Research Article ISSN Online: 2783-4859 Computational Mathematics and Computer Modeling with Applications

Received 9 May 2023
Accepted 30 June 2023

DOI: 10.52547/CMCMA.1.1.86
AMS Subject Classification: 68T07; 33C45

Hermite neural network for solving the Blasius equation

Aida Pakniyat[a] and Kourosh Parand[a]

In this paper, we propose a Hermite neural network method for solving the Blasius equation, a nonlinear ordinary differential equation defined on the semi-infinite interval. In this work, Hermite functions are transformed using variable transformation in a semi-infinite domain. Hermite functions are used for the first time in a neural network to solve Blasius differential equations, making this method better than existing networks. This method is efficient for solving differential equations. In this paper, we explore the benefits of using the backpropagation algorithm to update parameters for neural networks. By applying this approach, we can successfully avoid issues such as overflow and local minima, which are common challenges associated with other optimization methods. The results obtained are compared with other methods to validate the proposed method and presented in both graphical and tabular form. Copyright © 2022 Shahid Beheshti University.

Keywords: Hermite Functions; Neural Network; Collocation Method; The Blasius equation; Nonlinear ODE.

1. Introduction

The study of fluids has gained popularity in physics and fluid mechanics due to the description of flat plates representing boundary layers with zero-pressure gradients. Boundary layer flow is one of the fundamental problems in fluid dynamics that has long been studied by researchers. Most of the boundary layer is made up of a turbulent zone with eddies and fluctuations of flow parameters. However, in the vicinity of the wall, the fluctuations are damped, and laminar flow conditions prevail.

In 1931, Falkner and Skan expressed the Falkner-Skan equation, which is a nonlinear differential equation in fluid dynamics. This equation allows for some velocity profiles [7, 41]. If a fluid passes over a flat surface near the solid surface boundary, a layer of fluid called the boundary layer forms, and there are significant viscosity effects in this area. The law of mass conservation states that the fluid in a material volume within a closed material surface cannot disappear or appear without the existence of internal sources or sinks. Blasius proposed a solution method for solving equations in the parallel flow of a flat plate, which is induced by changing the pipe geometry.

Several methods have been proposed for solving unbounded domain problems [2, 3, 39]. Spectral methods [1, 15, 29, 40] are one of the most popular methods for solving such problems. Boyd studied some spectral methods in the semi-infinite domain [4]. Parand et al. [29, 34] compared rational and fractional polynomials to solve nonlinear ordinary differential equations on unbounded domains. Guo [10, 11] used the Jacobi and Gegenbauer collocation method. Shen [20] considered the Hermite spectral method on unbounded intervals.

Numerous numerical methods have been used to solve this problem [13, 18, 21, 35, 43]. In their study, Parand et al. used the generalized Laguerre function to solve the Blasius equation [34].

One of the methods for solving differential equations is the implementation of an artificial neural network. Recently, this method has attracted the attention of researchers because of its convergence and fast run time. In machine learning, the universal approximation theorems state that artificial neural networks can approximate any function and learn and represent it with any given precision. The use of machine learning for numerical solving began in the late 1990s [8, 17, 23]. For solving higher-order ordinary differential equations, hybrid methods based on machine learning and optimization methods have been used [24, 28, 31, 44]. Support vector machines have recently been considered as one method of solving differential equations [6, 22, 30, 33, 42].

[a] *Department of Computer and Data Sciences, Faculty of Mathematical Sciences, Shahid Beheshti University, Tehran, Iran. Email: a_pakniyat@sbu.ac.ir; k_parand@sbu.ac.ir.*
[*] *Correspondence to: K. Parand.*

Parand et al. have used a neural network and deep neural network for solving differential equations on an unbounded domain [25, 26, 38]. Gao [9] applied convolutional neural networks for solving parameterized steady-state PDEs on irregular domains. The authors [5, 23, 27, 36] used artificial neural networks for the solution of differential equations. Parand [38] also introduced a mixture of a new set of fractional orthogonal polynomials and the artificial neural network to solve nonlinear differential equations on unbounded domains. In this paper, we present a numerical method for the Blasius differential equations, which is one of the fundamentals of fluid dynamics, with the Hermite neural network. By using this method, there is no need to recalculate the parameters of the system model, which can be described based on the condition of the system's status.

The remainder of this letter is organized as follows: In section 2, we explain the implementation of deep learning for solving differential equations, and in section 3, we demonstrate the numerical experiment's results. Finally, in section 4, we present our conclusions

2. Implement

In this section, we summarize the method for solving the Blasius equation.

2.1. Blasius equation

Ludwig Prandt used scaling arguments to prove that about half of the terms in the Navier-Stokes equations are negligible in boundary layer flows (except near the leading edge of a plate). In fluid dynamics, the boundary layer that forms near a flat plate with fluid moving by it is described using the boundary layer condition and its initial condition [14, 18, 34].

