una función bien desarrollada *NSolve* para la solución numérica de ecuaciones diferenciales por medio del método de elementos finitos. Una descripción breve del código utilizado para el desarrollo y extracción de los datos se muestra a continuación:

Snippet 4.1: Código Mathematica - Inercia constante

```
s = NDSolve[{( y''''[x] == -N^2 y''[x] , y[0] == 0, y[2] == 0,
   y'[0] == 0, y'[2] == 0}, y, {x, 0, 2}, StartingStepSize -> 0.001,
  Method -> {"FixedStep", Method -> "ExplicitRungeKutta"}]
```

Con N dado por los eigenvalores del conjunto de ecuaciones (4.2).

Para el desarrollo de las simulaciones en Python, a continuación se muestra parte del código:

Snippet 4.2: Código Python - Inercia constante

```
import deepxde as dde
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import pandas as pd
import math

dde.config.set_default_float('float64')

def pde(x, y):

    dy_xx = dde.grad.hessian(y, x)
    pi = tf.constant(math.pi, dtype = tf.float64)
    E = 1
```

```
    I = 1
    N = 2*pi
    L = 2
    p = (N/L)**2
    A = E*I*dy_xx

    return dde.grad.hessian(A, x) + p*dy_xx

pi = tf.constant(math.pi)

def boundary_l(x, on_boundary):
    return on_boundary and np.isclose(x[0], 0)

rightSize = 2
def boundary_r(x, on_boundary):
    return on_boundary and np.isclose(x[0], rightSize)

def func_right(x):
    return tf.constant((1/x)**200.0)

def func_left(x):
    return tf.constant(x**200.0)

geom = dde.geometry.Interval(0, rightSize)
bc_l_1 = dde.icbc.DirichletBC(geom, lambda x: 0, boundary_l)
bc_l_2 = dde.icbc.NeumannBC(geom, lambda x: 0, boundary_l)
bc_r_1 = dde.icbc.DirichletBC(geom, lambda x: 0, boundary_r)
bc_r_2 = dde.icbc.NeumannBC(geom, lambda x: 0, boundary_r)
data = dde.data.PDE(geom, pde, [bc_l_1, bc_l_2, bc_r_1, bc_r_2],\
510, 100, solution=None, num_test=200)

layer_size = [1] + [60] * 4 + [1]
activation = "tanh"
initializer = "Glorot␣uniform"
net = dde.nn.FNN(layer_size, activation, initializer)

net.apply_output_transform(lambda x, y: x*(x-2.0)*(x-2.0)*y)

model = dde.Model(data, net)
model.compile("L-BFGS")
losshistory, train_state = model.train()

dde.saveplot(losshistory, train_state, issave=True, isplot=True)
```

Las soluciones $w_j(x)$ de la Ecuación 4.2 están bien definidas salvo un factor multiplicativo constante, indicando que las soluciones no están normalizadas. Para realizar la comparación se han normalizado los datos de ambas simulaciones a través de la siguiente fórmula:

$$w_{norm} = \frac{(w - w_{min})}{(w_{max} - w_{min})} \tag{4.3}$$

De esta forma podemos comparar las soluciones. Se generaron tres gráficas que se muestran a continuación con los primeros tres eigenvalores $N_{1,2,3} = 2\pi, 2.86\pi, 4\pi$. En cada gráfica observamos que el número de eigenvalor corresponde con el número de posiciones de equilibrio de la viga.

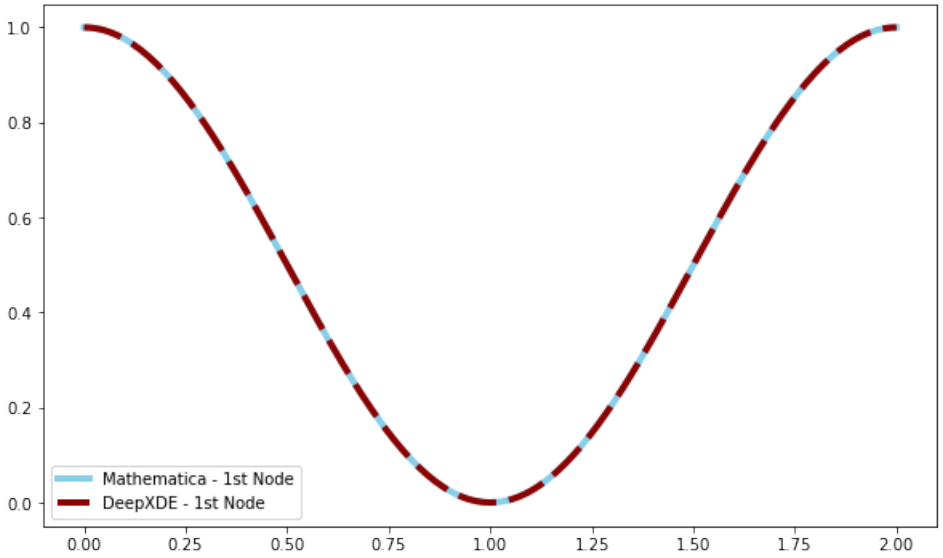

Fig. 4.2: Comparación entre solución numérica entre Mathematica y DeepXDE para el primer eigenvalor, $N_1 = 2\pi$.

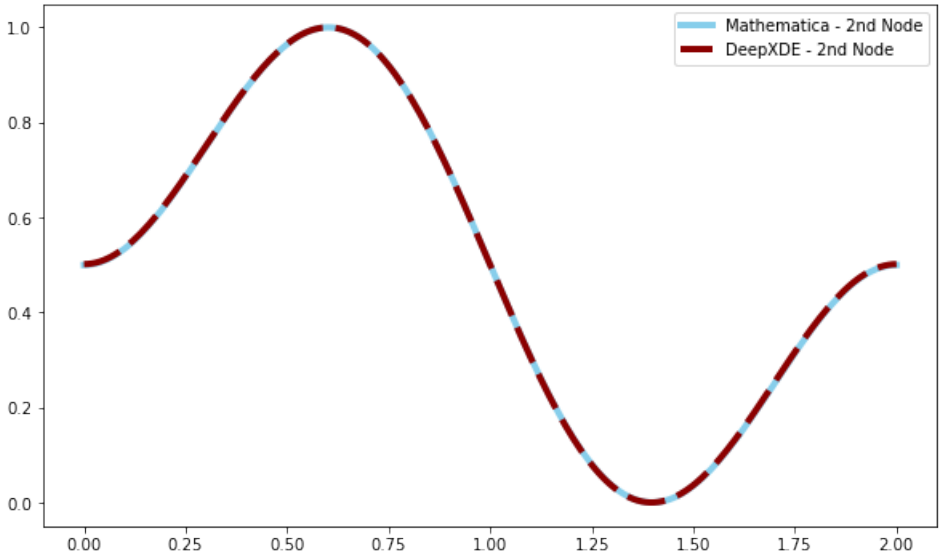

Fig. 4.3: Comparación entre solución numérica entre Mathematica y DeepXDE para segundo eigenvalor, $N_2 = 2.86\pi$

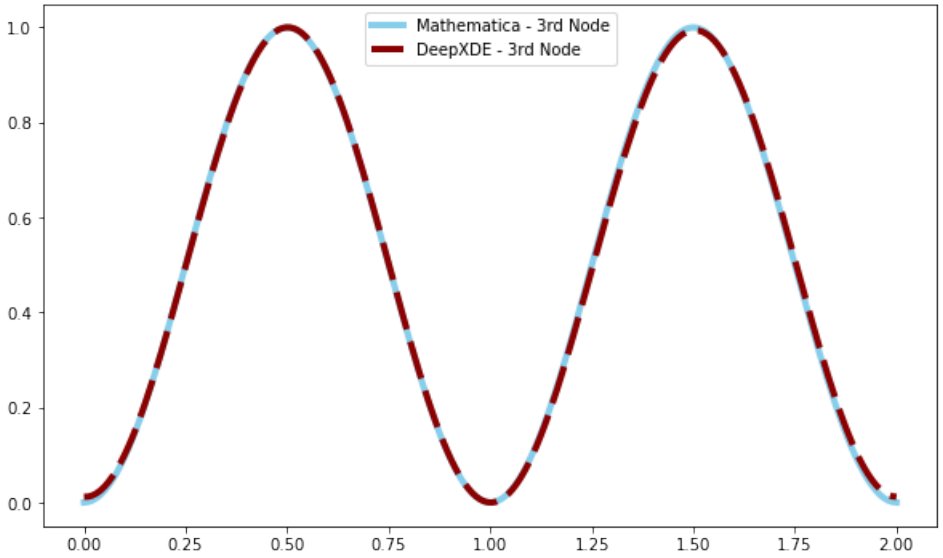

Fig. 4.4: Comparación entre solución numérica entre Mathematica y DeepXDE para el tercer eigenvalor, $N_3 = 4\pi$

4.1.2. Mecanismos ópticos biestables con momento de inercia variable

En la óptica moderna, los mecanismos biestables han ganado una gran atención debido a su capacidad para existir en uno de los dos estados estables sin ninguna entrada externa. Estos mecanismos, que son fundamentales en muchas aplicaciones tecnológicas, tienen el potencial de revolucionar la forma en que diseñamos y operamos los dispositivos ópticos. Sin embargo, se añade un nuevo grado de complejidad y fascinación cuando se introduce el concepto de momento de inercia variable. A continuación exploramos cómo cambian las soluciones cuando el momento de inercia no es constante a lo largo de la viga.

4.1.2.1. Modelo matemático con momento de inercia variable

4.1.2.1.1. Solución analítica Cuando el momento de inercia es variable, es decir, $I = I(x)$, tenemos la ecuación:

$$\frac{d^2}{dx^2}\left[EI(x)u''(x) + pu(x)\right] = 0 \tag{4.4}$$

Integrando un par de veces, tenemos:

$$EI(x)u''(x) + pu(x) = c_3x + c_4 \tag{4.5}$$

Aquí consideraremos el caso particular $I(x) = \frac{I_0}{1+x}$ con $x \in [0, 2]$. Sustituyendo en la ecuación, obtenemos:

$$\frac{EI_0}{1+x}u''(x) + pu(x) = c_4 x + c_3 \tag{4.6}$$

Por lo que, como solución particular podemos proponer $u_p(x) = c_4 x + c_3$. Por otro lado, para la solución homogénea se tiene:

$$\frac{EI_0}{1+x}u''(x) + pu(x) = 0 \tag{4.7}$$

Y reescribiendo, tenemos:

$$u''(x) + \left(\frac{p}{EI_0}\right)(1+x)u(x) = 0 \tag{4.8}$$

Eligiendo un parámetro de la ecuación diferencial como $-\lambda^3 = \left(\frac{p}{EI_0}\right)$, con lo que tenemos la ecuación:

$$u''(x) - \lambda^3(1+x)u(x) = 0 \tag{4.9}$$

De donde se obtienen soluciones de la forma:

$$u(x) = c_1 A_i(\lambda(1+x)) + c_2 B_i(\lambda(1+x)) + c_3 x + c_4 \tag{4.10}$$

Donde A_i y B_i son las dos soluciones independientes de la ecuación diferencial de Airy. Considerando, al igual que en el caso de inercia constante, las condiciones de borde $u(0) = u(2) = 0$ y $u'(0) = u'(L = 2) = 0$. Por lo que, de forma matricial encontramos las siguientes ecuaciones lineales independientes:

$$\begin{pmatrix} A_i(\lambda) & B_i(\lambda) & 0 & 1 \\ A_i(3\lambda) & B_i(3\lambda) & 2 & 1 \\ A_i'(\lambda) & B_i'(\lambda) & 1 & 0 \\ A_i'(3\lambda) & B_i'(3\lambda) & 1 & 0 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \vec{0} \tag{4.11}$$

4.1.2.1.2. Obtención de eigenvalores de la ecuación diferencial Para encontrar los eigenvalores de la ecuación diferencial, es decir, los valores λ para los cuales obtenemos una solución no trivial. Así entonces, buscamos que el determinante de la matriz en la Ecuación 4.11) se anule:

$$\begin{vmatrix} A_i(\lambda) & B_i(\lambda) & 0 & 1 \\ A_i(3\lambda) & B_i(3\lambda) & 2 & 1 \\ \lambda A_i'(\lambda) & \lambda B_i'(\lambda) & 1 & 0 \\ \lambda A_i'(3\lambda) & \lambda B_i'(3\lambda) & 1 & 0 \end{vmatrix} = 0 \tag{4.12}$$

Con lo que obtenemos:

$$\begin{aligned} & A_i(\lambda)[\lambda B_i'(\lambda) - \lambda B_i'(3\lambda)] \\ & -A_i'(\lambda)[\lambda B_i'(\lambda) - \lambda B_i'(3\lambda)] \\ & +A_i(\lambda)[-B_i(\lambda) + B_i(3\lambda) - 2\lambda B_i'(3\lambda)] \\ & -\lambda A_i'(3\lambda)[-B_i(\lambda) + B_i(3\lambda) - 2\lambda B_i'(3\lambda)] = 0 \end{aligned} \tag{4.13}$$

A continuación se muestra una gráfica de la Ecuación 4.13. Los ceros (puntos en los que corta el eje de las abscisas) de la función coinciden con los eigenvalores para los cuales la Ecuación 4.10 tiene solución no trivial.

