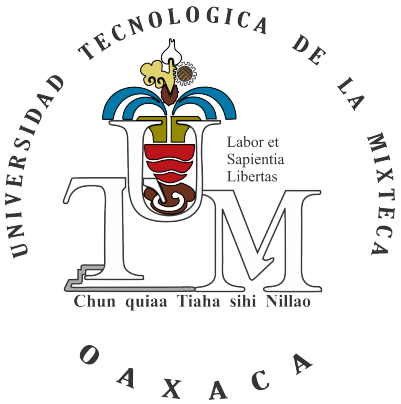


UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

SOLUCIÓN DE LA ECUACIÓN DE CALOR UNIDIMENSIONAL ESTACIONARIA EMPLEANDO UNA RED NEURONAL FÍSICAMENTE INFORMADA

TESIS

PARA OBTENER EL TÍTULO DE

INGENIERO EN FÍSICA APLICADA

PRESENTA:

ESTEBAN ELÍAS GONZÁLEZ MÉNDEZ

DIRECTOR:

DR. HUGO DAVID SÁNCHEZ CHÁVEZ

HUAJUAPAN DE LÉON, OAXACA
MARZO DE 2025

A mis padres.

Agradecimientos

Ser agradecido es esencial para apreciar y disfrutar lo que tenemos en esta vida. Implica reconocer el valor de aquello que nos rodea y su impacto en nuestras vidas. Así que...

Por su apoyo incondicional, por su gran cariño y amor hacia mí a lo largo de toda mi vida, por estar para mí en todo momento, creer en mí y motivarme a buscar ser mejor cada día.

Gracias, padres.

Por siempre sorprenderme con su increíble imaginación y creatividad. Por mostrarme una perspectiva diferente de ver el mundo y por compartir innumerables risas y apoyarme siempre que le es posible.

Gracias, hermano.

Por brindar su infraestructura y ofrecer espacios tranquilos en los que pude aprender, adquirir conocimiento, encontrar inspiración y reflexionar sobre mis pensamientos.

Gracias, Universidad Tecnológica de la Mixteca.

Por compartirme amablemente su conocimiento y mostrarme lo que la curiosidad y pasión pueden lograr. En especial, a mi director de tesis, el Dr. Hugo David Sánchez Chávez, por estar dispuesto a aconsejarme y apoyarme a lo largo de mi formación profesional.

Gracias, profesores.

Por compartir momentos en clase y dedicar tiempo a proyectos y tareas. A quienes estuvieron conmigo no solo estudiando, sino también compartiendo comidas, juegos y conversaciones.

Gracias, compañeros y amigos.

¡Gracias a todos!

Resumen

La transferencia de calor es fundamental en la física y la ingeniería, y resolver estos problemas es complejo debido a la necesidad de equilibrar precisión y eficiencia computacional.

Las redes neuronales artificiales han mostrado su potencia en tareas de predicción y optimización, pero su aplicación directa en problemas físicos ha sido limitada por la falta de integración de leyes físicas, afectando la precisión.

Este trabajo utiliza una solución innovadora: redes neuronales físicamente informadas, que incorporan conocimiento físico en su estructura para modelar mejor la propagación del calor. Se demuestra cómo estas redes mejoran la predicción de la temperatura en sistemas unidimensionales.

Abstract

Heat transfer is fundamental in physics and engineering, and solving these problems is complex due to the need to balance precision and computational efficiency.

Artificial neural networks have demonstrated their power in prediction and optimization tasks, but their direct application to physical problems has been limited by the lack of integration of physical laws, affecting precision.

This work employs an innovative solution: physics informed neural networks, which incorporate physical knowledge into their structure to better model heat propagation. It is demonstrated how these networks improve temperature prediction in one-dimensional systems.

Índice general

Índice de figuras

Índice de tablas

Aspectos preliminares

Planteamiento del problema

La transferencia de calor en una dimensión es un problema físico complejo cuando se trata de situaciones prácticas en las que se consideran múltiples factores como propiedades de materiales o condiciones iniciales y de frontera. Aunque las redes neuronales artificiales han demostrado ser eficaces en tareas de aprendizaje automático, su capacidad para resolver problemas de transferencia de calor en una dimensión también puede ser limitada debido a que carecen de una comprensión interna de las ecuaciones físicas subyacentes que rigen estos problemas.
Por lo tanto, la necesidad de desarrollar redes neuronales físicamente informadas que integren las características físicas sigue siendo pertinente en el contexto de la transferencia de calor en una dimensión. Al hacerlo, se puede mejorar la precisión y la efectividad de las soluciones generadas, lo que es esencial en aplicaciones que involucran la gestión térmica de componentes electrónicos, diseño de sistemas de calefacción o refrigeración, análisis de materiales conductores de calor, entre otros. La combinación de aprendizaje automático y conocimiento físico en estas redes puede conducir a una mejora significativa en la precisión de las predicciones de la transferencia de calor en una dimensión. Por lo tanto, en este trabajo, se plantea dar solución a la ecuación de calor en una dimensión estacionaria empleando una red neuronal físicamente informada.

Justificación

La transferencia de calor es un fenómeno crucial en una amplia gama de aplicaciones de ingeniería y física aplicada, desde la gestión térmica de componentes electrónicos en dispositivos modernos hasta la optimización de procesos de manufactura en la industria. La capacidad de modelar y predecir la distribución de temperatura en sistemas es esencial para el diseño eficiente, la toma de decisiones informadas y la prevención de problemas relacionados con la temperatura. Por lo tanto, resolver la ecuación de calor en contextos específicos es de gran relevancia.
La elección de abordar esta tarea mediante el uso de redes neuronales físicamente informadas agrega un elemento innovador y actual a la investigación en esta área. La intersección de principios físicos y técnicas de aprendizaje automático representa una convergencia multidisciplinaria que se encuentra en la vanguardia de la investigación y la aplicación práctica.

Hipótesis

Se espera que la implementación de una red neuronal físicamente informada solucione de manera efectiva y precisa la ecuación de calor unidimensional estacionaria al combinar la capacidad de aprendizaje automático de las redes neuronales con la incorporación de las características físicas del problema en comparación con enfoques tradicionales de resolución de problemas físicos.

Objetivos

Objetivo general

Desarrollar y aplicar una metodología basada en redes neuronales físicamente informadas para la solución eficiente y precisa de la ecuación de calor estacionaria en una dimensión, con el propósito de modelar y predecir la distribución de temperatura en sistemas unidimensionales, aplicando conocimientos de física y aprendizaje automático.

Objetivos específicos

- Desarrollar una metodología de resolución que integre redes neuronales y principios físicos.
- Diseñar y configurar una arquitectura de red neuronal adecuada para el problema de la ecuación de calor en una dimensión.
- Formular una función de pérdida que refleje las restricciones físicas y las condiciones iniciales/frontera.
- Recopilar y preparar datos de entrenamiento relevantes para la aplicación.
- Implementar el proceso de entrenamiento de la red, incluyendo parámetros y condiciones de entrenamiento.
- Evaluar la precisión de la red neuronal en la resolución de la ecuación de calor.
- Analizar y discutir los resultados obtenidos en términos de su coherencia con las restricciones físicas y su utilidad práctica.
- Realizar ajustes y mejoras en la metodología y la red neuronal si es necesario para abordar cualquier limitación o desafío identificado durante la investigación.

Metas

- Desarrollar una metodología efectiva para la solución de la ecuación de calor estacionaria en una dimensión.
- Diseñar una arquitectura de red neuronal eficiente.
- Formular una función de pérdida coherente con las restricciones físicas.
- Obtener resultados precisos y eficientes en la predicción de la distribución de temperatura.
- Contribuir al avance del conocimiento en la intersección de la física aplicada y el aprendizaje automático.

Metodología

Para cumplir con el objetivo general y los objetivos específicos planteados se siguió la metodología que se observa en la figura 1 y que se describe posteriormente.

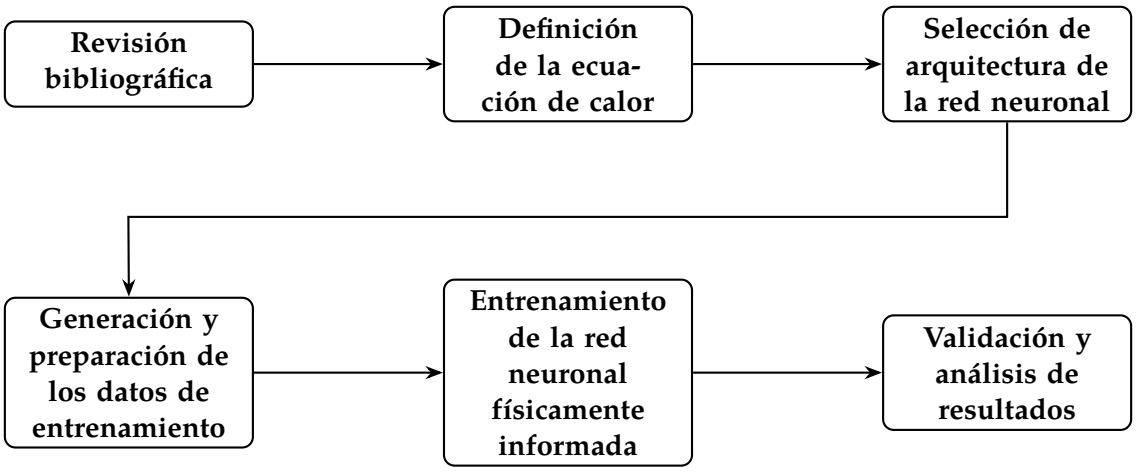


Figura 1: Metodología seguida para realizar este trabajo de tesis.

1. **Revisión bibliográfica**: Revisión de la literatura relacionada con la resolución de ecuaciones diferenciales, especialmente la ecuación del calor, mediante el uso de métodos numéricos y con redes neuronales. Se examinan investigaciones previas que han abordado problemas similares.

2. **Definición de la ecuación de calor**: En esta etapa, se identifica la ecuación diferencial correspondiente a la transferencia de calor unidimensional en estado estacionario. Esto incluye también las especificaciones de las condiciones iniciales y de frontera necesarias.

3. **Selección de arquitectura de la red neuronal**: En este paso, se selecciona la arquitectura de la red neuronal que mejor se adapte al problema en cuestión. Considerando aspectos como el número de capas, el tipo de neuronas, la función de activación, etcétera.

4. **Generación y preparación de datos de entrenamiento**: Para alimentar el proceso de entrenamiento de la red, se genera un conjunto de datos que contiene información relevante sobre las condiciones iniciales y/o frontera y otros datos necesarios.

5. **Entrenamiento de la red neuronal físicamente informada**: Durante el proceso de entrenamiento se busca minimizar la función de pérdida definida, lo que implica que la red se ajusta a la ecuación física y se adapta a los datos de entrenamiento.

6. **Validación y análisis de resultados**: Finalmente, se validará el rendimiento de la red utilizando datos de prueba independientes. Si se observa una discrepancia en la precisión, se realizarán ajustes en los parámetros para mejorar la calidad de la red.

Introducción

La transferencia de calor representa un campo de estudio esencial en la física y la ingeniería, y desempeña un papel fundamental en innumerables aplicaciones industriales y científicas. Desde la gestión térmica de sistemas electrónicos hasta el diseño eficiente de sistemas de calefacción, la capacidad de predecir con precisión cómo el calor se propaga a lo largo de objetos es crítica para el funcionamiento eficiente de numerosos sistemas. Sin embargo, la resolución efectiva de problemas de transferencia de calor sigue siendo un desafío debido a la necesidad de equilibrar la precisión con la eficiencia computacional [2].

En los últimos años, las redes neuronales artificiales han demostrado ser herramientas poderosas en el campo del aprendizaje automático, capaces de abordar tareas complejas de predicción y optimización en una variedad de áreas. No obstante, su aplicación directa en problemas físicos ha enfrentado limitaciones derivadas de la falta de comprensión interna de las leyes físicas que rigen los fénomenos físicos. En consecuencia, las soluciones generadas por estas redes pueden carecer de la precisión necesaria en comparación con los enfoques tradicionales que incorporan explícitamente las ecuaciones físicas [3].