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \text{ (continuity)}, \tag{1}$$

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\frac{\partial^2 u}{\partial y^2} \text{ (momentum)}, \tag{2}$$

a flat plate problem has the following boundary conditions

$$\begin{cases} u(0,y) = U_\infty, \quad u(x,0) = v(x,0) = 0, \\ u(x,\infty) = U_\infty, \end{cases} \tag{3}$$

Here, ν is the kinematic viscosity and U_∞ denotes the constant free stream velocity. The velocity components in the x coordinate parallel to the plate in the direction of the flow are denoted by u and v, while the y coordinate is normal to the plate. The pressure is denoted by p, and the density by ρ.

According to Blasius, the Prandtl x-momentum equation has a self-similar solution when $\frac{\partial p}{\partial x} = 0$. By introducing the dimensionless similarity variable η, the stream function f becomes:

$$\eta = \sqrt{\frac{m+1}{2}}\sqrt{\frac{U}{vx}}y, \quad \phi = \sqrt{\frac{2}{m+1}}\sqrt{vxU}f(\eta), \tag{4}$$

and $u = \frac{\partial\phi}{\partial y}$ and $v = \frac{\partial\phi}{\partial x}$ velocity components can be expressed as follows:

$$u = Uf^{'}(\eta), \quad v = -\frac{1}{2}\sqrt{\frac{m+1}{2}}\sqrt{\frac{U}{vx}}(f + \frac{m-1}{m+1}\eta f^{'}), \tag{5}$$

$$m = \frac{\beta}{2-\beta} \Rightarrow \beta = \frac{2m}{m+1}, \tag{6}$$

and substituting u and v in the equation (1) into governing equations after manipulations gives

$$f^{'''}(\eta) + \frac{1}{2}f(\eta)f^{''}(\eta) + \beta[1 - f^{'2}(\eta)] = 0, \quad \eta\epsilon(0,\infty), \tag{7}$$

the boundary conditions are as follows:

$$f(0) = f^{'}(0) = 0, \quad f^{'}(+\infty) = 1. \tag{8}$$

2.2. Hermite functions

In this section, we describe the properties of Hermite functions. The normalized Hermite function of degree n, denoted by $\widetilde{H}_n(x)$, is defined as [20, 32].

$$\widetilde{H}_n(x) = \frac{1}{\sqrt{2^n n!}}e^{\frac{-x^2}{2}}H_n(x), \quad n \geq 0, \ x \in \mathbb{R}. \tag{9}$$

Orthogonal relation for Hermite functions is as follows:

$$\int_{-\infty}^{+\infty} \widetilde{H}_n(x)\widetilde{H}_m(x) = \sqrt{\pi}\delta_{mn}, \tag{10}$$

here, δ_{mn} is the Kronecker delta function. Hermite functions are defined in the $(-\infty, +\infty)$ domain and have the following recurrence relation

$$\widetilde{H}_{n+1}(x) = x\sqrt{\frac{2}{n+1}}\widetilde{H}_n(x) - \sqrt{\frac{n}{n+1}}\widetilde{H}_{n-1}(x), \quad n \geq 1,$$
$$\widetilde{H}_0(x) = e^{\frac{-x^2}{2}}, \ \widetilde{H}_1(x) = \sqrt{2}xe^{\frac{-x^2}{2}}. \tag{11}$$

Using recurrent relation of Hermite functions and the above formula leads to

$$\widetilde{H'}_n(x) = \sqrt{2n}\widetilde{H}_{n-1}(x) - x\widetilde{H}_n(x) = \sqrt{\frac{n}{2}}\widetilde{H}_{n-1}(x) - \sqrt{\frac{n+1}{2}}\widetilde{H}_{n+1}(x), \tag{12}$$

and it becomes

$$\int_{-\infty}^{+\infty} \widetilde{H'}_n(x)\widetilde{H'}_m(x)dx = \begin{cases} -\frac{\sqrt{n\pi(n-1)}}{2}, & m = n-2, \\ (n+\frac{1}{2})\sqrt{\pi}, & m = n, \\ -\frac{\sqrt{\pi(n+1)(n+2)}}{2}, & m = n+2, \\ 0, & Otherwise. \end{cases} \tag{13}$$

$$\tilde{P} : \{u : u = e^{\frac{-x^2}{2}}\nu, \ \forall_\nu \epsilon P_N\}, \tag{14}$$

where P_N is Hermite polynomials of Ndegrees.

2.3. Hermite Deep Neural Network

Research has shown that artificial neural networks can effectively solve this type of differential equation. Deep neural networks are composed of multiple layers with densely connected perceptrons, making them ideal for solving differential equations due to their simplicity and computational efficiency [8, 12, 16, 19]. The structure of this model is illustrated in Figure 1, which includes an input layer with nodes $x_1, x_2, ..., x_n$, a flexible hidden layer, and an output layer with a single node.