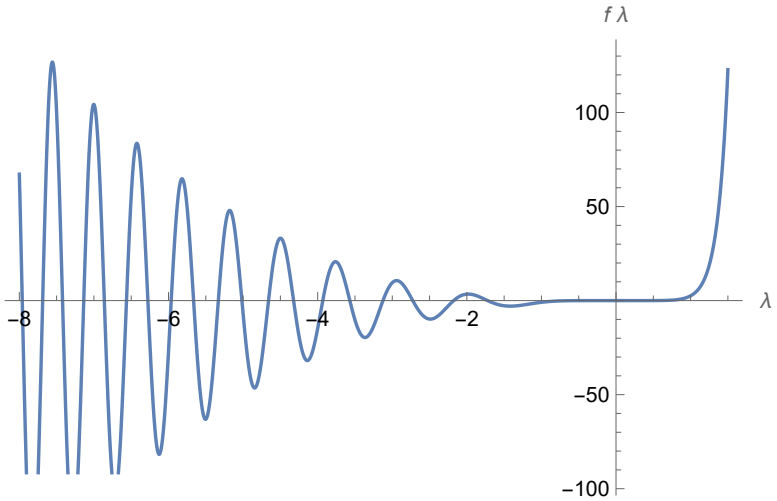

Fig. 4.5: Gráfica de la ecuación (4.13) en función de λ. Los ceros de la función corresponden a los eigenvalores de la ecuación diferencial.

4.1.2.2. Simulaciones numéricas: Mathematica vs PINN

Con la finalidad de evaluar y comparar el método de PINN con la solución obtenida por métodos tradicionales, en este caso, diferencias finitas, se generan soluciones a la ecuación diferencial en Mathematica y se compara contra las obtenidas por PINN.

Un pequeño snippet del código en Mathematica para realizar las simulaciones se muestra a continuación:

Snippet 4.3: Código Mathematica - Inercia Variable

```
s2 = NDSolve[{(1 + x)^2  y''''[x] - 2 (1 + x) y'''[x] +
     2  y''[x] == (lambda)^3 (1 + x)^3 y''[x] , y[0] == 0, y[2] == 0,
   y'[0] == 0, y'[2] == 0}, y, {x, 0, 2}, StartingStepSize -> 0.01,
  Method -> {"FixedStep", Method -> "ExplicitRungeKutta"}]

data = Table[Flatten[{x, Evaluate[{y[x]} /. s2]}], {x, 0, 2, 0.001}];

Export["Data_ODE_InertiaVariable_NthMode.csv", data];
```

De igual forma, una muestra del código utilizado en Python con el mismo objetivo está a continuación:

Snippet 4.4: Código Python - Inercia Variable

```
import deepxde as dde
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import pandas as pd
import math

def PlottingEigen(eigenvalue):
    dde.config.set_default_float('float64')

    def pde(x, y):

        dy_xx = dde.grad.hessian(y, x)
        #pi = tf.constant(math.pi, dtype = tf.float64)
        I = 1/(1+x)
        #eigenvalue = -1.72104490421 #First mode
        #eigenvalue = -2.17701849660 #Second mode
        #eigenvalue = -2.72517668085 #Third mode
        #eigenvalue = -3.12450566869660
        A = I*dy_xx

        return dde.grad.hessian(A, x) - (eigenvalue**3)*dy_xx

    pi = tf.constant(math.pi)

    def boundary_l(x, on_boundary):
        return on_boundary and np.isclose(x[0], 0)

    rightSize = 2
    def boundary_r(x, on_boundary):
        return on_boundary and np.isclose(x[0], rightSize)

    def func_right(x):
        return tf.constant((1/x)**200.0)

    def func_left(x):
        return tf.constant(x**200.0)

    geom = dde.geometry.Interval(0, rightSize)
    bc_l_1 = dde.icbc.DirichletBC(geom, lambda x: 0, boundary_l)
    bc_l_2 = dde.icbc.NeumannBC(geom, lambda x: 0, boundary_l)
    bc_r_1 = dde.icbc.DirichletBC(geom, lambda x: 0, boundary_r)
    bc_r_2 = dde.icbc.NeumannBC(geom, lambda x: 0, boundary_r)
```

```
numberPoints = 62
data = dde.data.PDE(geom, pde, [bc_l_1, bc_l_2, bc_r_1, bc_r_2],\
numberPoints, 4, solution=None, num_test=120)

layer_size = [1] + [60] * 3 + [1]
activation = "sigmoid"
initializer = "Glorot uniform"
net = dde.nn.FNN(layer_size, activation, initializer)

net.apply_output_transform(lambda x, y: y*10)
net.apply_output_transform(lambda x, y: x*(x-2.0)*(x-2.0)*y)
#net.apply_output_transform(lambda x, y: y * 10**2)

model = dde.Model(data, net)#, lr=0.001)
#model.compile("adam", lr=0.001)
#model.train(epochs=5000)
model.compile("L-BFGS")
#model.train_step.optimizer_kwargs = {'options': \
{'maxfun': 1e5, 'ftol': 1e-20, 'gtol': 1e-20, 'eps': 1e-20,\
'iprint': -1, 'maxiter': 1e5}}
losshistory, train_state = model.train()

dde.saveplot(losshistory, train_state,issave = False, isplot = False)

x = geom.uniform_points(100, True)
#x = geom.random_points(512,'Sobol')
y = model.predict(x, operator=None)

#DataDF['x'] = x
#DataDF['y'] = y
#DataDF.to_csv('Data_3rd_Mode.csv')
plt.plot(x, y, label = str(eigenvalue))
plt.xlabel("x")
plt.ylabel("PDE residual")
#plt.show()
tf.compat.v1.reset_default_graph()
```

Nuevamente, como en el caso anterior, podemos normalizar las soluciones por medio de la Ecuación 4.3 para realizar la comparación. Para los primeros dos eigenvalores, soluciones de la Ecuación 4.13, tenemos $\lambda_1 = -1.72104490421$ y $\lambda_2 = -2.17701849660$. A continuación, se muestran las dos gráficas comparativas entre ambos métodos. **Nota: El parámetro λ está definido como $-\lambda^3 = (\frac{p}{EI_0})$** (Figura 4.6 y Figura 4.7).

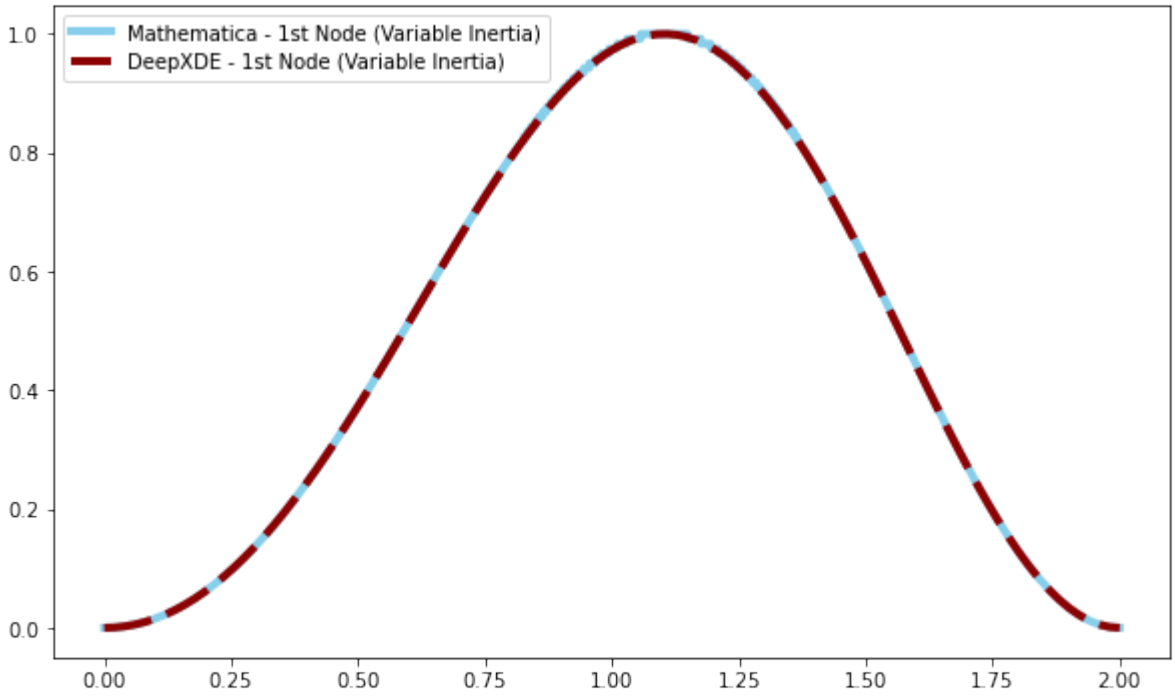

Fig. 4.6: Gráfica de la deformación del mecanismo biestable para el primer eigenvalor, $\lambda_1 = -1.72104490421$. Se muestran las soluciones normalizadas por dos métodos: diferencias finitas realizado en Mathematica, y la metodología PINN realizada en Python.

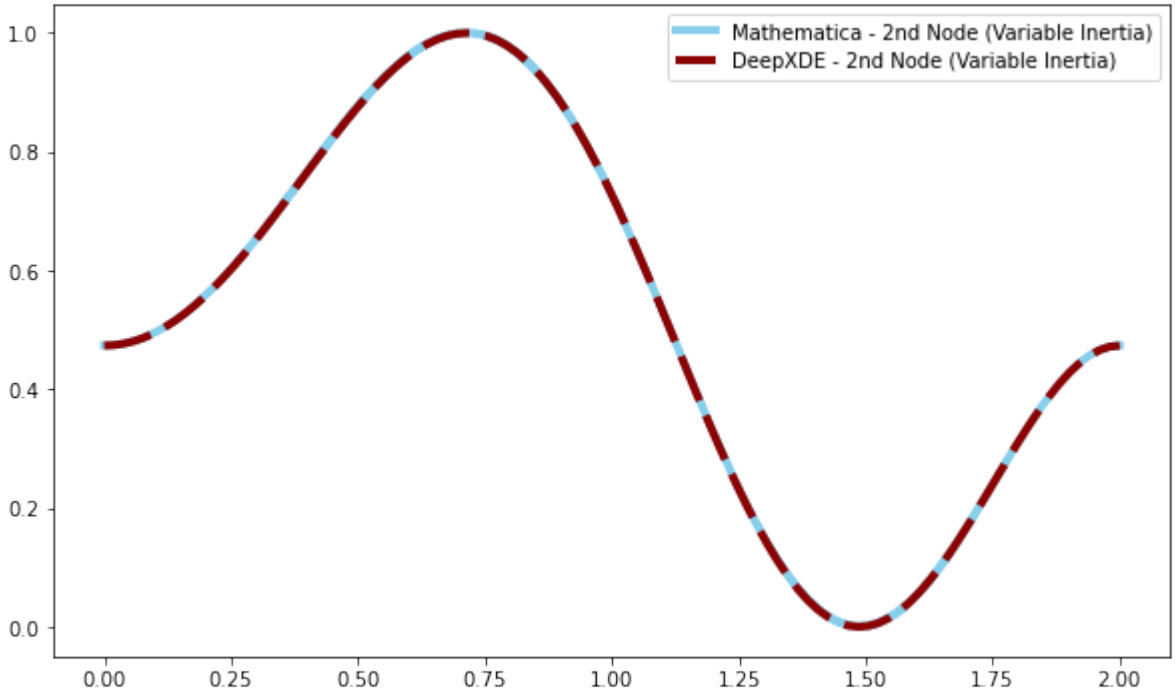

Fig. 4.7: Gráfica de la deformación del mecanismo biestable para el segundo eigenvalor, $\lambda_2 = -2.17701849660$. Se muestran las soluciones normalizadas por dos métodos: diferencias finitas realizado en Mathematica, y la metodología PINN realizada en Python.

4.2. Efecto túnel cuántico: Aprendizaje de las redes neuronales

4.2.1. Efecto túnel: solución analítica

Para resolver este problema, consideremos tres regiones en las que tendremos la función de onda, la primer región (de la izquierda) de donde viene la onda incidente, la región 2 es aquella donde se encuentra localizada la barrera de potencial y la región 3, una vez que la función de onda atravesó la barrera de potencial.

