Digital Object Identifier

Physics-Informed Neural Network: Solución de un problema directo e inverso de Ecuaciones Diferenciales Parciales.

SEBASTIÁN JARA CIFUENTES.[1]
[1]Magister en Estadística (c), Universidad de Valparaíso, Valparaíso, Chile. (e-mail: jarac.sebastian@gmail.com)

ABSTRACT En el año 2017, Raissi, Perdikaris y Karniadakis exponen lo que hoy conocemos como *Physics-Informed Neural Networks* (PINN) [3], [2]: redes neuronales que están capacitadas para resolver tareas de aprendizaje supervisado incorporando principios físicos -en forma de ecuaciones diferenciales- en el aprendizaje automático. En esta investigación, inspirados en Raissi [3], usamos una (PINN) para resolver un problema directo, y un problema inverso, asociado a una ecuación diferencial parcial (EDP) de segundo orden. En particular resolveremos $\Delta u = f$, sujeta a condiciones de borde Dirichlet. Este problema es conocido como la ecuación de Poisson, presente en una amplio espectro de problemas en físicos e ingeniería. Los códigos computacionales utilizados en los experimentos numéricos están disponibles en https://github.com/sevaseba/Estadistica-Computacional-2022-1.git.

INDEX TERMS Aprendizaje Automático, Redes Neuronales, Physics-Informed Neural Networks, Ecuaciones diferenciales, Ecuación de Poisson.

I. INTRODUCCIÓN

Resolver ecuaciones diferenciales es extremadamente importante hoy en día para diversas área de la ciencia: Ingenieria, Biología, Química, Matemática, Física, etc. En la mayoría de los casos, no se pueden encontrar soluciones exactas o analíticas. Por lo tanto, en tales casos se busca implementar un método numérico para calcular una solución aproximada de la ecuación diferencial. Los métodos estándar para resolver ecuaciones diferenciales numéricamente incluyen el método de Runge-Kutta, métodos lineales de múltiples pasos, métodos de elementos finitos o de volumen finito y métodos espectrales [4]. En la década de los noventa, se propone un nuevo enfoque para resolver ecuaciones diferenciales utilizando redes neuronales artificiales. Actualmente, el explosivo avance en los recursos computacionales ha permitido implementar con éxito simulaciones numéricas de ecuaciones diferenciales mediante redes neuronales, área conocida como *Physics-Informed Neural Networks* (PINN) [3], [2].

En está investigación se está interesado en resolver la ecuación de Poisson de manera directa e inversa, usando PINN. En particular, el problema directo que se resolverá es:

Hallar u tal que

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \tag{1}$$

para $(x, y) \in \Omega \subset \mathbb{R}^2$, f una función distinta de cero y continua y u sujeto a una condición de borde Dirichlet. Por otro lado, el problema inverso que se propone resolver es: Conocida la solución de (1), determinar $\lambda \in \mathbb{R}$ tal que

$$\frac{\partial^2 u}{\partial x^2} + \lambda\frac{\partial^2 u}{\partial y^2} = f(x, y), \tag{2}$$

para $(x, y) \in \Omega \subset \mathbb{R}^2$, f una función distinta de cero y continua.

La ecuación (1) surge en múltiples aplicaciones físicas en los campos de electroestática, termodinámica, ingeniería mecánica y física teórica, entre otras. La forma estándar de resolver (1) y (2) es a través de un método analítico, aunque a veces resulta imposible aplicar tales ideas para encontrar la solución, y el enfoque es buscar soluciones aproximadas mediante métodos numéricos, como diferencias finitas o elementos finitos, los cuales han mostrado ser muy eficientes para resolver este problema. Sin embargo, su desventaja radica en los casos donde los dominios son irregulares ge-

ométricamente y cuando no se conoce de manera completa las condiciones de borde del problema.

Como se mencionó, para resolver el problema definido por (1) y (2) se utilizarán las técnicas de redes neuronales basadas en la física (PINN). Inspirados por las ideas expuestas en [2] y [3], se buscará obtener una solución numérica para estos problemas a partir de una red neuronal feed forward fully connected.

Finalmente, aseguramos que muchas de las características de este problema, sobre todo los relativos a la implementación numérica de la solución a través de PINN, se extienden a ecuaciones diferenciales parciales más complicadas. Entonces, comprender uno de los casos más simples es el puntapie inicial para comenzar el estudio de soluciones aproximadas de EDP mediante una red neuronal.