In this model, the activation function of the hidden layers is based on the Hermite function. However, the values of the weights can sometimes become very large, which can lead to network instability and cause the weights to overflow, resulting in NaN values. To solve this problem, orthogonal Hermite functions can be used as activation functions, as their roots prevent this issue.

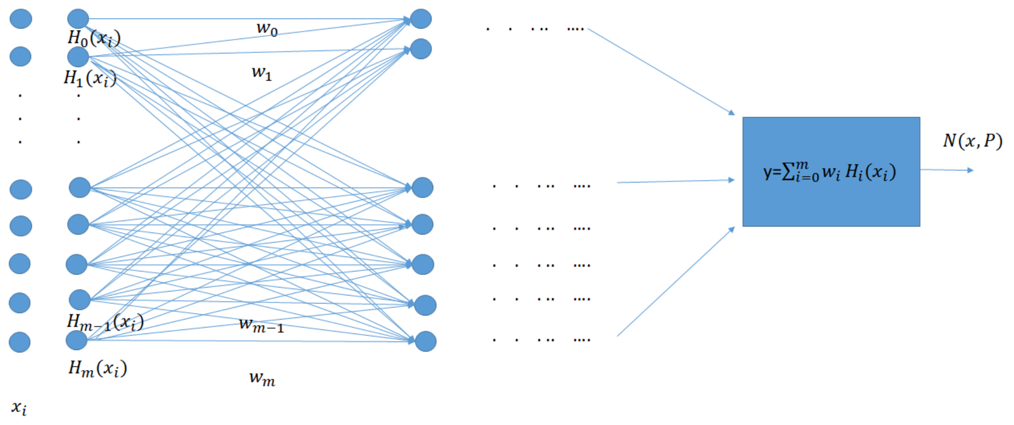

Figure 1. This is an illustrative example of using Hermite functions as activation functions in a neural network. The nodes in the input layer are denoted as x_i, and the Hermite functions used as activation functions are denoted as $H_i(x)$. The weight vectors from the hidden layers are denoted as $w_0, ..., w_m$, and the output layer is denoted as y. The network, denoted as $N(x, P)$, utilizes a backpropagation neural network with multiple layers.

2.4. Solving Blasius equation

This paper proposes a Hermite deep neural network to solve equation (7) with the boundary conditions stated in equation (8). The Hermite neural network can be used as a mixed method for semi-infinite domains, and is suitable for both infinite and semi-infinite intervals.

The first step in implementing the Hermite neural network model is to expand its structure, and then the proposed algorithm can be explained. Hermite neural networks consist of input, output, and hidden layers.

The trial solution of the Hermite neural network model, denoted as (HNN), is written as follows:

$$y_{HNN}(x, W) = A(x) + F(x, N(x, W)), \tag{15}$$

the function $A(x)$ specifies only the initial/boundary conditions, while $N(x, W)$ is a neural network with inputs x, weights, and biases, which are represented by W. The output of the Hermite neural network, denoted as $y_{HNN}(x)$, satisfies the condition when N is zero.

The proposed method can be divided into three steps:

- Step 1: Creating a feed-forward neural network and selecting the appropriate activation functions.
- Step 2: Defining the loss function.
- Step 3: Implementing a backpropagation neural network and using an optimized method.

Figure 2 illustrates the steps involved in creating a feed-forward network. The bias input is not emphasized here, as it is a constant input (usually one) connected to each neuron in the hidden layer by a weight. The only difference between the bias input and other inputs is that it remains the same as the other inputs change. Similarly, the weights connecting it to the next layer are updated as well.

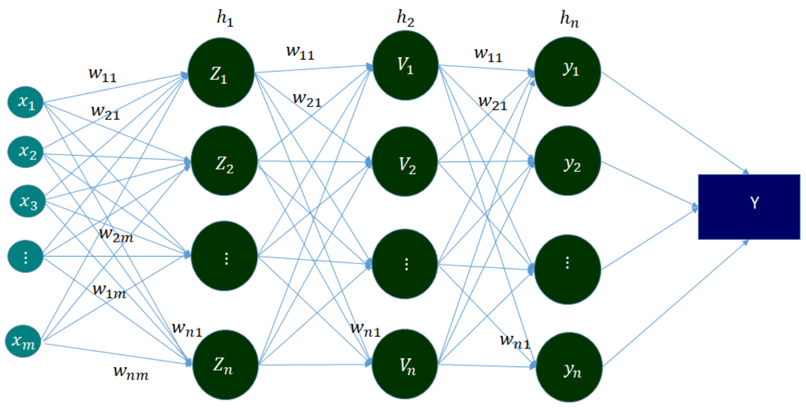

Figure 2. The structure of feed-forward neural network

The feed-forward neural network can be defined as follows:

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} & \cdots & w_{n1}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & \cdots & w_{n2}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} & \cdots & w_{n3}^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ w_{1m}^{(1)} & w_{2m}^{(1)} & \cdots & w_{nm}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \tag{16}$$