Para una partícula cuántica propagándose de izquierda a derecha con energía $E < V_0$, la solución para la función de onda en las regiones 1 y 2 contienen términos de propagación en ambos sentidos (la función de onda incidente y la que choca y regresa), mientras que en la tercera región sólo tenemos término hacia la derecha. Véase el esquema de la Figura 4.8.

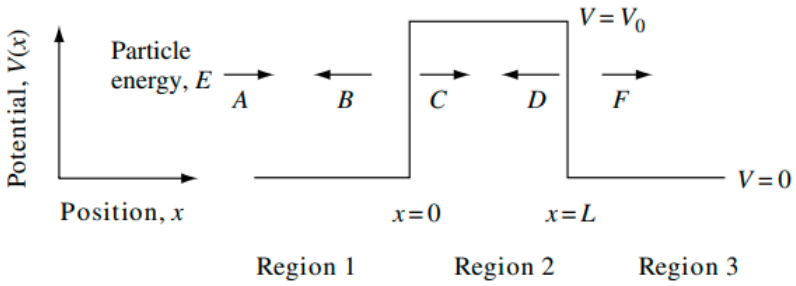

Fig. 4.8: Ilustración de la forma de la función de onda en las distintas regiones. Esquema tomado de [74].

En la región 1, la función de onda tiene la forma:

$$\psi_1(x) = Ae^{ikx} + Be^{-ikx}, \tag{4.14}$$

mientras que en la región 2, tenemos:

$$\psi_2(x) = Ce^{k_b x} + De^{-k_b x}, \tag{4.15}$$

donde $k = \sqrt{2mE}/\hbar$ y $k_b = \sqrt{2m(V_0 - E)}/\hbar$. Observemos que en la región 3, tenemos sólo función de onda hacia la derecha, por lo que $\psi_3(x) = Fe^{ikx}$.

La ecuación de Schrodinger para barreras finitas exige que tanto la función de onda como su derivada deben ser continuas en las fronteras, i.e, en $x = 0$ y $x = L$. Con esto obtenemos cuatro ecuaciones:

$$A + B = C + D \tag{4.16}$$

$$(A - B) = \frac{ik_b}{k}(D - C) \tag{4.17}$$

$$Ce^{k_b L} + De^{-k_b L} = Fe^{ikL} \tag{4.18}$$

$$Ce^{k_b L} - De^{-k_b L} = \frac{ik}{k_b} Fe^{ikL} \tag{4.19}$$

Dado que la probabilidad de transmisión[1] está dada por $|F/A|^2$, basta realizar la manipulación de las ecuaciones arriba listadas para obtener:

$$T = \left|\frac{F}{A}\right|^2 = \frac{1}{1 + \left[(k^2 + k_b^2)/(2kk_b)\right] \sinh^2(k_b L)}. \tag{4.20}$$

4.2.2. Coeficientes de transmisión y reflexión: simulaciones en Mathematica

En esta sección, se describe el algoritmo para la obtención de los coeficientes de transmisión y reflexión, así como un *snippet* del código utilizado en Mathematica para la integración de la ecuación.

Primero, analicemos las soluciones numéricas obtenidas para tres energías distintas para un mismo potencial.

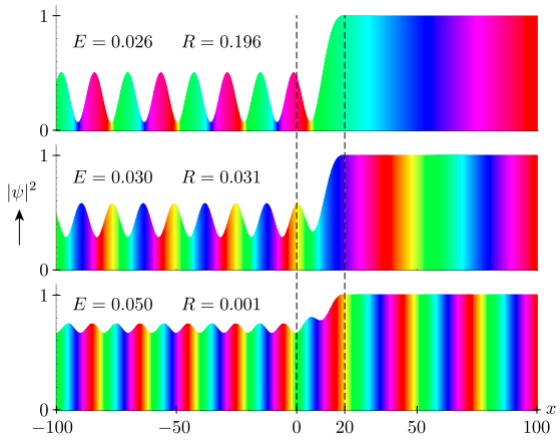

Fig. 4.9: Soluciones numéricas para tres distintas energías, considerando el mismo potencial. Las líneas punteadas marcan la posición del potencial rectangular. Observemos que a menor energía $E < V_0$, el coeficiente de reflexión aumenta, además podemos ver que los patrones de interferencia aumenta cuando aumenta el coeficiente R. Esquema tomado de [74].

[1] El coeficiente de transmisión, T, se define como la relación entre el flujo o densidad de corriente de la onda transmitida y el flujo de la onda incidente. Por otro lado, el coeficiente de reflexión, R, es la probabilidad de que la partícula sea reflejada por una barrera o escalón de potencial.

Veamos que del lado izquierdo de la barrera, tenemos una similitud con patrones de interferencia, pues la función de onda del lado izquierdo de la barrera muestra interferencia debido a la onda incidente y a la reflejada al chocar contra la barrera. A continuación, se muestra una forma de obtener el coeficiente de reflexión utilizando estos máximos y mínimos.

Como se ha descrito anteriormente, del lado izquierdo tenemos una función de onda con la forma de la Ecuación 4.14. De esta forma, la densidad de probabilidad, en el lado izquierdo, viene dada por:

$$|\psi_1(x)|^2 = \left(A^* e^{-ikx} + B^* e^{ikx}\right)\left(Ae^{ikx} + Be^{-ikx}\right), \tag{4.21}$$

dado que estos coeficientes A y B son complejos, por lo que son de la forma:

$$A = |A|e^{i\alpha}, \quad B = |B|e^{i\beta}, \tag{4.22}$$

por lo que sustituyendo en la Ecuación 4.21 obtenemos:

$$|\psi_1(x)|^2 = |A|^2 + |B|^2 + 2|A||B|\cos(2kx + \alpha - \beta). \tag{4.23}$$

Observemos que el máximo y mínimo de la densidad de probabilidad coincide cuando el coseno vale +1, o -1, respectivamente. Por lo tanto, tenemos:

$$|\psi_1|_{max} = \sqrt{|A|^2 + |B|^2 + 2|A||B|} = |A| + |B| \tag{4.24}$$

$$|\psi_1|_{min} = \sqrt{|A|^2 + |B|^2 - 2|A||B|} = |A| - |B|, \tag{4.25}$$

y finalmente obtenemos una relación para el coeficiente de reflexión como:

$$R = \frac{|B|^2}{|A|^2} = \left[\frac{|\psi_1|_{max} - |\psi_1|_{min}}{|\psi_1|_{max} + |\psi_1|_{min}}\right]^2 \tag{4.26}$$

Nota: Los coeficientes de reflexión y transmisión mantienen la relación $T + R = 1$.

4.2.3. Algoritmo de obtención de coeficientes de transmisión y reflexión

Para la obtención de los coeficientes de transmisión y reflexión para este problema, se ha utilizado Mathematica para realizar las simulaciones, a continuación se muestra un *snippet* de código utilizado:

Snippet 4.5: Obtención de coeficientes de transmisión y reflexión

```
Clear["Global'*"];

v[x_,h1_,h2_,h3_,w1_,w2_,w3_] := If[x >= -w1/2 && x <= w1/2, h1, 0] +
If[x > w1/2 && x <= w2 + (w1/2), h2, 0]\
+   If[x > w2 + (w1/2) && x <= w3 + w2 + (w1/2), h3, 0];

RefCoeff[h1_,h2_,h3_,w1_,w2_,w3_,energy_]:= Block[{xMax = 5,\
k = Sqrt[2 energy]}, solution = NDSolveValue[{psi''[x] =\
-2(energy - v[x,h1,h2,h3,w1,w2,w3]) psi[x],
psi[xMax] == 1, psi'[xMax] == I*k}, psi, {x, -xMax, xMax}];
max = FindMaximum[Abs[solution[x]], {x, -0.8 xMax, -0.1 xMax}];
min = FindMinimum[Abs[solution[x]], {x, -0.8 xMax, -0.1 xMax}];
Ref = ((max - min)/(max + min))^2;
Ref[[1]]];

args = {}
For[i = 1, i <= 100 000, i++, AppendTo[args, {RandomReal[{6,15}],\
RandomReal[{6,30}],RandomReal[{6,10}],RandomReal[{0.05,0.25}],
RandomReal[{0.05,0.25}],RandomReal[{0.05,0.25}],RandomReal[{1,5}]}]];

Data = Table[{a[[1]],a[[2]],a[[3]],a[[4]],a[[5]],a[[6]],a[[7]],
RefCoeff[a[[1]],a[[2]],a[[3]],a[[4]],a[[5]],a[[6]],a[[7]]]},{a,args}]\
// TableForm;

Export["AlturaGrosor_E_variable_tripleBarrier_100k.csv",Data];
```

Observemos que finalmente, lo que se obtiene a través del código son archivos con formato *csv* que guardan el conjunto de datos para la creación de las simulaciones, así como los resultados para el coeficiente de transmisión, es decir, duplas de entrada y salida para una ANN.

4.2.4. Red neuronal en Python

4.2.4.1. Preprocesamiento de los datos

En la Subsección 4.2.3 se ha descrito el proceso de obtención de las bases de datos para la construcción del modelo, finalmente lo que se obtienen son archivos de formato *csv* que contienen los potenciales y anchos de las barreras de potencial, así como el coeficiente de reflexión para cada una de las simulaciones (100k en cada simulación). Este proceso se ha realizado para una, dos y tres barreras. Al final, se han concatenado los datos de cada una de las simulaciones en un solo archivo para entrenar al modelo.

Posteriormente, se ha divido la muestra en dos (de un total de 300k), la mitad para entrenar el modelo, y la otra para probarlo.

4.2.4.2. Estructura de la red

La estructura de la red neuronal consiste en una capa de entrada, ocho capas intermedias y una capa de salida. La capa inicial consiste de 7 neuronas, y una capacidad para 7 datos iniciales, con una función de activación 'relu'. Las capas intermedias consisten de no más de 24 neuronas, con

una combinación de funciones de activación entre 'relu' y 'sigmoid'. Además, como optimizador se utilizó 'Adam' con una tasa de aprendizaje del 0.001. Se realizaron 50 iteraciones de aprendizaje (epochs) entre los datos de entrada y salida (se han compararado las entradas, descritas arriba, contra el coeficiente de transmisión T).

4.2.5. Comparación entre soluciones: diferencias finitas (Mathematica) vs ANN

Utilizando la red descrita en subsección anterior, se evaluó el modelo usando los datos de prueba, luego se ha calculado la diferencia porcentual entre el resultado de la simulación obtenida en mathematica y lo obtenido por el modelo en Tensor Flow.

Se ha obtenido que el 25 % de los datos (150k total) tiene un error menor al 2.14 %, el 50 % menor al 5.54 %, y el 75 % menor al 13.44 %. En cuestión de tiempo computacional, el modelo ANN evalúa mucho más rápido en un orden de 10^4.

4.3. Problemas inversos de eigenvalores

4.3.1. Introducción

El problema de eigenvalores inverso consiste en la reconstrucción de una matriz a través de su espectro[2]. El espectro de eigenvalores puede consistir de todos o sólo de una parte de los eigenvalores. El objetivo principal de una problema inverso de eigenvalores consiste en construir una matriz que mantenga cierta estructura. Existen dos preguntas principales asociadas a un problema de eigenvalores inverso: la solvencia teórica del problema en cuestión así como la capacidad de los recursos computacionales. En la búsqueda de una solución teórica del problema, se deben determinar las condiciones mínimas para el cual un espectro tiene solución, así como el desarrollo de un mecanismo de reconstrucción numérico de la matriz [75].

Un conjunto amplio de aplicaciones forma parte de la solución de este tipo de problemas, entre éstos destacan los sistemas de control, de identificación, tomografía sísmica, análisis de componentes principales, geofísica, espectroscopía, análisis estructural, entre otros. En particular para este trabajo, se estudia el caso del problema de eigenvalores inversos para matrices tridiagonales, aún más, consideramos que es una matriz simétrica, a la cual se le llama Matriz de Jacobi (Un ejemplo se encuentra más adelante en la Ecuación 4.27 para el caso finito) [76].

Una metodología común que tienen este conjunto de aplicaciones es la reconstrucción del sistema físico, i.e. encontrar sus parámetros físicos, a través del conocimiento de la dinámica del sistema. La dinámica del sistema puede consistir en múltiples conceptos, por ejemplo, un problema de vibraciones depende de las frecuencias naturales y los modos normales de un sistema, o que en un sistema de control, la estabilidad depende de los eigenvalores [3] [77].

