Results in Nonlinear Analysis 8 (2025) No. 2, 148–153
https://doi.org/10.31838/rna/2025.08.02.013
Available online at www.nonlinear-analysis.com

Results in Nonlinear Analysis
ISSN 2636-7556

Peer Reviewed Scientific Journal

Optimization of ordinary differential equations (ODE) solutions using modified recurrent neural networks

Aseel Najih Abbas Hassan Al-Maamouri

University of Al-Mustansiriya, College of Administration and Economics, Department of Information Technology

Abstract

This paper introduces a hybrid approach for solving ordinary differential equations (ODE) using modified recurrent neural networks (mRNNs). The approach combines mRNNs with novel optimization techniques. Crucially, when training an mRNN, training data points should be selected from the open interval (a, b) to avoid training the network with the boundary points. This approach reduces computational errors by avoiding boundary region training. Furthermore, we propose a transformation that maps training points from a potentially broader interval [a, b] into corresponding points within the open interval (a, b), before training. This allows the network to be trained on points that are similar in the open interval, which leads to improved accuracy. The proposed model demonstrates higher accuracy compared to existing mRNN models. A numerical example and corresponding simulations demonstrate the mathematical effectiveness of this approach.

Mathematics Subject Classification (2020): 65L05, 68T07, 93A30

Key words and phrases: S-Curve Function; Modified Recurrent Neural Networks (mRNN); Ordinary Differential Equations (ODE); Alternative Activation Function (AAF); Identity Function (IdFn).

1. Introduction

Differential equations are fundamental tools across numerous disciplines, including physics, chemistry, mechanics, and economics, providing a framework for problem-solving. Many real-world problems and natural phenomena can be modeled using differential equations. Solving these equations enables

Email address: al9625510@gmail.com (Aseel Najih Abbas Hassan Al-Maamouri)

Received April 7, 2025; Accepted May 19, 2025; Online June 30, 2025

us to describe and predict the behavior of these systems [1]. However, many differential equations lack analytical solutions, or their solutions are difficult to interpret. Consequently, various numerical algorithms have been developed to approximate solutions, such as Runge-Kutta, Adomian, Adams-Bashforth, Adams-Moulton, and Predictor-Corrector methods. More recently, neural networks have emerged as a powerful technique for approximating solutions to ODEs, partial differential equations (PDEs), and fuzzy differential equations (FDEs). As early as 1990, Lee and Kang demonstrated the use of Hopfield neural networks with parallel processing to solve first-order differential equations [2]. Later, Meade and Fernandez employed feed-forward neural networks combined with degree-one B-splines to tackle both linear and nonlinear ODEs. However, scaling these methods to multidimensional problems presents a significant challenge [3]. In contrast, a numerical technique proposed by Malek, Shekari, and Jamme, building upon work by Jamme and Liu, utilizes neural networks and optimization methods to approximate solutions to higher-order differential equations. This method seeks a solution as a combination of two analytical functions: one satisfying initial/boundary conditions, and the other incorporating parameters within the neural network [4]. The hidden layer of the neural network typically employs an activation function (AAF), often a sigmoid or hyperbolic tangent function. This paper is structured into five sections. Section 2 provides foundational definitions and theorems necessary for understanding the methodology. Section 3 introduces the proposed model for solving ordinary differential equations. Section 4 presents illustrative examples, and Section 5 offers concluding remarks.

2. Backgrounds

2.1 The S-curve function

S-curves are also known as Logistic Function, which depict patterns of gradual growth at the beginning, rapid acceleration midway, and leveling off near the end of a project. An S-shaped graph is often used to visualize the progress of a project over time, such as in project management, where cumulative data such as cost or hours worked are tracked [5, 6].

2.2 Activating Functions

The activation function in a RNN is used for limiting the output neurons in the network. A combination of IdFn and AAF was used in this paper. RNN algorithm is introduced that uses one conversion function [7, 8].

2.3 Multilayer Perceptron Network Training (MLP Network Training)

RNN training utilizes backpropagation (BP), an error correction method. This requires calculating derivatives of the neuron activation functions to determine sensitivities across the multi-layer perceptron (MLP) network [9]. Therefore, differentiable activation functions are necessary. We previously covered the characteristics of activation functions (specifically AAF), and the following sections will detail the error function [10].