II. MATERIALES Y METODOS

A. ECUACIÓN DE POISSON: PROBLEMA DIRECTO E INVERSO

Se propone resolver el siguiente problema directo de ecuaciones diferenciales parciales : Dado $\Omega = [0,2]\times[0,2] \subset \mathbb{R}^2$, hallar u tal que

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= f(x,y), \quad x \in \Omega, && (3)\\ u|_{\partial\Omega} &= 0. && (4) \end{aligned}$$

La condición (4) significa que la solución u debe ser igual a cero en la frontera de Ω. La solución del problema anterior depende en gran medida de como elegimos la función f. Entonces, con el objetivo de conocer a priori la solución del problema (3)-(4), se propone como solución la función

$$u(x,y) = 6xy(x-2)(y-2). \tag{5}$$

Notar que (5) satisface (4). Reemplazando (5) en (3), se obtiene que

$$f(x,y) = 12(x^2+y^2) - 24(x+y). \tag{6}$$

Lo anterior es suficiente para conocer de manera completa la EDP de interés y su solución.

Para el problema inverso consideramos nuevamente $\Omega = [0,2]\times[0,2] \subset \mathbb{R}^2$, u y f como en (5) y (6), respectivamente. Se propone hallar $\lambda \in \mathbb{R}$ tal que

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2} + \lambda\frac{\partial^2 u}{\partial y^2} &= f(x,y), \quad x \in \Omega, && (7)\\ u|_{\partial\Omega} &= 0. && (8) \end{aligned}$$

Dado lo anterior, es claro que λ necesariamente debe ser 1. El objetivo principal es resolver numéricamente, a partir de una red neuronal, los problemas directo e inverso definidos anteriormente.

B. RED NEURONAL FEED FORWARD FULLY CONNECTED

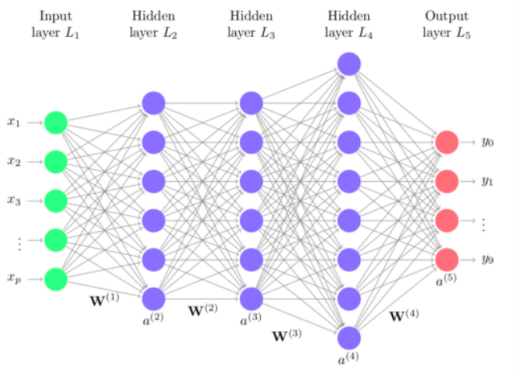
La estructura de una red neuronal feed forward fully connected, o comunmente conocida en espñol como perceptrón multicapa, consta de una capa de entrada, de una o varias capas ocultas y una capa de salida. Cada capa esta formada por un conjunto de neuronas, donde cada neurona está conectada a todas las neuronas de la capa anterior y a todas las neuronas de la capa posterior (figura 1).

En la figura 1, $\mathbf{x}$ es el vector de neuronas de entrada, $\mathbf{a}^{(i)}$ es el vector de neuronas de la capa $i-$ésima, $\mathbf{y}$ el vector de neuronas de salida, $g^{(i)}$ la función de activación aplicada a cada una de las neuronas de la capa i, $\mathbf{W}^{(i)}$ una matriz de pesos que pondera las neuronas de la capa $i-1$ y $\mathbf{b}^i$ un vector de sesgos para las neuronas de la capa i. Entonces, una red neuronal de L capas puede ser escrita como:

$$\begin{aligned} \mathbf{a}^{(1)} &= \mathbf{x} \\ \mathbf{a}^{(i)} &= g^{(i)}\left(\mathbf{W}^{(i-1)}\mathbf{a}^{(i-1)} + \mathbf{b}^{(i)}\right), \quad i = 2,3,...,L-1. \\ \mathbf{y} &= g^{(L)}(\mathbf{W}^{(L-1)}\mathbf{a}^{(L-1)} + \mathbf{b}^{(L)}) \end{aligned}$$

Combinando múltiples capas ocultas y funciones de activación no lineales, estos modelos pueden aprender prácticamente cualquier patrón. Está demostrado [12] que con suficientes neuronas una red perceptrón multicapa es un aproximador universal para cualquier función.

