Transactions of the Nigerian Association of Mathematical Physics
Volume 11, (January - June, 2020 Issue), pp123 – 130

A NEW PERSPECTIVE FOR SOLVING THIRD ORDER INITIAL VALUE PROBLEMS USING ARTIFICIAL NEURAL NETWORKS

Okereke Roseline Ngozi and Maliki Olaniyi Sadik

Department of Mathematics Michael Okpara University of Agriculture Umudike, Nigeria

Abstract

A new perspective for obtaining solutions of initial value problems using Artificial Neural Networks (ANN) stems from the fact that most solutions in previous research works rely on weight updating to finding approximate solutions to initial value problems. In this paper we develop a Neural Network algorithm using MathCAD 14 software, which enables us to slightly adjust the intrinsic biases involved in solving third order homogeneous and non-homogeneous ODE. In the homogeneous and non-homogeneous cases, we employ a Statistical Package for Social Sciences (SPSS 23) and Gaussian Radial Basis Function (GRBF) to obtain the weights, which need not be adjusted, from input layer to the hidden layer, and from the hidden layer to the output layer, respectively. We compare exact results with the neural network results for our example ODE problems and find the results to be in good agreement.

Keywords: Artificial Neural Network, Weights, Biases, IVP, MathCAD 14, SPSS 23, GRBF.

1.0 Introduction

An Artificial neural network (ANN) is fundamentally a mathematical model, and its structure consists of a series of processing elements which are inter-connected and their operation resemble that of the human neurons. These processing elements are also known as units or nodes. The ability of the network to process information is embedded in the connection strengths, simply called weights, which, when exposed to a set of training patterns, adapts to it. [1]. The conventional way of solving differential equations using artificial neural network involves updating of all the parameters, weights and biases, during the neural network training [2 - 7]. This is caused by the inability of the neural network to predict a solution with an acceptable minimum error. In order to reduce the error, the error function is minimized. Minimizing the error function demands finding its gradient. This gradient involves the computation of multivariate partial derivatives of the error function with respect to all the parameters, weights and biases, and the independent variable. This computational complexity is quite evident when handling first order ordinary differential equations, as shown in [8,9]. It is even more difficult when solving higher order ODE where you need to compute higher order derivatives of the error function.

2.0 Mathematical Model of ANN

ANNs are constructed with layers of units, which are termed *multilayer* ANNs. A layer of units in an ANN is made up of units that perform similar tasks. There are two functions that govern the behaviour of a unit in a particular layer, which normally are the same for all units within the whole ANN. These are;

1) The input function and
2) The output/activation function

Input into a node is a weighted sum of output from nodes connected to it.

A neuron receives a set of n inputs, $S = \{x_j \mid j = 1,2,...,n\}$. In Fig. 1, each input is weighted before reaching the main body of a neuron N_i by connection strength or weight factor w_{ij} for $j = 1,2,...,n$. In addition, it has a bias b or w_0, a threshold value θ_k, which has to be reached or exceeded for the neuron to produce an output signal [10].

2.1 Activation Function

The activation of a neuron is computed by applying a threshold function (popularly known as activation function) to the weighted sum of the inputs plus a bias [10]. A function $f(s)$ acts on the produced weighted signal. This function is called an *activation*

Correspondence Author: Okereke R.N., Email: okerekeroseline@yahoo.com, Tel: +2347036838240, +2348035401200 (MOS)

*function***.** Mathematically, the output of the i^{th} neuron N_i is given by;

$$O_i = f\left[w_0 + \sum_{j=1}^{n} w_{ij} x_j \right] \quad (1)$$

and figure 1.1 shows detailed computational steps of the working principle of an artificial neuron (AN) in a neural network. Now the input signal for the i^{th} neuron N_i is,

$$s_i = w_0 + \sum_{j=1}^{n} w_{ij} x_j \quad (2)$$

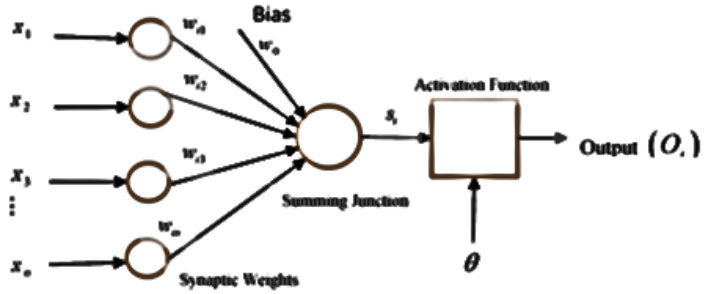

Fig.1 Mathematical Model of Artificial Neural Network. [8]