[2]En matemáticas, el espectro de una matriz consiste en el conjunto de eigenvalores de una matriz.
[3]En particular, en un sistema de control, el signo de cada eigenvalor determina la estabilidad del sistema en

En particular para este trabajo, proponemos una aproximación a la solución del problema siguiente:

Problema 4.3.1 *Dado un conjunto de escalares* $\Omega \in \mathrm{R}$*, encuentre una matriz* $\mathbf{M} \in \mathcal{N}$ *que satisface* $\omega(\mathbf{M}) = \Omega$*, donde* $\mathcal{N}$ *es el conjunto de matrices de Jacobi.*

La historia del estudio de este tipo de problemas no es reciente, y podemos remontarnos a los primeros trabajos en el tema realizados por Hochstadt [78]. Posteriormente, G. Guseinov y Gasymov [79] desarrollaron el estudio del análisis espectral para matrices de Jacobi infinitas, matrices de Jacobi no-autoadjuntas desde la perpesctiva de la función espectral generalizada. Guseinov ha estudiado también el problema asociado a uno o dos espectros, desde 2010 hasta 2013. Durante 2009 estudió además el problema de espectro inverso para matrices $N \times N$ para matrices simétricas tridiagonales, problema estudiado en el desarrollo de este trabajo [80]. Desde una perspectiva física, Bairamov ha estudiado el espectro y dispersión para la ecuación de Schrödinger discreta, así como para sistemas de Dirac [81]. Durante el mismo lapso, Huseynov estudia el problema de dispersión inversa en sistemas de Dirac [82]. Finalmente, Manafov et. al. desde el 2016 y hasta el 2018, realizaron el estudio de la igualdad de Parseval para ecuaciones de Sturm-Liouville discretas, así como para problemas de espectro inverso para la ecuación de Sturm-Liouville dependiente de la energía con interacción delta [83].

4.3.2. Metodología a través de redes neuronales

Un método de aproximación del Hamiltoniano referido a un espectro de eigenvalores se propone, en esta sección, a través del uso de redes neuronales artificiales. En esta sección construimos la matriz tridiagonal $n \times n$:

$$M = \begin{bmatrix} b_1 & a_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_1 & b_2 & a_2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a_2 & b_3 & a_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_3 & b_4 & a_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_4 & ... & ... & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ... & ... & ... & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & ... & b_{n-2} & a_{n-2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{n-2} & b_{n-1} & a_{n-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{n-1} & b_n \end{bmatrix} \tag{4.27}$$

donde $a_n, b_n \in [0, 1]$.

En particular, en este trabajo consideramos matrices simétricas, las cuales son llamadas Matrices de Jacobi. La metodología de construcción del espectro para este caso es la siguiente:

1. Se construyen matrices aleatorias con valores $a_n, b_n \in [0, 1]$ como se menciona anteriormente.

una dirección (eigenvector asociado a cada eigenvalor).

Para cada matriz $n \times n$ se requieren $(2n-1)$ parámetros.

2. Se construye un modelo de red neuronal con una capa de entrada de tamaño $(2n-1)$ para los parámetros de salida, La salida tiene tamaño diferente correspondiente a los casos:
   - Sólo eigenvalores (para cada matriz $n \times n$ corresponden n eigenvalores).
   - Eignenvalores + 1 eigenvector.
   - Eigenvalores + 2 eigenvectores.
   - ...
   - Eigenvalores + n eigenvectores.
3. Las capas ocultas corresponden a 8 capas de tamaños entre 12 y hasta 200 neuronas con funciones de activación tanh en cada capa.
4. Para la compilación del modelo se utilizo como medida 'mae', que es la medida de error de *error absoluto medio* con un optimizador *Adam* a tasa de $3e^{-4}$.
5. Para realizar el ajuste se itera 50 veces (50 epochs) sobre el conjunto de pares de datos, con una paralelización de 4 *workers*.
6. Con la finalidad de comparar los resultados, se obtienen nuevamente matrices aleatorias de tamaño $n \times n$ con $(2n-1)$ parámetros, y se obtienen sus eigenvalores. Posteriormente, se utiliza el modelo creado a partir de la red neuronal y se determina la 'mejor matriz' que acopla a los eigenvalores.
7. Finalmente, se obtienen nuevamente los eigenvalores de la matriz calculada en el paso anterior y se comparan con los originales. El cálculo de la norma para comparación se realiza mediante:
$$norm = ||w - w_2||/||w||$$

Un *snippet* del código se muestra a continuación, para el cálculo de la metodología anterior, esto para el caso particular de una matriz 3×3:

Snippet 4.6: Problema Inverso de Eigenvalores

```
% Carga de librerias necesarias
% Construccion de matriz y obtencion de eigenvalores.
def eigen3x3():
    a, b, c, d, e = np.random.uniform(0,1,5)
    M = np.array([[a, d, 0],
                  [d, b, e],
                  [0, e, c]])
    w, v = la.eig(M)
    M = np.array([a, b, c, d, e])
    % v1 = v[0].flatten()
    % v2 = v[1].flatten()
    % v3 = v[2].flatten()
    e_n = w

    return M,e_n

% Creacion de simulaciones. Construccion de matrices y obtencion de eigenvalores
```

```
% para un conjunto grande de casos, N = 10k, 100k.
M, E_n = [],[]
for i in range(0,100000):
    m, en = eigen3x3()
    M.append(m)
    E_n.append(en)
M = np.array(M)
E_n = np.array(E_n)

% Construccion de red neuronal para estructura de datos de entrada y salida.
capa1 = tf.keras.layers.Dense(units = 3, input_shape = (3,), activation = 'tanh')
capa2 = tf.keras.layers.Dense(units = 6,activation = 'tanh')
capa3 = tf.keras.layers.Dense(units = 12,activation = 'tanh')
capa4 = tf.keras.layers.Dense(units = 24,activation = 'tanh')
capa5 = tf.keras.layers.Dense(units = 96,activation = 'tanh')
capa6 = tf.keras.layers.Dense(units = 48,activation = 'tanh')
capa7 = tf.keras.layers.Dense(units = 24,activation = 'tanh')
capa8 = tf.keras.layers.Dense(units = 12,activation = 'tanh')
capa9 = tf.keras.layers.Dense(units = 6,activation = 'tanh')
salida = tf.keras.layers.Dense(units = 5, input_shape = (5,))

% Construccion de modelo secuencial de capas.
modelo = tf.keras.Sequential([capa1,capa2,capa3,capa4,capa5,capa6,\
capa7,capa8,capa9,salida])
modelo.compile(
    optimizer = tf.keras.optimizers.Adam(1e-3),
    loss='mae',
)

% Ajuste de modelo
historial = modelo.fit(E_n,M, epochs=50, workers = 8, verbose = 2)
```

4.3.3. Resultados

Para el análisis de resultados, consideramos la última parte de la metodología anterior (Ecuación 4.3.2), utilizando dos formas de evaluación: (i) calculando la norma

$$norm_1 = ||w - w_2||/||w||,$$

y (ii) ordenando los eigenvalores por cada una de las matrices y calculando la norma

$$norm_2 = ||w_{sorted} - w_{2_{sorted}}||/||w_{sorted}||.$$

En la Figura 4.10 se muestra la comparativa de 2 modelos distintos para matrices 3×3 (con dos diferentes metodologías de evaluación de éstos): (i) utilizando únicamente los eigenvalores como entradas para la red neuronal y (ii) utilizando el espectro de eigenvalores así como uno de los eigenvectores. Para bajas dimensiones, como en este caso (n = 3), encontramos que agregar información referente a los eigenvalores mejora, en general, el modelo. En este caso puede verse que el modelo de eigenvalores con un eigenvector (color rosa) se comporta mejor que el modelo únicamente con eigenvalores (color verde). Lo mismo sucede con la metodología de evaluación con ordenamiento, es decir, el cálculo de $norm_2$.

Al aumentar la dimensión de las matrices en estudio, encontramos que el comportamiento cambia completamente. En la Figura 4.11 observamos que, a diferencia de la comparativa para modelos 3×3, al agregar información referente a uno de los eigenvalores del sistema afecta el desempeño del modelo para matrices 5×5, pues obsérvese que el modelo con eigenvalores y un eigenvector asociado tiene un peor desempeño, por ambas metodologías de evaluación. En el caso sin ordenar, observamos que el histograma verde (únicamente eigenvalores) tiene una distribución de menores errores que el histograma púrpura (eigenvalores más un eigenvector). Lo mismo sucede para la metodología de ordenamiento (histograma azul comparado con histograma rosa).

Además, en la Figura 4.12 se comparan dos modelos distintos para evaluar matrices 9×9, en el cual se pueden observar dos diferentes casuísticas al aumentar la dimensión a un número mayor (n=9): (i) los errores son considerablemente mayores que en los casos anteriores (n = 3 y n = 5) y (ii) se mantiene la tendencia que al aumentar la dimensión de análisis incorporar información referente a los eigenvalores resulta contraproducente. Cabe destacar que utilizar la metodología de ordenamiento de los eigenvalores muestra un mejor comportamiento al evaluar los errores.

Por último, en la Figura 4.13 se muestra el comportamiento de afectación en los modelos al aumentar la dimensión de estudio, puesto que los errores van en aumento conforme a la dimensión. Además, en la Figura 4.14 se muestra que al aumentar la dimensión, como se ha mencionado antes, incorporar al modelo información de los eigenvalores resulta en un empeoramiento del modelo, pues veamos que el histograma azul (n = 9, únicamente eigenvalores) tiene un mejor desempeño que el modelo para (n = 8 y n = 9) incorporando un eigenvalor (histogramas verde y púrpura, respectivamente).

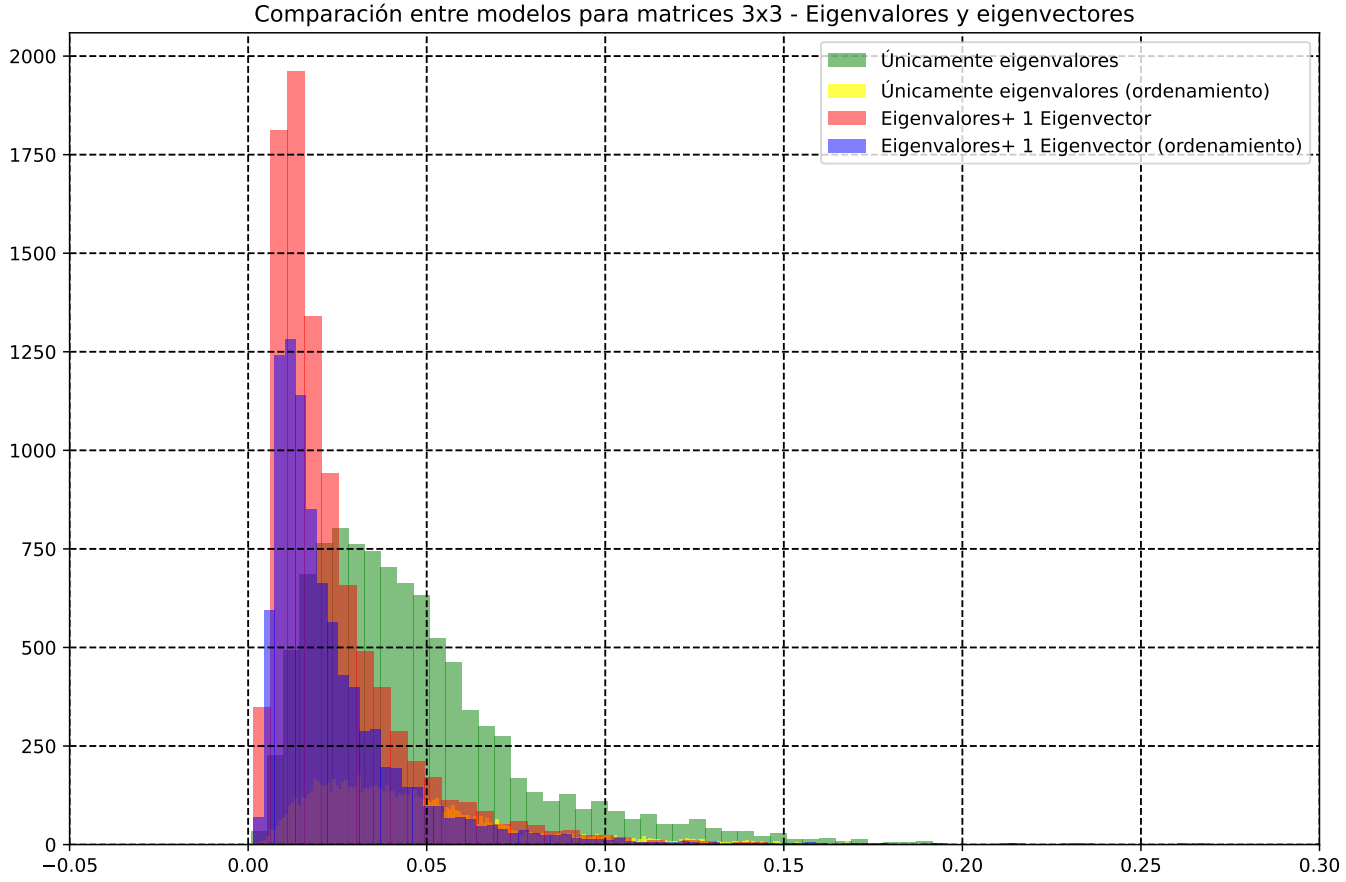

Fig. 4.10: Comparación para modelos de matrices 3x3. Se comparan los modelos de eigenvalores únicamente, así como eigenvalores con un eigenvector. Además se muestra la comparativa, cuando se hace cálculo de la norma ordenada. En general, muestra que un ordenamiento numérico de los eigenvalores mejora la calidad de los resultados. Para modelos de bajas dimensiones, se observa que agregar información referente a los eigenvectores mejora los modelos.