2.4 Tools for Building World Approximations

A multilayer perceptron (MLP) with one hidden layer, using an S-curve activation in the hidden layer and a linear output layer, can approximate any continuous function to any desired degree of accuracy. The quality of this approximation is measured by the integral of the squared error [6, 11].

3. Motivation

This work aims to present a hybrid two-layer RNN method for solving ODEs. The approach involves increasing the dimensionality of the input data and employing optimization techniques. The RNN,

initially developed for function approximation, is adapted for ODE solutions. The initial weights of the two-layered network model are randomly initialized.

4. Problem Statement

An initial value differential equation with a first-order value:

$$\left\{\begin{array}{c}\dfrac{dy(x)}{dx}=f(x,y(x),x\epsilon\left[a,b\right],\\ y(z)=a,\end{array}\right\} \tag{1}$$

Trial functions can be expressed in the form below:

$$y_T(x,y)=\alpha+(x-a)N(x,p) \tag{2}$$

This function is composed of two separate sections. The first section establishes the initial condition, and the second integrates a neural network with adjustable parameters. The error minimization process follows a specific pattern:

$$E(p)=\sum_{i=1}^{m}\left(\frac{dy_T(x_i,p)}{dx}\right)f(x_i,y_i,p))^2 \tag{3}$$

A discrete point with coordinates $[u_i]_{i=0}^{n-1}$ belongs to the interval [a, b]. In (3), we get the following result when we differentiate from trial function $y_T(x_i,p)$

$$\frac{d_{yT}(x,p)}{dx}=N(x,p)+(x-a)\frac{dN(x,p)}{dx} \tag{4}$$

Because the derivatives of the Activation Approximation Functions (AAFs) used in feed-forward networks and S-curves are directly linked to the transformation function's value, differentiation is not required [12]. This characteristic makes AAFs useful in the hidden layers of neural networks. While less precise than functions like the Sech function [4, 13], AAFs offer flexibility. Our proposed mRNN model uses a single hidden layer with AAFs and an Identity Function (IdFn) in the output layer. The key advantage lies in the capacity to choose activation functions (AAFs) within the hidden layer, allowing for precise control over the accuracy achieved. Our model offers a simpler alternative to existing neural network methods for solving ordinary and fuzzy differential equations. These current methods frequently rely on complex architectures to reach comparable accuracy. In our model, wj represents the weight connecting the input to the jth hidden unit, vj represents the weight from the jth hidden unit to the output, bj is the bias for the jth hidden unit, and nj is the output of that unit. Building on this, Equation (4) defines N(x, P) and its derivative, dN(x,p)/dx [1, 15].

$$N=\sum_{j=1}^{H}VjS(nj) \tag{5}$$

$$S(nj)=\frac{2}{e^{nj}+e^{-nj}},nj=wjQ(x)+bj \tag{6}$$

So that,

$$Q(x)=(x+1)\varepsilon,\varepsilon\in(0,1) \tag{7}$$

Therefore,

$$Q(x) \in (a,b) \tag{8}$$

The training process involves selecting points based on their similarity within the open interval (a, b). First, points similar to those in (a, b) are transformed, and then a neural network is trained using these transformed points, considering the distance represented by the interval [a, b] [16].

$$\frac{dN}{dx} = \sum_{j=1}^{H} v_j w_j S'(n_j) = \sum_{j}^{H} v_j w_j \frac{-2\left(e^{wj\,Q(x)+bj} + e^{-wj\,Q(x)-bj}\right)}{e^{wj\,Q(x)+bj} + e^{-wj\,Q(x)-bj})^2} \tag{9}$$

Based on the interpretations described in (5) and (9), (3) can be rewritten in the following form:

$$E(p) = \sum_{i=1}^{m}((xi-a)\sum_{j=1}^{H} v_j w_j \frac{-2\left(e^{wj\,Q(xi)+bj} + e^{-wj\,Q(xi)-bj}\right)}{e^{wj\,Q(xi)+bj} + e^{-wj\,Q(xi)-bj})^2} \tag{10}$$

$$-\sum_{j=1}^{H} v_j \frac{2}{e^{wj\,Q(xi)+bj} + e^{-wj\,Q(xi)-bj}} + f(xi, yi(xi,p))^2$$

