ISBN: 978-628-95207-4-3, ISSN: 2414-6390, Digital Object Identifier: 10.18687/LACCEI2023.1.1.553

Solving Ordinary Differential Equations through an Artificial Neural Network

Roberto Martínez1, Luis Lara2, Rosa Corti3
1Universidad del Centro Educativo Latinoamericano-UCEL, Universidad Nacional de Rosario, Argentina, romamar@fceia.unr.edu.ar, 2Universidad del Centro Educativo Latinoamericano-UCEL, Instituto de Fisica Rosario (CONICET), Argentina, lplara@gmail.com, 3Universidad Nacional de Rosario, Argentina, rcorti@fceia.unr.edu.ar.

Abstract– ***This paper describes a method for the numerical solution of ordinary differential equations using an artificial neural network. Unlike other reported methods, the loss function is constructed from the same formalization of the differential equation and is done using Pytorch computational resources for self-differentiation. The proposed method is applied in the resolution of four types of differential equations and the results obtained are shown. The design of the resulting neural network was oriented to its subsequent implementation in reconfigurable logic.***

Keywords: Deep learning, Differential equations, Artificial intelligence

Digital Object Identifier: (only for full papers, inserted by LACCEI).
ISSN, ISBN: (to be inserted by LACCEI).
DO NOT REMOVE

Resolución de Ecuaciones Diferenciales Ordinarias por medio de una Red Neuronal Artificial

Roberto Martínez[1], Luis Lara[2], Rosa Corti[3]
[1]Universidad del Centro Educativo Latinoamericano-UCEL, Universidad Nacional de Rosario, Argentina, romamar@fceia.unr.edu.ar, [2]Universidad del Centro Educativo Latinoamericano-UCEL, Instituto de Fisica Rosario (CONICET), Argentina, lplara@gmail.com, [3]Universidad Nacional de Rosario, Argentina, *rcorti@fceia.unr.edu.ar*.

Resumen– Este artículo describe un método para la solución numérica de ecuaciones diferenciales ordinarias utilizando una red neuronal artificial. A diferencia de otros métodos reportados, se construye la función de pérdida a partir de la misma formalización de la ecuación diferencial y se lo hace haciendo uso de los recursos computacionales de Pytorch para la auto diferenciación. Se aplica el método propuesto en la resolución de cuatro tipos de ecuaciones diferenciales y se muestran los resultados obtenidos. El diseño de la red neuronal resultante, estuvo orientado a su posterior implementación en lógica reconfigurable.

.

Palabras clave: Aprendizaje profundo, Ecuaciones diferenciales, Inteligencia artificial