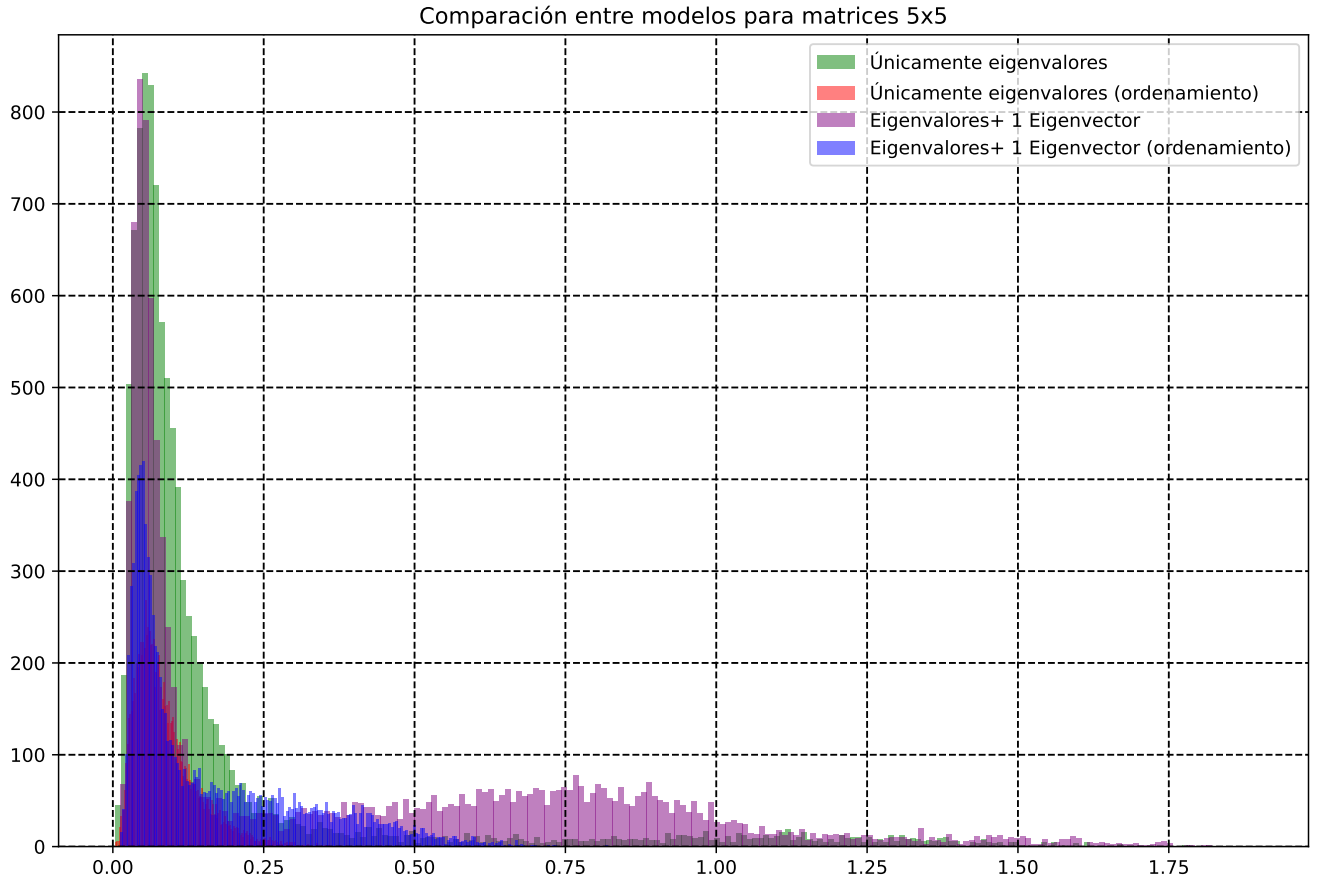

Fig. 4.11: Comparación para modelos de matrices 5x5. Se comparan los modelos de eigenvalores únicamente, así como eigenvalores con un eigenvector. Además se muestra la comparativa, cuando se hace cálculo de la norma ordenada. En general, muestra que un ordenamiento numérico de los eigenvalores mejora la calidad de los resultados. En este caso, además se observa que agregar información referente a un eigenvector es perjudicial para el modelo, si no se ordenan los eigenvalores.

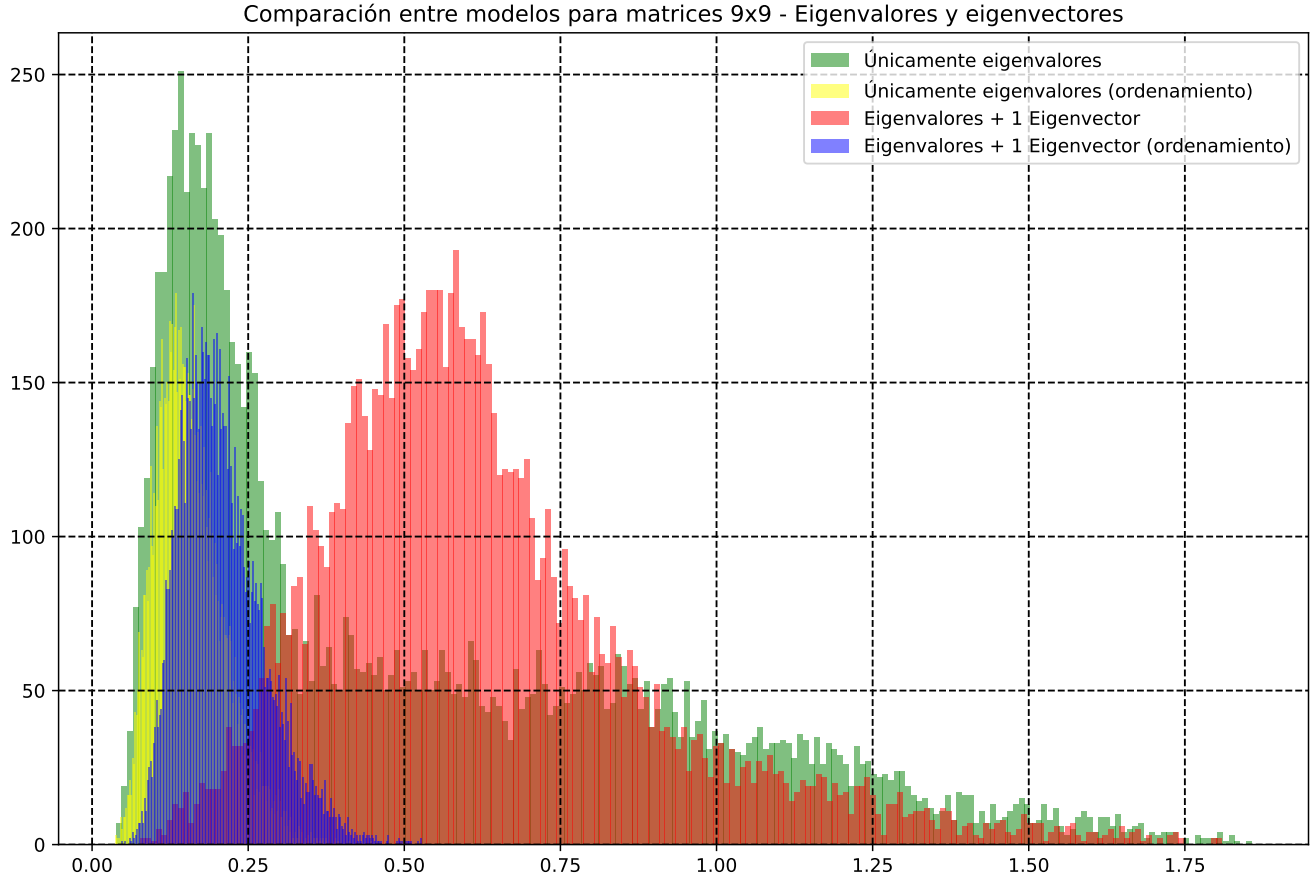

Fig. 4.12: Comparación para modelos de matrices 9x9. Se comparan los modelos de eigenvalores únicamente, así como eigenvalores con un eigenvector. Además se muestra la comparativa, cuando se hace cálculo de la norma ordenada. En general, muestra que un ordenamiento numérico de los eigenvalores mejora la calidad de los resultados. Para modelos de altas dimensiones, se observa que agregar información referente a los eigenvectores empeora los modelos.

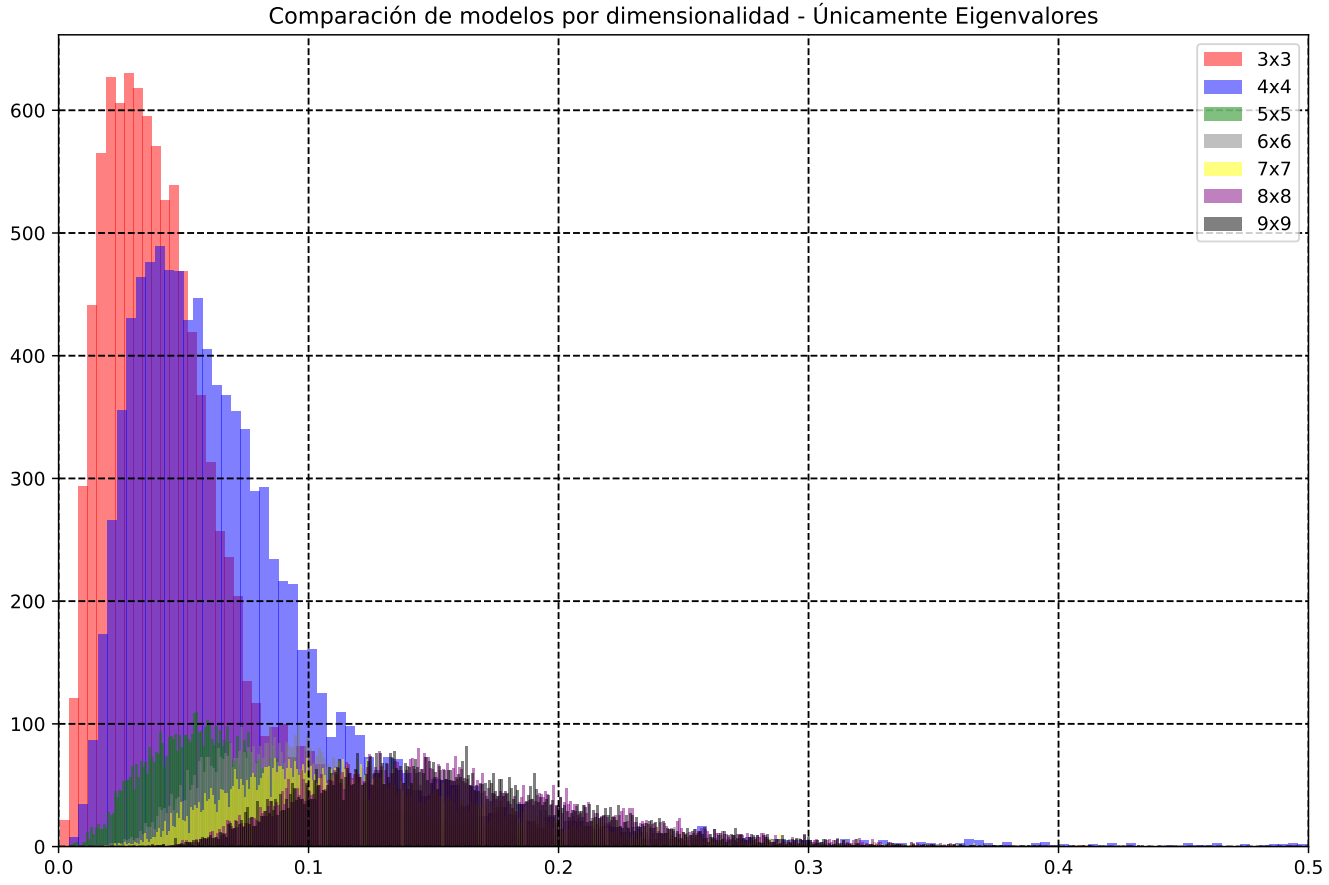

Fig. 4.13: Comparación de modelos para distintas dimensiones de matriz (n = 3,4,5,6,7,8,9). Se utiliza la metodología de únicamente eigenvalores y se comparan los errores para cada unas de las dimensiones. En general, muestra que los resultados de los modelos empeoran al aumentar la dimensión.