5. Numerical results and discussion

5.1 Numerical Examples

This section details the behavior and properties of the new method. A single example is used to discuss the simulation results. The simulation was conducted on Matlab 2012 [17]. Randomly selected weights were used as the initial weights.

$$\left\{\begin{array}{c} \frac{dy(x)}{dz} = 4.x^3 - 3x^2 + 2, x \in [0,1] \\ y(0) = 0 \end{array}\right\} \tag{11}$$

It is clear from the solution to (7) that y(x) is the product of :

$$y(x) = x^4 - x^3 + 2x \tag{11}$$

The trail solution can be summed up as follows:

$$y_T(x) = xN(x,p) = x\sum_{j=1}^{H} \frac{2}{e^{wj\,Q(xi)+bj} + e^{-wj\,Q(xi)-bj}} \tag{12}$$

In this case, the error function will take the following form:

$$E(p) = \sum_{i=1}^{m}(\sum_{j=1}^{H} v_j \frac{2}{e^{wj\,Q(xi)+bj} + e^{-wj\,Q(xi)-bj}} + \sum_{j=1}^{H} v_j wj \frac{-2\left(e^{wj\,Q(xi)+bj} + e^{-wj\,Q(xi)-bj}\right)}{e^{wj\,Q(xi)+bj} + e^{-wj\,Q(xi)-bj})^2} - \left(4x^3 - 3x^2 + 2\right))^2 \tag{13}$$

This example demonstrates training an error function with a neural network that includes a hidden layer of 5 AAF units. The training will use " = 0.4 and m = 6, representing 6 equally spaced points within the interval [0, 1]. Table 1 displays the optimal weights and biases obtained from this training. Table 2 presents the values of both the analytical solution and the trial function. Finally, Figure 1 illustrates the transient behavior of the model in relation to changes in its weights and biases.

There are several ways in which a network's E(p) error can be expressed. For example, a network's E(p) error would be: 2.7557e 07. From that value, it would be : 2.7557e 0.007.

Table 1: Shows an optimal weighting and biasing formula the optimal values of weights and biases.

Version No.	Ver_No.1	Ver_No.2	Ver_No.3	Ver_No.4	Ver_No.5
V_i	4.0386	3.5615	-0.7124	8.5625	5.0682
Wi	3.606	7.0895	3.1273	-3.6432	6.7945
bi	0.3072	3.3185	1.2374	7.4252	8.7846

Table 2: Illustrates the comparison between the precise solution (ya) and its approximation (yt).

Version No.	Ver_No.1	Ver_No.2	Ver_No.3	Ver_No.4	Ver_No.5	Ver_No.6
xi	0	1.2231	1.4231	1.6231	1.8231	2.0231
yi	0	1.4167	1.7847	2.1366	2.5207	3.023
ya	0	1.4167	1.7847	2.1367	2.5207	3.0231

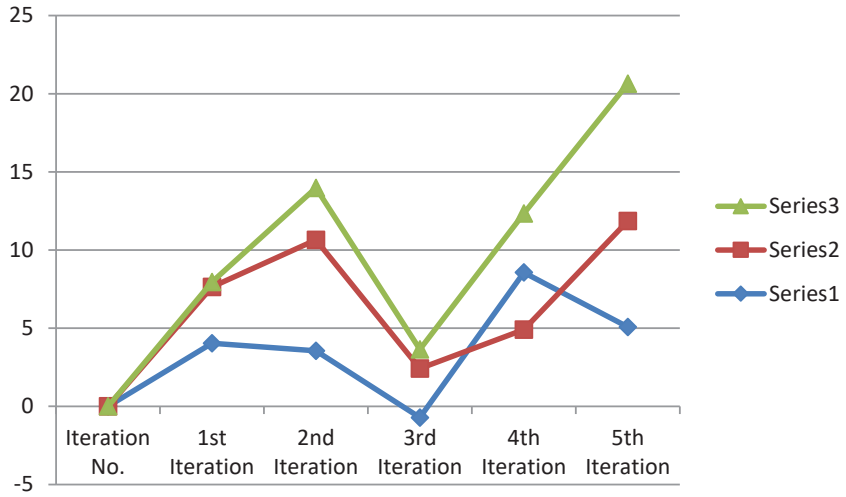

Figure 1: A formula for determining the optimal weights and biases.