En este trabajo, se explora una solución innovadora para mejorar la resolución de problemas de transferencia de calor en una dimensión: las redes neuronales físicamente informadas. Estas redes integran el conocimiento físico directamente en su estructura, lo que les permite comprender y modelar con mayor precisión la propagación del calor. En esencia, combinan la potencia del aprendizaje automático con la profundidad de la comprensión física, lo que promete revolucionar la manera en que se abordan los problemas de naturaleza térmica [3, 4].

A lo largo de este trabajo, se examinarán los fundamentos de las redes neuronales físicamente informadas y se mostrará cómo pueden mejorar la precisión en la predicción de la temperatura en sistemas unidimensionales. Además, se analizarán las implicaciones de esta innovación en campos como la ingeniería, la física y la simulación térmica, y cómo podría cambiar la forma en que se abordan los problemas de transferencia de calor en una dimensión.

En resumen, este trabajo se sumerge en la sinergia entre el aprendizaje automático y la física, destacando cómo las redes neuronales físicamente informadas representan una nueva frontera en la resolución de problemas de transferencia de calor en una dimensión, con el potencial de ofrecer soluciones más efectivas y precisas en el mundo real.

Capítulo 1

Marco Teórico

1.1. Transferencia de calor y ecuación de calor

1.1.1. Transferencia de calor

La **transferencia de calor** es el proceso en el cual se transporta y transfiere energía de un sistema a otro debido a una diferencia de temperatura. Por lo tanto, si los medios o sistemas interactuantes se encuentran a la misma temperatura, no puede existir una transferencia neta de calor. Aquella energía que es transportada en este proceso, se le denomina **calor**. El calor es una cantidad física no observable a simple vista, sin embargo, se puede identificar y cuantificar mediante mediciones indirectas y análisis, para lo cual, es necesario recurrir a otras ramas de la ciencia [5, 6].

La transferencia de calor se encuentra en muchos sistemas y aparatos de ingeniería, como los condensadores, los radiadores, los hornos y colectores de energía solar, así como aparatos de uso doméstico, como acondicionadores de aire, refrigeradores, sistemas de calefacción, planchas y calentadores de agua. En la ingeniería, el problema clave para realizar el diseño de todos estos aparatos es la determinación de la tasa de transferencia de calor para una diferencia de temperatura especificada [5, 6].

1.1.2. Ecuación de calor unidimensional estacionaria

Un problema de transferencia de calor es unidimensional si la temperatura en el medio varía y se transfiere en una sola dirección; al mismo tiempo, la variación de temperatura y, como consecuencia, la transferencia de calor en otras direcciones es despreciable o cero [5].

De manera general, si la temperatura T está en función de la posición x y el tiempo t, tal que $T = T(x, t)$, la ecuación de calor en una dimensión está dada por:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\dot{e}_{gen}}{k} = \frac{1}{\alpha}\frac{\partial T}{\partial t}, \tag{1.1}$$

donde $\alpha = k/\rho C$ es la difusividad térmica del material, k es la conductividad térmica (cuyo valor es considerado constante), ρ es la densidad, C es el calor específico y $\dot{e}_{gen}$ es la razón de generación de calor en el material [5, 6].

Si el sistema se encuentra en estado estacionario, es decir, la temperatura no varía con el tiempo y no existe generación de calor interna, la ecuación de calor (1.1) se reduce a una ecuación diferencial ordinaria conocida como la ecuación de Laplace en una dimensión:

$$\nabla^2 T = \frac{d^2 T}{dx^2} = 0. \tag{1.2}$$

Se puede resolver la ecuación (1.2) integrando, obteniéndose la solución general dada por

$$T(x) = C_1 x + C_2. \tag{1.3}$$

Para obtener una solución única para un problema, no basta con especificar únicamente la ecuación diferencial que lo describe. Es necesario establecer ciertas condiciones, como el valor de la función o sus derivadas en un determinado punto de la variable independiente. Estas condiciones obligan a la solución a cumplir con ciertos requisitos en puntos específicos, lo que permite determinar valores únicos para las constantes arbitrarias y, por ende, obtener una solución única. Esta información adicional es proporcionada por separado en forma de condiciones iniciales o de frontera. Para describir completamente un problema de transferencia de calor, es necesario especificar dos condiciones en la frontera para cada dirección del sistema de coordenadas a lo largo de la cual la transferencia de calor es relevante [5].

1.2. Métodos numéricos

Resolver analíticamente la ecuación de calor puede ser desafiante en situaciones más complejas, por lo que se recurre a los métodos numéricos para obtener soluciones aproximadas. Estos métodos, al contrario de las soluciones analíticas, se centran en utilizar técnicas computacionales para simular el comportamiento de sistemas físicos en situaciones diversas. A continuación, se describe la metodología de algunos de los métodos numéricos más comunes y utilizados para la resolución de ecuaciones diferenciales.

1.2.1. Método de diferencias finitas

Este método discretiza tanto el espacio como el tiempo, aproximando las derivadas con diferencias finitas. La ecuación de calor se transforma en un sistema de ecuaciones algebraicas, que se resuelve de manera iterativa. Este método es simple y efectivo [7]. Los pasos del método de diferencias finitas incluyen:

1. **Discretización del Dominio:** Se selecciona un conjunto de puntos espaciados uniformemente en el dominio, llamados nodos.

2. **Aproximación de Derivadas:** Se utilizan expresiones algebraicas (diferencias finitas) para aproximar las derivadas de la función en términos de los valores en los nodos.

3. **Discretización de la Ecuación Diferencial:** Se sustituyen las aproximaciones de derivadas en la ecuación diferencial, generando una versión discreta de la ecuación.

4. **Formulación de un Sistema de Ecuaciones:** Se obtiene un sistema de ecuaciones algebraicas relacionando los valores de la función en los nodos.

5. **Aplicación de Condiciones de Contorno:** Imposición de restricciones y condiciones en el sistema global.

6. **Resolución del Sistema:** El sistema de ecuaciones se resuelve numéricamente para obtener los valores de la función en los nodos.

1.2.2. Método de los elementos finitos

El método de los elementos finitos divide el dominio en elementos más pequeños, aproximando la solución dentro de cada elemento mediante funciones de forma. Al ensamblar estos

elementos se obtiene un sistema de ecuaciones que se resuelve para obtener la temperatura en cada nodo. Este método es altamente versátil y se adapta bien a geometrías complejas [8]. Los pasos del método del elemento finito son:

1. **Discretización del Dominio:** División del dominio en elementos finitos.

2. **Definición de Variables y Ecuaciones:** Establecimiento de variables y ecuaciones.

3. **Formulación del Problema:** Expresión del problema para cada elemento finito.

4. **Ensamblaje Global:** Combinación de las ecuaciones de los elementos finitos en un sistema global.

5. **Aplicación de Condiciones de Contorno:** Imposición de restricciones y condiciones en el sistema global.

6. **Resolución del Sistema Global:** Solución del sistema global de ecuaciones.

En términos generales, la metodología de estos métodos implica dividir el dominio del problema en elementos más pequeños, formulando ecuaciones discretas que describen la evolución del sistema. Se incorporan condiciones iniciales y de frontera, y se utilizan métodos de integración para hallar la solución del sistema.

La elección de un método específico puede depender de la naturaleza del problema, y la implementación precisa puede requerir un equilibrio entre la precisión y la eficiencia computacional. En algunos casos, la convergencia puede ser un problema, y la elección inadecuada de parámetros o la falta de comprensión del comportamiento del sistema pueden llevar a soluciones no realistas o divergentes [2].

1.3. Redes neuronales

La eficacia de métodos numéricos tradicionales, como las Diferencias Finitas y los Elementos Finitos, es innegable, pero enfrentan desafíos particulares. Problemas complejos con geometrías irregulares, condiciones de contorno variables y comportamientos no lineales pueden hacer que las soluciones aproximadas no capturen completamente la realidad física. En este contexto, las redes neuronales artificiales (ANN, por sus siglas en inglés) emergen como una alternativa revolucionaria. Inspiradas en la capacidad del cerebro humano para aprender y adaptarse, estas redes han demostrado su eficacia en la resolución de problemas complejos y no lineales. A diferencia de los métodos numéricos tradicionales, las ANN no requieren la formulación explícita de ecuaciones diferenciales y pueden aprender patrones directamente de datos.

Las ANN son modelos computacionales inspirados en la estructura y funcionamiento del sistema nervioso biológico, específicamente en las redes de neuronas presentes en el cerebro. Estas redes se utilizan en aprendizaje automático y procesamiento de datos para realizar tareas como reconocimiento de patrones, clasificación, regresión y otras tareas relacionadas con la inteligencia artificial [3].

1.3.1. Redes neuronales biológicas

El sistema nervioso humano contiene células conocidas como **neuronas**, las cuales se encuentran conectadas a otras mediante los axones y las dendritas. Las **dendritas** son el extremo de entrada de una neurona. Existe una gran variedad de formas dendríticas, sin embargo, de manera general, son muy parecidas a las ramas de un árbol, varían en forma y tamaño, lo cual se ve reflejado en la capacidad de procesamiento de información de cada neurona. El cuerpo de la célula es llamado **soma**. Un **axón** es una fibra delgada tubular, se encuentra en el extremo de salida de una neurona, se ramifican en sus extremos en lo que se conoce como **arborización terminal** y forman conexiones con otras neuronas mediante las **sinapsis**, las cuales se encuentran en la parte terminal de los axones, como se observa en la figura 1.1. La fuerza de estas conexiones varía y cambia en respuesta a los estímulos externos presentes. Además, estas conexiones permiten que una neurona mande impulsos o descargas eléctricas a las células con las que se encuentra conectada. El impulso es transportado por el soma hasta el axón, el cual se encarga de seguir transmitiendo descargas a las neuronas vecinas cuando se acumula cierta cantidad de energía [9, 10, 11].

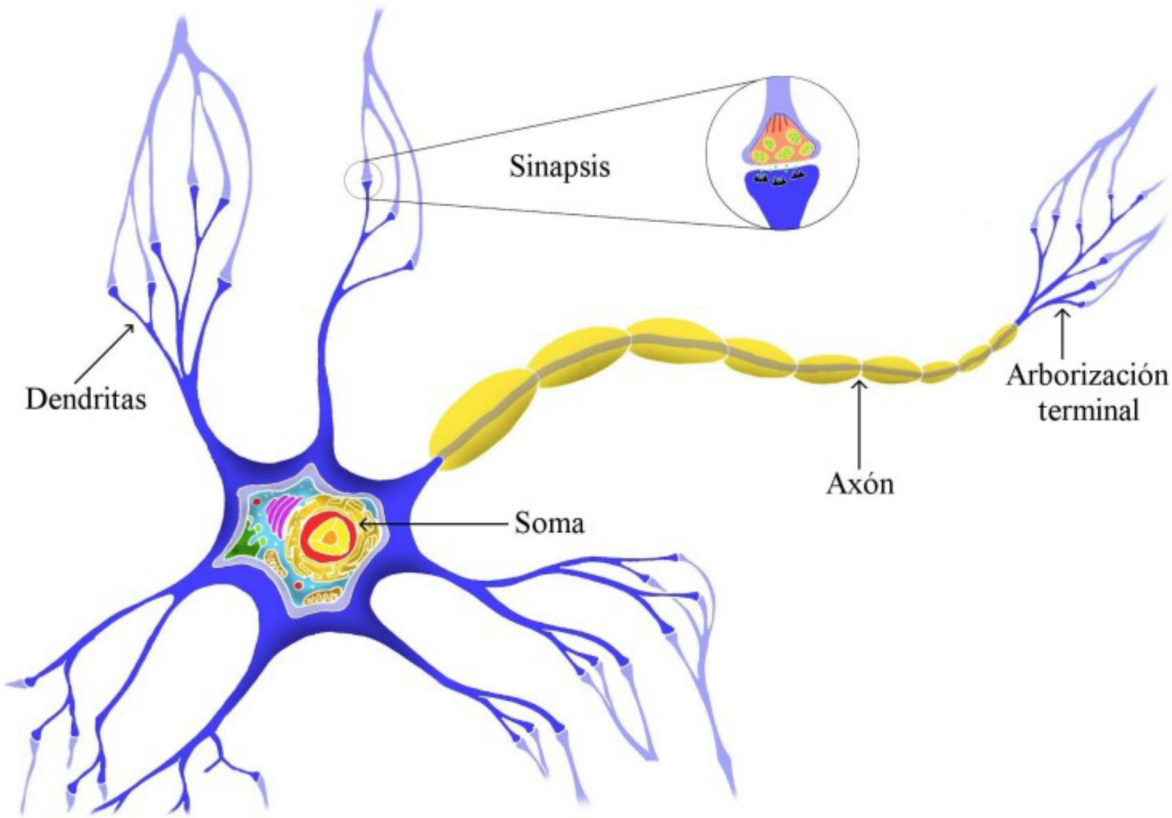


Figura 1.1: Neurona biológica [1].