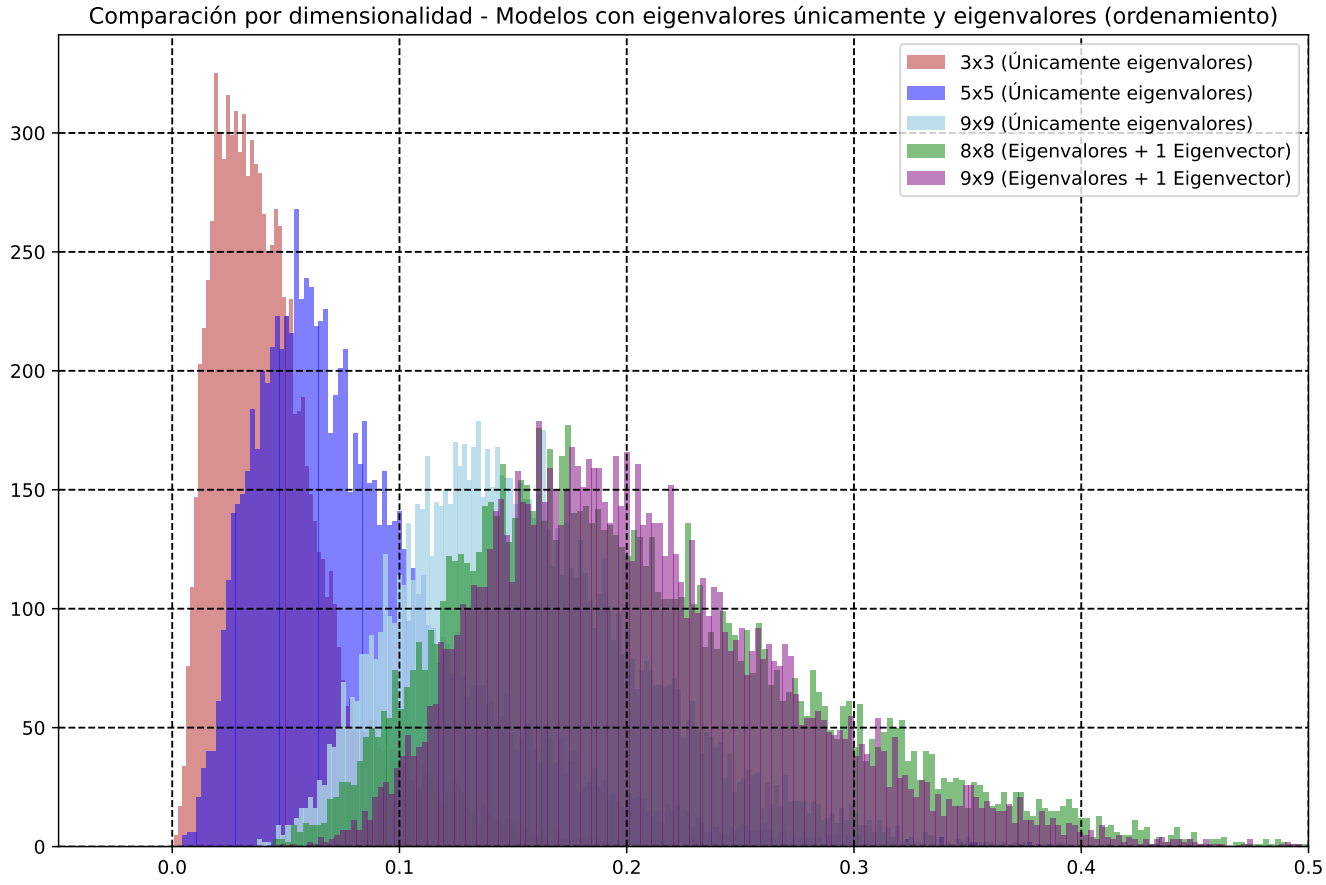

Fig. 4.14: Comparación de modelos para distintas dimensiones de matriz (n = 3,5,8,9). Se utiliza la metodología de únicamente eigenvalores y la metodología de eigenvalores + 1 eigenvector. Se comparan los errores para cada unas de las dimensiones. En general, muestra que los resultados de los modelos empeoran al aumentar la dimensión. Además, resalta que al aumentar la dimensión resulta más conveniente únicamente usar los eigenvalores.

Capítulo 5

Conclusiones

Las notables capacidades de procesamiento de información de las ANNs así como su habilidad de aprender de ejemplos, convierte esta metodología en un nuevo paradigma eficiente de solución de problemas. El incremento en la utilización de las ANNs son debidos a dos principales factores: **(i)** la habilidad de reconocer y aprender de patrones de entrada y salida sin ninguna consideración física, incluyendo problemas no lineales o de alta dimensionalidad; **(ii)** alta tolerancia a datos con errores de medición o ruido. Las ANNs también tienes limitaciones que se deben considerar: **(i)** dependencia tanto en la calidad como cantidad de los datos, **(ii)** la imposibilidad de explicar de forma comprensible los cálculos y procesos intermedios del resultado de una ANN y **(iii)** la falta de reglas fijas o de guía para la elección de parámetros de la red neuronal.

A lo largo de este trabajo de investigación, se ha confirmado la eficacia de las redes neuronales asistidas por la física (PINNs) en la solución de Ecuaciones Diferenciales Parciales (EDPs). La ventaja principal de usar PINNs radica en su capacidad para modelar soluciones que se alinean intrínsecamente con las leyes físicas subyacentes del problema en cuestión, minimizando la dependencia de una gran cantidad de datos de entrada. En nuestro estudio detallado presentado en la Sección 4.1, abordamos dos ecuaciones diferenciales distintas. Validamos inicialmente las soluciones de PINN para el caso del dispositivo mecánico biestable con un momento de inercia constante. Para este propósito, comparamos dos métodos distintos: (i) un enfoque basado en diferencias finitas utilizando el método de Runge-Kutta en el software Mathematica y (ii) nuestra implementación de PINN en DeepXDE con Python. Intrigantemente, al normalizar las soluciones, encontramos que eran idénticas entre los dos enfoques. Posteriormente, extendimos nuestro análisis al dispositivo mecánico biestable con un momento de inercia variable, seleccionando la función $I(x) = \frac{1}{1+x}$ para el intervalo $x \in [0, 2]$. Tras un análisis profundo, derivamos los eigenvalores que conferían relevancia física a la EDP, identificando estos como $\lambda_1 = -1.72104490421$ y $\lambda_2 = -2.17701849660$. La solución a través de ambos enfoques numéricos indicó un acuerdo sobresaliente entre ellos, reafirmando la robustez y precisión de las PINNs en este contexto. Por lo tanto, lo desarrollado en esta tesis respalda la viabilidad y exactitud de las PINNs en la resolución de problemas físicos complejos, posicionándolas como una herramienta invaluable en el ámbito de la investigación de sistemas dinámicos.

En la Sección 4.2, hemos logrado construir con éxito una Red Neuronal Artificial (ANN) dedicada a desentrañar el enigmático fenómeno del efecto túnel cuántico, enfocándonos en la determinación precisa de los coeficientes de transmisión y reflexión, ya sea para una, dos o tres barreras de potencial. El camino para llegar a este logro comenzó con la meticulosa construcción de bases de

datos, que comprendían información crucial como las alturas y grosores de las barreras en cuestión y el coeficiente de reflexión. Este vital conjunto de datos fue construido en el software Mathematica. Avanzando en nuestro proceso, procedimos a segmentar nuestros datos en conjuntos destinados para entrenamiento y validación. El diseño final de nuestra ANN consistió en una estructura compuesta por una capa de entrada, ocho capas ocultas, y una capa de salida. La primera capa albergaba 7 neuronas y empleaba la función de activación 'relu'. Por otro lado, las capas intermedias, con una diversidad de hasta 24 neuronas, alternaban funciones de activación 'relu' y 'sigmoid'. Optimizamos nuestro modelo utilizando el optimizador 'adam' y establecimos una tasa de aprendizaje de 0.001. Al evaluar el desempeño de nuestro modelo utilizando el conjunto de datos de validación, encontramos que el modelo era extremadamente prometedor. Específicamente, el 25 % de los datos evaluados arrojó un error menor al 2.14 %, y, de manera alentadora, el 50 % y el 75 % presentaron un error menor al 13.44 %. Además, uno de los hallazgos más destacados fue la eficiencia computacional del modelo: superó las expectativas al ser aproximadamente 10000 veces más rápido por iteración que los métodos tradicionales. Estos resultados reafirman la potencia y la relevancia de las ANN en la ingenieria cuántica moderna.

En la Sección 4.3 de este trabajo académico se abordó de manera meticulosa el desafío del problema inverso de eigenvalores en matrices de distintas dimensiones, en específico, rangos de $n = 3$ hasta $n = 9$. Empleando técnicas supervisadas de redes neuronales, nuestro enfoque se centró en las llamadas matrices de Jacobi (relacionado con los Hamiltonianos a primeros vecinos), que se caracterizan por ser matrices tridiagonales compuestas por números reales. En las fases iniciales, generamos matrices de Jacobi que abarcaban un espectro de valores reales entre $I = [0, 1]$. Posteriormente, evolucionamos hacia la creación de redes neuronales adaptadas, con una estructura de entrada basada en las dimensiones $(2n1)$ o $2n$ para modelos que incluían tanto eigenvalores como un eigenvector. La arquitectura neuronal se basó en un diseño que variaba entre 12 y 200 neuronas distribuidas en un máximo de 8 capas ocultas, todas empleando la función de activación tanh. Durante la fase de compilación, se adoptó el error absoluto medio ('mae') como métrica de precisión y se empleó el optimizador 'adam', estableciendo una tasa de aprendizaje de 3×10^{-4}. A su vez, definimos un proceso de entrenamiento de 50 epochs con una paralelización optimizada con 4 workers. Para una evaluación robusta y detallada, generamos matrices aleatorias de tamaño n×n, calculamos sus eigenvalores y los introdujimos en la red neuronal. El resultado fue una matriz que, idealmente, se ajustaría al espectro deseado. Evaluamos la precisión de este ajuste mediante dos enfoques distintos: $norm = ||w - w_2||/||w||$ y $norm_2 = ||w_{sorted} - w_{2_{sorted}}||/||w_{sorted}||$. Nuestro análisis detallado sobre el comportamiento modelado en relación con la dimensión de las matrices reveló observaciones notables. Notamos que, para matrices de dimensiones más pequeñas ($n < 4$), la inclusión de datos relacionados con los eigenvectores potencia notablemente el rendimiento del modelo. Sin embargo, al incrementar la dimensión, esta inclusión tiende a desestabilizar el desempeño. Además, identificamos una tendencia preocupante: al expandir la dimensión bajo estudio, los modelos muestran una disminución en su capacidad de generalización. Esta revelación es crucial para investigaciones futuras y aplicaciones de este enfoque sobre la solución de problemas inversos.

Los resultados de este trabajo muestran que la construcción de dispositivos y/o sistemas híbridos entre métodos tradicionales y de redes neuronales prometen ser todo un nuevo paradigma de investigación. La posibilidad de aplicación de esta metodología recae en la predicción y medición de procesos físicos, modelos multiescala, simulaciones y control de sistemas dinámicos, por mencionar algunos. Sin embargo, no se puede dejar de mencionar la falta de guías o buenas prácticas en el

desarrollo y estructura de una ANN, por lo que el estudio de las ANNs, PINNs y sus aplicaciones tiene un camino de mejora y desarrollo bastante amplio.

Apéndice A

Algoritmo *Back-Propagation*

En general, la finalidad del algoritmo es encontrar los pesos tales que por cada entrada produce una salida lo más cerca a un valor predefinido. Una agradable introducción a este método puede encontrarse en [84].

- **Paso 1** Inicializar los pesos, la tasa de aprendizaje y el criterio para detener el entrenamiento.
- **Paso 2** Elegir aleatoramiento un par de entrada y salida.
- **Paso 3** Calcular las entradas de cada capa, así como la salida final de la red.
- **Paso 4** Calcular los componentes de sensibilidad.
- **Paso 5** Calcular los componentes del gradiente y actualizar los pesos de la red.
- **Paso 6** Iterar el proceso hasta llegar al criterio de detención.