$Z = [z_1, z_2, \ldots z_n]^T \subset \mathbb{R}^n$ output vector are the inner product of $X = [x_1, x_2, \ldots x_m] \subset \mathbb{R}^m$ inputs and h_1 hidden layers.

$$Z_j = W_{ij}.X_i, \tag{17}$$

W_{ij} is the weight matrix.This relationship can also be written for other layers. In this stage, the active function is applied

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_k \end{bmatrix} = \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} & \cdots & w_{n1}^{(n)} \\ w_{12}^{(2)} & w_{22}^{(2)} & \cdots & w_{n2}^{(2)} \\ w_{13}^{(2)} & w_{23}^{(2)} & \cdots & w_{n3}^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ w_{1k}^{(2)} & w_{2k}^{(2)} & \cdots & w_{nk}^{(2)} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}, \tag{18}$$

The output vector of the desired hidden layer is obtained using the inner product of the combined matrix and input vector, and this process continues until the output layer is obtained.

$$V_k = W_{jk}.Z_j = W_{ij}.W_{jk}.X_i = W_{ijk}.X_i, \tag{19}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_l \end{bmatrix} = \begin{bmatrix} w_{11}^{(n)} & w_{21}^{(n)} & \cdots & w_{n1}^{(n)} \\ w_{12}^{(n)} & w_{22}^{(n)} & \cdots & w_{n2}^{(n)} \\ w_{13}^{(n)} & w_{23}^{(n)} & \cdots & w_{n3}^{(n)} \\ \vdots & \vdots & \vdots & \vdots \\ w_{1l}^{(n)} & w_{2l}^{(n)} & \cdots & w_{nl}^{(n)} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}, \tag{20}$$

Finally, the output vector is obtained

$$Y_l = W_{kl}.V_k = W_{ijk}.W_{kl}.X_i = W_{ijkl}.X_i, \tag{21}$$

Hermite functions are suitable as activation functions for nonlinear problems in semi-infinite intervals because of their proper convergence. The network output $N(x, W)$, where $N : \mathbb{R}^m \to \mathbb{R}$, is defined as follows:

$$N(x, W) = \sum_{j=0}^{m} w_j H(z_j), \tag{22}$$

where $w_j = [w_0, ..., w_m]$ denotes the weight from the jth hidden layer, $z_j, \quad j = 0, .., m$ is the Hermite collocation point, and $H_j = [H_0, ..., H_m]$ are Hermite functions whose properties are defined in section 2.2. Hermite functions are defined on $(-\infty, +\infty)$, so domain truncation is used to solve this problem [32].

At this stage, since the output is different from the actual output, we move on to the next stage. The network returns and recomputes the weights and biases. This change is made in the direction that reduces the amount of error and minimizes the difference between the predicted output and the actual value. One optimization method used is gradient descent. This algorithm is used to train a neural network through a method called the chain rule.

This paper considers the supervised version of the back-propagation neural network to minimize the loss function. In contrast to other neural networks, this method cannot get stuck in a local minimum. Hermite neural networks can be divided into the following setups: Hermite functions are selected as activation functions using the roots of Hermite functions as training data."

First, let us consider the boundary value problem

$$f^{'''}(x) + \frac{1}{2} f(x) f^{''}(x) = 0, \quad x\epsilon(0, \infty). \tag{23}$$

The loss function is used to evaluate the performance of our Hermite neural networks (HNN) model. The loss function for m inputs with $x_i, \ i = i, ..., m$ nodes is minimized as follows:

$$HermiteCost(x, W) = \frac{1}{m} \sum_{i=1}^{m} (y^{'''}_{HNN}(x) + \frac{1}{2} y_{HNN}(x) y^{''}_{HNN}(x) - Y)^2, \tag{24}$$

where Y is the exact solution. The kth-derived Hermite functions mappings can be approximated by

$$\frac{\partial}{\partial x} N(x, W) = \sum_{j=0}^{m} w_j \frac{\partial}{\partial x} H(z_j), \tag{25}$$

$$\frac{\partial^2}{\partial x^2} N(x, W) = \sum_{j=0}^{m} w_j \frac{\partial^2}{\partial x^2} H(z_j), \tag{26}$$

To minimize error, the loss function was optimized by gradient descent. The hidden layers of the model include the weights and biases of each neuron, but the input layer does not.

As a result, the approximating derivatives are defined

$$\frac{\partial HermiteCost(x, W)}{\partial x_i} = (\frac{\partial HermiteCost(x, W)}{\partial x_1}, ..., \frac{\partial HermiteCost(x, W)}{\partial x_m}). \tag{27}$$

We conclude the kth derivatives of m that

$$\frac{\partial^k}{\partial x_i^k} HermiteCost = \frac{\partial}{\partial x_k} (\sum_{i=1}^{m} HermiteCost). \tag{28}$$

The output of the HNN model is as follows:

$$\nabla^{(k)} HermitCost = \sum_{i=1}^{m} w_i D H^{(k)}(z_i), \tag{29}$$

where ∇ is a differential operator, D is denoted as the sum of the weights where $\{\omega_1, ...\omega_m\}$ is the weights. We discussed one neuron as the general model in the description but for the other neurons, we followed the description.