I. INTRODUCCIÓN

Este trabajo se enfoca en la solución numérica de ecuaciones diferenciales ordinarias (EDO) utilizando una Red Neuronal Artificial (RNA). Las ecuaciones diferenciales desempeñan un rol fundamental en la descripción de fenómenos en una amplia gama de aplicaciones científicas e industriales, incluidas la física y la astronomía, la meteorología, la química, la biología, la ecología y el modelado de poblaciones y la economía. Son esenciales para una descripción matemática de la naturaleza y se encuentran en el núcleo de diferentes teorías físicas. Debido a su importancia, existe una profusa literatura sobre procedimientos para su solución. En general, los métodos tradicionales requieren la discretización del dominio en un número de elementos finitos, donde las funciones se aproximan localmente. La solución numérica de ecuaciones diferenciales puede efectuarse por diversos métodos, diferencias finitas (FDM), elemento finito (FEM), volumen finito (FVM) y métodos de Galerkin [1] [2] entre otros. Estos métodos tienen en común el hecho de requerir una malla que da el soporte a la solución numérica con un orden de convergencia algebraico. Aunque estos métodos proporcionan buenas aproximaciones a la solución, la discretización del dominio puede ser un desafío en problemas de dos o más dimensiones. Además, las soluciones aproximadas derivadas son discontinuas y pueden afectar seriamente su estabilidad. Así mismo, para obtener una solución de precisión satisfactoria, puede ser necesario trabajar con discretización muy fina que aumenta significativamente el costo computacional. En los últimos años se han propuestos varios procedimientos de inteligencia artificial, tales como RNA, algoritmos genéticos (AG) y lógica difusa, como técnicas poderosas para resolver una variedad de problemas del mundo real debido a su excelente capacidad de aprendizaje [3][4][5]. En particular, las RNA son aproximaciones de funciones que, entrenadas adecuadamente, pueden generar la función que resuelve aceptablemente las EDO. La mayoría de los métodos numéricos tradicionales de integración, suelen tener un proceso secuencial para avanzar desde un tiempo inicial hasta un punto de tiempo final, mientras que los esquemas de redes neuronales buscan soluciones simultáneamente en todos los puntos de tiempo. Este enfoque tiene sus antecedentes en [6] [7] y se valora por sus ventajas, como el aprendizaje, la adaptación, el cálculo de errores, la tolerancia a fallas y su velocidad de respuesta. Un estado del arte sobre aprendizaje profundo y resolución de educaciones diferenciales puede consultarse en [8]. En los últimos años, las RNA se han empleado con éxito para la solución de ecuaciones diferenciales y han contribuido a esto, los avances en las herramientas computacionales, que facilitan el cálculo diferencial. Los autores en [9] sintetizan cinco ventajas al respecto a) La solución es diferenciable y tiene una forma analítica cerrada, b) El método proporciona una solución con muy buenas propiedades de generalización. c) La complejidad computacional no aumenta rápidamente cuando aumenta el número de puntos de muestreo d) Se pueden incluir parámetros libres en la solución, evitando repetir simulaciones en diferentes condiciones y e) El método se puede implementar en arquitecturas paralelas.

En [10], los autores proponen un método desarrollado en *Pytorch* para resolver ecuaciones diferenciales ordinarias con una RNA. Encuentran, que al usar una función de pérdida convexa y no lineal, se obtiene una solución no única y también concluyen que es más costoso que usar un esquema clásico de diferencias finitas. Sin embargo prevén que el aprendizaje automático y la inteligencia artificial tendrán un gran impacto y consecuencias en la resolución de estas ecuaciones. Los autores en [11] presentan un algoritmo de aproximación y analizan errores de las redes neuronales informadas por la física (PINN) para resolver diferentes tipos de ecuaciones diferenciales y proponen un método de refinamiento adaptativo basado en residuos (RAR) para mejorar la distribución de puntos residuales durante el proceso de entrenamiento y, por lo tanto, aumentar la eficiencia del entrenamiento.

En este trabajo, proponemos un enfoque diferente a los tradicionales basados en mallas y se basa en el aprendizaje profundo con RNA. Asimismo, a diferencia de trabajos reportados sobre RNA en la integración de ecuaciones diferenciales, el método propuesto no hace uso del clásico

entrenamiento supervisado, en el cual se provee al modelo de datos numéricos de entrada y etiquetado, ni se hace necesario suponer previamente una función solución para satisfacer condiciones iniciales.

Las RNAs pueden considerarse como modelos naturales de cómputo paralelo, marcando una diferencia con los AG, respecto a su implementación en lógica reconfigurable. Este proceso de cómputo concurrente es debido a los diferentes tipos de paralelismo que una RNA exhibe. Una RNA posee a) paralelismo en la etapa de entrenamiento b) paralelismo a nivel de capa c) paralelismo a nivel de nodo (externo) d) paralelismo a nivel de nodo y e) paralelismo a nivel de bit [12]. Por estas características, la arquitectura de un FPGA (*Field programable gate array*) resulta ideal para explotar el paralelismo presente en una RNA. En [13] se describe una metodología de creación de prototipos para implementar modelos de redes neuronales profundas en hardware. A partir de un modelo desarrollado en lenguaje de programación C o C ++, desarrollan una arquitectura de hardware utilizando una plataforma virtual SoC (System on Chip) y verifican la funcionalidad mediante placa FPGA. Se muestra que es posible implementar un diseño FPGA que ejecute los cálculos de inferencia requeridos por una red neuronal. Cuando el diseño se implementó en una FPGA, encuentran un aumento de velocidad de más de 5 veces en comparación con procesadores ARM7.