All inputs are multiplied by their weights and added together to form the net input to the neural called net. Mathematically, we can write

$$net = w_{i1}x_1 + w_{i2}x_2 + ... + w_{in}x_n + \theta \quad (3)$$

where θ is a threshold value that is added to the neurons. The neuron behaves as activation or mapping function $f(net)$ to produce an output y which can be expressed as:

$$y = f(net) = f\left(\sum_{j=1}^{n} w_{ij} x_j + \theta \right) \quad (4)$$

where $f(net)$ is called the neuron activation function or the neuron transfer function [10].

In this paper we employ the sigmoid activation function given by

$$\sigma(x) = \left(1 + e^{-Tx}\right)^{-1} \quad (5)$$

It is an *S* shaped smooth function, where input is mapped into values between +1 and 0.

2.2 Function Approximation

Function approximation attempts to describe the behaviour of very complicated functions by sets of simpler functions, therefore we shall exploit its potentials in this paper.

2.3Theorem (Universal approximation theorem for MLP)

Let I_n represent an *n*-dimensional unit cube containing all possible input samples *x*, that is, $x_i \in [0,1]$, $i = 1,2,...,n$, and $C(I_n)$ be the space of continuous functions on I_n . If $\sigma(.)$ is a continuous sigmoid function, then the finite sums of the form

$$y_k = y_k(x,w) = \sum_{i=1}^{N_2} w_{ki}^3 \sigma\left(\sum_{j=0}^{n} w_{ij}^2 x_j \right) \quad k = 1,2,...,m \quad (6)$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum $y(x,w)$ of the form (2.6) that satisfies $|y(x,w) - f(x)| < \varepsilon$ for all $x \in I_n$. As such, there always exists a 3-layer perceptron that can approximate an arbitrary nonlinear, continuous, multi-dimensional function f with any desired accuracy[11].Thus, any continuous function can be approximated to a given precision using artificial neural networks with just one hidden layer.

3.0 Neural Network method for solving third order ODE

Lagaris *et al.* [4] gave the following as the general formulation for the solution of first order ODE

$$\frac{d\psi}{dx} = f(x,\psi), \quad x \in [a,b] \quad (7)$$

with initial condition $\psi(a) = A$.To employ ANN for the solution, we define a trial solution, $\psi_t(x,p)$ written as the sum of two terms, i.e.

$$\psi_t(x,p) = A(x) + F(x, N(x,p)), \tag{8}$$

where $A(x)$ satisfies the initial conditions and contains no adjustable parameters, where $N(x,p)$ is the output of feed forward neural network with the parameters p and input data x. The function $F(x, N(x,p))$ is actually the operational model of the neural network. Feed forward neural network (FFNN) converts differential equation problem to function approximation problem. The neural network $N(x,p)$ is given by

$$N(x,p) = \sum_{j=1}^{m} v_j \sigma(z_j) \tag{9}$$

where

$$z_j = \sum_{i=1}^{n} w_{ji} x_i + u_j, \tag{10}$$

w_{ji} denotes the weight from input unit i to the hidden unit j, v_j denotes weight from the hidden unit j to the output unit, u_j denotes the biases, and $\sigma(z_j)$ is the sigmoid activation function.