FIGURE 1. Representación de una red neuronal perceptrón multicapa con una capa de entrada, tres capas ocultas y una capa de salida. Fuente: Computer Age Statistical Inference 2016



Si escribimos la red neuronal perceptrón multicapa como $\mathbf{y} = \hat{u}(\mathbf{x},\theta)$, donde θ es un vector que continene los pesos y sesgos, entonces el proceso de entrenamiento de la red neuronal consiste en hallar $\hat{\theta}$ que minimice una función de pérdida que mida el error entre los datos observados y los datos que ofrece la predicción de la red neuronal. Matemáticamente esto se representa como :

$$\hat{\theta} = \min_{\theta} \sum_{i=1}^{N_\tau} \|u_i - \hat{u}(\mathbf{x}_i,\theta)\| \tag{9}$$

donde $\{\mathbf{x}_i, u_i\}_{i=1}^{N_r}$ es un conjunto finito de datos de entrenamiento y u_i es el output observado para el input $\mathbf{x}_i$. Para resolver (9), se implementa un algoritmo de descenso de gradiente. Este algoritmo de descenso, en coordinación con la técnica de Backpropagation[1] y la diferenciación automática, hace eficiente computacionalmente el proceso de minimización (9) cuantificando la influencia que tiene cada peso y sesgo de la red neuronal en sus predicciones. El algoritmo irá ajustando lo pesos y sesgos para minimizar el promedio de los errores de las predicciones.

C. REDES NEURONALES INFORMADAS POR LA FÍSICA (PINN)

Las redes neuronales basadas en la física (PINN) consideran la información de las mediciones del proceso físico y la ecuación diferencial asociada al proceso. Las mediciones del proceso físico y la ecuación diferencial son usadas para construir una función, llamada usualmente función de pérdida, que mide el error entre la red neuronal y la solución exacta del problema. La información física del fenómeno viene en forma de ecuaciones diferenciales ordinarias (ODE) o ecuaciones diferenciales parciales (PDE). En la literatura encontramos aplicaciones para PDE de orden entero, ecuaciones diferenciales integrales, PDE fraccionales o PDE estocásticas.

La forma general de una gran cantidad de ecuaciones diferencial paramétrica es

$$u_t(t,x) + \Gamma[u(t,x);\lambda] = f(t,x), \quad x \in \Omega \subset \mathbb{R}^n, \ t \in [0,T], \tag{10}$$

donde $u(t,x)$ denota una función que satisface la ecuación diferencial (10), $\Gamma[\,\cdot\,;\lambda]$ un operador diferencial no lineal, y λ es un parámetro [2].

Notar que, en los problemas de Poisson que se está interesado en resolver [(1),(2], se tiene:

$$\Gamma[\,\cdot\,;\lambda] := \frac{\partial^2}{\partial x^2} + \lambda \frac{\partial^2}{\partial y^2}. \tag{11}$$

En el caso de interés, $x := (x,y) \in \Omega = [0,2] \times [0,2]$ y la función u no depende de t, entonces $u_t = 0$.

Si una red neuronal $\hat{u}(t,x,\theta)$ es una solución aproximada de (10), entonces el vector θ de pesos y sesgos de la red neuronal debe minimizar la siguiente función de pérdida:

$$L(\theta) = \frac{1}{N_b}\sum_{j=1}^{N_b} |u(t_b^j, x_b^j) - \hat{u}(t_b^j, x_b^j, \theta)|^2 + \frac{1}{N_i}\sum_{j=1}^{N_i} |\hat{u}_t(t_i^j, x_i^j, \theta) + \Gamma[\hat{u}(t_i^j, x_i^j, \theta);\lambda] - f(t_i^j, x_i^j)|^2 \tag{12}$$

[1]Backpropagation se basa principalmente en el uso abusivo de la regla de la cadena para calcular los gradiente de la función de pérdida, con respecto a los pesos y sesgos.

Aquí $\{u(t_b^j, x_b^j)\}_{j=1}^{N_b}$ representa los datos de entrenamiento para la red neuronal, que en este contexto equivalenten a las condiciones de borde de la ecuación diferencial[2]. Por otro lado, el segundo término de la derecha de (12) representa la restricción fisica que debe satisfacer la red neuronal y pude pensarse como una penalización para los pesos y sesgos de la red sobre Ω, que está discretizado por el conjunto $\{t_i^j, x_i^j\}_{j=1}^{N_i}$.