Este documento se organiza de tal forma que en la sección 2, se describe el método utilizado, en la sección 3 se plantean los resultados obtenidos al resolver ecuaciones diferenciales con una RNA, y en la sección 4 se reportan conclusiones y trabajos futuros.

II. MÉTODO PROPUESTO

Una ecuación diferencial ordinaria de primer orden queda definida por las expresiones

$$y' = f(x, y) \quad (1)$$
$$y(x_0) = y_0 \quad (2)$$
$$x_0 = a$$

donde:
a < x < b
f real y continua
$' = d/dx$

Resolver la ecuación (1) significa encontrar la función *y(x)* que satisfaga esta ecuación para un dominio determinado y las condiciones iniciales indicadas en (2). El método reportado en este trabajo se enfoca en el desarrollo de una RNA multicapa *feedforward* densamente conectada, con dos funciones de costo (*loss function* - LF) a verificar en su entrenamiento. La primera función, orientanda a cumplir con (1) se define como

$$LF1 = |y' - f(x, y)| \quad (3)$$

Para resolver el problema del valor inicial (2), se define una segunda función

$$LF2 = |y(x_0) - y_0| \quad (4)$$

A partir de las funciones de costo definidas, los pesos *w* y sesgos *b* se deben encontrar mediante alguna de las diferentes técnicas de optimización que existen en el entrenamiento de redes neuronales. Lo expresado anteriormente significa que la predicción de la RNA, para una determinada entrada (*x*), deberá cumplir con las ecuaciones siguientes:

$$y'_p = f(x, y_p) \quad (5)$$
$$y_p(x_0) = y_0 \quad (6)$$

donde y_p es la predicción (salida) de la red una vez entrenada.

El diseño y entrenamiento de la RNA se desarrolla en *Python*. Para resolver lo planteado, se ha innovado en la lógica del método de entrenamiento haciendo intervenir a la predicción de la RNA y a su derivada, en la función de costo (pérdida) a minimizar. Esto lleva a reformular el planteo usual de la función de costo, que evalúa la desviación entre las predicciones de la RNA y los valores deseados. Las ecuaciones (5) y (6) indican que se deberán cumplir simultáneamente las dos funciones de costo, *LF1* y *LF2*. La función, *LF1*, deberá evaluar la desviación de la predicción de la RNA para cumplir con (5). Esto es, evaluar el grado de cumplimiento de la relación que debe darse entre la entrada de la red, la salida de la red (predicción) y su derivada. En función de esta evaluación, el algoritmo de entrenamiento de la red, deberá ajustar progresivamente los parámetros de pesos y sesgos para cumplir con ella. La (3) reformulada para para el diseño del modelo es ahora:

$$LF1_p = y'_p - f(x, y_p) \quad (7)$$

El ambiente *Pytorch* provee de una herramienta para el cálculo del grafico computacional, que permite visualizar la estructura de las operaciones matemáticas que se realizan en un modelo de aprendizaje profundo. En el proceso de entrenamiento de la red, es particularmente de utilidad esta herramienta para el cálculo de los gradientes de manera eficiente [14]. Para el cálculo de y'_p se utiliza el módulo *Autograd* (diferenciación automática o auto diferenciación) que hace uso del grafico computacional, y facilita el cálculo automático de derivadas de una función [15].

La función *LF2* se incorpora al método de cálculo para que la salida de la red cumpla también con el problema del valor inicial, se reformula entonces la (4) para su incorporación al algoritmo de entrenamiento:

$$LF2_p = y_p(x_0) - y_0 \quad (8)$$

De esta forma, a la RNA se le exige que cumpla, simultáneamente, con las dos funciones de pérdida. Para minimizar la función final resultante, se aplica el método de Error Cuadrático Medio (*MSELoss*).