To solve this problem using neural network (NN), a NN architecture with three layers - one input layer with one neuron; one hidden layer with three neurons and one output layer with one output unit, as shown in figure 2.2, is employed. Each neuron is connected to other neurons of the previous layer through adaptable synaptic weights w_j and biases u_j. Lagaris *et al.* [4] applied the same procedure to second order IVP.

$$\frac{d^2\psi}{dx^2} = f\left(x, \psi, \frac{d\psi}{dx}\right), \ \psi(0) = A, \psi'(0) = B \tag{11}$$

with the trial solution cast as

$$\psi_t(x) = A + Bx + x^2 N(x_i, p) \tag{12}$$

where $A, B \in \mathbb{R}$ and $N(x_i, p) = \sum_{j=1}^{3} v_j \sigma(z_j)$.

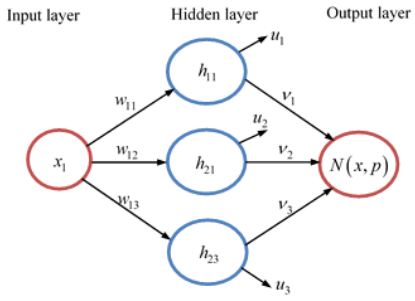

Fig.2 Schematic for $N(x,p)$

For third order initial value problem:

$$\frac{d^3\psi}{dx^3} = f\left(x, \psi, \frac{d\psi}{dx}, \frac{d^2\psi}{dx^2}\right), \psi(0) = A, \psi'(0) = B, \psi''(0) = C, \tag{13}$$

We extended the trial solution for first and second order ODE to get the following for third order, viz:

$$\psi_t(x) = A + Bx + Cx^2 + x^3 N(x_i, p), \ A, B, C \in \mathbb{R}$$

[8] proposed the following method in solving linear ordinary homogeneous differential equations with constant coefficients using NN novel approach. First, a solution is assumed to the homogeneous part and the assumed solution is approximated with a polynomial function using SPSS model. The choice of polynomial functions was motivated by the fact that every polynomial function $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, is continuous $\forall x \in \mathbb{R}$, and the graph of continuous function in a given interval is unbroken in that interval [12]. This enables us to estimate the regression coefficients in the nonlinear regression model. These coefficients are then employed as the weights from the input layer to the hidden layer. The task of obtaining the weights from hidden layer to the output layer is done by finding $f(x)$, a real function of a real valued vector $x = [x_1, x_2, ..., x_d]^T$ and a set of functions, $\{\varphi_i(x)\}$ called the elementary functions such that

$$\hat{f}(x,v) = \sum_{i=1}^{N} v_i \varphi_i(x) \tag{14}$$

is satisfied, where v_i are real valued constants such that

$$\left| f(x) - \hat{f}(x,v) \right| < \varepsilon \tag{15}$$

When one can find coefficients v_i that make ε arbitrarily small for any function $f(\cdot)$ over the domain of interest, we say that the elementary function set $\{\varphi_i(\cdot)\}$ has the property of universal approximation over the class of functions $f(\cdot)$ [8].We shall use the GRBF given by

$$\varphi_i(x) = \exp\left[-\frac{|x - x_i|^2}{2\sigma^2} \right], \quad \sigma^2 = \frac{1}{N}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{16}$$

as our elementary function and find $f(x)$ such that;

$$v = \phi^{-1} f \tag{17}$$

Here v becomes a vector with the coefficients, f is a vector composed of the values of the function at the *N* points, and ϕ the matrix with entries given by values of the elementary functions at each of the *N* points in the domain. The major task here is finding $f(x)$. We shall apply what [8,9] proposed for first and second order ode to this effect. Now to compute the weights from hidden layer to the output layer, we find a function $F(x)$ such that $v = \phi^{-1} f$ where $f(x) = \left[F(x_1), F(x_2), F(x_3) \right]^T$. We form a linear function based on the default sign of the differential equation, i.e. $F(x) = ax + b$, where a is the coefficient of the derivative of y and b is the coefficient of y. Extending this to the third order IVP, $F(x) = ax^3 + bx^2 + cx + d$, where $a, b, c, d \in \mathbb{R}$. Then we employ MathCAD 14 [13], a numerical algebra software, to slightly adjust the biases. These are the only parameters to be adjusted according to this new procedure. The Mean-Squared Error (MSE), given by:

$$E_n = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2 \tag{18}$$

will be employed to analyze the error. A demonstration of how the neural network solutions agree with exact solutions are presented in the subsequent tables and simulations.