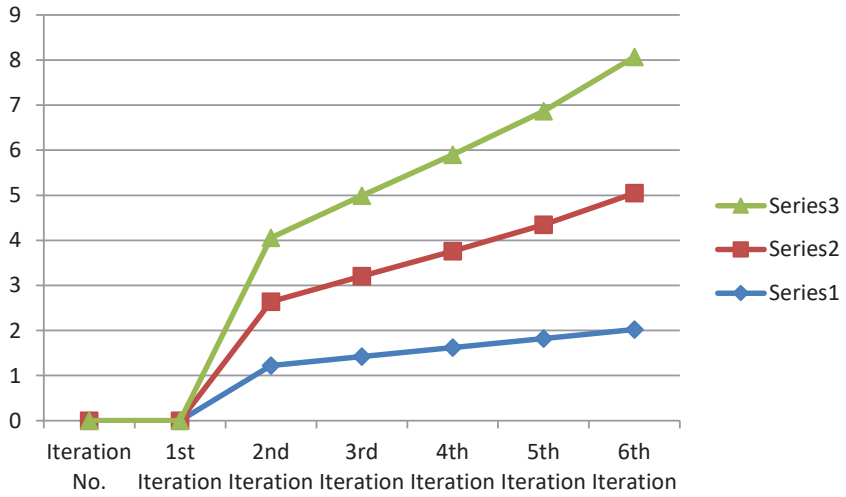

Figure 2: The algorithm computes the deviation of the approximate solution (yt) from the precise solution (ya).

Conclusion

The derivatives of the activation functions (AAFs) in RNNs (Recurrent Neural Networks) and S-Curves are directly determined by the transformation function itself, so we don't need to differentiate them separately. The Sech activation function, although a possibility for hidden layers in neural networks, generally yields lower accuracy compared to alternatives like the exponential secant. However, our research demonstrates that using our modified neural network architecture from Section 3, the desired accuracy can be regained even with Sech in the hidden layer. This approach allows us to achieve similar accuracy to standard neural networks when tackling ordinary and fuzzy differential equations.

References

[1] M. Abadi et al., TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
[2] A. Bachouch et al., Deep neural networks algorithms for stochastic control problems on finite horizon: Numerical applications, arXiv:1812.05916, 2020.
[3] C. Basdevant et al., Spectral and finite difference solutions of the Burgers equation, Comput. & Fluids 14 (1986), 23–41.
[4] A. G. Baydin et al., Automatic differentiation in machine learning: A survey, J. Mach. Learn. Res. 18 (2018), 43 Paper No. 153.
[5] C. Beck and A. Jentzen, Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations, J. Nonlinear Sci. 29 (2019), 1563–1619.
[6] C. Beck et al., Solving stochastic differential equations and Kolmogorov equations by means of deep learning, arXiv:1806.00421, 2018.
[7] C. Beck et al., Deep splitting method for parabolic PDEs, arXiv:1907.03452, 2019.
[8] C. Beck et al., Overcoming the curse of dimensionality in the numerical approximation of Allen–Cahn partial differential equations via truncated full-history recursive multilevel Picard approximations, J. Numer. Math. 28 (2020), 197–222.
[9] C. Beck et al., An overview on deep learning-based approximation methods for partial differential equations, arXiv:2012.12348, 2020.
[10] R. Bellman, Dynamic programming, Princeton University Press, Princeton, NJ, 1957.
[11] J.-D. Benamou, B. D. Froese, and A. M. Oberman, Two numerical methods for the elliptic Monge-Ampère equation, M2AN. Math. Modell. Numer. Anal. 44 (2010), 737–758.
[12] C. Bender and R. Denk, A forward scheme for backward SDEs, Stochastic Process. Appl. 117 (2007), 1793–1812.
[13] C. Bender, N. Schweizer, and J. Zhuo, A primal–dual algorithm for BSDEs, Math. Finance 27 (2017), 866–901.
[14] P. Beneventano et al., High-dimensional approximation spaces of artificial neural networks and applications to partial differential equations, arXiv:2012.04326, 2020.
[15] Y. Bengio, Learning deep architectures for AI, Now Publishers Inc., 2009.
[16] J. Berg and K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018), 28–41.
[17] J. Berner, M. Dablander, and P. Grohs, Numerically solving parametric families of high-dimensional Kolmogorov partial differential equations via deep learning, Adv. Neural Inform. Process. Syst. 33 (2020).