3. Numerical results

This section presents numerical results for solving the nonlinear Blasius equation with boundary conditions using Python software version 3.9.0 and the autograd package of Python. The trial solution for the differential equation (7) with the boundary conditions (8) has the following form:

$$y_{HNN}(x, W) = (\frac{x}{x+1} - x).N(x, W). \tag{30}$$

This paper reports numerical results using the HNN model and obtains values of $f^{''}(\eta)$, $f^{'}(\eta)$, and $f(\eta)$ for various η, which are then compared with other methods. In order to reduce the loss function, we used the gradient descent method. This method consists of three hidden layers with a total of 21, 23, and 37 neurons.
Table 1 shows a comparison of $f^{''}(0)$ between various methods for solving the Blasius equation

Table 1. The comparison of $f^{''}(0)$ with this present method and the various methods

	Number of layers	This persent method	Parand [34]	Howarth [14]	Rafael [35]
$f^{''}(0)$	[1,21,23,37,1]	0.332057102	0.332057335	0.33206	0.33206

Shown below are the approximations of $y_{HNN}(x, W)$ for the Blasius equation obtained by the Hermite functions neural network and a comparison with other methods. Tables 2, 3, and 4 present the resulting values of $f(\eta)$, $f^{'}(\eta)$, and $f^{''}(\eta)$ using this method and other methods.

Table 2. The Comparison of $f(\eta)$ for various η between the Hermite neural network and some other methods

η	The present method	Parand [34]	Howarth [14]	Rafael[35]	Relative Error
1.0	0.16541105	0.1655731	0.16557	0.16557	9.60E-04
2.0	0.64967773	0.6500351	0.65002	0.65003	5.42E-04
3.0	1.39647618	1.3968254	1.39681	1.39682	2.46E-04
4.0	2.30574561	2.3057619	2.30575	2.30576	6.24E-06
5.0	3.28384346	3.2832913	3.28327	3.28330	1.66E-04
6.0	4.28098773	4.2796473	4.27962	4.27965	3.13E-04

Table 3. Resulting values $f^{'}(\eta)$ for the present method with the other methods

η	The present method	Parand [34]	Howarth [14]	Rafael[35]	Relative Error
1.0	0.329566712	0.3297956	0.32979	0.32978	6.47E-04
2.0	0.629571785	0.6297737	0.62977	0.62977	3.15E-04
3.0	0.846109525	0.8460586	0.84605	0.84605	7.04E-05
4.0	0.956103098	0.9555253	0.95552	0.95552	6.10E-04
5.0	0.992407319	0.9915583	0.99155	0.99155	8.65E-04
6.0	0.999894820	0.9989882	0.99898	0.99898	9.16E-04

Table 4. Comparison of $f^{''}(\eta)$ between the present method and the other numerical methods

η	The present method	Parand [34]	Howarth [14]	Rafael[35]	Relative Error
0.0	0.332057102	0.332057335	0.33206	0.33206	8.73E-06
1.0	0.322884293	0.3230174	0.32301	0.32301	3.89E-04
2.0	0.266958108	0.2667514	0.26675	0.26675	7.80E-04
3.0	0.161256399	0.1613615	0.16136	0.16136	6.42E-04
4.0	0.063990964	0.0642426	0.06424	0.06423	3.72E-03
5.0	0.016547122	0.0159142	0.01591	0.01591	4.00E-02
6.0	0.003129914	0.0024067	0.00240	0.00240	3.04E-01

The Blasius graph obtained by approximating $f(\eta)$ and $f'(\eta)$ using the present method is shown in Fig. 3. It can be seen that the graph tends to zero as η increases.
We plot the logarithmic graph of the relative error $f^{''}(\eta)$ for various η in the solution of the Blasius equation in Fig 4.

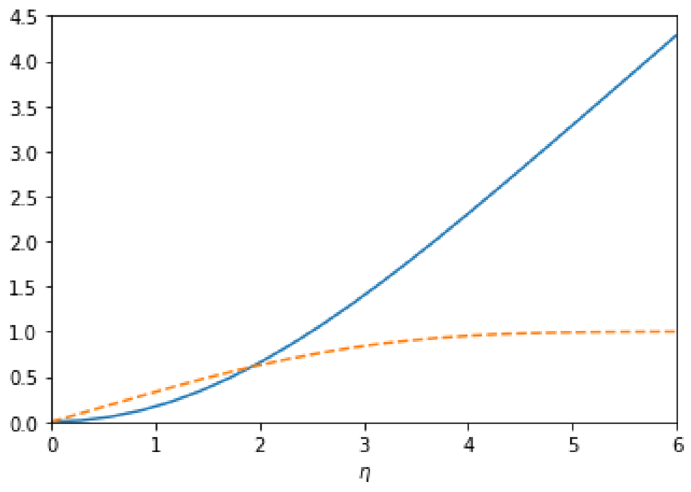

Figure 3. Plot of numerical approximation for $f(\eta)$ (solid) and $f'(\eta)$ (dashed) by Hermite neural network