Como la red $\hat{u}(x,t,\theta)$ es una compuesta de funciones diferenciable, aplicadas a productos de matrices, entonces las derivadas se pueden obtener analíticamente mediante la regla de la cadena [1] y computacionalmente por diferenciación automática con Tensorflow [6]. Entonces es posible calcular $\Gamma[\hat{u}(t_i^j, x_i^j, \theta);\lambda]$, en (12), está expresión define para Raissi lo que es una red neuronal basada en la física [2].

Los parámetros contenidos en el vector θ de la red neuronal $\hat{u}(t,x,\theta)$ se obtienen minimizando la función de pérdida (12). Para tal efecto, frecuentemente se utilizan optimizadores iterativos basados en gradientes, para lograr que la pérdida (12) sea menor que un umbral o luego de un número finito de épocas o pasos del esquema numérico de entrenamiento. El esquema estándar viene dado por:

$$\theta_{k+1} = \theta_k - \eta_k \nabla_\theta L(\theta_k), \tag{13}$$

donde η_k es la tasa de aprendizaje en el paso k, y θ_k es el vector de parámetros de la red neuronal en el paso k. Para avanzar en el esquema (13), es necesario obtener las derivadas en $\nabla_\theta L(\theta_k)$, con respecto a los parámetros contenidos en el vector θ, las cuales se obtienen computacionalmente por diferenciación automática con Tensorflow, Keras, Pytorch, SciAnn u otra libreria usada en deep learning [6].

D. SCIANN: UN ENVOLTORIO DE KERAS PARA CÁLCULOS CIENTÍFICOS Y APRENDIZAJE PROFUNDO DE PINN

Redes neuronales para computación científica [13], SciANN por sus siglas en inglés, es una API de redes neuronales artificiales de alto nivel desarrollado sobre Tensorflow y Keras, diseñada para realizar cálculos científicos y entrenar computacionalmente redes neuronales basadas en la física (PINN).

La interfaz de programación que ofrece SciANN están orientada a aplicaciones en ingeniería como el ajuste de modelos de regresión, la solución de ecuaciones diferenciales ordinarias y parciales, y para resolver problemas inversos (identificación de parámetros de una ecuación diferencial).

Pese a estar diseãda para una audiencia con experiencia en computación científica o ciencia e ingeniería computacionales, su simpleza la hace abordable para cualquier persona que desea hacer una transición desde Tensorflow y Keras a una interface donde se hace más simple la implementación y entrenamiento de una PINN.

[2]También tienen la posibilidad de ser restricciones para las derivadas de la solución en el borde

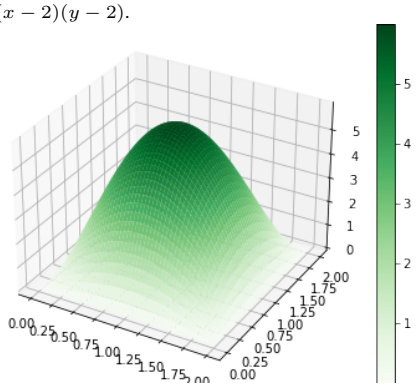
Por lo anterior, los experimentos numéricos de la presente tesis serán realizados en esta API de Tensorflow y Keras.

III. RESULTADOS

A. PROBLEMA DIRECTO

A continuación, se presenta la solución numéricamente obtenida para el problema directo definido en la subsección II.A. En este caso, la gráfica de la solución exacta (5) se presenta en (figura 2).

FIGURE 2. Gráfica de la solución exacta de (3)-(4). En este caso $u(x, y) = 6xy(x-2)(y-2)$.



Para el experimiento numérico se consideró una red neuronal con dos neuronas de entradas, dos capas ocultas con diez neuronas cada una, más una neurona de salida. Las funciones de activiación utilizadas en las capas ocultas han sido tangentes hiperbólicas, el algoritmo de aprendizaje usando es Adam con una tasa de aprendizaje inicial de 0.02 y se usó un número conservador de 10.000 épocas en el esquema (13) (figura 4). La función de pérdida a minimizar ha sido:

$$L(\theta) = \frac{1}{N_b}\sum_{j=1}^{N_b} |u(t_b^j, x_b^j) - \hat{u}(x_b^j, y_b^j, \theta)|^2 + \frac{1}{N_i}\sum_{j=1}^{N_i} \left| \frac{\partial^2 \hat{u}(x_i^j, y_i^j, \theta)}{\partial x^2} + \frac{\partial^2 \hat{u}(x_i^j, y_i^j, \theta)}{\partial y^2} - f(x_i^j, y_i^j) \right|^2, \tag{14}$$

con f definida en (6), $\{(x_b^j, y_b^j)\}_{j=1}^{N_b}$ representa un número finito de puntos distribuidos uniformemente en toda la frontera de $\Omega = [0,2] \times [0,2]$, y $\{x_i^j, y_i^j\}_{j=1}^{N_i}$ representa un conjunto finito de puntos distribuidos uniformemente en todo el dominio Ω. La gráfica de la solución aproximada ofrecida por la red neuronal es graficada en (figura 3).