La neurona recibe impulsos de las entradas procedentes de sinapsis. La célula se activa o no dependiendo de si la suma de estos impulsos en el soma supera un valor característico.

1.3.2. Redes neuronales artificiales

Las ANN son una técnica del aprendizaje automático que pretende simular el aprendizaje de los organismos y seres biológicos [10]. Estas redes están organizadas o compuestas por capas de neuronas interconectadas. Generalmente, hay una capa de entrada, una o varias capas ocultas y una capa de salida, como se muestra en la figura 1.2.

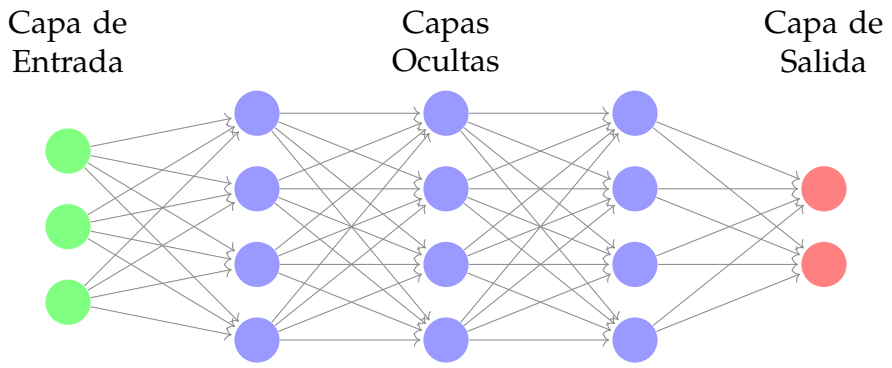


Figura 1.2: Estructura de una red neuronal artificial.

La información fluye desde la capa de entrada, a través de las capas ocultas, y finalmente produce una salida en la última capa.

1.3.3. Neurona artificial

Las ANN, al igual que las redes neuronales biológicas, contienen neuronas artificiales, las cuales se encargan de realizar cálculos. Una neurona es una unidad de procesamiento de información fundamental para el funcionamiento de las ANN [11].
El modelo más simple de una neurona artificial, incluye de manera análoga a las neuronas biológicas, las señales que recibe de otras neuronas, la intensidad de la sinapsis y una función que define si se activa o no en caso de alcanzar un umbral o valor. En una neurona artificial se tienen los siguientes elementos:

- Una **entrada** x_j.
- Un **conjunto de sinapsis** caracterizadas por un **peso sináptico** propio. Una entrada x_j en la sinapsis j conectada a la neurona k se multiplica por el peso sináptico w_{kj} donde k, corresponde a la neurona en cuestión y j se refiere al extremo de la sinapsis j.
- Un **sumador** que se encarga de sumar las entradas que recibe, ponderadas por los respectivos pesos de cada neurona.
- Una **función de activación** que limita la amplitud de la salida de la neurona.

El modelo de una neurona artificial simple, incluye también un **sesgo** aplicado, b_k, como se observa en la figura 1.3. Dependiendo de si el sesgo b_k es positivo o negativo, la entrada neta que recibe la función de activación aumenta o disminuye [11].

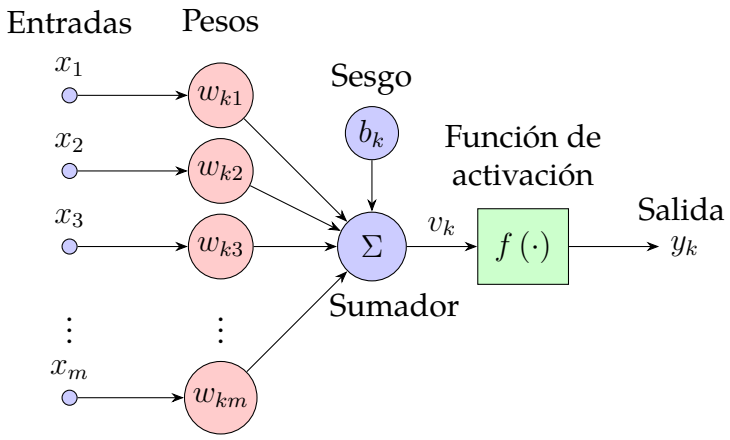


Figura 1.3: Modelo de neurona artificial.

De manera matemática, la salida de la neurona artificial mostrada en la figura 1.3 está dada por:

$$y_k = f\left(v_k + b_k\right),$$

donde,

$$v_k = \sum_{j=1}^{m} w_{kj} x_j.$$

1.3.4. Función de activación

La función de activación en una red neuronal es una función matemática aplicada a la salida de cada neurona, determina si la neurona debe "activarse"y transmitir su señal a la siguiente capa de la red. La función de activación f define la salida de la neurona en términos de la entrada neta v.

La no linealidad introducida por las funciones de activación es esencial para que las redes neuronales puedan aproximar funciones no lineales y complejas [12]. Algunas de las funciones de activación aplicadas a una salida v más comunes son:

- **Función umbral**

$$f(v) = \begin{cases} 1, \text{ si } v \geq 0, \\ 0, \text{ si } v < 0\,. \end{cases}$$

- **Función de rectificación lineal (ReLU)**

$$f(v) = \begin{cases} v, \text{ si } v > 0, \\ 0, \text{ si } v \leq 0\,. \end{cases}$$

- **Función sigmoide**: La función sigmoide, transforma la entrada en un valor en el rango $(0, 1)$:

$$f(v) = \frac{1}{1 + e^{-v}}\,.$$

- **Función *tanh***: La función tangente hiperbólica transforma la entrada en un valor en el rango $(-1, 1)$:

$$f(v) = \frac{e^{-2v} - 1}{e^{-2v} + 1}\,.$$

La elección de la función de activación impacta en la capacidad del modelo para aprender y generalizar a partir de los datos. Algunas funciones, como la sigmoide o la tangente hiperbólica, fueron populares en el pasado, pero en la actualidad, la rectificación lineal (ReLU) es preferida debido a la facilidad para entrenar redes neuronales multicapa con esta función de activación y a su eficacia en la convergencia durante el entrenamiento [10, 11].

1.3.5. Arquitecturas de las redes neuronales artificiales

La forma en que están conectadas las neuronas de una red neuronal está íntimamente ligada al algoritmo de aprendizaje utilizado para entrenar la red [11]. En general, se distinguen tres tipos de arquitecturas (estructuras) de redes neuronales diferentes:

- **Redes de capa simple con conexiones hacia adelante**:
  Este tipo de red consta de una única capa de neuronas, y cada neurona está conectada a todas las entradas, formando así una estructura de conexiones hacia adelante. La información fluye directamente desde las entradas a través de la capa de neuronas hacia las salidas. Aunque estas redes son sencillas, su capacidad para capturar patrones complejos puede ser limitada.

- **Redes multicapa con conexión hacia adelante**:
  Las redes multicapa, o redes neuronales feedforward, constan de múltiples capas de neuronas organizadas en una estructura de capas interconectadas. La información fluye en una dirección, desde la capa de entrada a través de las capas ocultas hasta la capa de salida. La introducción de capas ocultas permite a estas redes aprender representaciones más abstractas y complejas de los datos, lo que las hace más adecuadas para tareas sofisticadas de aprendizaje.

- **Redes recurrentes**:
  Las redes recurrentes incorporan conexiones que permiten la retroalimentación de las salidas de la red hacia sus propias entradas, creando ciclos en la estructura. Esta capacidad de mantener y utilizar información previa en el proceso de toma de decisiones hace que las redes recurrentes sean particularmente útiles para modelar secuencias temporales y dependencias a largo plazo.

Cada una de estas arquitecturas tiene aplicaciones específicas y ventajas en función de la naturaleza del problema que se está abordando. La elección de la arquitectura de la red neuronal es esencial para lograr un rendimiento óptimo en tareas de aprendizaje automático, ya que la estructura de la red influye directamente en su capacidad para interpretar patrones complejos en los datos [11].

1.3.6. Aprendizaje y entrenamiento

Aprendizaje

El aprendizaje en redes neuronales se fundamenta en el ajuste de pesos y sesgos para mejorar el rendimiento de la red mediante la evaluación con datos y retroalimentación. Este proceso se lleva a cabo a través de un conjunto de datos representativo, permitiendo enseñar a la red a través de diferentes enfoques de aprendizaje [11, 1].

Una de las clasificaciones que se realiza en las redes neuronales corresponde al tipo de aprendizaje que utilizan, existen tres tipos de aprendizajes:

- **Aprendizaje supervisado**: Este tipo de redes neuronales requieren datos previamente etiquetados para llevar a cabo su proceso de aprendizaje. En otras palabras, se les proporciona información de entrada junto con la correspondiente salida esperada. La red ajusta sus pesos y conexiones internas con el objetivo de minimizar la discrepancia entre sus predicciones y las etiquetas proporcionadas. El aprendizaje supervisado es común en tareas de clasificación y regresión [11, 1].

- **Aprendizaje no supervisado**: Las redes neuronales que utilizan aprendizaje no supervisado no dependen de la presencia de un asesor externo que proporcione etiquetas para los datos de entrenamiento. Estas redes buscan patrones o estructuras inherentes en los datos sin conocimiento previo de las salidas deseadas. El aprendizaje no supervisado es útil en la identificación de agrupamientos, reducción de dimensionalidad y otras tareas donde la estructura subyacente de los datos es desconocida [1].

- **Aprendizaje por refuerzo**: En el aprendizaje por refuerzo, las redes neuronales interactúan con un entorno y toman decisiones en función de las entradas recibidas. Posteriormente, son recompensadas o penalizadas de acuerdo con la calidad de sus decisiones. El objetivo es maximizar la recompensa acumulada a lo largo del tiempo. Este tipo de aprendizaje se encuentra comúnmente en aplicaciones como juegos, robótica y sistemas de control, donde la red aprende a través de la retroalimentación continua de su desempeño [11].

Estos distintos tipos de aprendizaje ofrecen maneras específicas de entrenar a las redes neuronales, cada una con sus propias áreas de aplicación. La elección de la forma de aprendizaje más apropiada se basa en las características particulares del problema que se está tratando y en la disponibilidad de datos para entrenar la red [10, 11].

Entrenamiento

El proceso de entrenamiento implica alimentar a la red con datos de entrada y ajustar los pesos para minimizar la diferencia entre las salidas predichas y las salidas reales. El algoritmo de retropropagación es comúnmente utilizado para este propósito [10].

Durante el entrenamiento, la red neuronal pasa por múltiples épocas. Cada época representa una iteración completa sobre todo el conjunto de datos de entrenamiento. En cada época, los pesos de la red son ajustados de acuerdo con el error calculado, y el objetivo es reducir progresivamente la función de pérdida. En redes neuronales profundas, las épocas permiten que la red aprenda gradualmente representaciones más complejas de los datos.

En el caso de una red neuronal con una sola capa, el proceso de entrenamiento resulta sencillo, ya que el error o función de pérdida se puede calcular directamente en función de los pesos, lo que simplifica el cálculo del gradiente. Sin embargo, en redes de varias capas,

surge una complicación debido a que la pérdida es una función compleja que resulta de la composición de pesos en capas previas. Para abordar esta complejidad, se recurre al uso del algoritmo de retropropagación, el cual se fundamenta en la aplicación de la regla de la cadena del cálculo diferencial.

1.3.7. Backpropagation

El algoritmo de retropropagación o **backpropagation** (abreviatura de "backward propagation of errors") es una técnica fundamental utilizada para entrenar redes neuronales artificiales. Fue introducido por [13] como un método eficiente para ajustar los pesos de la red, con el objetivo de minimizar el error de predicción mediante el cálculo del gradiente de la función de pérdida con respecto a los pesos.

El algoritmo de backpropagation es un algoritmo utilizado para entrenar redes neuronales. Es una técnica de optimización que ajusta los pesos de la red neuronal para minimizar la diferencia entre las salidas predichas y las salidas deseadas.