Finalmente, se realiza el proceso de propagación hacia atrás [16], y optimización para actualizar los pesos y sesgos de la red neuronal. El optimizador Adam [17] de *PyTorch* que se adopta, es un algoritmo de optimización de gradiente descendente estocástico que combina las técnicas de gradiente descendente y gradiente descendente con momento y utiliza un método de adaptación de tasas de aprendizaje por parámetro, permitiendo una convergencia más eficiente del entrenamiento. En Fig. 1 se esquematiza este proceso.

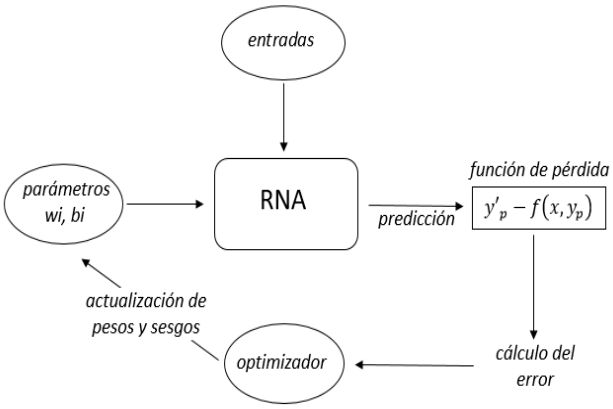

Fig. 1 Proceso de entrenamiento de la RNA

III. RESULTADOS

En esta sección evaluamos el desempeño del modelo propuesto para integrar EDOs, comparando sus resultados con soluciones analíticas en diversos ejemplos. Como se mencionó en la sección 2, desarrollamos la red neuronal en Python utilizando el *framework Pytorch*, el cual está diseñado para el cálculo con tensores.

Debido a que preveíamos necesitar recursos computacionales adicionales, optamos por trabajar en la nube utilizando *Google Colaboratory*, que nos brindó la posibilidad de acceder a una GPU (*Graphics Processing Unit*).

El flujo de trabajo fue el siguiente:

- Arquitectura de la red: definición de hiperparámeros (tasa de aprendizaje, función de activación, cantidad capas internas y neuronas por capa, tamaño de lotes de datos, cantidad de épocas)
- Planteo de las funciones de pérdida
- Definición del método computacional para el cálculo de la derivada de las salidas de la red
- Definición de variables y métricas a evaluar
- Desarrollo del modelo y su entrenamiento
- Ensayos y ajustes de los hiperparámetros

A. *Ejemplo 1*

El primer caso que se toma es la resolución de la EDO descripta por

$$y' = 2x + 1 \tag{9}$$

La solución explicita de (9) con el valor inicial *y(0)=0* , resulta ser

$$y = x^2 + x \tag{10}$$

Para resolver la EDO planteada utilizando el modelo numérico se siguieron los pasos establecidos al comienzo de esta sección cuyo resultado es el desarrollo de una red con las siguientes características:

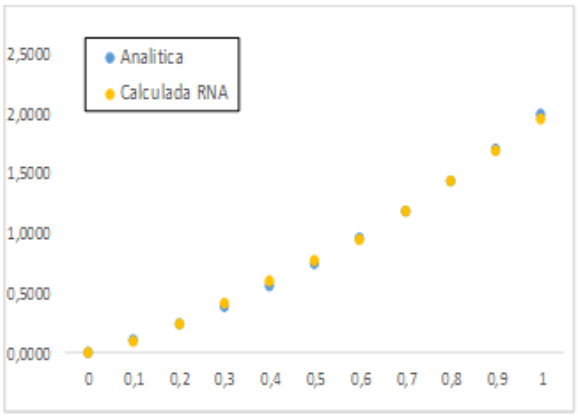

Fig. 2 Comparación entre los valores de y(x) calculados analiticamente y los obtenidos de la RNA