Finalmente, para la simulación que se presenta, el número de puntos N_b en la frontera de Ω es de 400 y el número de puntos N_i dentro de Ω ha sido de 10.000.

Finalmente, al comparar la solución exacta (5) y la solución aproximada, se alcanzó un error en norma L_2 de 0.966657, en la época 10.000 del entrenamiento (figura 3).

FIGURE 3. Gráfica de la solución aproximada de (3)-(4). Para el entrenamiento de la red neuronal se consideran dos capas oculta de 10 neuronas cada una y 10.000 épocas en el entrenamiento.

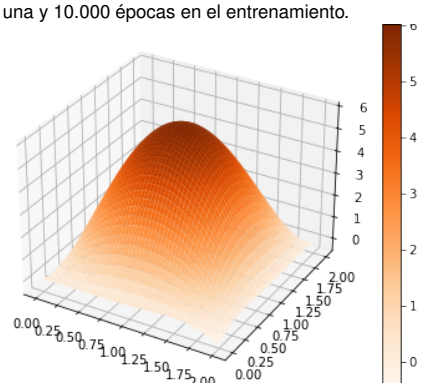
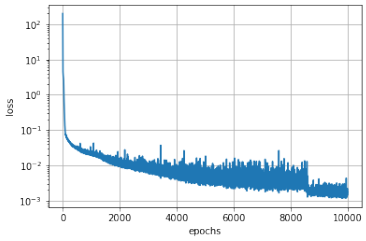


FIGURE 4. Al cabo de las 10.000 pasos en el entrenamiento, la función de pérdida alcanzó una cota inferior de aproximadamente del orden de 10^{-2}.



B. PROBLEMA INVERSO

A continuación, se presenta la solución numéricamente obtenida para el problema inverso definido en la subsección II.A. Se busca determinar el parámetro λ para el problema (7)-(8), con f definido como en (6).

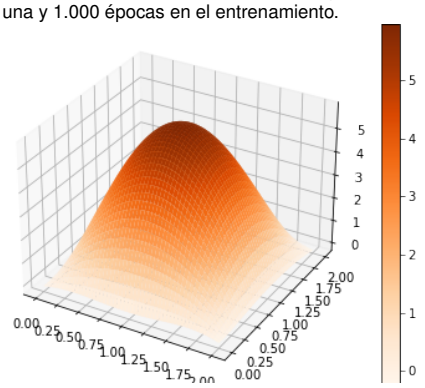
Para el experimiento numérico se consideró una red neuronal configurada de manera análoga a lo hecho en el problema directo de la subsección anterior. La función de pérdida a minimizar ha sido:

$$L(\theta, \lambda) = \frac{1}{N_i}\sum_{j=1}^{N_i} |u(t_i^j, x_i^j) - \hat{u}(x_i^j, y_i^j, \theta)|^2 + \frac{1}{N_i}\sum_{j=1}^{N_i} \left| \frac{\partial^2 \hat{u}(x_i^j, y_i^j, \theta)}{\partial x^2} + \lambda \frac{\partial^2 \hat{u}(x_i^j, y_i^j, \theta)}{\partial y^2} - f(x_i^j, y_i^j) \right|^2, \tag{15}$$

Note que ahora en (15), el parámetro λ es incorporado en la función de pérdida como una variable que participa en el proceso de optimización. Entonces, el algoritmo de aprendizaje ofrecerá la solución aproximada de la ecuación diferencial (figura 5) y el valor aproximada del parámetro λ. Además, en (15) consideramos un único conjunto de puntos de entrenamiento $\{(x_i^j, y_i^j)\}_{j=1}^{N_i}$ en todo el dominio Ω, porque a priori conocemos la solución exacta $u = u(x, y)$ del problema (7)-(8).