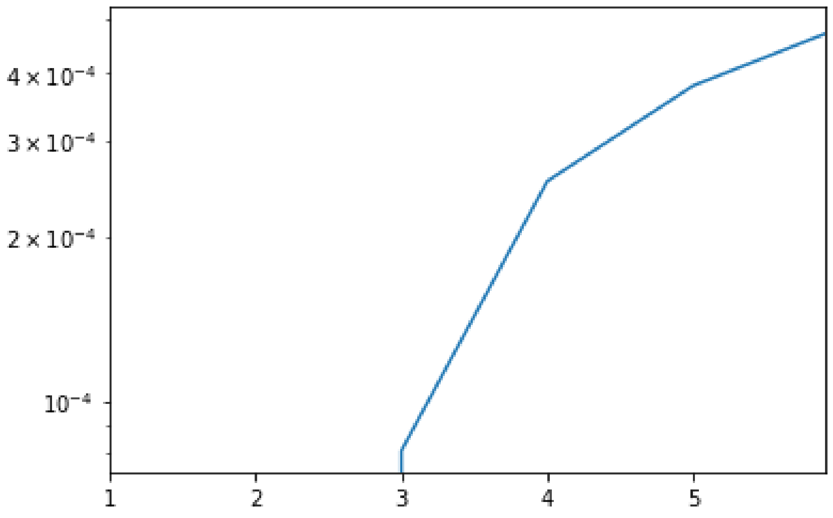

Figure 4. Plot of the logarithmic of the relative error $f''(\eta)$ for various η by Hermite neural network

Table 5 shows the numerical results for $f''(\eta)$ when $\eta = 0$ for the present method and the Physics-informed neural networks (PINN) [37] with varying numbers of layers. Figure 5 illustrates the accuracy of this method for solving $f''(\eta)$ when $\eta = 0$. The absolute error for the various layers is shown on the left side, while the right graph shows the absolute error versus solution time. The present method is compared with the PINN model for solving this problem, and the results are shown in Figure 5. The PINN model uses Hyperbolic Tangent as an activation function, which can lead to the problem of Vanishing Gradient. This occurs when the derivative becomes very small for very large or very small values of x, making it difficult for the network to be trained.

Table 5. The numerical results for $f''(\eta)$ when $\eta = 0$ by Hermite neural network

$f''(0)$			
The number of layers	The present method	PINN	Parand [34]
[1,19,1]	0.331062275	0.320283547	0.332057335
[1,21,23]	0.332203264	0.330709275	0.332057335
[1,21,23,27]	0.332057102	0.332389979	0.332057335

According to Figure 5, as the number of layers increases, the absolute error decreases, and the solution becomes closer to the ideal state. The plots show that the time order of the absolute error is exponential. The implementation algorithm has a time order that is polynomial in the degree, determined by the number of hidden layers.

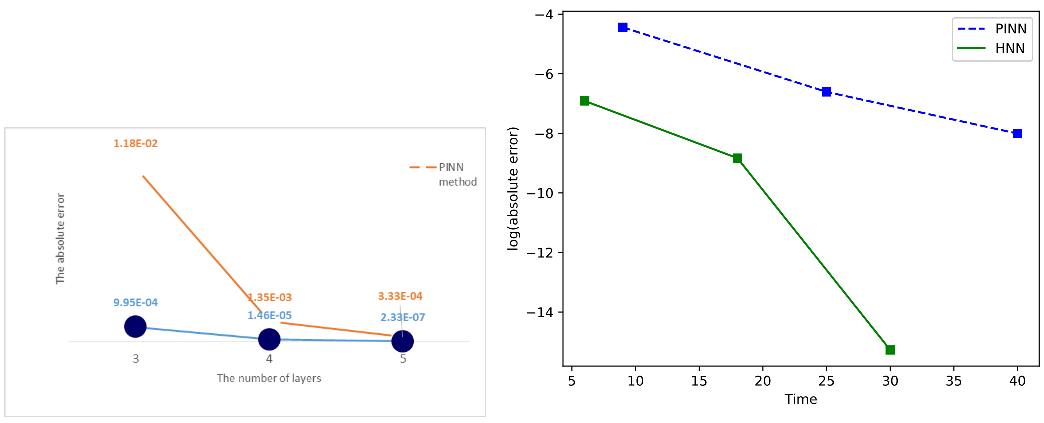

Figure 5. The plot on the left shows the absolute error of the present method and the PINN model versus the number of layers for $f''(\eta)$ when $\eta = 0$. The plot on the right shows the absolute error of this method versus solution time (in minutes) for $f''(\eta)$ when $\eta = 0$.