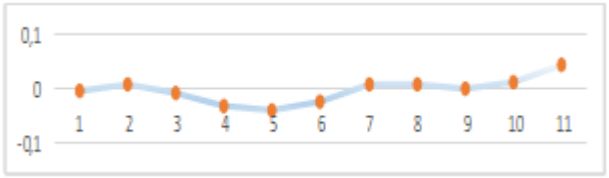

Fig. 3 Diferencia de valores entre las soluciones

- Red multicapa *feedforward* densamente conectada con dos capas internas de cinco neuronas y función de activación *ReLu* (rectificación lineal) cada una. una capa de salida con función de activación lineal
- Factor de aprendizaje 0,005
- Lotes de entrenamiento de 20 números aleatorios en el intervalo [0,1]
- Entrenamiento con 500 épocas

En la Fig. 2 se grafican los valores de *y(x)* calculados a partir de la solución analítica y los hallados con la RNA. En la Fig. 3 se grafica la diferencia de valores entre la solución analítica y la obtenida por la RNA.

En el dominio de datos utilizados, el error relativo medio resultó de 0,0169.

Se realizó una prueba cambiando únicamente la función de activación por una sigmoidal (*Sigmoid*). Se entrena la red con este único cambio y en 246 epocas, un cantidad significativamente menor que en el esquema anterior, se obtiene un error relativo medio de 0,012.

B. *Ejemplo II*

En este caso se ensaya una ecuación diferencial con una función trigonométrica:

$$y' = \cos(x) \tag{11}$$

La solución explicita de (11) con el valor inicial *y(0)=0*, resulta

$$y = \text{sen}(x) \tag{12}$$

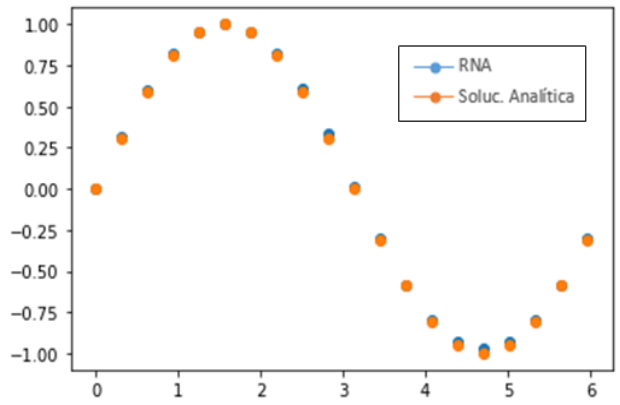

Fig. 4 Valores de *y(x)* calculados a partir de la solución analítica y los obtenidos por medio de la RNA del ej.*II*

Para el cálculo numérico de integración de la EDO se diseña un RNA con las siguientes características:

- Red multicapa *feedforward* densamente conectada con tres capas internas de cinco neuronas y función de activación *Sin* (función seno) cada una. Capa de salida de una neurona con función de activación lineal
- Factor de aprendizaje 0,001
- Lotes de entrenamiento de 50 números aleatorios en el intervalo [0, 2π]
- Entrenamiento con 500 épocas

En la Fig. 4 se grafican los valores de *y(x)* calculados a partir de la solución analítica y los obtenidos por medio de la RNA, para una entrada representada por 20 valores distribuidos entre 0 y 2π. En la Tabla I se indican los resultados del análisis de regresión.

En los ensayos para resolver esta ecuación diferencial, se probaron también las funciones de activación rectificador lineall (*ReLu*) y sigmoidal (*Sigmoid*). En el primer caso no fue posible hallar un desempeño aceptable de la RNA, pese a haber probado con distintos cantidad de niveles ocultos y neuronas por nivel. Por el contrario, con la función sigmoidal, la red demostró resolver la EDO aunque con un error mayor del logrado con la función seno.

TABLA I
ANALISIS DE REGRESIÓN DEL EJEMPLO. II

Estadísticas de la regresión	
Coeficiente de correlación múltiple	0,999532551
Coeficiente de determinación R^2	0,99906532
R^2 ajustado	0,99898743
Error típico	0,018885157
Observaciones	20

.