Apéndice A.1 Detalles del algoritmo

1. Inicializar los pesos w_{ij}^l con valores pequeños en el intervalo $[-1, 1]$. Seleccionar los valores de la tasa de aprendizaje y el criterio de detención.
2. Elegir aleatoriamiento un vector de entrada de $\mathcal{I} = \{I_1^1, I_2^1, ..., I_{N_1}^1\}$ y su correspondiente salida de $\mathcal{T} = \{t_1, t_2, ..., t_{N_1}\}$.
3. Calcular la entrada para cada capa, $\{I_1^l, I_2^l, ..., I_{N_l}^l\}$ y las salidas de la capa L, $\{O_1, O_2, ..., O_{N^{L+1}}\}$.

$$I_j^2 = \sum_{i=0}^{N^1} I_i^1 w_{ij}^1, \quad j = 1, 2, 3, ..., N^2.$$

$$I_j^l = \sum_{i=0}^{N^{l-1}} \mathcal{A}(I_i^{l-1}) w_{ij}^{l-1}, \quad j = 1, 2, 3, ..., N^l, \quad l = 1, 2, 3, ..., L.$$

$$O_j = \sum_{i=0}^{N^L} \mathcal{A}(I_i^L) w_{ij}^L, \quad j = 1, 2, 3, ..., N^{L+1}.$$

Los términos I_0^1, $\mathcal{A}(I_0^{l-1})$ y $\mathcal{A}(I_0^L)$ son inicializados en uno.

4. Calcular los componentes de sensibilidad $\{\delta_1^l, \delta_2^l, ..., \delta_{N_l}^l\}$.

$$\delta_i^{L+1} = 2(O_i - t_i), \quad i = 1, 2, 3, ..., N^{L+1}.$$

$$\delta_i^l = \mathcal{A}'(I_i^l) \sum_{j=1}^{N^{l+1}} w_{ij}^l \delta_j^{l+1}, \quad l = L, L-1, ..., 2. \; i = 1, 2, 3, ..., N^L.$$

5. Calcular las componentes del gradiente g_{ij}^l (derivada parcial del error respecto a los pesos) y actualizar los pesos w_{ij}^L.

$$g_{ij}^l = \frac{\partial e}{\partial w_{ij}^l} = \mathcal{A}(I_i^l)\delta_j^{l+1} \quad l = 1, 2, 3, ..., L. \; i = 0, 1, 2, ..., N^l.$$

$$w_{ij}^l = w_{ij}^l - r g_{ij}^l. \quad j = 1, 2, 3, ..., N^{l+1}.$$

6. Verificar el criterio de detención del algoritmo.

   a) Incrementando el índice de iteración, $k = k + 1$. Si $k > k_{max}$, detenemos y regresamos los pesos w_{ij}^l.

   b) Verificar la convergencia. Si $|g_{ij}^l| \leq g_{min}$ con $l = 1, 2, 3, ..., L.$, $i = 0, 1, 2, ..., N^l$ y $j = 1, 2, 3, ..., N^{l+1}$, detenemos y regresamos los pesos.

   c) Si el criterio de detención anterior no se satisface, iteramos de nuevo.

Bibliografía

[1] E. C. Smith y M. S. Lewicki. Efficient auditory coding. *Nature*, **439**(7079):978-982, feb. de 2006. ISSN: 1476-4687. DOI: 10.1038/nature04485. eprint: https://doi.org/10.1038/nature04485. URL: https://doi.org/10.1038/nature04485 (citado en la pág. 1).

[2] A. Halevy, P. Norvig y F. Pereira. The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems*, **24**(2):8-12, 2009. DOI: 10.1109/MIS.2009.36 (citado en la pág. 1).

[3] A. Krizhevsky, I. Sutskever y G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. En F. Pereira, C. Burges, L. Bottou y K. Weinberger, edición, *Advances in Neural Information Processing Systems*, volumen 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (citado en la pág. 1).

[4] K. He, X. Zhang, S. Ren y J. Sun. Deep Residual Learning for Image Recognition. *CoRR*, **abs/1512.03385**, 2015. arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (citado en la pág. 1).

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser e I. Polosukhin. Attention is All you Need. En I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan y R. Garnett, edición, *Advances in Neural Information Processing Systems*, volumen 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (citado en la pág. 1).

[6] K. B. N. R. A. K.-J. S. S. M. B. H. M. T. S. Esteva Andre. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, **542**, 2017. eprint: 1476-4687. URL: https://doi.org/10.1038/nature21056 (citado en la pág. 1).

[7] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao y K. Zieba. End to End Learning for Self-Driving Cars, 2016. arXiv: 1604.07316 [cs.CV] (citado en la pág. 1).

[8] 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 y X. Zheng. TensorFlow: A system for large-scale machine learning, 2016. arXiv: 1605.08695 [cs.DC] (citado en la pág. 1).

[9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai y S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library, 2019. arXiv: 1912.01703 [cs.LG] (citado en la pág. 1).

[10] G. Carleo y M. Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, **355**(6325):602-606, 2017. DOI: 10.1126/science.aag2302. URL: https://doi.org/10.1126%2Fscience.aag2302 (citado en la pág. 2).

[11] P. Broecker, J. Carrasquilla, R. G. Melko y S. Trebst. Machine learning quantum phases of matter beyond the fermion sign problem. *Scientific Reports*, **8823**(-):602-606, 2017. DOI: 10.1038/s41598-017-09098-0. URL: https://doi.org/10.1038/s41598-017-09098-0 (citado en la pág. 2).

[12] P. Baldi, P. Sadowski y D. Whiteson. Searching for Exotic Particles in High-Energy Physics with Deep Learning. *Nature communications*, **5**:4308, jul. de 2014. DOI: 10.1038/ncomms5308 (citado en la pág. 2).

[13] M. Raissi, P. Perdikaris y G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, **378**:686-707, 2019. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125 (citado en las págs. 2, 20).

[14] Z. Mao, A. D. Jagtap y G. E. Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, **360**:112789, 2020. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2019.112789. URL: https://www.sciencedirect.com/science/article/pii/S0045782519306814 (citado en la pág. 2).

[15] R. Qiu, R. Huang, X. Yao, J. Wang, Z. Zhang, J. Yue, Z. Zeng e Y. Wang. Physics-informed neural networks for phase-field method in two-phase flow. *Physics of Fluids*, **34**, abr. de 2022. DOI: 10.1063/5.0091063 (citado en la pág. 2).

[16] S. Cuomo, V. S. di Cola, F. Giampaolo, G. Rozza, M. Raissi y F. Piccialli. Scientific Machine Learning through Physics-Informed Neural Networks: Where we are and What's next, 2022. arXiv: 2201.05624 [cs.LG] (citado en las págs. 2, 3).

[17] I. H. M. Basheer. Artificial Neural Networks: Fundamentals, computing, desing and application. *Journal of Microbiological Methods*, **43**:3-31, 2000. DOI: 10.1016/S0167-7012(00)00201-3 (citado en las págs. 3, 5).

[18] A. Andina Diego Pham D. T. Vega-Coronado. Neural Networks Historical Review. *Computational Intelligence*, **43**:39-65, 2007. ISSN: 978-0-387-37450-5. DOI: 10.1007/0-387-37452-3_2 (citado en las págs. 5, 10).

[19] B. Li, C. Delpha, D. Diallo y A Migan-Dubois. Application of Artificial Neural Networks to photovoltaic fault detection and diagnosis: A review. *Renewable and Sustainable Energy Reviews*, **138**:110512, 2021 (citado en la pág. 5).

[20] C. Kingston. Neural networks in forensic science. *Journal of Forensic Science*, **37**(1):252-264, 1992 (citado en la pág. 5).

[21] A. Vidaki, D. Ballard, A. Aliferi, T. H. Miller, L. P. Barron y D. S. Court. DNA methylation-based forensic age prediction using artificial neural networks and next generation sequencing. *Forensic Science International: Genetics*, **28**:225-236, 2017 (citado en la pág. 5).

[22] R. Patidar, L. Sharma y col. Credit card fraud detection using neural network. *International Journal of Soft Computing and Engineering (IJSCE)*, **1**(32-38), 2011 (citado en la pág. 5).

[23] W. G. Baxt. Application of artificial neural networks to clinical medicine. *The lancet*, **346**(8983):1135-1138, 1995 (citado en la pág. 5).

[24] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed y H. Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, **4**(11):e00938, 2018 (citado en la pág. 5).

[25] A. Jain, J. Mao y K. Mohiuddin. Artificial neural networks: a tutorial. *Computer*, **29**(3):31-44, 1996. DOI: 10.1109/2.485891 (citado en las págs. 5, 10, 11).

[26] K. Gurney. *An Introduction to Neural Networks*. UCL Press, first edition edición, 1997 (citado en la pág. 5).

[27] A. Krogh. What are artificial neural networks? *Nature biotechnology*, **26**(2):195-197, 2008 (citado en la pág. 6).

[28] J Zou, Y. Han, y S. So. Overview of Artificial Neural Networks. *Methods in molecular biology (Clifton, N.J.)*, **458**:15-23, 2008. DOI: 10.1007/978-1-60327-101-1_2 (citado en la pág. 9).

[29] M. Fernandez, J. Caballero, L. Fernández y A. Sarai. Genetic Algorithm Optimization in Drug Design QSAR: Bayesian-Regularized Genetic Neural Networks (BRGNN) and Genetic Algorithm-Optimized Support Vectors Machines (GA-SVM). *Molecular diversity*, **15**:269-89, mar. de 2010. DOI: 10.1007/s11030-010-9234-9 (citado en la pág. 9).

[30] A. Cherkasov, E. N. Muratov, D. Fourches, A. Varnek, I. I. Baskin, M. Cronin, J. Dearden, P. Gramatica, Y. C. Martin, R. Todeschini, V. Consonni, V. E. Kuz'min, R. Cramer, R. Benigni, C. Yang, J. Rathman, L. Terfloth, J. Gasteiger, A. Richard y A. Tropsha. QSAR Modeling: Where Have You Been? Where Are You Going To? *Journal of Medicinal Chemistry*, **57**(12):4977-5010, 2014. DOI: 10.1021/jm4004285. URL: https://doi.org/10.1021/jm4004285. PMID: 24351051 (citado en la pág. 9).

[31] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, **65**(6):386, 1958 (citado en las págs. 10, 11).

[32] I. C. Education. Unsupervised learning. 2020. URL: https://www.ibm.com/cloud/learn/unsupervised-learning (visitado 25-09-2022) (citado en la pág. 11).

[33] M. W. M. G. Dissanayake y N. Phan-Thien. Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, **10**(3):195-201, 1994. DOI: https://doi.org/10.1002/cnm.1640100303. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cnm.1640100303. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.1640100303 (citado en la pág. 19).

[34] H. Owhadi. Bayesian Numerical Homogenization. *Multiscale Modeling & Simulation*, **13**(3):812-828, 2015. DOI: 10.1137/140974596. eprint: https://doi.org/10.1137/140974596. URL: https://doi.org/10.1137/140974596 (citado en la pág. 20).

[35] M. Raissi, P. Perdikaris y G. E. Karniadakis. Inferring solutions of differential equations using noisy multi-fidelity data. *Journal of Computational Physics*, **335**:736-746, 2017. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2017.01.060. URL: https://www.sciencedirect.com/science/article/pii/S0021999117300761 (citado en la pág. 20).

[36] M. Raissi y G. E. Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, **357**:125-141, 2018. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2017.11.039. URL: https://www.sciencedirect.com/science/article/pii/S0021999117309014 (citado en la pág. 20).

[37] M. Raissi, P. Perdikaris y G. E. Karniadakis. Machine learning of linear differential equations using Gaussian processes. *Journal of Computational Physics*, **348**:683-693, 2017. ISSN: 0021-

9991. DOI: https://doi.org/10.1016/j.jcp.2017.07.050. URL: https://www.sciencedirect.com/science/article/pii/S0021999117305582 (citado en la pág. 20).

[38] E. Haghighat, M. Raissi, A. Moure, H. Gomez y R. Juanes. A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, **379**:113741, 2021. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2021.113741. URL: https://www.sciencedirect.com/science/article/pii/S0045782521000773 (citado en la pág. 23).

[39] S. Cai, Z. Wang, S. Wang, P. Perdikaris y G. E. Karniadakis. Physics-Informed Neural Networks for Heat Transfer Problems. *Journal of Heat Transfer*, **143**(6), abr. de 2021. ISSN: 0022-1481. DOI: 10.1115/1.4050542. eprint: https://asmedigitalcollection.asme.org/heattransfer/article-pdf/143/6/060801/6688635/ht_143_06_060801.pdf. URL: https://doi.org/10.1115/1.4050542. 060801 (citado en la pág. 23).