4. Conclusion

Spectral methods make it possible to increase the convergence and accuracy of a problem, and by using machine learning methods, it is possible to predict the state of the system in the future. By combining these two methods, not only can we obtain reasonably convergent results, but we can also describe the future state of the output system.

In this paper, we applied the orthogonal neural network approach to solve the Blasius equation. The accuracy of the present method is desirable. Our scheme is based on a Hermite neural network with domain truncation to semi-infinite intervals. The Adam algorithm was used for training the neural network, with the hidden layer consisting of the weights and Hermite functions as activation functions. Finally, we proposed numerical results and compared them with other methods for this problem. Table 1 shows that our convergence rate is better than other methods. The resulting values of $f(\eta)$, $f^{'}(\eta)$, and $f^{''}(\eta)$ are presented in Tables 2, 3, and 4. Additionally, the values of $f(\eta)$ and $f^{'}(\eta)$ when $\eta = 0$ are shown in Figure 3. The logarithmic relative error of the proposed method is plotted in Figure 4. The complexity of this model is evaluated to be $O(n^3)$.

The advantage of this method is that it avoids overflow of training and testing data by using roots of Hermite functions, and it also avoids getting stuck in local minimum.

References

1. M. Abbazadeh, M. Bayat, M. Dehghan, and M. I. Azis. Investigation of generalized Couette hydromagnetic flow of two-step exothermic chemical reaction in a channel via the direct meshless local Petrov–Galerkin method. *Engineering Analysis with Boundary Elements*, 125:178–189, 2021.
2. A. A. Aghaei and K. Parand. Hyperparameter optimization of orthogonal functions in the numerical solution of differential equations. *arXiv e-prints*, page arXiv:2304.14088, 2023.
3. M. A. Arefin, M. A. Nishu, M. N. Dhali, and M. H. Uddin. Analysis of reliable solutions to the boundary value problems by using shooting method. *Mathematical Problems in Engineering*, 2022:1–9, 2022.
4. J. P. Boyd. *Chebyshev and Fourier Spectral Methods*. Dover Publications, Mineola, New York, 2000.
5. F. Calabr'o, G. Fabiani, and C. Siettos. Extreme learning machine collocation for the numerical solution of elliptic PDEs with sharp gradients. *Computer Methods in Applied Mechanics and Engineering*, 387:114188, 2021.
6. M. Delkhosh and K. Parand. A new computational method based on fractional lagrange functions to solve multi-term fractional differential equations. *Numerical Algorithms*, 88(2):729–766, 2021.
7. C. Duque-Daza, D. Lockerby, and C. Galeano. Numerical solution of the Falkner-Skan equation using third-order and high-order-compact finite difference schemes. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 33(4):381–392, 2011.
8. M. Frank, D. Drikakis, and V. Charissis. Machine-learning methods for computational science and engineering. *Computation*, 8(1):1–35, 2020.
9. H. Gao, L. Sun, and J. X. Wang. Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. *Journal of Computational Physics*, 428:110079, 2021.
10. B. Guo. Gegenbauer approximation and its applications to differential equations with rough asymptotic behaviors at infinity. *Applied Numerical Mathematics*, 38:403–425, 2001.
11. B. Y. Guo. Jacobi approximations in certain Hilbert spaces and their applications to singular differential equations. *Journal of Mathematical Analysis and Applications*, 243:373–408, 2000.
12. Z. Hajimohammadi, F. Baharifard, A. Ghodsi, and K. Parand. Fractional Chebyshev deep neural network (FCDNN) for solving differential models. *Chaos, Solitons & Fractals*, 153:111530, 2021.