El algoritmo de backpropagation consta de dos etapas o fases:

- **Propagación hacia adelante**: Durante esta fase, las entradas de una instancia de entrenamiento se introducen en la red neuronal. A través de una secuencia de cálculos que involucran los pesos actuales, la información se propaga hacia adelante a través de las capas de la red, generando una predicción final. Esta predicción se compara con la salida real, y se calcula la derivada de la función de pérdida con respecto a la salida [10, 11].

- **Propagación hacia atrás**: En esta etapa, se calculan las derivadas parciales de la función de pérdida con respecto a los pesos en todas las capas. Estas derivadas se utilizan para ajustar los pesos, contribuyendo así a la mejora del rendimiento de la red en función de la retroalimentación proporcionada durante la fase hacia adelante. Este proceso de ajuste iterativo es fundamental para el aprendizaje de la red neuronal [10].

Estas fases son esenciales para ajustar los pesos de la red y minimizar la función de pérdida, facilitando así la mejora en la capacidad predictiva y de aprendizaje del modelo neuronal. Este ciclo de propagación hacia adelante y hacia atrás se repite hasta que la red alcance un rendimiento deseado [10, 11].

Para un análisis más detallado de las etapas y los cálculos involucrados en el algoritmo de **backpropagation**, se puede consultar el apéndice A.

1.3.8. Optimizador

El optimizador es una parte esencial en el entrenamiento de redes neuronales, ya que es responsable de ajustar los pesos y sesgos del modelo para minimizar la función de pérdida, que mide el error entre las predicciones de la red y los valores reales. El proceso de ajuste se realiza de forma iterativa, con el objetivo de mejorar la precisión de las predicciones al modificar los parámetros de la red de manera eficiente [14].

El optimizador se basa en los gradientes de la función de pérdida con respecto a los pesos de la red neuronal. Estos gradientes indican la dirección y magnitud en la que deben modificarse los parámetros para reducir el error. De esta manera, el optimizador busca la configuración de pesos que minimice el error, lo que lleva a una red más precisa [15].

Existen diversos tipos de optimizadores que varían en su método de ajuste de los pesos. Entre los más utilizados se encuentran:

- **Gradiente Descendiente Estocástico (SGD)**: Este es uno de los optimizadores más básicos, que ajusta los pesos utilizando el gradiente de la pérdida sobre pequeños lotes de datos en lugar de todo el conjunto, lo que acelera el proceso de entrenamiento [14].
- **Momentum**: Extiende el SGD al acumular el gradiente de las iteraciones anteriores para suavizar la convergencia y evitar que el modelo se quede atrapado en mínimos locales [15].
- **Adam (Adaptive moment estimation)**: Uno de los optimizadores más avanzados y populares, combina la ventaja del Momentum y la tasa de aprendizaje adaptativa, ajustando el tamaño del paso para cada peso de manera automática según los gradientes acumulados. Adam es eficaz en la mayoría de los problemas de aprendizaje profundo debido a su capacidad para converger rápidamente y manejar problemas complejos [16]. Se explica más a detalle en el Apéndice B.

La elección del optimizador es un factor clave en el rendimiento de la red, ya que afecta tanto la velocidad de convergencia como la precisión final del modelo [15].

1.3.9. Tasa de aprendizaje

La tasa de aprendizaje es un hiperparámetro crucial en el entrenamiento de redes neuronales, ya que controla el tamaño de los pasos que el optimizador da para ajustar los pesos y minimizar la función de pérdida [14].

Una tasa adecuada asegura una convergencia eficiente hacia una solución óptima. Si es demasiado alta, puede provocar una convergencia inestable, mientras que si es demasiado baja, el aprendizaje será lento y puede quedar atrapado en mínimos locales, impidiendo una mejora significativa en el modelo.

1.4. Redes Neuronales Físicamente Informadas

La idea de resolver ecuaciones diferenciales utilizando redes neuronales se planteó hace ya más de 25 años [17], derivado del hecho de que las redes multicapa con propagación hacia adelante son aproximadores universales [18]. Sin embargo, en el año 2017 se publicó un artículo en dos partes, reducido a un único artículo en el 2019, en el cual se presentaron las **redes neuronales físicamente informadas** como un nuevo tipo de solucionador basado en datos [19].

Las redes neuronales físicamente informadas (**PINNs**, por sus siglas en inglés, Physics-Informed Neural Networks) son un enfoque específico en el diseño de redes neuronales que incorporan principios físicos o conocimientos a priori sobre un sistema físico específico en su estructura y entrenamiento.

La integración de conocimientos físicos en el diseño de estas redes no sólo aporta precisión a los resultados, sino que también ofrece la posibilidad de interpretar y comprender de manera más profunda los procesos subyacentes en situaciones donde las leyes físicas son bien conocidas y fundamentales para el comportamiento del sistema [4].

De manera general, las PINN pueden resolver ecuaciones diferenciales expresadas de la forma:

$$\begin{aligned} F\left(u(z);\gamma\right) &= f(z), \quad z \in \Omega, \\ B\left(u(z)\right) &= g(z), \quad z \in \partial\Omega, \end{aligned} \tag{1.4}$$

donde $\Omega \subset \mathbb{R}^d, d \in N$ y $\partial\Omega$ denota la frontera. Además, $z = [x_1, x_2, ..., x_{d-1}; t]$ es el vector con coordenadas espacio-tiempo, mientras que u es la función desconocida que será aproximada, γ representa los parámetros físicos del problema, F es un operador diferencial, f representa las condiciones iniciales, B es otro operador para las condiciones de frontera y g la función en la frontera [4].
La ANN que aproxima a la función $u(z)$ se denota por $\mathbf{NN}_\theta(z)$, donde θ representa el conjunto de parámetros con los que se aproxima $u(z)$. Es decir,

$$\mathbf{NN}_\theta(z) \approx u(z).$$

Finalmente, lo que realiza la ANN es aprender los parámetros θ que mejor aproximen la solución de la ecuación diferencial mediante el proceso de entrenamiento, esto se realiza minimizando una función de pérdida.

1.4.1. Función de pérdida

La función de pérdida que guía el entrenamiento de la red incluye términos que penalizan la desviación de la ecuación diferencial, las condiciones iniciales y de frontera y los datos conocidos correspondientes al dominio Ω. La función de pérdida total $\mathcal{L}$ es la suma de estos tres términos:

$$\mathcal{L} = \mathcal{L}_{DE} + \mathcal{L}_c + \mathcal{L}_{data}, \tag{1.5}$$

donde

$$\mathcal{L}_{\text{DE}} = \frac{1}{N_{DE}} \sum_{j=1}^{N_{DE}} |F(\mathbf{NN}_\theta(z)) - f(z_j)|^2, \tag{1.6}$$

$$\mathcal{L}_{\text{c}} = \frac{1}{N_c} \sum_{j=1}^{N_c} |B(\mathbf{NN}_\theta(z)) - g(z_j)|^2, \tag{1.7}$$

$$\mathcal{L}_{\text{data}} = \frac{1}{N_d} \sum_{j=1}^{N_d} |\mathbf{NN}_\theta(z) - u(z_j)|^2, \tag{1.8}$$

$\mathcal{L}_{DE}$ es el término que satisface la ecuación diferencial, $\mathcal{L}_c$ penaliza el error de las condiciones iniciales y de frontera y $\mathcal{L}_{data}$ indica la discrepancia entre las predicciones y los datos obtenidos mediante los valores conocidos. Las constantes N_{DE}, N_c y N_d indican la cantidad de puntos o datos para cada término.

Se minimiza la función de pérdida total utilizando el conjunto de datos preparados para realizar el proceso de entrenamiento. Durante el entrenamiento, la red ajusta sus pesos y conexiones para minimizar la función de pérdida, incorporando simultáneamente las restricciones físicas definidas en las ecuaciones. Finalmente, se realiza la validación del rendimiento de la red utilizando conjuntos de datos no utilizados durante el entrenamiento y si es necesario, se ajustan los parámetros para mejorar el rendimiento de la red [19, 4].

1.4.2. Aplicaciones y problemas resueltos con las PINN

En los últimos años, las PINN se han empleado para resolver una amplia variedad de ecuaciones diferenciales, desde ecuaciones diferenciales ordinarias (ODE), así como ecuaciones diferenciales parciales (PDE) [4]. A continuación se muestran algunas ecuaciones resueltas mediante el uso de la metodología e implementación de las PINN.

Ecuación de Schrödinger

En [19] se resuelve la ecuación de Schrödinger unidimensional no lineal:

$$i\frac{\partial \psi}{\partial t} + \frac{1}{2}\frac{\partial^2 \psi}{\partial x^2} + |\psi|^2\psi = 0,$$

donde ψ es la solución de valor complejo. Esta es una ecuación de campo clásica utilizada para estudiar sistemas mecánico cuánticos, incluida la propagación de ondas no lineales en fibras ópticas y/o guías de ondas, condensados de Bose Einstein y ondas de plasma.
Con este ejemplo, Raissi et al. resaltan la capacidad de las PINN para manejar condiciones de frontera periódicas, soluciones de valores complejos, así como diferentes tipos de no linealidades en las ecuaciones diferenciales parciales gobernantes.

Ecuación eikonal

En [20] se utiliza la metodología de las PINN para resolver la ecuación eikonal, la cual es una ecuación diferencial parcial de la forma:

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

donde v es la velocidad y $u(z)$ es un tiempo de activación desconocido.

A través de pruebas en modelos sintéticos de referencia, se demuestra que la precisión del enfoque de las PINN es mejor que la solución obtenida con algunos de los algoritmos más utilizados para resolver la ecuación eikonal.
Además, a diferencia de los métodos convencionales, la naturaleza sin malla del método propuesto permite una fácil incorporación de la topografía de la superficie, que suele ser una consideración importante para los datos sísmicos terrestres [20].

Ecuación de advección-dispersión

En [21] se abordan múltiples ecuaciones de advección-dispersión dadas por la ecuación (1.4.2).

$$\frac{\partial u}{\partial t} + \nabla \cdot (-\mathbf{D}\nabla u + \mathbf{v}u) = s.$$

Los autores encuentran que el método de las PINN es preciso y produce resultados superiores a los obtenidos utilizando métodos típicos basados en discretización.

Capítulo 2

Metodología y desarrollo

En este capítulo, se describe y detalla el proceso seguido para realizar la implementación de la metodología de una PINN para resolver la ecuación de calor unidimensional estacionaria.

En primer lugar, se describe el problema abordado, seguido de un resumen de la metodología utilizada en la configuración y entrenamiento de cada modelo de PINN, abarcando sus arquitecturas, los parámetros e hiperparámetros seleccionados.

Posteriormente, se describe el proceso de generación de datos para entrenar la PINN y el proceso de entrenamiento que incluye la implementación de una función de pérdida específica para el problema.

2.1. Definición del problema

A continuación se procede con la descripción del problema abordado y las características que se consideraron para dar solución a la ecuación de calor unidimensional estacionaria.

Recordando, la ecuación de calor unidimensional estacionaria está dada por la ecuación (1.2):

$$\frac{d^2T(x)}{dx^2} = 0, \quad x \in [0, L],$$

donde $T(x)$ es la función de temperatura que se busca determinar en el intervalo $[0, L]$. Para este trabajo, se utilizaron las siguientes condiciones de frontera: $T_0 = 5$ en $x = 0$, y $T_f = 10$ en $x = 1$. Con estos valores, la ecuación (1.3) se reescribe como

$$T(x) = \left(\frac{T_f - T_0}{L}\right) x + T_0 = 5x + 5. \tag{2.1}$$

El objetivo de este trabajo es desarrollar y aplicar una metodología basada en redes neuronales físicamente informadas (PINNs) que permita resolver de manera eficiente y precisa la ecuación de calor unidimensional estacionaria. La implementación incluye diversas configuraciones de PINNs, que integran la ecuación diferencial (1.2) y las condiciones de frontera en su función de pérdida para aproximar la solución $T(x)$. Cada modelo es evaluado y comparado con la solución exacta dada por la ecuación (2.1), lo que permite analizar la efectividad de las configuraciones para modelar y predecir la distribución de temperatura en sistemas unidimensionales. Este análisis facilita la identificación de configuraciones óptimas y proporciona una comprensión de cómo los parámetros de la red influyen en la precisión de las predicciones.