[40] B. Moseley, A. Markham y T. Nissen-Meyer. Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations, 2021. arXiv: 2107.07871 [physics.comp-ph] (citado en la pág. 23).

[41] A. Griewank y A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008 (citado en la pág. 23).

[42] M. Bücker, G. Corliss, U. Naumann, P. Hovland y B. Norris. Automatic Differentiation: Applications, Theory, and Implementations. *Lecture Notes in Computational Science and Engineering, Springer-Verlag Berlin Heidelberg 2006.*, **50**:XVIII, 370, 2017. ISSN: 1439-7358. DOI: https://doi.org/10.1007/3-540-28438-9. URL: https://link.springer.com/book/10.1007/3-540-28438-9 (citado en la pág. 23).

[43] C. C. Margossian. A Review of automatic differentiation and its efficient implementation. *CoRR*, **abs/1811.05031**, 2018. arXiv: 1811.05031. URL: http://arxiv.org/abs/1811.05031 (citado en la pág. 23).

[44] G. Pang, L. Lu y G. E. Karniadakis. fPINNs: Fractional Physics-Informed Neural Networks. *SIAM Journal on Scientific Computing*, **41**(4):A2603-A2626, 2019. DOI: 10.1137/18M1229845. eprint: https://doi.org/10.1137/18M1229845. URL: https://doi.org/10.1137/18M1229845 (citado en la pág. 24).

[45] Y. T. Y. John Xu Zhi-QinZhang y Z. Ma. Frequency Principle: Fourier Analysis Sheds Light on Deep Neural Networks. *Communications in Computational Physics*, **28**(5):1746-1767, 2020. ISSN: 1991-7120. DOI: https://doi.org/10.4208/cicp.OA-2020-0085. URL: http://global-sci.org/intro/article_detail/cicp/18395.html (citado en la pág. 25).

[46] L. Sun, H. Gao, S. Pan y J.-X. Wang. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, **361**:112732, 2020. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2019.112732. URL: https://www.sciencedirect.com/science/article/pii/S004578251930622X (citado en la pág. 25).

[47] T. De Ryck, S. Lanthaler y S. Mishra. On the approximation of functions by tanh neural networks. *Neural Networks*, **143**:732-750, 2021. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2021.08.015. URL: https://www.sciencedirect.com/science/article/pii/S0893608021003208 (citado en la pág. 26).

[48] Y. Shin. On the Convergence of Physics Informed Neural Networks for Linear Second-Order Elliptic and Parabolic Type PDEs. *Communications in Computational Physics*,

28(5):2042-2074, 2020. DOI: 10.4208/cicp.oa-2020-0193. URL: https://doi.org/10.4208%2Fcicp.oa-2020-0193 (citado en la pág. 26).

[49] Z. Lai, C. Mylonas, S. Nagarajaiah y E. Chatzi. Structural identification with physics-informed neural ordinary differential equations. *Journal of Sound and Vibration*, **508**:116196, 2021. ISSN: 0022-460X. DOI: https://doi.org/10.1016/j.jsv.2021.116196. URL: https://www.sciencedirect.com/science/article/pii/S0022460X21002686 (citado en la pág. 27).

[50] R. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems (Classics in Applied Mathematics Classics in Applied Mathemat)*. Society for Industrial y Applied Mathematics, USA, 2007. ISBN: 0898716292 (citado en la pág. 27).

[51] E. Süli. *Finite Element Methods*. En *Encyclopedia of Applied and Computational Mathematics*. B. Engquist, edición. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, páginas 527-533. ISBN: 978-3-540-70529-1. DOI: 10.1007/978-3-540-70529-1_450. URL: https://doi.org/10.1007/978-3-540-70529-1_450 (citado en la pág. 27).

[52] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002. DOI: 10.1017/CBO9780511791253 (citado en la pág. 27).

[53] W. F. Ames. *Numerical Analysis of Partial Differential Equations*. Academic Press, 1992 (citado en la pág. 27).

[54] J. C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. SIAM, 2004 (citado en la pág. 27).

[55] E. Kharazmi, Z. Zhang y G. E. Karniadakis. hp-VPINNs: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, **374**:113547, 2021. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2020.113547. URL: https://www.sciencedirect.com/science/article/pii/S0045782520307325 (citado en la pág. 27).

[56] V. Dwivedi y B. Srinivasan. Physics Informed Extreme Learning Machine (PIELM)–A rapid method for the numerical solution of partial differential equations. *Neurocomputing*, **391**:96-118, 2020. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.12.099. URL: https://www.sciencedirect.com/science/article/pii/S0925231219318144 (citado en la pág. 27).

[57] A. A. Ramabathiran y P. Ramachandran. SPINN: Sparse, Physics-based, and partially Interpretable Neural Networks for PDEs. *Journal of Computational Physics*, **445**:110600, 2021. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2021.110600. URL: https://www.sciencedirect.com/science/article/pii/S0021999121004952 (citado en la pág. 27).

[58] Z. Fang y J. Zhan. A Physics-Informed Neural Network Framework for PDEs on 3D Surfaces: Time Independent Problems. *IEEE Access*, **8**:26328-26335, 2020. DOI: 10.1109/ACCESS.2019.2963390 (citado en la pág. 27).

[59] U. bin Waheed, E. Haghighat, T. Alkhalifah, C. Song y Q. Hao. PINNeik: Eikonal solution using physics-informed neural networks. *Computers Geosciences*, **155**:104833, 2021. ISSN: 0098-3004. DOI: https://doi.org/10.1016/j.cageo.2021.104833. URL: https://www.sciencedirect.com/science/article/pii/S009830042100131X (citado en la pág. 28).

[60] F. Sahli Costabal, Y. Yang, P. Perdikaris, D. E. Hurtado y E. Kuhl. Physics-Informed Neural Networks for Cardiac Activation Mapping. *Frontiers in Physics*, **8**:42, 2020 (citado en la pág. 28).

[61] J. D. Smith, K. Azizzadenesheli y Z. E. Ross. EikoNet: A deep neural networking approach for seismic ray tracing. https://github.com/charlespwd/project-title, 2020 (citado en la pág. 28).

[62] J. D. Smith, K. Azizzadenesheli y Z. E. Ross. EikoNet: Solving the Eikonal Equation With Deep Neural Networks. *IEEE Transactions on Geoscience and Remote Sensing*, **59**(12):10685-10696, 2021. DOI: 10.1109/tgrs.2020.3039165. URL: https://doi.org/10.1109%2Ftgrs.2020.3039165 (citado en la pág. 28).

[63] S. Amini Niaki, E. Haghighat, T. Campbell, A. Poursartip y R. Vaziri. Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems during manufacture. *Computer Methods in Applied Mechanics and Engineering*, **384**:113959, 2021. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2021.113959. URL: https://www.sciencedirect.com/science/article/pii/S0045782521002966 (citado en la pág. 28).

[64] W. Hu, Z. Guo y W. Li. Physics-Informed Neural Networks for Excited States of Molecules. *The Journal of Physical Chemistry Letters*, **11**(13):5135-5141, 2020 (citado en la pág. 29).

[65] G. Carleo y M. Troyer. Solving the Schrödinger equation with a neural network. *Science*, **355**(6325):602-606, 2017 (citado en la pág. 29).

[66] J. Wang y J. Kohler. Learning to solve quantum mechanics: A recurrent neural network learns the Schrödinger equation. *The Journal of Chemical Physics*, **150**(16):161101, 2019 (citado en la pág. 29).

[67] H. Chen, H. Sun y J. Liu. Solving the Schrödinger equation using a deep autoencoder neural network. *The Journal of Chemical Physics*, **154**(11):114111, 2021 (citado en la pág. 29).

[68] M. Raissi, U. Faghihi y A. Rasekh. A Physics-Informed Deep Learning Framework for Modeling COVID-19 Dynamics. *Scientific Reports*, **11**(1):7214, 2021 (citado en la pág. 29).

[69] Z. Tan, Y. Zhou e Y. Sun. Physics-Informed Neural Networks for Non-Invasive Estimation of Pulmonary Artery Pressure Waveform. *IEEE Journal of Biomedical and Health Informatics*, **24**(11):3113-3122, 2020 (citado en la pág. 29).

[70] S. Ham, P. Song, A. Manduca, D. S. Lake y S. Chen. Physics-Informed Neural Networks for Inferring Tissue Stiffness from Dynamic Magnetic Resonance Elastography. *IEEE Transactions on Medical Imaging*, **40**(5):1452-1461, 2021 (citado en la pág. 29).

[71] J. Qiu, J. Lang y A. Slocum. A curved-beam bistable mechanism. *Journal of Microelectromechanical Systems*, **13**(2):137-146, 2004. DOI: 10.1109/JMEMS.2004.825308 (citado en las págs. 31, 32).

[72] W. R. Inc. Mathematica, Version 13.3. URL: https://www.wolfram.com/mathematica. Champaign, IL, 2023 (citado en la pág. 33).

[73] L. Lu, X. Meng, Z. Mao y G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, **63**(1):208-228, 2021. DOI: 10.1137/19M1274067 (citado en la pág. 33).

[74] D. V. Schroeder. Notes on Quantum Mechanichs. *Weber State University*, 2022 (citado en las págs. 42, 43).

[75] H. Pickmann, R. L. Soto, J. Egaña y M. Salas. An inverse eigenvalue problem for symmetrical tridiagonal matrices. *Computers Mathematics with Applications*, **54**(5):699-708, 2007. ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2006.12.035. URL: https:

//www.sciencedirect.com/science/article/pii/S0898122107002489 (citado en la pág. 46).

[76] S. Yuan, A. Liao e Y. Lei. Inverse eigenvalue problems of tridiagonal symmetric matrices and tridiagonal bisymmetric matrices. *Computers Mathematics with Applications*, **55**(11):2521-2532, 2008. ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2007.10.006. URL: https://www.sciencedirect.com/science/article/pii/S0898122107007535 (citado en la pág. 46).

[77] M. T. Chu y G. H. Golub. Structured inverse eigenvalue problems. *Acta Numerica*, **11**:1–71, 2002. DOI: 10.1017/S0962492902000016 (citado en la pág. 46).

[78] K. M. Hochstadt H. On a singular inverse eigenvalue problem. *Arch. Rational Mech. Anal.*, **37**:243–254, 1970. DOI: https://doi.org/10.1007/BF00251605. URL: https://link.springer.com/article/10.1007/BF00251605 (citado en la pág. 47).

[79] G. I. N. I. Gasymov M.G. An inverse problem for the Sturm-Liouville operator with nonseparable self-adjoint boundary conditions. *Sib Math J*, **31**:910–918, 1990. DOI: https://doi.org/10.1007/BF00970056. URL: https://link.springer.com/article/10.1007/BF00970056 (citado en la pág. 47).

[80] G. S. Guseinov. Inverse Spectral Problems for Tridiagonal N by N Complex Hamiltonians. *Symmetry, Integrability and Geometry: Methods and Applications*, 2009. DOI: 10.3842/sigma.2009.018. URL: https://doi.org/10.3842%2Fsigma.2009.018 (citado en la pág. 47).

[81] E. Bairamov y C. Coskun. Jost solutions and the spectrum of the system of difference equations. *Applied Mathematics Letters*, **17**(9):1039-1045, 2004. ISSN: 0893-9659. DOI: https://doi.org/10.1016/j.aml.2004.07.006. URL: https://www.sciencedirect.com/science/article/pii/S089396590481663X (citado en la pág. 47).

[82] H. M. Huseynov. AN INVERSE SCATTERING PROBLEM FOR A SYSTEM OF DIRAC EQUATIONS WITH DISCONTINUITY CONDITIONS. *Proceedings of the Institute of Mathematics and Mechanics, National Academy of Sciences of Azerbaijan*, **40**(Special Issue):1039-1045, 2014. ISSN: 215-225. URL: https://proc.imm.az/volumes/40-s/40-s-17.pdf (citado en la pág. 47).

[83] M. D. Manafov, A. Kablan y B. Bala. Parseval equality of discrete Sturm-Liouville equation with periodic generalized function potentials. *AIP Conference Proceedings*, **1991**(1):020023, jul. de 2018. ISSN: 0094-243X. DOI: 10.1063/1.5047896. eprint: https://pubs.aip.org/aip/acp/article-pdf/doi/10.1063/1.5047896/13162490/020023_1_online.pdf. URL: https://doi.org/10.1063/1.5047896 (citado en la pág. 47).

[84] S. Sathyanarayana. A Gentle Introduction to Backpropagation. *Numeric Insight, Inc Whitepaper*, jul. de 2014 (citado en la pág. 61).