4.0 Main results

In this section we present our main results. This concerns the solution of third order initial value problems using the novel artificial neural network methods developed in [8]. We shall consider both homogeneous and non homogeneous third order differential equations.

4.1 Third order homogeneous and non-homogeneous ODE

We observe here that in the literature, we did not see, to the best of our knowledge, any example on third order differential equations to compare our results with. Thus, we shall be contented to make a comparison of our results with exact solutions. It is also important to mention here that the methods we have developed thus far, can solve linear constant coefficient ordinary differential equations of any order. We begin with the initial value problem [14];

$$y''' - y'' + 100y' - 100y = 0;\ y(0) = 4, y'(0) = 11,\ y''(0) = -299,\ x \in [0,1] \tag{19}$$

Our trial solution is $y_t(x) = A + Bx + cx^2 + x^3 N(x,p)$. Applying the initial conditions gives

$$A = 4, B = 11, C = -229/2$$

Hence $y_t(x) = 4 + 11x - \frac{299}{2}x^2 + x^3 N(x,p)$, the exact solution is $y_a(x) = e^{-x} + \cos(10x) + \sin(10x)$.

We use excel spreadsheet to find values of $y_a(x)$ at all the specified x values, displayed in Table 4.1

Table 4.1 Values of $\left(x, y_a(x) = e^{-x} + \cos(10x) + \sin(10x)\right)$ for problem (19)

x	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	2	2.4869	1.7146	0.50099	0.0814	0.97346	2.5029	3.4246	3.0694	1.961	1.335

Using SPSS regression model[15], we compute the weights from input layer to hidden layer. From Table 4.2 and Figure 3, the cubic curve shows perfect fit. Using the cubic, we pick our weights from input layer to hidden layer as: $w_{11} = -17.581, w_{12} = 44.840, w_{13} = -28.698$.

Table 4.2 Model Summary and Parameter Estimates (SPSS Output) for problem (19)
Dependent Variable: Y

	Model Summary	Parameter Estimates			
Equation	R Square	Constant	b1	b2	b3
Linear	.040	1.509	.628		
Quadratic	.066	1.778	-1.165	1.793	
Cubic	.536	2.811	-17.581	44.840	-28.698

The independent variable is X

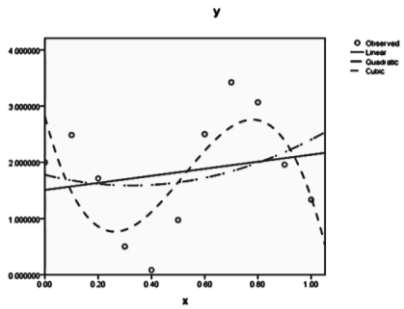

Fig. 3Polynomial Curve – Fit Regression of $y_a(x) = e^{-x} + \cos(10x) + \sin(10x)$

In order to compute the weights from hidden layer to the output layer, we form $F(x)$ by forming a cubic equation with the default sign of the ODE. That is, $F(x) = ax^3 - bx^2 + cx - d$, where $a = 1,\ b = -1,\ c = 100,\ d = -100$. Therefore,

$$F(x) = x^3 + x^2 + 100x + 100,\ f(x) = [110.011, 120.048, 130.117]^T \quad (20)$$

as obtained from Table 4.1, and $v = \phi^{-1} f$. Hence, the weights from the hidden layer to the output layer are given by;

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 1 & 0.94 & 0.78 \\ 0.94 & 1 & 0.94 \\ 0.78 & 0.94 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 110.011 \\ 120.048 \\ 130.117 \end{bmatrix} = \begin{bmatrix} 518.28 \\ -940.225 \\ 609.67 \end{bmatrix} \quad (21)$$

Hence the weights from the hidden layer to the output layer are;

$v_1 = 518.28,\ v_2 = -940.225,\ v_3 = 609.67$

The biases are fixed between -50 and 50. We now train the network with the available parameters using our MathCAD 14 algorithm [13] as follows:

