Simulación de transferencia de calor en un disipador mediante Redes Neuronales Informadas por Física (PINN)

José Antonio Angeles Guerrero

Facultad de Ciencias, Universidad Nacional Autónoma de México.

4 de diciembre de 2025

1. Introducción

Las Redes Neuronales son herramientas computacionales que predicen un resultado al recibir un conjunto de datos, esto ocurre gracias al entrenamiento de estas,donde se aprenden patrones mediante el ajuste de pesos de cada neurona acorde a su respectiva función de activación, con el fin de minimizar una función de pérdida despues de cada época entrenada, volviendo más precisa su predicción.

Recientemente ha tenido relevancia un tipo de red neuronal al resolver problemas fisicos, estas son las **Redes Neuronales Informadas por Física (PINN)** las cuales implementan las ecuaciones diferenciales del problema dentro de la función de pérdida al entrenarse, forzando a que su predicción cumpla con las leyes físicas impuestas.

Las PINNs son una buena alternativa a los métodos númericos, ya que no necesita mallas para discretizar los datos con los que se trabajan, lo cual implica que la solución que genera es continua y diferenciable, lo que simplifica la solución de problemas.

Para el proyecto se decidió resolver la ecuación de calor en una geometría relativamente compleja como lo es un disipador de calor.

1.1. Objetivo

Lograr entrenar una Red Neuronal Informada por Física para que resuelva la ecuación de calor en una geometría compleja, con el fin de demostrar la capacidad de las Redes Neuronales Informadas por Física.

2. Desarrollo

2.1. Planteamiento matemático

Se buscó resolver la ecuación de calor con la siguiente forma:

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u$$

Donde el $\alpha = 97.4\ \text{mm}^2/\text{s}$ siendo este el valor de la difusividad térmica del aluminio.

Para simplificar el modelo se decidió usar como condiciones de frontera $u = 80°C$ en la base del disipador y $u = 20°C$ en el resto del contorno, asumiendo que el disipador recibe un flujo de aire tan alto que la temperatura de las paredes del disipador se iguala instantáneamente a la temperatura ambiente de $20°C$. Finalmente como condición inicial se usó $u_0 = 20°C$.

2.2. Herramientas utilizadas

Para todo el codigo se usó como lenguaje Python3 por su amplio uso en el computo científico y especificamente en machine learning. Más aún, para la arquitectura de la red neuronal y su entrenamiento se usó la librería de PyTorch la cual es una de las más utilizadas para este fin.

2.3. Generación de los datos

Para poder entrenar la red neuronal esta tiene que recibir los datos con los que va a trabajar, los cuales son el dominio de la simulación.

Para el proyecto se decidió simular un disipador de aire de aletas extruidas como el siguiente:

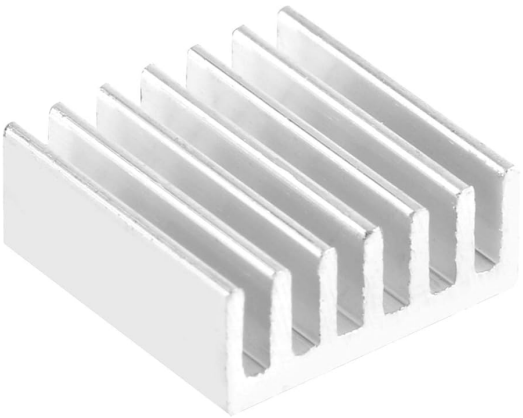

Figura 1: Disipador de aletas extruidas.

Este disipador permite aprovechar su simetria y poder simular solamente un corte transversal de este en 2d. El modelo 2d tiene dimensiones de 50 mm × 50 mm con un grosor de la base de 10 mm y grosor de las aletas de 5 mm.

Los puntos se generaron usando lógica booleana, con coordenadas $X = (x, y, t)$,divididos en 3 conjuntos:

- 70,000 puntos interiores aleatorios.
- 20,000 puntos en las fronteras.
- 10,000 puntos en la condicion inicial $t = 0$ s.