Para más detalles sobre la implementación de estos parámetros y la solución exacta, consulte el código en el Apéndice C.

2.2. Implementación de PINN

La implementación de la metodología basada en PINNs se llevó a cabo utilizando la biblioteca TensorFlow. El proceso incluyó la configuración de diferentes arquitecturas de red, la implementación de una función de pérdida física para asegurar que la solución cumpliera con las ecuaciones diferenciales y las condiciones de frontera, y la evaluación de los modelos generados en términos de precisión y eficiencia.

2.2.1. TensorFlow

TensorFlow es una biblioteca de código abierto para el aprendizaje automático y el aprendizaje profundo, desarrollada por Google [22]. Su flexibilidad y escalabilidad la han convertido en una de las herramientas más populares para implementar redes neuronales en diversos campos, incluidos la física y la ingeniería. En este trabajo, se utilizó TensorFlow para implementar y evaluar varias configuraciones de PINNs con el objetivo de resolver la ecuación de calor unidimensional estacionaria.

El entorno de desarrollo fue configurado con TensorFlow versión 2.17.0 en Google Colab, lo que facilitó tanto el desarrollo como la visualización de los resultados y la experimentación con distintas arquitecturas de red. La arquitectura de cada PINN fue diseñada utilizando la API de Keras de TensorFlow, ajustando los hiperparámetros para encontrar la configuración que ofreciera el mejor equilibrio entre precisión y eficiencia en la predicción de la distribución de temperatura.

2.3. Configuración de modelos de PINNs

Se implementaron cinco modelos de red neuronal con configuraciones variadas. Las características principales de cada red son:

- **Arquitectura:** Las redes implementadas son redes multicapa completamente conectadas (conexiones hacia adelante), tienen **entre 2 y 3 capas ocultas**, con variaciones en el número de neuronas por capa (entre 20 y 100).
- **Función de activación:** Se utilizaron distintas funciones de activación para las capas ocultas: relu, sigmoide y *tanh*.
- **Inicialización de pesos y sesgos:** Los **pesos y sesgos** fueron inicializados de manera aleatoria.
- **Optimización:** Para todos los modelos se utilizó el optimizador **Adam**, configurado con una tasa de aprendizaje inicial de 0.001.

La cantidad de capas, neuronas por capa y la función de activación fueron seleccionadas sin seguir un criterio específico, con el propósito de explorar el desempeño de las redes bajo diferentes configuraciones. Esto permite analizar cómo estas características impactan en la precisión y capacidad de generalización de los modelos.

Cada una de las cinco configuraciones fue entrenada durante 500 épocas y evaluada para determinar su precisión en la predicción de la distribución de temperatura.

Esta evaluación permitió seleccionar la configuración que ofrece el mejor equilibrio entre precisión y tiempo de entrenamiento, considerando tanto la capacidad de cada modelo para cumplir con las condiciones de frontera como su exactitud en la resolución del problema.

Las características principales de estos modelos se resumen en la Tabla 2.1.

Tabla 2.1: Configuración de parámetros principales para los cinco modelos de red neuronal.

Modelo	**Capas ocultas**	**Neuronas por capa**	**Función de activación**
PINN 1	2	50	$sigmoide$
PINN 2	2	100 y 50	$ReLU$
PINN 3	3	50	$tanh$
PINN 4	2	20	$ReLU$
PINN 5	3	70, 40 y 20	$sigmoide$

2.4. Generación de los datos de entrenamiento

Para generar los datos de entrenamiento necesarios para resolver la ecuación de calor unidimensional estacionaria, se utilizó una estrategia basada en la generación de puntos aleatorios dentro del dominio $[0, 1]$ y la aplicación de las condiciones de frontera.

Los pasos seguidos para la generación de datos fueron los siguientes:

- **Puntos interiores:** Se generaron 100 puntos distribuidos uniformemente en el intervalo $[0, 1]$, empleando una distribución aleatoria uniforme. Para cada punto, se calculó la temperatura exacta $T(x)$ utilizando la solución analítica de la ecuación de calor. Esta solución representa una interpolación lineal entre las temperaturas de las fronteras T_0 y T_f.

- **División de datos:** Los puntos generados se separaron en conjuntos de entrenamiento y validación. El $80\,\%$ de los datos se asignaron al conjunto de entrenamiento, mientras que el $20\,\%$ restante conformó el conjunto de validación. Esta división se realizó de forma aleatoria mediante la permutación de índices, garantizando una selección imparcial.

- **Condiciones de frontera:** Las condiciones de frontera se incorporaron en la función de pérdida de la red neuronal. En los extremos del dominio ($x = 0$ y $x = L$), se penalizó a la red para que las predicciones coincidieran con los valores impuestos T_0 y T_f, asegurando el cumplimiento de estas restricciones.

La generación y partición de los datos garantiza que la red neuronal pueda aprender la solución de la ecuación y, al mismo tiempo, verificar su capacidad de generalización. Este enfoque permite que la red neuronal cumpla tanto con las restricciones físicas del problema como con la representación adecuada de la distribución de temperatura en el dominio.

Para ver los detalles sobre la implementación de la generación de datos y la partición en conjuntos de entrenamiento y validación, consulte el Apéndice D.

2.5. Proceso de entrenamiento

El entrenamiento de los modelos PINN se llevó a cabo minimizando una función de pérdida diseñada para incorporar tanto la ecuación diferencial como las condiciones de frontera impuestas. Esta función de pérdida está definida como:

$$\mathcal{L} = \mathcal{L}_{\text{ED}} + \mathcal{L}_{\text{c}}, \tag{2.2}$$

donde:

$$\mathcal{L}_{\text{ED}} = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{d^2 \mathbf{NN}(x_i)}{dx^2} \right)^2 \tag{2.3}$$

representa el error asociado al cumplimiento de la ecuación diferencial, y:

$$\mathcal{L}_{\text{c}} = \left(\mathbf{NN}(0) - T_0 \right)^2 + \left(\mathbf{NN}(L) - T_f \right)^2 \tag{2.4}$$

corresponde a los términos que penalizan el incumplimiento de las condiciones de frontera.

En estas ecuaciones, N es el número de puntos de muestreo en el dominio, y $\mathbf{NN}(x_i)$ denota la predicción de la red neuronal en el punto x_i. El término $\mathcal{L}_{\text{ED}}$ mide el error de la red neuronal al aproximar la solución de la ecuación diferencial (en este caso, la ecuación de calor unidimensional estacionaria), mientras que $\mathcal{L}_{\text{c}}$ asegura que las predicciones de la red en los extremos del dominio $x = 0$ y $x = L$ respeten las temperaturas impuestas, T_0 y T_f, respectivamente.

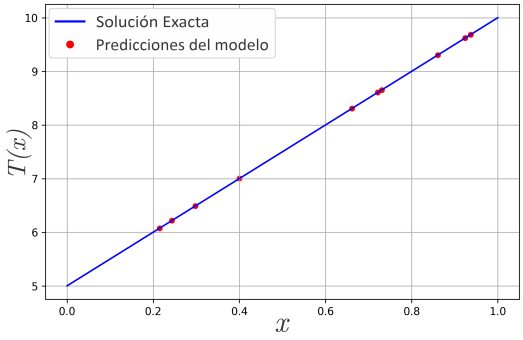
El entrenamiento fue realizado utilizando el optimizador Adam con un valor de tasa de aprendizaje inicial de 0.001. La implementación detallada de esta función de pérdida se presenta en el Apéndice E, y el procedimiento completo para entrenar y evaluar los modelos PINN, incluyendo el cálculo de pérdidas en cada iteración y la medición del tiempo total, se describe en el Apéndice F. Esta formulación asegura que la red neuronal no solo respete las restricciones impuestas por la ecuación diferencial, sino que también cumpla adecuadamente con las condiciones de frontera, garantizando así una solución física consistente.

Capítulo 3

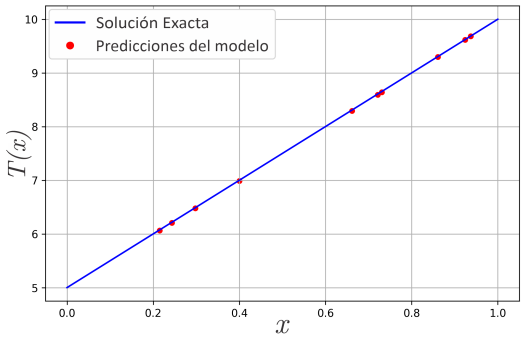
Resultados

3.1. Soluciones generadas por PINNs

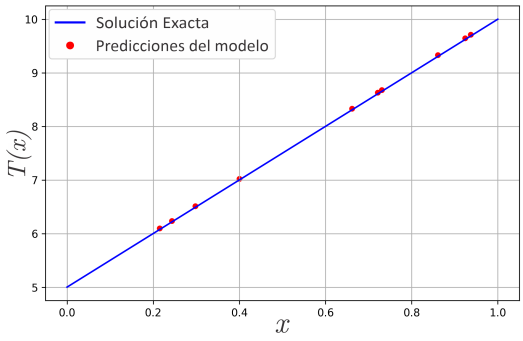
Los resultados se obtuvieron mediante la evaluación de puntos de muestreo aleatorios en el dominio $x \in [0, 1]$, utilizando las condiciones de frontera establecidas durante el entrenamiento. Para evaluar la generalización, se generó un conjunto de datos de prueba independiente, empleado en las predicciones de los cinco modelos. La Figura 3.1 muestra las predicciones de cada modelo.



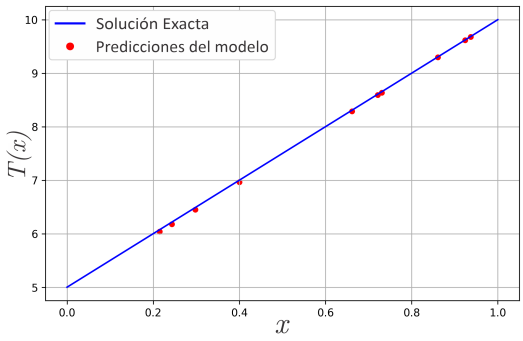
(a) PINN 1



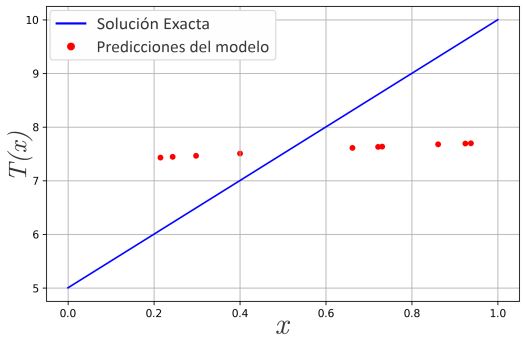
(b) PINN 2



(c) PINN 3



(d) PINN 4



(e) PINN 5

Figura 3.1: Comparación de las soluciones aproximadas generadas por los cinco modelos de PINN frente a la solución exacta.

3.2. Validación de resultados

Para validar la efectividad de la PINN, se analizó su convergencia durante el entrenamiento y se calcularon métricas de error en un conjunto de datos de prueba independiente.

3.2.1. Convergencia durante el entrenamiento

En primer lugar, se monitoreó la convergencia de la función de pérdida durante el entrenamiento utilizando un conjunto de validación. Esto permitió verificar que la red estuviera aprendiendo de manera efectiva y prevenir el sobreajuste. La evolución de la pérdida se registró en función del número de épocas para evaluar la estabilidad del aprendizaje.

3.2.2. Cálculo de métricas de error

Posteriormente, se calcularon dos métricas de error para evaluar el desempeño de la red neuronal: el Error Cuadrático Medio (MSE) y el Error Absoluto Medio (MAE). Ambas métricas se calculan comparando la predicción de la red neuronal con la solución exacta en puntos específicos del dominio, tanto en los datos de entrenamiento como en los de prueba.

- **Error Cuadrático Medio (MSE):**

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(T(x_i) - \mathbf{NN}(x_i)\right)^2,$$

  donde x_i son los puntos de muestreo, N es el número total de puntos, $T(x_i)$ es la solución exacta y $\mathbf{NN}(x_i)$ es la predicción de la red neuronal.

- **Error Absoluto Medio (MAE):**

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|T(x_i) - \mathbf{NN}(x_i)|,$$

  donde las variables tienen el mismo significado que en el MSE.