C. *Ejemplo III*

En este ejemplo tomamos una EDO no lineal, como es el caso de la ecuación

$$y' = -y^2 \tag{13}$$

cuya solución explicita con valor inicial *y(0)=1* , resulta

$$y = \frac{1}{1+x} \tag{14}$$

La red neuronal que se desarrolla para resolver la EDO es de dos capas internas de quince neuronas cada una y función de activación *ReLu*. Capa de salida de una neurona con función de activación lineal. El entrenamiento se realizó con lotes de entrenamiento de 50 números aleatorios en el intervalo [0, 1] y 800 épocas. El factor de aprendizaje se estableció en 0,001. En la Fig. 5 se representa la evolución de la función de pérdida en función de la época de entrenamiento y se puede observar el rápido decrecimiento de aquella, obteniéndose valores cercanos a cero a partir de la época 300.

En la Fig. 6 se compara la solución analítica de la EDO estudiada, con la calculada por la RNA, para una entrada representada por el vector [0, 0,2, 0,4, 0,6, 0,8, 1].

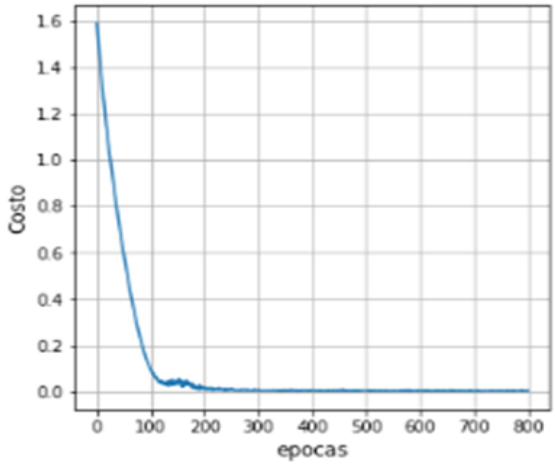

Fig. 5 Evolución de la función de perdida del ejemplo III

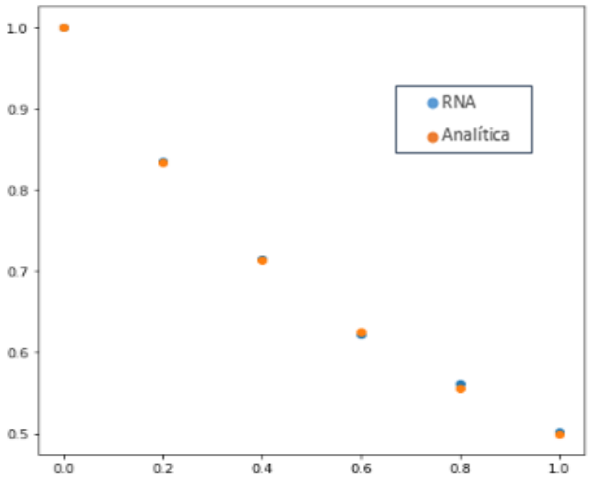

Fig. 6 Valores de *y(x)* calculados a partir de la solución analítica y los obtenidos por medio de la RNA del ejemplo III

D. *Ejemplo IV*

En este último ejemplo se resuelve la EDO:

$$y' = x.y \quad (15)$$

Cuya solución analítica con el valor inicial $y(0) = 1$ resulta

$$y = e^{-x2/2} \quad (16)$$

Para el cálculo numérico de (15) se diseñó una RNA multicapa *feedforward* densamente conectada de tres capas ocultas de ocho neuronas y función de activación ReLu cada y una capa de salida de una neurona con función de activación lineal. Se entrena la red en 500 épocas con lotes de 50 números aleatorios en el intervalo [0, 1].