$w_1 := -17.581 \quad w_2 := 44.840 \quad w_3 := -28.698 \quad x := 1$

$v_1 := 518.28 \quad v_2 := -940.225 \quad v_3 := 609.67 \quad u_1 := 16.004 \quad u_2 := -50 \quad u_3 := 26.3$

$z_1 := w_1 \cdot x + u_1 = -1.577 \quad z_2 := w_2 \cdot x + u_2 = -5.16 \quad z_3 := w_3 \cdot x + u_3 = -2.398$

$\sigma(z_1) := [1 + \exp(-z_1)]^{-1} = 0.171189$

$\sigma(z_2) := [1 + \exp(-z_2)]^{-1} = 5.709 \times 10^{-3}$

$\sigma(z_3) := [1 + \exp(-z_3)]^{-1} = 0.083325$

$N := v_1 \cdot \sigma(z_1) + v_2 \cdot \sigma(z_2) + v_3 \cdot \sigma(z_3) = 134.157043$

$y_p(x) := 4 + 11 \cdot x - \frac{299}{2} \cdot x^2 + x^3 \cdot N = -0.342957$

$y_d(x) := e^{-x} + 10 \cdot \cos(10 \cdot x) + \sin(10 \cdot x) = -0.342954$

$E := 0.5 \cdot (y_d(x) - y_p(x))^2 = 5.862 \times 10^{-12}$

Next we consider the initial value problem;

$$y''' + 6y'' + 11y' + 6y = 6x - 7; y(0) = 1,\ y'(0) = -1,\ y''(0) = 4,\ x \in [0,1]. \quad (22)$$

The trial solution is $y_t(x) = A + Bx + cx^2 + x^3 N(x, p)$. Applying the initial conditions gives $A = 1, B = -1, C = 2$. Hence,

$y_t(x) = 1 - x + 2x^2 + x^3 N(x, p),\ y_a(x) = e^{-x} + e^{-2x} + e^{-3x}$.

We use excel spreadsheet to find values of $y_a(x)$ at all the x points as displayed in Table 3.3.

Table 4.3 Values of $\left(x, y_a(x) = e^{-x} + e^{-2x} + e^{-3x}\right)$ for problem (22)

x	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	3	2.4644	2.0379	1.6962	1.4208	1.19754	1.0153	0.8656	0.7419	0.639	0.553

Using SPSS regression model [15],we compute the weights from input layer to hidden layer. From Table 4.4 and Figure 4, the cubic curve shows a perfect fit. Using the cubic, we pick our weights from input layer to hidden layer as: $w_{11} = -5.651,\ w_{12} = -4.986,\ w_{13} = -1.781$.

Table 4.4 Model Summary and Parameter Estimates (SPSS Output) for problem (22)
Dependent Variable: Y

	Model Summary	Parameter Estimates			
Equation	**R Square**	**Constant**	**b1**	**b2**	**b3**
Linear	.925	1.509	.628		
Quadratic	.066	1.778	-1.165	2.315	
Cubic	.536	2.811	-17.581	4.986	-1.781

The independent variable is X

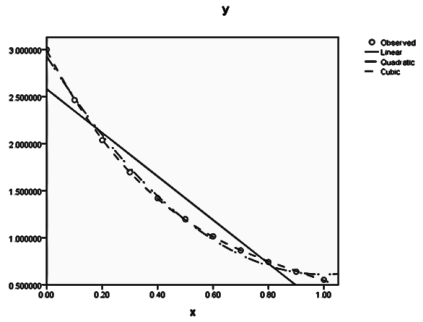

Fig. 4Polynomial curve – fit regression of $y_a(x) = e^{-x} + e^{-2x} + e^{-3x}$

Now to compute the weights from hidden layer to the output layer, we form

$$F(x) = 6x - 7, \quad f(x) = [-6.4, -5.8, -5.2]^T \tag{23}$$

as obtained from Table 4.3and $v = \phi^{-1} f$. Hence, the weights from the hidden layer to the output layer are

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 1 & 0.94 & 0.78 \\ 0.94 & 1 & 0.94 \\ 0.78 & 0.94 & 1 \end{bmatrix}^{-1} \begin{bmatrix} -6.4 \\ -5.8 \\ -5.2 \end{bmatrix} = \begin{bmatrix} -29.915 \\ 45.312 \\ -24.46 \end{bmatrix} \tag{24}$$

The weights from the hidden layer to the output layer are $v_1 = -29.915, v_2 = 45.312, v_3 = -24.46$.