Estas métricas, junto con el uso de un conjunto de prueba, aseguran que los resultados obtenidos sean precisos y confiables, proporcionando una base sólida para la evaluación del rendimiento de la red neuronal.

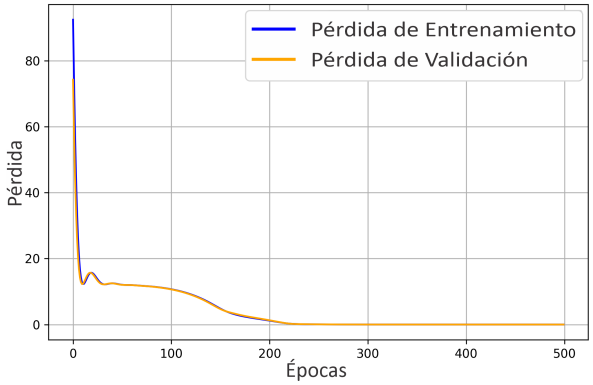
3.3. Comparación de modelos

La Tabla 3.1 presenta los valores de la función de pérdida para cada uno de los modelos, tanto durante el entrenamiento como en la validación, así como el tiempo de entrenamiento asociado a cada modelo.

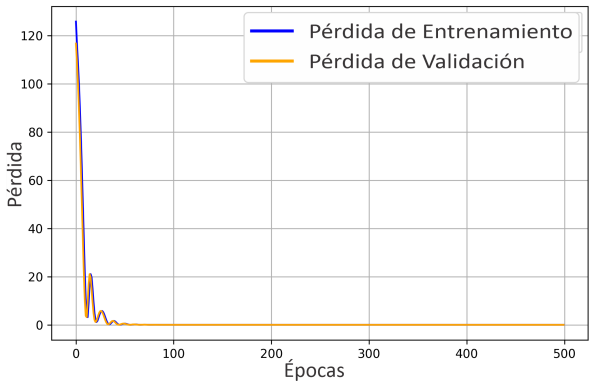
Tabla 3.1: Comparación de pérdida y tiempo de entrenamiento para los cinco modelos de PINN.

Modelo	**Pérdida (Entrenamiento)**	**Pérdida (Validación)**	**Tiempo de Entrenamiento (s)**
PINN 1	0.001213	0.000585	45.26
PINN 2	0.0	0.0	41.74
PINN 3	0.000820	0.001201	59.23
PINN 4	9.095×10^{-13}	9.095×10^{-13}	42.16
PINN 5	11.067	11.057	60.75

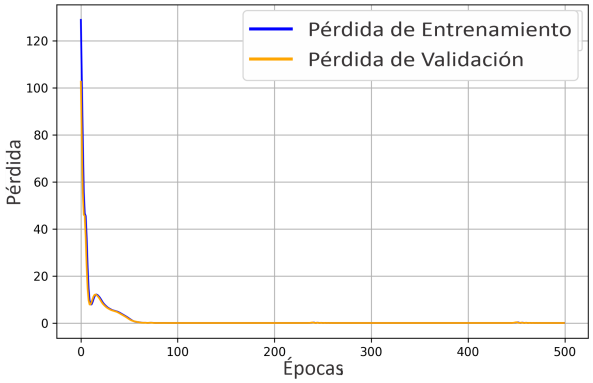
La Figura 3.2 muestra la evolución de la función de pérdida en función del número de épocas para cada modelo, permitiendo comparar la velocidad de convergencia y estabilidad de los modelos.



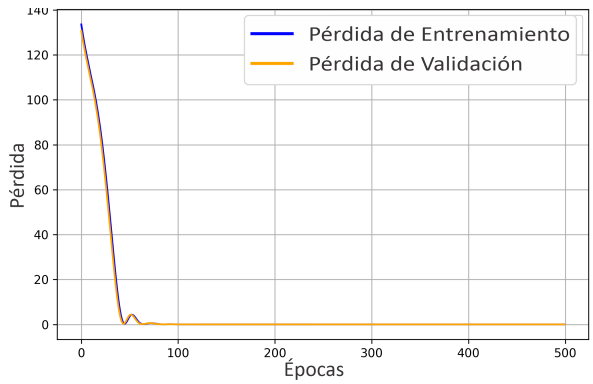
(a) PINN 1



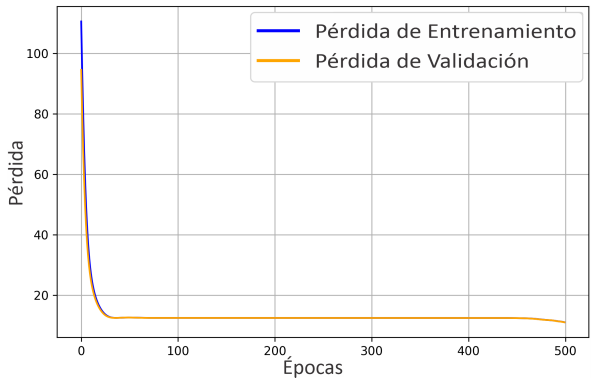
(b) PINN 2



(c) PINN 3



(d) PINN 4



(e) PINN 5

Figura 3.2: Evolución de la función de pérdida para los cinco modelos de PINN.

Adicionalmente, se calcularon las métricas de error MSE y MAE para comparar las predicciones de los modelos con la solución exacta de la ecuación de calor en un conjunto de datos de prueba independiente, validando así su precisión. También se registró el tiempo de entrenamiento de cada modelo, proporcionando un análisis de su eficiencia computacional.

Los resultados obtenidos para las métricas de error se presentan en la Tabla 3.2.

Tabla 3.2: Comparación de MSE y MAE para los cinco modelos de PINN.

Modelo	**MSE**	**MAE**
PINN 1	1.049×10^{-6}	0.000918
PINN 2	1.034×10^{-4}	0.009474
PINN 3	5.961×10^{-4}	0.024325
PINN 4	6.398×10^{-4}	0.021362
PINN 5	1.735	1.230266

Esta validación proporciona una visión integral de la precisión de los modelos al aproximar la solución de la ecuación de calor, al mismo tiempo que permite analizar su estabilidad durante el proceso de entrenamiento y su eficiencia computacional. De esta manera, se puede valorar no solo la capacidad de los modelos para generalizar a datos no vistos, sino también su rendimiento práctico en términos de tiempo de entrenamiento y capacidad de aprendizaje estable.

Para más detalles sobre la implementación de las pruebas y comparaciones realizadas entre los modelos, véase el Apéndice G.

Capítulo 4

Análisis de resultados

4.1. Comparación con solución exacta

La figura 3.1 permite evaluar y analizar cómo cada modelo aproxima la solución del problema. Dado que la solución exacta es lineal, actúa como una referencia clara para validar el desempeño de los modelos.

Se observa que las soluciones aproximadas de los primeros cuatro modelos coinciden estrechamente con la solución exacta en todo el dominio, lo que indica un desempeño adecuado y una alta precisión en la resolución del problema. Sin embargo, las predicciones del modelo PINN 5 muestran discrepancias notables, desviándose significativamente de la solución exacta. Esto sugiere posibles problemas como un sobreajuste a los datos de entrenamiento, una arquitectura inapropiada o una elección subóptima de los parámetros del modelo. Estas observaciones subrayan la importancia de ajustar cuidadosamente las configuraciones de la PINN para garantizar su eficacia y fiabilidad en la solución de ecuaciones diferenciales.

4.2. Análisis de convergencia

Se observa que los modelos con arquitecturas más simples, como el PINN 2, alcanzan una pérdida de entrenamiento y validación de cero, lo que indica una convergencia exitosa. Este modelo, con una arquitectura sencilla, logra una solución precisa rápidamente y sin riesgo de sobreajuste.

En contraste, los modelos más complejos, como el PINN 5, muestran pérdidas mucho más altas y un tiempo de entrenamiento considerablemente mayor, lo que sugiere que enfrentan dificultades de convergencia y son más propensos al sobreajuste. Esto resalta que, en algunos casos, los modelos más simples pueden converger más rápidamente y con mayor precisión.

Al observar las gráficas, se destacan varios comportamientos:

- El PINN 2 muestra una pérdida de entrenamiento y validación de cero, lo que indica una convergencia completa y precisa. Esto sugiere que el modelo tiene una arquitectura adecuada para el problema, logrando una solución exacta sin riesgo de sobreajuste. Además, el tiempo de entrenamiento (41.74 segundos) es relativamente corto, lo que refuerza la eficiencia del modelo.

- El PINN 4 también presenta pérdidas prácticamente nulas en ambas fases, lo que indica una convergencia extremadamente precisa, similar a la del PINN 2. Su tiempo de entrenamiento (42.16 segundos) es también corto, lo que implica que ha alcanzado la precisión deseada sin necesidad de un cómputo excesivo.

- El PINN 1 y el PINN 3 muestran pérdidas bajas (0.001213 y 0.00082, respectivamente) y una caída en la pérdida durante las primeras épocas, lo que indica que están aprendiendo adecuadamente. Sin embargo, el PINN 1 presenta una mayor diferencia entre la pérdida de entrenamiento y la de validación, lo que sugiere un ligero sobreajuste en comparación con el PINN 3, que muestra un comportamiento más equilibrado.

- Finalmente, el PINN 5 presenta pérdidas significativamente más altas tanto en entrenamiento como en validación (11.067 y 11.057, respectivamente). Aunque la curva de la función de pérdida disminuye de manera suave a lo largo de las épocas, el modelo no logra reducir eficazmente la pérdida. A pesar del tiempo de entrenamiento considerablemente largo (60.75 segundos), la convergencia es deficiente, lo que podría indicar que el modelo es demasiado complejo o mal configurado para este problema, lo que resulta en un sobreajuste y una incapacidad para generalizar adecuadamente.

4.3. Comparativa de modelos mediante métricas de error

Se observa que los modelos con configuraciones más complejas, como el PINN 5, presentan los valores más altos de MSE y MAE, lo que indica un mayor error en la aproximación. Esto sugiere que el aumento en la cantidad de capas y neuronas puede haber generado sobreajuste, dificultando el entrenamiento debido a la saturación de activaciones.

En contraste, el PINN 4, con una estructura más sencilla (dos capas ocultas y 20 neuronas por capa con función de activación ReLU), exhibe los valores más bajos de MSE y MAE. Esto indica que una configuración más simple, combinada con una función de activación que evita la saturación, facilita un aprendizaje más eficiente y estable.

El PINN 1, con una arquitectura similar (dos capas ocultas y 50 neuronas por capa, pero con activación sigmoide), también obtiene buenos resultados en términos de MSE y MAE. A pesar del uso de funciones sigmoides, su estructura relativamente sencilla favorece la generalización y precisión en la solución.

Los modelos PINN 2 y PINN 3, con configuraciones intermedias, presentan errores mayores que los modelos más simples, pero menores que los del modelo más complejo. El PINN 2, con 100 y 50 neuronas por capa y activación ReLU, muestra una mayor sensibilidad a la elección de hiperparámetros. Por su parte, el PINN 3, con tres capas ocultas y función tanh, evidencia que un mayor número de capas no garantiza necesariamente un mejor desempeño, y que la selección de la función de activación juega un papel clave.

El teorema de Hornik establece que una red neuronal con al menos una capa oculta y suficientes neuronas es capaz de aproximar cualquier función continua en un dominio compacto [18]. Sin embargo, este resultado solo garantiza la existencia de una red con dicha capacidad, pero no implica que cualquier arquitectura específica alcance una solución precisa en la práctica. La adecuada selección de parámetros, hiperparámetros, funciones de activación y estrategias de entrenamiento sigue siendo fundamental para obtener resultados satisfactorios.

Estos resultados confirman que un aumento en la complejidad del modelo no siempre conduce a un mejor desempeño. Es esencial encontrar un equilibrio en la estructura de la red y la configuración de sus parámetros, ya que una arquitectura excesivamente compleja puede dificultar el entrenamiento y causar sobreajuste, mientras que un diseño bien balanceado tiende a ofrecer mejores resultados en términos de precisión y generalización.