En la Fig. 7 se compara la solución analítica de la EDO, con la calculada por la RNA, para una entrada representada por el vector [0, 0,2, 0,4, 0,6, 0,8, 1]. Las estadísticas de regresión se muestran en la Tabla II

TABAL II
ANALISIS DE REGRESIÓN DEL EJEMPLO IV

Estadísticas de la regresión	
Coeficiente de correlación múltiple	0,99893217
Coeficiente de determinación R^2	0,99786548
R^2 ajustado	0,99733185
Error típico	0,00798532
Observaciones	6

Para resolver este mismo caso, se modificó la red original, suprimiendo una capa interna y cambiando las funciones de activación de tipo *ReLu* a *Sigmoid*. En Fig. 8 se representa la evolución de la función de costo en las 400 épocas de entrenamiento de la RNA con funciones de activación rectificador lineal. En Fig 9, la misma representación resultado de una RNA de una capa oculta menos y con funciones de activación sigmoidal. Obsérvese que se lograron prestaciones similares, con una capa menos, pero con el costo de usar una función de activación más compleja de sintetizar en hardware.

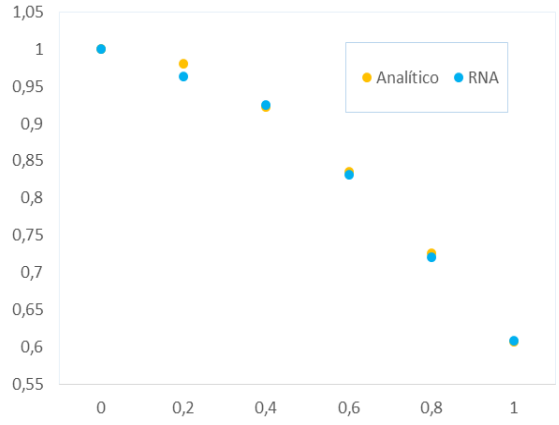

Fig. 7 Comparación de los valores de y(x) obtenidos analíticamente y los calculados por la RNA.