The biases are fixed between -10 and 10. We now train the network with the available parameters using our MathCAD 14 algorithm as follows:

$w_1 := -5.651 \quad w_2 := 4.986 \quad w_3 := -1.781 \quad x := 1$

$v_1 := -29.915 \quad v_2 := 45.312 \quad v_3 := -24.46 \quad u_1 := 6 \quad u_2 := -5.464 \quad u_3 := -1$

$z_1 := w_1 \cdot x + u_1 = 0.349 \quad z_2 := w_2 \cdot x + u_2 = -0.478 \quad z_3 := w_3 \cdot x + u_3 = -2.781$

$\sigma(z_1) := \left[1 + \exp(-z_1)\right]^{-1} = 0.586375$

$\sigma(z_2) := \left[1 + \exp(-z_2)\right]^{-1} = 0.382821$

$\sigma(z_3) := \left[1 + \exp(-z_3)\right]^{-1} = 0.05836$

$N := v_1 \cdot \sigma(z_1) + v_2 \cdot \sigma(z_2) + v_3 \cdot \sigma(z_3) = -1.622483$

$y_p(x) := 1 - x + 2 \cdot x^2 + x^3 \cdot N = 0.377517$

$y_d(x) := 9 \cdot e^{-x} - 8 \cdot e^{-2 \cdot x} + 3 \cdot e^{-3 \cdot x} + x - 3 = 0.377594$

$E := 0.5 \cdot (y_d(x) - y_p(x))^2 = 2.969 \times 10^{-9}$

Finally, we consider the initial value problem;

$$y''' - 5y'' - 2y' + 24y = x^2 e^{3x}; y(0) = 1, \; y'(0) = 1, \; y''(0) = 3, \; x \in [0,1] \tag{25}$$

The trial solution is $y_t(x) = A + Bx + cx^2 + x^3 N(x, p)$. Applying the initial conditions gives $A = 1, B = 1, C = 3/2$. Therefore,

$$y_t(x) = 1 + x + \frac{3}{2}x^2 + x^3 N(x, p), \quad y_a(x) = e^{-2x} + e^{3x} + e^{4x}.$$

We use excel spreadsheet to find values of $y_a(x)$ at all the x points as displayed in Table 3.5.

Table 4.5 Values of $\left(x, y_a(x) = e^{-2x} + e^{3x} + e^{4x}\right)$ for problem (25)

x	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	3	3.6604	4.718	6.32853	8.7225	12.2386	17.374	24.857	35.758	51.64	74.82

Using SPSS regression model [15], we compute the weights from input layer to hidden layer. From Table 4.6 and Figure 5, the cubic curve shows perfect fit. Using the cubic, we pick our weights from input layer to hidden layer as: $w_{11} = 26.881,\ w_{12} = -78.908,\ w_{13} = 123.643$.

Table 4.6 Model Summary and Parameter Estimates (SPSS Output) for problem (25)
Dependent Variable: Y

	Model Summary	Parameter Estimates			
Equation	**R Square**	**Constant**	**b1**	**b2**	**b3**
Linear	.801	-9.255	62.714		
Quadratic	.982	6.728	-43.843	106.557	
Cubic	.999	2.277	26.881	-78.908	123.643

The independent variable is X

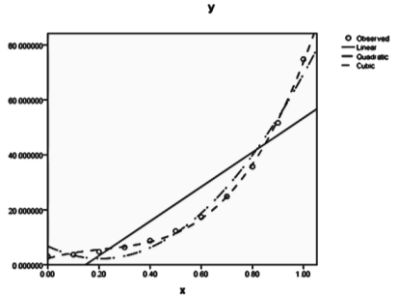

Fig. 5 Polynomial Curve – Fit Regression of $y_a(x) = e^{-2x} + e^{3x} + e^{4x}$

In order to compute the weights from hidden layer to the output layer, we form

$$F(x) = x^2 e^{3x}, \quad f(x) = [0.020086, 0.072885, 0.221364]^T, \tag{26}$$

obtained from Table 4.5, and $v = \phi^{-1} f$. Hence, the weights from the hidden layer to the output layer are