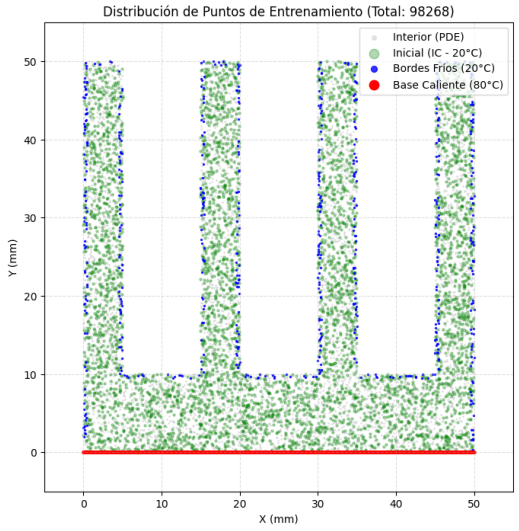

Figura 2: Representación gráfica de los puntos de entrenamiento.

2.4. Arquitectura de la red neuronal

La configuración de la red neuronal fue una de perceptrón multicapa (MLP) configurada como totalmente conectada (Feed-Forward Neural Network) con la información moviendose en una sola dirección pasando por todas las capas. La arquitectura de la red se optimizó para poder reducir el sesgo espectral, que es la tendencia de los modelos a aprender las componentes de baja frecuencia (patrones grandes, como el interior del disipador) e ignorar las de alta frecuencia (detalles finos, como las fronteras y las condiciones iniciales), la arquitetura fue la siguiente:

- Primero los datos se normalizan para que la red los pueda procesar de mejor forma.
- Fourier Feature Mapping: Despues de que los datos estén normalizados se someten a una proyección de Fourier para elevarlos a un espacio de dimensión mayor:

$$\gamma(X) = [\sin(2\pi BX), \cos(2\pi BX)]$$

Donde B es una matriz de pesos con una distribución Gaussiana. Al hacer esto se expande de

3 dimensiones a 40 características (20 senos y 20 cosenos) lo que permite que el modelo pueda detectar los patrones finos.

- Perceptrón Multicapa: Tras pasar la proyección de Fourier los datos entran al perceptrón multicapa, con 4 capas ocultas densas y 64 neuronas por capa. Como función de activación se usó Tanh (tangente hiperbólica) pues a diferencia de ReLU, esta es infinitamente diferenciable, lo cual es necesario para poder calcular las segundas derivadas para la ecuación de calor.
- Capa de salida: Mediante una transformación lineal se proyectan las caracteristicas latentes a un escalar, el cual es la temperatura adimensional, la cual despues de la predicción debe de ser desnormalizada.

2.5. Entrenamiento

Durante el ciclo de entrenamiento, se utilizó el motor de diferenciación automática de Pytorch (torch.autograd.grad) para poder calcular las derivadas para la ecuación diferencial. Este aplica la regla de la cadena sobre el grafo computacional dinámico, lo que hace que devuelva derivadas exactas.

En el entrenamiento se buscó minimizar una función de pérdida total $\mathcal{L}_{total}$ la cual es la siguiente suma ponderada:

$$\mathcal{L}_{total} = \mathcal{L}_{PDE} + \lambda_{BC}\mathcal{L}_{BC} + \lambda_{IC}\mathcal{L}_{IC}$$

Donde:

- $\mathcal{L}_{PDE}$ es el error cuadrático medio del residuo de la ecuación de calor: $|u_t - \alpha\nabla^2 u|^2$
- $\mathcal{L}_{BC}$ es el error de la condición de frontera y $\lambda_{BC} = 10$ para que le de más importancia a minimizar esta función de pérdida.
- $\mathcal{L}_{IC}$ es el error de la condición inicial, igualmente con $\lambda_{IC} = 10$

Como algoritmo optimizador se utilizó el Adam con una tasa de aprendizaje inicial de 1×10^{-3}. Se implementó un planificador StepLR para que reduzca la tasa de aprendizaje cada 5,000 epocas a la mitad.

El entrenamiento duró 20,000 épocas.

3. Resultados

Con esta configuración de la red neuronal el modelo pudo converger y reducir la función de pérdida total:

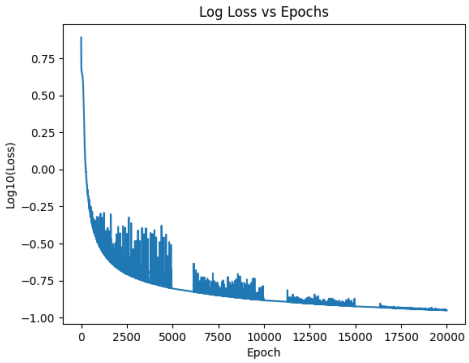

Figura 3: Gráfica de la función de perdida.