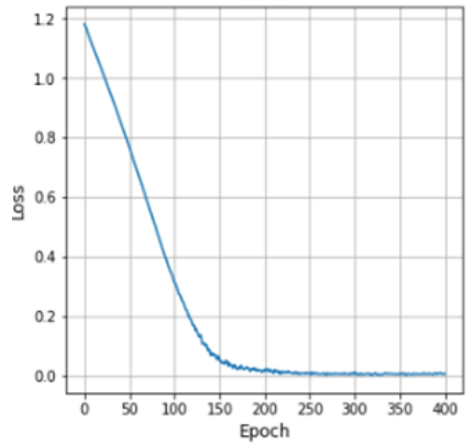

Fig. 8 Función de costo con activación *ReLu*

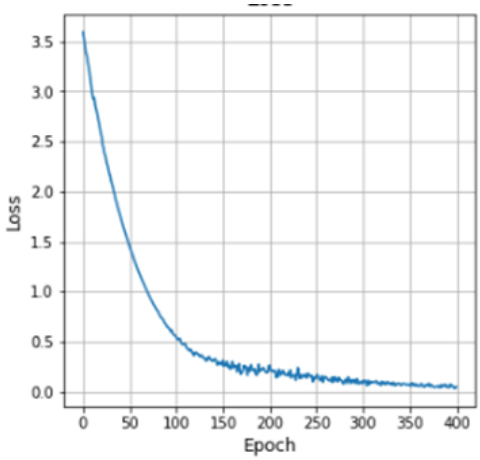

Fig. 9 Función de costo con activación *Sigmoid*

IV. CONCLUSIONES

Se desarrolló un método para la solución de ecuaciones diferenciales ordinarias basado en una red neuronal artificial. A diferencia de otros métodos propuestos en trabajos que se relacionan, no se hace uso de un entrenamiento supervisado clásico, proveyendo a la red datos de entrenamientos y sus etiquetas correspondientes (*target*), sino que se construye la función de pérdida a partir de la misma ecuación diferencial como objetivo de aprendizaje. Para lograrlo, se hizo uso de los recursos computacionales de *Pytorch* para la auto diferenciación. Como se mostró en los resultados de los ejemplos desarrollados, la función solución es continua en el dominio definido y la diferencia con la solución explicita puede hacerse tender a cero a expensa del costo computacional (complejidad de la RNA y ciclos de entrenamiento). En los ensayos realizados, los diseños se realizaron utilizando funciones de activación que facilitaran que la RNA pueda ser implementada en hardware, sobre una plataforma de lógica reconfigurable (*Field Programmable Gate Array – FPGA*). Si bien existen desarrollos de implementación de estas redes en FPGA, hacer que estos diseños sean parametrizables, en lo que se refiere a número de entradas y salidas, cantidad de capas, función de activación para cada capa y número de neuronas por capa, es un aspecto

no tenido en cuenta en los trabajos consultados. Además, se prevé definir mediante parámetros, el número de bits de entradas, pesos y sesgos, teniendo en cuenta las definiciones adoptadas de forma que los resultados estén correctamente representados. Es precisamente este enfoque el que se prevé desarrollar en trabajos futuros.

REFERENCIAS

[1] Duran Ricardo, "Galerkin approximations and finite element methods", Departamento de Matemática, Facultad de Ciencias Exactas, Universidad de Buenos Aires, 1428 Buenos Aires, Argentina. http://mate.dm.uba.ar/~rduran/class_notes/fem.pdf

[2] Pirozzoli, S., "Numerical methods for high-speed flows". Annual review of fluid mechanics, 43, 163-194, 2011

[3] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deeplearning framework for solving forward and inverse problems involving nonlinear partial differential equations", J. Comput. Phys., 378, pp. 686—707, 2019.

[4] Samaniego E et al, "An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications", Computer Methods Methods in Applied Mechanics and Engineering. 362 112790, 2020.-

[5] Michoski, Craig, et al. "Solving differential equations using deep neural networks." Neurocomputing 399: 193-212, 2020.

[6] Lagaris I., Likas A., Fotiadis D., "Artificial neural networks for solving ordinary and partial differential equations", IEEE Trans. Neural Netw. 9 (5) 987–1000, 1998 http://dx.doi.org/10.1109/72.712178.

[7] Meade Jr, Andrew J., Fernandez Alvaro A.. "The numerical solution of linear ordinary differential equations by feedforward neural networks." *Mathematical and Computer Modelling* 19.12: 1-25, 1994.

[8] Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. , "DeepXDE: A deep learning library for solving differential equations". *SIAM review*, *63*(1), 208-228, 2021.

[9] Yadav, Neha, Anupam Yadav, Manoj Kumar. "An introduction to neural network methods for differential equations". Vol. 1. Berlin: Springer, 2015.

[10] Knoke, Tobias, Wick, Thomas, "Solving differential equations via artificial neural networks: Findings and failures in a model problem." Examples and Counterexamples 1:100035, 2021.

[11] Raissi, M., Perdikaris, P., Em Karniadakis, G., " 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, pp 686-707, ScienceDirect, 2019.

[12] Omondi, A. R., Rajapakse, J. C., and Bajger, M. "FPGA neurocomputers". In FPGA Implementations of Neural Networks, Springer, PP. 1–36. 2006.

[13] W. Kim and H. Jun, "Fast Prototyping of a Deep Neural Network on an FPGA," 2020 International SoC Design Conference (ISOCC), 2020, pp. 214-215, doi: 10.1109/ISOCC50952.2020.9333030.

[14] Pytorch, Diferenciación automática https://pytorch.org/docs/stable/autograd.html?highlight=autograd

[15] Pytorch Grafico computacional, https://pytorch.org/blog/computational-graphs-constructed-in-pytorch/

[16] Xinghuo Yu, M. O. Efe and O. Kaynak, "A general backpropagation algorithm for feedforward neural networks learning," in IEEE Transactions on Neural Networks, vol. 13, no. 1, pp. 251-254, Jan. 2002.

[17] Pytorch, Optimizador Adam https://pytorch.org/docs/stable/optim.html#torch.optim.Adam