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 1 & 0.94 & 0.78 \\ 0.94 & 1 & 0.94 \\ 0.78 & 0.94 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0.020086 \\ 0.072885 \\ 0.221364 \end{bmatrix} = \begin{bmatrix} 3.622 \\ -7.596 \\ 4.537 \end{bmatrix} \tag{27}$$

The weights from the hidden layer to the output layer are $v_1 = 3.622, v_2 = -7.596, v_3 = 4.537$.

The biases are fixed between –5 and 30. We now train the network with the available parameters using MathCAD 14 software as follows:

$w_1 := 3.622 \quad w_2 := -7.596 \quad w_3 := 4.537 \quad x := 1$

$v_1 := 26.881 \quad v_2 := -78.908 \quad v_3 := 123.643 \quad u_1 := 0.925 \quad u_2 := 30 \quad u_3 := -3.807$

$z_1 := w_1 \cdot x + u_1 = 4.547 \quad z_2 := w_2 \cdot x + u_2 = 22.404 \quad z_3 := w_3 \cdot x + u_3 = 0.73$

$\sigma(z_1) := \left[1 + \exp(-z_1)\right]^{-1} = 0.98952$

$\sigma(z_2) := \left[1 + \exp(-z_2)\right]^{-1} = 1$

$\sigma(z_3) := \left[1 + \exp(-z_3)\right]^{-1} = 0.674805$

$N := v_1 \cdot \sigma(z_1) + v_2 \cdot \sigma(z_2) + v_3 \cdot \sigma(z_3) = 31.126026$

$y_p(x) := 1 + x + \frac{3}{2} \cdot x^2 + x^3 \cdot N = 34.626026$

$y_d(x) := 0.769 \cdot e^{4 \cdot x} + 0.196 \cdot e^{3 \cdot x} - \frac{1}{375}\left(x \cdot e^{-3 \cdot x}\right)\left(25 \cdot x^2 + 60 \cdot x + 126\right) = 34.626017$

$E := 0.5 \cdot \left(y_d(x) - y_p(x)\right)^2 = 3.789 \times 10^{-11}$

5.0Simulations

We present the graphical profile of the accuracy of our method of solution as compared with the analytical. The conventional method, as mentioned earlier, involves complicated multivariate partial derivatives, while adjusting the parameters (weights and biases). This becomes even more apparent when second and higher order differential equations are involved. Therefore, our novel approach reduces this complexity to the barest minimum by eliminating completely weight adjustments. Table 5.1 and Figure 6 compares the analytic result of problem (19) with our neural network result, and there was satisfactory agreement. Likewise Table 5.2, Table 5.3, Figure 7 and Figure 8 for problems (22) and (25) respectively.

Table 5.1 Comparison of the results of Problem (19)

Input Data (X)	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Y Exact	4	3.5675	0.8823	-1.479	-1.2259	1.5408	4.4232	4.9324	2.7784	0.1383	-0.343
Y Pred	4	3.5675	0.8823	-1.479	-1.2259	1.5408	4.4233	4.9325	2.7784	0.1383	-0.343

Table 5.2 Comparison of the results of Problem (22)

Input Data (X)	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Y Exact	1	0.9162	0.8525	0.7966	0.7418	0.6851	0.6257	0.5639	0.5009	0.4384	0.3776
Y Pred	1	0.9161	0.8525	0.7965	0.7418	0.6852	0.6256	0.5639	0.5009	0.4383	0.3775

Table 5.3 Comparison of the results of Problem (25)

Input Data (X)	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Y Exact	1	1.3928	1.9569	2.7667	3.9299	5.6039	8.0179	11.5075	16.5651	23.9151	34.626
Y Pred	1	1.2655	1.9569	2.7667	3.9299	5.6039	8.0179	11.5074	16.5651	23.9151	34.626