4.4. Implicaciones de los resultados

Los resultados obtenidos mediante las PINNs demuestran que este enfoque es una alternativa eficiente para resolver ecuaciones diferenciales, como la ecuación de calor unidimensional estacionaria. Las métricas de error, como el MSE y el MAE, muestran que los modelos con arquitecturas simples (PINN 1 y PINN 4) lograron una aproximación precisa de la solución analítica, con errores significativamente menores en comparación con modelos más complejos. Esto sugiere que la simplicidad en la estructura del modelo puede ser clave para obtener buenos resultados, ya que una mayor complejidad no siempre mejora el desempeño.

La elección de la función de activación también influye en la convergencia y estabilidad del modelo. En particular, la ReLU, utilizada en el PINN 4, parece haber favorecido una convergencia más eficiente, evitando la saturación de las activaciones. Este comportamiento refuerza la idea de que una red neuronal bien configurada, incluso con una arquitectura sencilla, puede ofrecer resultados de alta calidad.

Por otro lado, los resultados también resaltan que la configuración de la red, los hiperparámetros y la función de activación son factores determinantes en el desempeño del modelo. El caso del PINN 5, con su arquitectura compleja, muestra que un modelo más grande no necesariamente conduce a mejores resultados y puede llevar al sobreajuste si no se eligen adecuadamente los parámetros. Este hallazgo subraya la importancia de un enfoque balanceado en la selección de la estructura de la red.

Las PINNs presentan ventajas significativas en términos de flexibilidad y capacidad de generalización. A diferencia de los métodos tradicionales, que requieren discretizar el dominio espacial y temporal, las PINNs permiten una formulación continua de la solución. Esto es especialmente útil en problemas con geometrías irregulares o condiciones de frontera complejas, donde las técnicas de discretización pueden resultar difíciles de implementar.

4.5. Limitaciones

A pesar de los buenos resultados obtenidos, el enfoque de las PINNs presenta varias limitaciones. Una de las principales es su dependencia de una correcta configuración de la red, así como la selección adecuada de parámetros e hiperparámetros. El comportamiento subóptimo del PINN 5, que mostró sobreajuste y tiempos de entrenamiento largos, evidencia que una mayor complejidad no siempre mejora los resultados, y resalta la necesidad de una selección cuidadosa de los parámetros.

El tiempo de entrenamiento es otro desafío importante. Mientras que los modelos más sencillos (como el PINN 1 y el PINN 4) lograron resultados en tiempos relativamente cortos, los modelos más complejos (como el PINN 5) requieren más recursos computacionales y tiempo sin una mejora significativa en la precisión. Esto puede limitar la escalabilidad de las PINNs, especialmente en problemas de mayor tamaño o dimensionalidad.

Además, las PINNs requieren una cantidad adecuada de datos de calidad para entrenar correctamente el modelo. Si los datos son limitados o ruidosos, el desempeño del modelo puede verse comprometido. Aunque las PINNs son robustas y pueden generalizar bien con datos de alta calidad, los datos de entrada deben ser representativos del problema para obtener buenos resultados.

Finalmente, otro desafío importante de las PINNs es su limitada interpretabilidad. Aunque estas redes son poderosas para resolver ecuaciones diferenciales, los resultados que gene-

ran pueden carecer de una explicación clara sobre los mecanismos subyacentes. Esta falta de transparencia dificulta su aplicación en contextos donde se requiere entender profundamente el comportamiento de la solución y los factores que la determinan, lo que limita su utilidad en áreas que demandan interpretaciones detalladas de los modelos.

4.6. Posibles mejoras e investigación futura

Las PINNs se posicionan como una herramienta innovadora para la resolución de ecuaciones diferenciales. Sin embargo, aún existen oportunidades significativas para optimizar su desempeño. Una posible mejora sería optimizar la arquitectura de la red neuronal mediante estrategias como la búsqueda en cuadrícula o algoritmos evolutivos, lo que permitiría identificar configuraciones óptimas de hiperparámetros. Además, implementar técnicas avanzadas de regularización podría ser clave para prevenir el sobreajuste, particularmente en modelos con arquitecturas más profundas.

En los experimentos realizados, se observó que la convergencia se alcanzó en un número relativamente bajo de épocas: cerca de 100 en la mayoría de los casos, excepto para el PINN 1, que requirió unas 250 épocas. Este resultado sugiere que las PINNs pueden entrenarse de manera eficiente con menos iteraciones de lo esperado inicialmente.

Una dirección prometedora para futuras investigaciones sería expandir las capacidades predictivas de las PINNs, entrenándolas con condiciones iniciales variables en lugar de fijas. Esto ampliaría su aplicabilidad, permitiéndoles enfrentarse a escenarios más dinámicos y resolver problemas de mayor complejidad. Además, sería interesante evaluar su desempeño utilizando datos reales, lo cual permitiría medir su robustez y precisión en situaciones prácticas, como en la simulación de fenómenos físicos o en la predicción de sistemas dinámicos con ruido inherente.

Por otro lado, las PINNs podrían integrarse con métodos tradicionales, como los de elementos finitos, para mejorar la precisión y eficiencia en la resolución de problemas de alta dimensionalidad o geometrías complejas. Esta combinación de enfoques podría aprovechar lo mejor de ambas metodologías, fusionando la flexibilidad y adaptabilidad de las PINNs con la fiabilidad y precisión de los métodos numéricos clásicos, optimizando así el rendimiento general de las simulaciones. Además, el uso conjunto de PINNs y técnicas de optimización podría abrir nuevas áreas de aplicación en simulaciones industriales y en investigaciones científicas avanzadas.

Conclusiones

En este trabajo, se desarrolló y evaluó un modelo de red neuronal físicamente informada (PINN) para resolver la ecuación de calor unidimensional estacionaria con condiciones de frontera específicas. Se probaron cinco modelos con variaciones en su arquitectura y parámetros de entrenamiento, lo que permitió identificar el que ofrecía el mejor rendimiento en términos de precisión y eficiencia computacional.

Los resultados indican que las PINNs son herramientas efectivas para modelar y predecir la distribución de temperatura en sistemas unidimensionales, integrando principios físicos durante el proceso de entrenamiento. Los modelos con arquitecturas simples lograron soluciones precisas en tiempos de entrenamiento relativamente cortos, mientras que los modelos más complejos, aunque igualmente precisos, implicaron mayores costos computacionales y tiempos de entrenamiento más largos, lo que resalta la importancia de configurar adecuadamente la red y seleccionar los hiperparámetros.

A pesar de las ventajas de las PINNs, como la formulación continua de la solución y su capacidad para adaptarse a condiciones de frontera, su efectividad depende en gran medida de una correcta configuración de parámetros, incluidos los hiperparámetros y las funciones de activación. Además, las redes más complejas tienden a ser más susceptibles al sobreajuste, lo que puede limitar su escalabilidad y eficiencia en problemas de mayor tamaño o con datos más complejos.

En resumen, este trabajo desarrolló y aplicó una metodología basada en redes neuronales físicamente informadas (PINNs) para resolver la ecuación de calor estacionaria en una dimensión, con el objetivo de modelar y predecir la distribución de temperatura en sistemas unidimensionales. Aunque las PINNs tienen un gran potencial, aún existen desafíos en la optimización de parámetros y la estabilidad del entrenamiento. Superar estos obstáculos podría ampliar su aplicabilidad en diversas áreas científicas e ingenieriles.

Bibliografía

[1] N. P. Chávez. *Aprendizaje no supervisado y el algoritmo de wake-sleep en redes neuronales*. Tesis de Licenciatura, Universidad Tecnológica de la Mixteca, 2012.

[2] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow*. Electro Skills Series. Hemisphere Publishing Corporation, 1980.

[3] E. Guerrero J. D. Meshir and A. Palafox. Redes neuronales físicamente informadas para aproximar la solución de ecuaciones diferenciales y sus aplicaciones en la vida real. *Komputer Sapiens*, 2, 15. Mayo-agosto 2023.

[4] S. Cuomo, V. Schiano, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. *Scientific Machine Learning Through PhysicsInformed Neural Networks: Where we are and Whats Next*. Journal of Scientific Computing, **92**, 2022.

[5] Y. A. Çengel. *Transferencia de calor y masa*. McGraw-Hill Interamericana de España S.L., 2011.

[6] F. Kreith, M. S. Bohn, and R. M. Manglik. *Principios de Transferencia de Calor*. Cengage Learning, 2012.

[7] R. J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial and Applied Mathematics, 2007.

[8] D. L. Logan. *A First Course in the Finite Element Method*. Cengage Learning, 2007.

[9] J. A. Anderson. *An Introduction to Neural Networks*. Bradford book. MIT Press, 1997.

[10] C. C. Aggarwal et al. *Neural Networks and Deep Learning*, volume 10. Springer, 2018.

[11] S. S. Haykin. *Neural Networks and Learning Machines*. Number v. 10 in Neural networks and learning machines. Prentice Hall, 2009.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. Diponible en `http://www.deeplearningbook.org`.

[13] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning representations by back-propagating errors*, volume 323. 1986.

[14] Léon Bottou. *Stochastic Gradient Descent Tricks*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[15] Sebastian Ruder. *An overview of gradient descent optimization algorithms*, volume abs/1609.04747. 2016.

[16] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*, volume abs/1412.6980. 2014.

[17] I. E. Lagaris, A. Likas, and D. I. Fotiadis. *Artificial neural networks for solving ordinary and partial differential equations*. IEEE Transactions on Neural Networks, **9**(5):987-1000, 1998.

[18] K. Hornik, M. B. Stinchcombe, and H. L. White. *Multilayer feedforward networks are universal approximators*. Neural Networks, **2**:359-366, 1989.

[19] M. Raissi, P. Perdikaris, and G. E. Karniadakis. *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*. Journal of Computational Physics, **378**:686-707, 2019.

[20] U. Waheed, E. Haghighat, T. Alkhalifah, C. Song, and Q. Hao. *PINNeik: Eikonal solution using physics-informed neural networks*. Computers Geosciences, **155**:104,833, 2021.

[21] Q. He and A. Tartakovsky. *PhysicsInformed Neural Network Method for Forward and Backward AdvectionDispersion Equations*. Water Resources Research, **57**, 2021.

[22] Agarwal A. Barham P. et al. Abadi, M. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Disponible en tensorflow.org.

Apéndices

Apéndice A
Algoritmo Backpropagation

El algoritmo de **Backpropagation** es un método utilizado para entrenar redes neuronales artificiales, introducido en [13] como una forma eficiente de ajustar los pesos de la red y minimizar una función de pérdida mediante el cálculo del gradiente con respecto a los pesos. En problemas tradicionales de aprendizaje supervisado, la función de pérdida suele medir la diferencia entre la salida de la red y un valor esperado. Sin embargo, en PINNs, la función de pérdida incluye términos que evalúan el cumplimiento de una ecuación diferencial y sus condiciones de frontera.

A continuación, se describen los pasos del algoritmo.

Pasos del algoritmo:

1. **Propagación hacia adelante**:

   - Se presenta una entrada x a la red y se calculan las salidas de cada neurona en cada capa hasta obtener la salida final $\mathbf{NN}(x)$.
   - En cada neurona, se aplica la función de activación correspondiente.

2. **Cálculo de la función de pérdida**:

   - En una red neuronal estándar, la función de pérdida mide la diferencia entre la salida de la red y el valor esperado y. Para un problema de regresión, esto se define como:
   $$\mathcal{L} = \frac{1}{2}\sum_i (y_i - \hat{y}_i)^2.$$
   - En una PINN, la función de pérdida incluye un término que evalúa el error en la ecuación diferencial y otro que impone restricciones en los valores de frontera.
   - En este trabajo, la función de pérdida utilizada se detalla en la sección 2.5.
   - Dado que la salida de la red $\mathbf{NN}(x)$ depende de los pesos w_{kj}, la función de pérdida también depende de estos parámetros, lo que permite calcular su gradiente.

3. **Propagación hacia atrás (Backpropagation)**:

   - Se calcula el gradiente de la función de pérdida con respecto a los pesos w_{kj}, aplicando la regla de la cadena:
   $$\frac{\partial \mathcal{L}}{\partial w_{kj}}.$$

4. **Actualización de los pesos**:

   - Los pesos de la red se actualizan utilizando un optimizado.