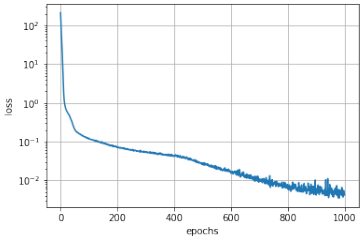
Para la simulación que se presenta, el número de puntos N_i dentro de Ω ha sido de 10.000. El error L_2 que se alcanzó

FIGURE 5. Gráfica de la solución aproximada de (7)-(8). Para el entrenamiento de la red neuronal se consideran dos capas oculta de 10 neuronas cada una y 1.000 épocas en el entrenamiento.



al comparar la solución exacta (5) y la solución aproximada obtenida con 1.000 pasos de entrenamiento (figura 6) ha sido de 0.9608973. El valor aproximado obtenido del parámetro λ a sido 1.005846, entonces al comparar con el valor real $\lambda = 1$, el error absoluto es de 0.005846.

FIGURE 6. Al cabo de las 1.000 pasos en el entrenamiento, la función de pérdida alcanzó una cota inferior de aproximadamente del orden de 10^{-2}.



IV. CONCLUSIONES Y FUTUROS TRABAJOS

El teorema de aproximación universal afirma que una red neuronal artificial que contiene una sola capa oculta puede aproximar a cualquier función arbitrariamente compleja con suficientes neuronas. En esta investigación seSe ha logrado implementar con éxito una solución numérica para un problema directo y un problema inverso asociado a la ecuación de Poisson. La métricas de evalaución resultaron satisfactorias para confirmar que la red neuronal aproxima la solución exacta de un problema de Poisson con condición dirichlet. Respecto al parámetro λ, el valor obtenido fue cercano al valor real, con no más de 1000 pasos de entrenamiento. Lo anterior se explica porque, a diferencia del problema directo, en el problema inverso contamos con una mayor cantidad de datos de entrenamiento. Finamente, la libreria SciAnn resultó eficiente, y cómoda, para realizar las simulaciones numéricas.

Como trabajo futuro, se pretenderá explorar la solución numérica mediante PINN de ecuaciones diferenciales parciales no lineales, como por ejemplo la ecuación de Navier-Stokes.

A modo de comentario final, hoy en día resulta necesario avanzar en resultados teóricos en esta área, para responder a las preguntas: ¿Cuántas neuronas, cuántas capas y cuántos puntos N_i o N_b se necesitan realmente para obtener una cierta precisión? ¿Para operadores diferenciales de orden superior se tendrá también una aceptable precisión con solo dos capas, diez unidades neuronales en cada capa y funciones de activación tangentes hiperbólicas?. Estas y más preguntas aún permanecen abiertas en el área de Physics Informed Neural Network.

V. REFERENCIAS

REFERENCES

[1] I. E. Lagaris, A. Likas, D.I. Fotiadis, "Artificial Neural Networks for Solving Ordinary and Partial Differential Equations" *IEEE Transactions on Neural Networks*. 1998

[2] M. Raissi, P. Perdikaris, G. E. Karniadakis, "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations," *https://arxiv.org/abs/1711.10561*. 2017.

[3] M. Raissi, P. Perdikaris, 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, Volume 378*. 2019.

[4] E. M. S. Springer Verlag GmbH, "Encyclo- pedia of mathematics," *Website, uRL: https: //www.encyclopediaofmath.org/.*

[5] F. Chollet et al., "Keras," *https://github.com/ fchollet/keras* (2015).

[6] M. Abadi et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," *arXiv:1603.04467 [cs.DC]*. 2016.

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

[8] de Figueiredo, D.G. and Metzger, R. and Neves, A.F., "Ecuaciones diferenciales aplicadas," *Textos del IMCA*. 2006.

[9] M. Piscopo, M. Spannowsky, P. Waite, "Solving differential equations with neural networks: Applications to the calculation of cosmological phase transitions," *arXiv:1902.05563v2*. 2019.

[10] K. Hornik, M. Stinchcombe, H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks, Volume 2, Issue 5*. 1989.

[11] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks, Volume 4, Issue 2,*. 1991.

[12] Cybenko, G. "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems (MCSS)*. 1989.

[13] E. Haghighat, R. Juanes, "Sciann: A keras wrapper for scientific computations and physics-informed deep learning using artificial neural networks". 2020.