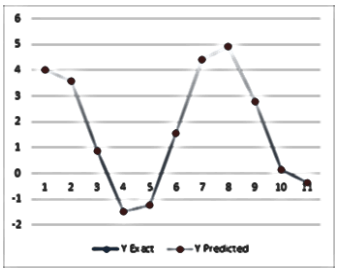

Fig. 6Y Exact and Y Predicted for problem (19)

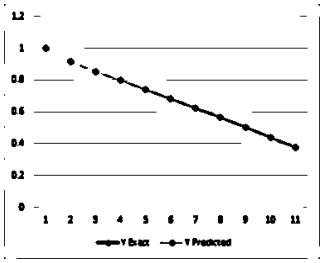

Fig. 7 Y Exact and Y Predicted for problem (22)

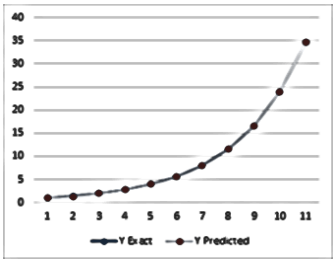

Fig. 8 Y exact and Y predicted for problem (25)

6.0Conclusion

In this paper we extended the novel approach developed in [8] to third order homogeneous and non-homogeneous linear ordinary differential equations. We employed Excel spreadsheet, IBM SPSS 23, and MathCAD 14 algorithm to achieve this task. Our result is validated by the excellent approximations achieved in comparison with the exact solutions.

References

[1] Graupe**,** D. (2007). Principles of Artificial Neural Networks (2nd Edition). Advanced Series on Circuits and Systems – World Scientific Publishing Co. Pte. Ltd., Singapore, Vol. 6

[2] Rumelhart, D. E. and McClelland, J. L. (1986). Parallel Distributed Processing, Explorations in the Microstructure of Cognition I and II. MIT Press, Cambridge.

[3] Werbos, P. J. (1974). Beyond Recognition, New Tools for Prediction and Analysis in the Behavioural Sciences. Ph.D. Thesis, Harvard University, Cambridge, M. A.

[4] Lagaris, I. E., Likas A. C. and Fotiadis D.I. (1997). Artificial Neural Network for Solving Ordinary and Partial Differential Equations. arXiv:physics/9705023v1, Greece

[5] Mall, S. and Chakraverty, S. (2013). Comparison of Artificial Neural Network Architecture in Solving Ordinary Differential Equations. Hindawi Publishing
Corporation.

[6] Otadi, M. and Mosleh M. (2011). Numerical Solution of Quadratic Riccati Differential Equations by Neural Network. *Mathematical Sciences*, 5:249-257.

[7] Tian Qi, C., Yulia, R., Jesse, B. and David, D. (2018). Neural Ordinary Differential Equations. arXiv:1806.07366v1, Canada.

[8] Okereke, R. N. (2019). A new perspective to the Solution of ordinary differential Equations using Artificial Neural Networks. Ph.D Dissertation, Mathematics Department, Michael Okpara University of Agriculture Umudike, Nigeria.

[9] Okereke R. N, Maliki O. S, Oruh B. I (2020).A novel method for solving ordinary differential equations with artificial neural networks. Applied Mathematics (AP), (to appear).

[10] Yadav, N., Yadav, A. and Kumar, M. (2015). An Introduction to Neural Network Methods for Differential Equations. Springer.

[11] Principe, J. C., Euliano, N. R., Lefebvre, W. Curt, (1997), Neural and Adaptive Systems: Fundamentals Through Simulation.

[12] Thomas,JR, G. B. and Finney, R. L. (1979). Calculus and Analytic Geometry. Addison – Wesley, London.

[13] Mathcad Version 14 (2007) PTC (Parametric Technology Corporation) Software Products. http://communications@ptc.com.

[14] Erwin, K. (2006). Advanced Engineering Mathematics. 9th Edition, John Wiley and Sons, Singapore.

[15] IBM SPSS Statistics 23 (2015). www.ibm.com.