Etapas del proceso:

- **Inicialización**: Los pesos de la red se inicializan aleatoriamente o utilizando un esquema especializado.
- **Propagación hacia adelante**: La entrada se pasa a través de la red desde la capa de entrada hasta la capa de salida, aplicando la combinación lineal de los pesos y las entradas, seguida de la función de activación en cada neurona.
- **Evaluación de la función de pérdida**: Se calcula la función de pérdida, que en problemas de aprendizaje supervisado mide la diferencia entre la predicción de la red y el valor esperado. En una PINN, la función de pérdida incorpora términos que evalúan el cumplimiento de la ecuación diferencial y las condiciones de frontera.
- **Propagación hacia atrás**: Se calcula el gradiente de la función de pérdida con respecto a cada peso de la red, propagando la información hacia atrás a través de las capas.
- **Actualización de los pesos**: Usando el gradiente calculado, los pesos se actualizan empleando un optimizador. Este proceso se repite en múltiples iteraciones hasta que la función de pérdida se minimiza o alcanza un valor aceptable.

A través de este proceso iterativo, la red ajusta sus parámetros para mejorar su aproximación y garantizar el cumplimiento de las restricciones impuestas por la ecuación diferencial y las condiciones de frontera.

Apéndice B
Optimizador Adam

Adam (Adaptive Momentum Estimation) se basa en la estimación de los primeros momentos (media) y los segundos momentos (varianza) de los gradientes durante el proceso de entrenamiento, lo que le permite ajustar de manera adaptativa los pasos que da en cada iteración para actualizar los pesos de la red. A diferencia de los métodos de optimización tradicionales, que utilizan una tasa de aprendizaje constante, Adam ajusta esta tasa para cada parámetro de manera individual [16].

El optimizador Adam actualiza los parámetros de la red neuronal utilizando las siguientes ecuaciones:

- **Estimación del primer momento (promedio de los gradientes)**:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

  donde:

  - m_t es la estimación del primer momento en el tiempo t.
  - g_t es el gradiente de la función de pérdida con respecto a los pesos w_{kj} en el tiempo t.
  - β_1 es el factor de decaimiento para la media (valor típico: 0.9).

- **Estimación del segundo momento (promedio de los cuadrados de los gradientes)**:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

  donde:

  - v_t es la estimación del segundo momento (varianza) en el tiempo t.
  - β_2 es el factor de decaimiento para la varianza (valor típico: 0.999).

- **Corrección por sesgo**:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

- **Actualización de los pesos**:

$$w_{\text{nuevo}} = w_{\text{actual}} - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

  donde:

  - w_{nuevo} son los nuevos pesos ajustados.
  - η es la tasa de aprendizaje.
  - ϵ es un pequeño valor para evitar divisiones por cero (típicamente 10^{-8}).

El optimizador Adam incluye varios hiperparámetros clave que afectan su comportamiento y eficiencia durante el entrenamiento de redes neuronales:

- **Tasa de aprendizaje** (η): Controla el tamaño de los pasos en cada actualización de los parámetros. Un valor típico inicial es 0.001, pero puede ajustarse dependiendo del problema.
- **Factor de decaimiento para el primer momento** (β_1): Afecta cómo se actualizan las medias de los gradientes. Usualmente se utiliza un valor de 0.9, lo que le permite al optimizador almacenar una memoria exponencialmente decaída de los gradientes pasados.
- **Factor de decaimiento para el segundo momento** (β_2): Controla la tasa de decaimiento para la varianza de los gradientes. Su valor común es 0.999, permitiendo que el optimizador tenga en cuenta tanto las variaciones grandes como pequeñas en los gradientes.
- **Epsilon** (ϵ): Un pequeño valor que se añade en el denominador para evitar divisiones por cero durante la actualización de los pesos. Generalmente se usa $\epsilon = 10^{-8}$.

Estos hiperparámetros pueden ajustarse manualmente en función del tipo de problema y de la red neuronal utilizada. En particular, la tasa de aprendizaje (η) es el hiperparámetro más crítico, ya que su valor impacta directamente en la velocidad de convergencia del modelo y en la estabilidad del entrenamiento.

Apéndice C
Definición de parámetros y solución exacta

En este apéndice se presenta el código utilizado para definir los parámetros y la solución exacta:

```
# Parámetros del problema
T0 = 5  # Temperatura en x = 0
Tf = 10  # Temperatura en x = L
L = 1    # Longitud del dominio

# Función para la solución exacta
def exact_solution(x, T0, Tf, L):
    return ((Tf - T0) / L) * x + T0
```

Este fragmento de código define los parámetros T_0, T_f y L, y calcula la temperatura $T(x)$ en cualquier punto x de la barra utilizando la expresión de la solución exacta.

Apéndice D
Generación de datos de entrenamiento

Los datos para entrenar la red neuronal se generan de manera aleatoria en el dominio $x \in [0, 1]$ con condiciones de frontera $T(0) = 5$ y $T(L) = 10$, donde L es la longitud del dominio (en este caso $L = 1$). Los puntos de entrenamiento se distribuyen aleatoriamente dentro de este intervalo, y la temperatura $T(x)$ en cada punto se calcula utilizando la solución analítica de la ecuación definida previamente.

A continuación, se muestra el código en Python utilizado para generar estos datos y realizar la división en conjuntos de entrenamiento y validación (80 %-20 %):

```
import numpy as np
import tensorflow as tf

# Generación de datos de entrenamiento
# Generar datos
num_points = 100
x_all = np.random.uniform(0, L, num_points)
T_all = exact_solution(x_all, T0, Tf, L)

# División de datos (80% entrenamiento, 20% validación)
train_size = int(0.8 * num_points)
val_size = num_points - train_size
indices = np.random.permutation(num_points)
train_indices = indices[:train_size]
val_indices = indices[train_size:]

x_train = x_all[train_indices]
T_train = T_all[train_indices]
x_val = x_all[val_indices]
T_val = T_all[val_indices]

# Conversión a tensores de TensorFlow
x_train = tf.convert_to_tensor(x_train.reshape(-1, 1), dtype=tf.float32)
T_train = tf.convert_to_tensor(T_train.reshape(-1, 1), dtype=tf.float32)
x_val = tf.convert_to_tensor(x_val.reshape(-1, 1), dtype=tf.float32)
T_val = tf.convert_to_tensor(T_val.reshape(-1, 1), dtype=tf.float32)
```

Este enfoque asegura que los datos sean divididos de manera aleatoria para evitar sesgos, permitiendo una evaluación más realista del rendimiento del modelo.

Apéndice E
Función de pérdida

En este apéndice se presenta la implementación de la función de pérdida utilizada durante el entrenamiento de la red neuronal PINN. Esta función combina términos que aseguran el cumplimiento tanto de la ecuación diferencial como de las condiciones de frontera impuestas. A continuación, se muestra el código:

```
# Función de pérdida para la PINN
def pinn_loss(x, T_pred, model):
    with tf.GradientTape() as tape1:
        tape1.watch(x)
        with tf.GradientTape() as tape2:
            tape2.watch(x)
            T = model(x)
        dT_dx = tape2.gradient(T, x)
    d2T_dx2 = tape1.gradient(dT_dx, x)

    # Pérdida física
    loss_physics = tf.reduce_mean(tf.square(d2T_dx2))

    # Pérdida por condiciones de frontera
    loss_bc_0 = tf.reduce_mean(tf.square(model(tf.constant([[0.0]],
    ↪  dtype=tf.float32)) - T0))
    loss_bc_L = tf.reduce_mean(tf.square(model(tf.constant([[L]],
    ↪  dtype=tf.float32)) - Tf))

    return loss_physics + loss_bc_0 + loss_bc_L
```

Apéndice F
Entrenamiento de la red

El entrenamiento de la red se basa en la minimización progresiva de la función de pérdida, la cual está diseñada para garantizar que las predicciones de la red neuronal respeten tanto la ecuación diferencial como las condiciones de frontera impuestas.

En cada iteración, el código calcula esta función, lo que permite evaluar el desempeño de la red. A lo largo del proceso, se almacena el valor de la función de pérdida en cada época para su posterior visualización y análisis, con el fin de entender cómo evoluciona la minimización de la pérdida. El proceso concluye cuando se han realizado el número de épocas determinado.

```python
import time
# Entrenamiento y evaluación de modelos
def train_and_evaluate_model(model, x_train, T_train, x_val, T_val,
↪   epochs=500):
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
    losses_train = []
    losses_val = []

    # Medición del tiempo de entrenamiento
    # Inicio
    start_time = time.time()

    # Entrenamiento
    for epoch in range(epochs):
        with tf.GradientTape() as tape:
            T_pred_train = model(x_train)
            loss_train = pinn_loss(x_train, T_pred_train, model)
        grads = tape.gradient(loss_train, model.trainable_variables)
        optimizer.apply_gradients(zip(grads, model.trainable_variables))
        losses_train.append(loss_train.numpy())

        T_pred_val = model(x_val)
        loss_val = pinn_loss(x_val, T_pred_val, model)
        losses_val.append(loss_val.numpy())

    # Final
    end_time = time.time()
    training_time = end_time - start_time
    print(f"El entrenamiento tomó {training_time:.2f} segundos")

    return losses_train, losses_val, training_time
```

Apéndice G
Pruebas y comparación de modelos

En este apéndice se presenta el código utilizado para realizar comparaciones entre diferentes arquitecturas de redes neuronales. Los modelos fueron entrenados con diversas configuraciones de capas y funciones de activación, y fueron evaluados utilizando las métricas de Error Absoluto Medio (MAE) y Error Cuadrático Medio (MSE). Estas métricas fueron calculadas utilizando la biblioteca scikit-learn (`sklearn.metrics`). Finalmente, se graficaron las predicciones obtenidas por cada modelo frente a la solución exacta para analizar su rendimiento.

A continuación se muestra el código utilizado para entrenar y evaluar los modelos:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Generar puntos de prueba para los modelos
num_test_points = 10  # Número de puntos para prueba
x_test = np.random.uniform(0, L, num_test_points).reshape(-1,
↪  1).astype(np.float32)
T_test = exact_solution(x_test, T0, Tf, L)  # Solución exacta para los puntos
↪  de prueba

# Entrenamiento iterativo de los modelos a probar
models_configs = [
    {"layers": [50, 50], "activation": 'sigmoid'},
    {"layers": [100, 50], "activation": 'relu'},
    {"layers": [50, 50, 50], "activation": 'tanh'},
    {"layers": [20, 20], "activation": 'relu'},
    {"layers": [70, 40, 20], "activation": 'sigmoid'},
]

for idx, (config, label) in enumerate(zip(models_configs, model_labels)):
    # Crear y entrenar el modelo
    model = Sequential()
    model.add(Dense(config['layers'][0], input_dim=1,
    ↪  activation=config['activation']))
    for layer_size in config['layers'][1:]:
        model.add(Dense(layer_size, activation=config['activation']))
    model.add(Dense(1, activation='linear'))

    print(f"Entrenando {label} con configuración: {config}")
    losses_train, losses_val, training_time = train_and_evaluate_model(model,
    ↪  x_train, T_train, x_val, T_val, epochs=500)

    # Prueba de modelos con datos de prueba independientes
    # Realizar predicciones en los datos de prueba
```

```
    T_pred = model.predict(x_test)

    # Calcular MAE y MSE
    mae = mean_absolute_error(T_test, T_pred)
    mse = mean_squared_error(T_test, T_pred)

    print(f"{label}: MAE: {mae}, MSE: {mse}")
    results[label]["mae"] = mae
    results[label]["mse"] = mse

    # Graficar pérdida vs épocas
    plt.figure(figsize=(8, 5))
    plt.plot(losses_train, label=f'Pérdida de Entrenamiento ({label})',
    ↪  color='blue')
    plt.plot(losses_val, label=f'Pérdida de Validación ({label})',
    ↪  color='orange')
    plt.title(f'Pérdida durante el entrenamiento ({label})')
    plt.xlabel('Épocas')
    plt.ylabel('Pérdida')
    plt.legend()
    plt.grid(True)
    plt.show()

    # Graficar solución exacta y predicciones
    plt.figure(figsize=(8, 5))
    plt.plot(np.linspace(0, L, 100), exact_solution(np.linspace(0, L, 100),
    ↪  T0, Tf, L),
             label="Solución Exacta", color="blue", linestyle="-")  #
             ↪  Solución exacta
    plt.scatter(x_test, T_pred, label="Predicciones del Modelo", color="red",
    ↪  s=20)
    plt.xlabel("x")
    plt.ylabel("T(x)")
    plt.title(f"Predicciones del Modelo - {label}")
    plt.legend()
    plt.grid(True)
    plt.show()
```