Obteniendo al final las pérdidas: $\mathcal{L}_{total} = 0.105$ y $\mathcal{L}_{PDE} = 0.019$ por lo que el modelo aprendió a resolver la ecuación de calor. Finalmente se creó una malla de puntos para utilizar el modelo para predecir la solución y poder hacer una animación.

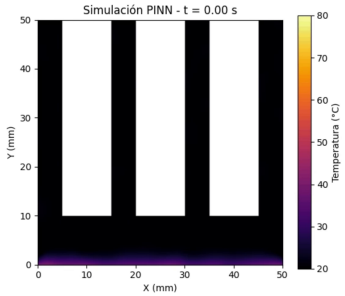

Figura 4: Predicción en $t \approx 0$: Se puede observar la condición inicial donde la base está caliente y el resto del disipador está a temperatura ambiente.

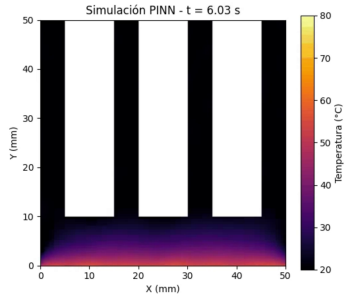

Figura 5: Predicción en $t \approx 6$: Se puede observar que conforme pasa el tiempo se propaga el calor y el gradiente se ve suave y continuo.

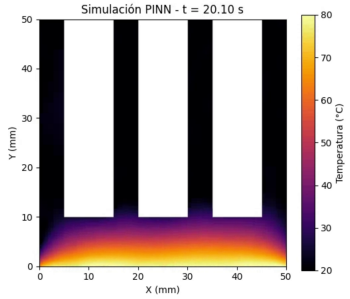

Figura 6: Predicción en $t \approx 20$: Se puede ver que pasado un tiempo el sistema entra en equilibrio.

En la animación completa se alcanza a apreciar que el calor nunca sube hasta las puntas de las aletas, esto tiene sentido pues al poner las condiciones de frontera tan restrictivas como que se enfríe el disipador instantáneamente y al tener las aletas muy delgadas, no permite que suba el calor, pero además se nota este comportamiento en los espacios donde no hay aletas, ya que se forman unos valles fríos debajo de estos, lo cual indica que la red predice exitosamente el comportamiento del calor.

4. Conclusiones

Se logró entrenar una red neuronal que pueda resolver la ecuación de calor en una geometría compleja como lo es el modelo del disipador, a pesar de que parezca un comportamiento extraño el que se ve en las aletas, esto es congruente con el modelo idealizado que se propuso.

Esto permite ver el potencial que pueden llegar a tener las PINNs a la hora de resolver problemas complejos de física, pues el modelo propuesto puede subir de complejidad y el proceso sería análogo, implementando técnicas y métodos mas sofisticados de la redes neuronales.

Referencias

[1] PyTorch Development Team, *Automatic differentiation package - torch.autograd*, PyTorch Documentation, 2024. Recuperado de `https://pytorch.org/docs/stable/autograd.html`

[2] PyTorch Development Team, *A Beginner PyTorch Tutorial*, PyTorch Tutorials, 2024. Recuperado de `https://pytorch.org/tutorials/beginner/blitz/intro_tutorial.html`

[3] Cengel, Y. A., Ghajar, A. J., & Boles, M. A., *Transferencia de calor y masa: Fundamentos y aplicaciones*, McGraw-Hill Education, 2011.

[4] Raissi, M., Perdikaris, P., & Karniadakis, G. E., *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, Journal of Computational Physics, 2019. Recuperado de `https://doi.org/10.1016/j.jcp.2018.10.045`

[5] Tancik, M., Srinivasan, P., Mildenhall, B., et al., *Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains*, Advances in Neural Information Processing Systems (NeurIPS), 2020. Recuperado de `https://proceedings.neurips.cc/paper/2020/hash/55053683268957697aa39fba6f231c68-Abstract.html`