13. Z. Hajimohammadi, F. Baharifard, and K. Parand. A new numerical learning approach to solve general Falkner-Skan model. *Engineering with Computers*, 38:121–137, 2022.
14. L. Howarth. On the calculation of steady flow in the boundary layer near the surface of a cylinder in a stream. *Aero Res. Counc. Lond. R & M*, 164:16–32, 1935.
15. A. Jalilian, A. A. bdi, and G. Hojjati. Variable stepsize SDIMSIMs for ordinary differential equations. *Applied Numerical Mathematics*, 168:115–126, 2021.
16. X. Jin, S. Cai, H. Li, and G. E. Karniadakis. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 426:1–42, 2021.
17. A. Kratsios. The universal approximation property: characterization, construction, representation, and existence. *Annals of Mathematics and Artificial Intelligence*, 89:435–469, 2021.
18. S. J. Liao. An explicit, totally analytic approximate solution for Blasius' viscous flow problems. *International Journal of Non-Linear Mechanics*, 34(4):759–778, 1999.
19. S. Mall and S. Chakraverty. Application of Legendre neural network for solving ordinary differential equations. *Applied Soft Computing*, 43:347–356, 2016.
20. Z. Mao and J. Shen. Hermite spectral methods for fractional PDEs in unbounded domains. *SIAM Journal on Scientific Computing*, 39(5):A1928–A1950, 2017.
21. H. D. Mazraeh, M. Kalantari, S. H. Tabasi, A. A. Aghaei, Z. Kalantari, and F. Fahim. Solving Fredholm integral equations of the second kind using an improved cuckoo optimization algorithm. *Global Analysis and Discrete Mathematics*, 7(1):33–52, 2023.
22. S. Mehrkanoon, T. Falck, and J. A. K. Suykens. Approximate solutions to ordinary differential equations using least squares support vector machines. *IEEE Transactions on Neural Networks and Learning Systems*, 23(9):1356–1367, 2012.
23. C. Michoski, M. Milosavljevi'c, T. Oliver, and D. R. Hatch. Solving differential equations using deep neural networks. *Neurocomputing*, 399:193–212, 2020.
24. M. B. Milovanovic, D. S. Antic, S. S. Nikolic, S. L. Peric, M. T. Milojkovic, and M. D. Spasic. Neural network based on orthogonal polynomials applied in magnetic levitation system control. *Elektronika ir elektrotechnika*, 23(3):24–29, 2017.
25. M. M. Moayeri, J. A. Rad, and K. Parand. Dynamical behavior of reaction-diffusion neural networks and their synchronization arising in modeling epileptic seizure: A numerical simulation study. *Computers & Mathematics with Applications*, 80(8):1887–1927, 2020.
26. M. Omidi, B. Arab, A. H. Rasanan, J. A. Rad, and K. Parand. Learning nonlinear dynamics with behavior ordinary/partial/system of the differential equations: looking through the lens of orthogonal neural networks. *Engineering with Computers*, pages 1–20, 2021.
27. N. Panda, M. G. Fernndez-Godino, H. C. Godinez, and C. Dawson. A data-driven non-linear assimilation framework with neural networks. *Computational Geosciences*, 25(1):233–242, 2021.
28. K. Parand, A. A. Aghaei, M. Jani, and A. Ghodsi. A new approach to the numerical solution of Fredholm integral equations using least squares-support vector regression. *Mathematics and Computers in Simulation*, 180:114–128, 2021.
29. K. Parand, A. A. Aghaei, M. Jani, and A. Ghodsi. Parallel LS-SVM for the numerical simulation of fractional Volterra's population model. *Alexandria Engineering Journal*, 60(6):5637–5647, 2021.
30. K. Parand, A. A. Aghaei, M. Jani, and R. Sahleh. Learning with fractional orthogonal kernel classifiers in support vector machines: Theory, algorithms and applications. In *Springer Nature Singapore*, pages 199–224. Springer, Singapore, 2023.
31. K. Parand, F. Baharifard, A. A. Aghaei, and M. Jani. Learning with fractional orthogonal kernel classifiers in support vector machines: Theory, algorithms and applications. In *Springer*, pages 19–36. Singapore, 2023.
32. K. Parand, Z. Kalantari, and M. Delkhosh. Solving the boundary layer flow of Eyring–Powell fluid problem via quasilinearization–collocation method based on Hermite functions. *INAE Letters*, 3(1):11–19, 2018.
33. K. Parand, M. Razzaghi, R. Sahleh, and M. Jani. Least squares support vector regression for solving Volterra integral equations. *Engineering with Computers*, 38:789–796, 2022.
34. K. Parand and A. Taghavi. Rational scaled generalized laguerre function collocation method for solving the blasius equation. *Journal of Computational and Applied Mathematics*, 233(4):980–989, 2009.
35. C. Rafael. Numerical solutions of the classical Blasius flat-plate problem. *Applied Mathematics and Computation*, 170:706–710, 2005.
36. M. Raissi and G. E. Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.
37. 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.
38. A. H. Rasanan, D. Rahmati, S. Gorgin, and K. Parand. A single layer fractional orthogonal neural network for solving various types of Lane–Emden equation. *New Astronomy*, 75:101307, 2020.
39. F. Salehi, S. H. Shahraki, M. K. Fallah, and M. Hemamia. Numerical investigation of differential biological models via Gaussian RBF collocation method with genetic strategy. *Computational Mathematics and Computer Modeling with Applications*, 1(2):46–64, 2022.
40. J. Shen, T. Tang, and L.-L. Wang. *Spectral Methods: Algorithms, Analysis and Applications*. Springer, New York, 2011.
41. A. P. Solodov and V. F. Ochkov. *Differential Models*. Springer, Berlin, Heidelberg, 2005.
42. J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.
43. M. H. Uddin, M. A. Arefin, M. A. Akbar, and M. Inc. New explicit solutions to the fractional-order Burgers equation. *Mathematical Problems in Engineering*, 2021:1–11, 2021.
44. N. Vukovi'c, M. Petrovi'c, and Z. Miljkovi'c. A comprehensive experimental evaluation of orthogonal polynomial expanded random vector functional link neural networks for regression. *Applied Soft Computing*, 70:1083–1096, 2018.